

# BANCO DE DADOS

AUTOR

**RICARDO BALIEIRO**

1ª EDIÇÃO

SESES

RIO DE JANEIRO 2015



**Estácio**

**Conselho editorial** REGIANE BURGER; ROBERTO PAES; GLADIS LINHARES; KAREN BORTOLOTI;  
HELCEMARA AFFONSO DE SOUZA

**Autor do original** GERALDO HENRIQUE NETO

**Projeto editorial** ROBERTO PAES

**Coordenação de produção** GLADIS LINHARES

**Coordenação de produção EaD** KAREN FERNANDA BORTOLOTI

**Projeto gráfico** PAULO VITOR BASTOS

**Diagramação** BFS MEDIA

**Revisão linguística** ROSELI CANTALOGO COUTO

**Revisão de conteúdo** RICARDO BALIEIRO

**Imagem de capa** SEMISATCH | DREAMSTIME.COM

Todos os direitos reservados. Nenhuma parte desta obra pode ser reproduzida ou transmitida por quaisquer meios (eletrônico ou mecânico, incluindo fotocópia e gravação) ou arquivada em qualquer sistema ou banco de dados sem permissão escrita da Editora. Copyright SESES, 2015.

#### Dados Internacionais de Catalogação na Publicação (CIP)

B186B BALIEIRO, RICARDO

Banco de dados / Ricardo Balieiro.

Rio de Janeiro : SESES, 2015.

168 p. : IL.

ISBN: 978-85-60923-93-9

1. Banco de dados. 2. Modelagem de dados. 3. Projeto conceitual de dados.
4. Diagrama entidade-relacionamento. I. SESES. II. Estácio.

CDD 005.1

Diretoria de Ensino — Fábrica de Conhecimento  
Rua do Bispo, 83, bloco F, Campus João Uchôa  
Rio Comprido — Rio de Janeiro — RJ — CEP 20261-063

# Sumário

Prefácio	7
1. Introdução aos Sistemas de Banco de Dados	9
Objetivos	10
1.1 Aprender a Diferença Entre Dado e Informação	11
1.1.1 Dado	12
1.1.2 Informação	12
1.2 Conceituar Banco de Dados, Sistema Gerenciador de Bancos de Dados e Sistemas de Bancos de Dados	16
1.3 Analisar as Diferenças Entre Sistemas de Bancos de Dados e Sistemas Baseados em Arquivos	18
1.3.1 Principais problemas de gerenciamento de dados do sistema de arquivos	20
1.4 Conhecer a História e a Evolução dos Bancos de Dados	23
1.5 Partes que Compõem um Banco de Dados	29
1.6 Conhecer os Usuários de um Ambiente de Banco de Dados	30
1.7 Aprender as Principais Características dos SGBDS	31
1.8 Aprender a Decidir Sobre o Emprego ou não de Bancos de Dados	32
1.9 Conhecer Conceitos Fundamentais de um Ambiente com SGBD	33
1.9.1 Conhecer a Arquitetura de 3 Esquemas (Conceitual, Lógico e Físico)	34
1.9.2 Aprender o Conceito e o Processo de Abstração de Dados	35
1.10 Modelagem de Dados	36
1.11 Identificar os Principais Objetos Conceituais (Entidades, Relacionamentos e Atributos)	37
1.12 Aprender a Identificar e Conhecer as Representações Básicas destes Objetos Conceituais	37
1.12.1 Entidade	37

1.12.2 Atributos	38
1.12.3 Relacionamentos	44
1.12.4 Conhecer o Modelo Entidade Relacionamento	45
1.12.5 Aprender a Criar um Modelo para o Negócio	45
Atividades	46
Reflexão	46
Referências bibliográficas	47

## 2. Modelagem Conceitual 49

Objetivos	50
2.1 Definir e Exemplificar os Conceitos de Cardinalidade	51
2.2 Aprender Sobre Limites Mínimos e Máximos	51
2.3 Conhecer as Possibilidades e Critérios para Nomear os Relacionamentos	55
2.4 Aprender Sobre Relacionamentos Recursivos	56
2.5 Aprender Sobre Atributos em Relacionamentos	57
2.6 Conhecer as Extensões do Modelo Entidade Relacionamento: Generalizações e Agregações	57
2.6.1 Especialização	58
2.6.2 Generalização	62
2.6.3 Agrupamento de entidades (entidade associativa)	64
2.7 Aprender Sobre a Modelagem Lógica dos Dados	66
2.8 Conhecer os Modelos Lógicos de Dados Existentes	67
2.9 Aprender a Base Conceitual para Modelo Relacional	71
2.9.1 Conhecer os Conceitos de Chave Candidata, Primária e Estrangeira	74
2.9.2 Compreender as Restrições de Integridade	79
2.9.3 Aprender o Método de Conversão do Modelo Conceitual para o Modelo Relacional	81
Atividades	85
Reflexão	86
Referências bibliográficas	86

### 3. Normalização 89

Objetivos	90
3.1 Aprender o Conceito de Normalização	91
3.1.1 Fases da normalização (Dependência funcional)	92
3.1.2 Problemas de Redundância	99
3.2 Conhecer as Principais Formas Normais:	
1ª Forma Normal (1Fn), 2ª Forma Normal (2Fn), 3ª Forma Normal (3Fn) E Forma Normal De Boyce-Codd	100
3.2.1 Primeira Forma Normal (1FN)	101
3.2.2 Segunda Forma Normal (2FN)	102
3.2.3 Terceira Forma Normal (3FN)	104
3.2.4 Forma Normal de Boyce-Codd	106
3.2.5 Problemas de Decomposições	108
3.3 Aprender Sobre as Linguagens de Manipulação de um Banco de Dados Relacional	109
3.4 Conhecer Conceitos Básicos de Álgebra Relacional	110
Atividades	115
Reflexão	116
Referências bibliográficas	117

### 4. Linguagem SQL (Parte 1) 119

OBJETIVOS	120
4.1 Conhecer a História da Linguagem da Linguagem SQL e suas Principais Características	121
4.2 Aprender os Comandos de Criação de Esquemas	122
4.3 Aprender os Comandos para Criação e Destruição de Tabelas	124
4.4 Inserir, Modificar e Excluir Registros nas Tabelas	135
4.5 Aprender Comandos Básicos para a Recuperação dos Dados Contidos nas Tabelas	140
4.6 Conhecer um Pouco mais Sobre a Linguagem SQL	140
4.7 Aprender Sobre Outros Operadores da Cláusula Where	141
4.8 Aprender Sobre Cláusulas de Ordenação, Cálculos e Apresentação de Dados	144

Atividades	147
Reflexão	147
Referências bibliográficas	148

## 5. Linguagem SQL (Parte 2) 149

Objetivos	150
5.1 Aprender Sobre Sinônimos e Qualificadores	151
5.2 Aprender como Recuperar Dados de mais de uma Tabela Simultaneamente	152
5.3 Aprender Sobre a Utilização e Construção de Consultas Aninhadas	155
5.4 Aprender Sobre o Conceito e as Propriedades das Transações em um Ambiente de Banco de Dados	158
5.5 Aprender a Criar Índices e Visões	160
5.6 Conhecer os Comandos que Tratam da Segurança do Banco de Dados	163
Atividades	164
Reflexão	165
Referências bibliográficas	166

## Gabarito 167

# Prefácio

Prezados(as) alunos(as),

No princípio da produção no mundo, bens de consumo eram construídos por artesãos em suas casas ou oficinas nas quais poucas pessoas trabalhavam. Porém, a sociedade ansiava por mais e com a revolução industrial introduzindo máquinas que faziam sozinhas o serviço de vários artesãos, as empresas começaram a produzir em escala para atender a ânsia da população e, consequentemente, passaram a contratar mais pessoas que não eram necessariamente da família. Essa nova forma de produção exigiu das empresas um novo tipo de organização e novas alternativas para manter, proteger e difundir as informações dentro da empresa. Então vários relatórios contábeis financeiros foram criados para melhor organizar as informações importantes da empresa “moderna”, com isso, cresce a produção em massa e o fator de competitividade entre elas, que passaram a buscar novas formas alternativas de prosperar, como inovação, diferenciação e qualidade dos produtos.

Nesse contexto a informação ganhou uma perspectiva maior dentro da organização passando a ser a atriz principal da ópera sendo responsável direta ou indiretamente pela inovação, diferenciação e qualidade dos produtos. Então armazenar, controlar, compartilhar e proteger informação passou a ser muito importante para as organizações e estas passaram a buscar apoio na Tecnologia da Informação para resolver os seus problemas.

Hoje, aplicações de *software* são construídas para localizar, inserir, alterar, apagar e relacionar dados presentes em um Banco de Dados. Essas aplicações suportam e automatizam os processos que toda a empresa moderna possui.

Veremos a partir de agora, como a área de Gerenciamento de Banco de Dados operacionaliza, gere, esquematiza e possibilita o acesso aos dados das empresas para a assertiva tomada de decisão organizacional.

**Bons estudos!**





# 1

## **Introdução aos Sistemas de Banco de Dados**

Iniciaremos os nossos estudos sobre banco de dados falando sobre os conceitos de dados e informações, bem como o quanto estes são fundamentais e valiosos para as organizações. Estudamos que a informação, tem se tornado o principal ativo das empresas, ajudando na tomada de decisões e, logo, se posicionando como um fator de competitividade.

Entendido a importância das informações nos dias presentes, iremos estudar sobre os Sistemas de Gerenciamento de Banco de Dados, que provêm meios eficientes para a administração dos dados das organizações, possibilitando a constituição de informações íntegras e úteis.

Nos dias presentes, todas as organizações de médio e grande porte, certamente fazem uso de SGBD's para administrarem seu grande contingente de informações, de modo a manterem-nas seguras, íntegras e disponíveis a consultas a qualquer tempo. Mas afinal, qual o papel de um SGBD? Como os SGBDs evoluíram? Quais os tipos de SGBDs? Certamente, você está ansioso por conhecer as respostas destas indagações. Então, vamos aos estudos!



## OBJETIVOS

- Discutir sobre dados e informação, bem como o papel de cada um nos contextos que vivenciamos;
  - Estudar o histórico e processo de evolução dos SGBD's;
  - Verificar as vantagens principais da utilização de um SGBD os problemas por este solucionados;
  - Aprofundar nossos conhecimentos sobre os modelos de dados;
  - Compreender a utilização dos modelos de dados e sua evolução;
  - Estudar os níveis de abstração de dados e sua importância para a constituição de um projeto de banco de dados.
-

## 1.1 Aprender a Diferença Entre Dado e Informação

É de comum conhecimento que vivemos num mundo globalizado; para o processo de globalização foi de fundamental importância o rápido e constante avanço tecnológico, em especial, o uso da tecnologia de informática associada à de telecomunicações. Pode-se destacar nesse ponto um fator importantíssimo: o surgimento e a popularização da Internet, que possibilitou às empresas fazerem parte do comércio eletrônico, não apenas como empresas físicas, mas também como negócios eletrônicos.

Nesse novo modelo social verifica-se um novo paradigma: a Sociedade do Conhecimento. Segundo Peter Drucker (1999), “as atividades que ocupam o lugar central das organizações não são mais aquelas que visam produzir ou distribuir objetos, mas aquelas que produzem e distribuem informação e conhecimento”. A informação passa a ser um patrimônio, algo de valor, e considerada como o principal ativo das empresas. As pessoas e as empresas devem exercitar as suas habilidades de aprender e reaprender, de forma rápida e constante.

Considerando as profundas mudanças que têm ocorrido no meio social, logo, também, no meio organizacional/corporativo, devemos compreender que a Tecnologia da Informação munida com seus equipamentos, programas, recursos de telecomunicações, entre outros, tornou-se não mais apenas um diferencial competitivo para as empresas, mas algo que de forma imprescindível deve estar no dia a dia das mesmas.

Segundo a visão de Laudon e Laudon (2006), com a crescente velocidade dos negócios e a evolução tecnológica no ambiente empresarial, os sistemas de informação tornaram-se ferramentas essenciais para criar empresas competitivas, gerenciar corporações globais e fornecer serviços e produtos úteis aos clientes. Neste cenário de profundas mudanças e inovações observamos uma constante referência a dados e informações, onde, muitas vezes, estes termos são referenciados como sinônimos, o que não é verdade! Vamos de fato compreender sua diferenciação

### 1.1.1 Dado

Em outras palavras dados são fatos brutos, ou seja, que não foram submetidos a nenhum processamento de modo a mostrar seu real significado. A partir destas definições entende-se que dados são apenas uma sequência de símbolos. Dados não transportam nenhum sentido ou significado dos fatos, pois, faltam elementos que propiciem o estabelecimento completo, necessitam de uma estrutura relacional interna para que originem uma função cognitiva.

Para ilustrar melhor o conceito de dados, vamos tomar como exemplo um número decimal qualquer, pois os números, na forma que conhecemos, são símbolos que compõem um sistema numérico na base 10 no qual cada número pode ser representado por um valor.

Um som também pode ser considerado dado. Apesar de o som ser algo contínuo no espaço, é possível quantificá-lo (ou discretizá-lo, ou ainda, dar valor a ele) escolhendo-se vários pontos da onda sonora e atribuindo a eles o valor de sua amplitude, por exemplo.

Essas amostras de pontos das ondas são tratadas como dados (ou sequência de dados), pois cada ponto é um valor discreto e juntos (por meio de processamento matemático) podem formar a onda novamente.

Assim como o som, uma imagem também pode ser considerada um dado, pois apesar da imagem ser uma sequência contínua de cores e texturas (percebidas assim pelo olho humano), ela pode ser subdividida em milhares ou milhões de pontos os quais podem ser quantificados em termos da intensidade das cores vermelho, azul e verde em cada um deles (é possível ver essa quantificação em monitores de vídeo ou ainda em fotos de jornais). Esses pontos são considerados dados, pois foi possível “valorar” toda a imagem por meio dos pontos que a constitui (SETZER & DA SILVA, 2005).

### 1.1.2 Informação

Estudamos até agora que dados por si só não possuem representatividade inteligível.

Será que dados têm algo referente à informação?

Para não deixá-lo muito curioso, vejamos: a informação é o resultado do processamento de dados brutos para revelar seu significado (ROB, 2011). Observe que a informação não possui, de fato, um conceito pronto e definido, haja vista que, esta é o resultado de um processo no qual os dados são submetidos.

O processamento destes dados dá-se de muitas formas, podendo ser simples, desde a análise do modo como os dados são organizados obtendo-se um padrão; ou também, de formas mais complexas, que vão desde a realização de previsões ou a extração de pressupostos fazendo-se uso de modelos estatísticos.

Observe a complexidade envolvida em torno do conceito de informação. Essa complexidade é oriunda de uma gama de processos e fatores que são empregados aos dados brutos de modo a originar a informação significativa. Logo, entendemos que a informação é o resultado da adição de um padrão específico de relações ao dado. Quando fazemos uso da informação obtida, não estamos, apenas, atuando sobre os dados que a originaram, mas também, sobre os padrões coletivos ou individuais de formação e, por meio destes, conseguimos obter a real percepção do quanto uma informação pode ou não ser útil.

Muito pode ser encontrado acerca de informação, o que não é inesperado; vivenciamos uma era onde esta é predominante, deve ser precisa, relevante e rápida, sendo determinante para a tomada de decisões estratégicas. Sabemos o quanto a assertividade de uma decisão é importante para entidades organizacionais, uma vez que estas inferirão na sobrevivência comercial. A capacidade de deter informação com as características ora descritas, contribuem de maneira sem precedentes para a tomada de decisões (ROB, 2005).

É difícil definir a palavra informação. Alguns autores dizem que esta é o resultado do processamento do dado. Porém, o conceito de processamento do dado fica um pouco complicado de ser explicado, então vamos considerar informação como o dado com algum significado para quem o analisa. Para que melhor compreendamos as conceituações, observe a tabela 1.1

WINDOWS				
	MASSA 1GB		MASSA 2GB	
	Inserção	Recuperação	Inserção	Recuperação
PostgreSQL	219754.6667	190265.6667	405001.3333	365854.3333
Oracle	209185.3333	124847.6667	5277267	241922.3333
Diferença	10569.33333	65418	-122265.6667	123932
	0.048096059	0.343824512	-0.301889541	0.338746842
	10.56933333	64.418	122265.6667	
			0.23188568	

LINUX				
	MASSA 1GB		MASSA 2GB	
	Inserção	Recuperação	Inserção	Recuperação
PostgreSQL	195184.3333	143377.3333	363737.3333	292033.3333
Oracle	178945.3333	172419	416393	344760.6667
	16239	-29041.66667	-52655.66667	-52727.33333
		29041.66667	52655.66667	52727.33333
	0.083198276	0.16843658	0.126456657	0.152938947

Tabela 1.1 – Dados brutos.

A figura 1.1 nos apresenta uma série de dados obtidos de um teste de desempenho entre bancos de dados. Observe o quanto é difícil extrairmos alguma compreensão (informação) apenas visualizando esta figura. Para que estes dados tenham significância deverão ser submetidos a um processamento, de modo a gerar significância e, em consequência, nossa compreensão. A figura 1.1 demonstrará o resultado dos dados trabalhados:

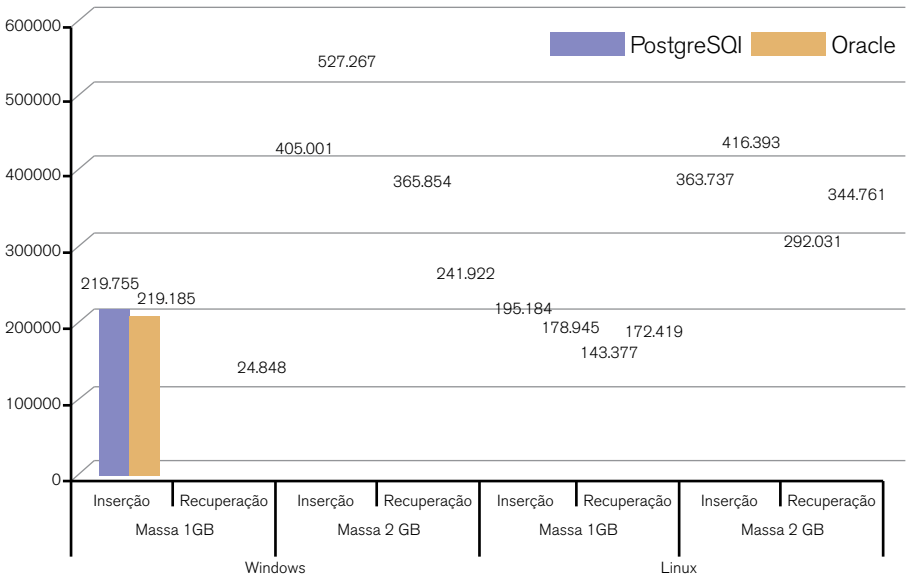


Figura 1.1 – Informação em formato de gráficos.

Veja como ficou muito mais fácil a compreensão da figura 1.2; sem muito esforço observamos que são informações relativas a banco de dados (conforme legenda do gráfico) referentes a operações de inserção e recuperação de dados em distintos sistemas operacionais. Observe que a informação pode ser originada de diversas maneiras, seja uma forma textual, imagens, gráficos, etc.

Neste cenário, os dados brutos representados na tabela 1.1 sofreram algum processamento e, como resultantes, obtivemos informação em forma de gráfico.

## CONEXÃO

Leia sobre a relação: Dados-Informação-Conhecimento em: <http://www.ime.usp.br/~vwsetzer/dado-info-Folha.html>

Este outro artigo trata sobre Inteligência Competitiva das Organizações, pautando-se na relação que estudamos: <http://www8.fgv.br/bibliodata/geral/docs/Intcomp.pdf>

A informação tem sua significância no contexto de diminuir uma incerteza. Num meio qualquer onde uma incerteza vem a ser vivenciada, em relação a alguma coisa, o detentor da informação tem condições de discernir de forma racional e razoável sua dúvida (incerteza), até um denominador comum.

Veja que este trecho traduz o que realmente aconteceu ao visualizarmos a tabela 1.1 e a figura 1.1, onde, respectivamente, não tínhamos certeza (dúvida) quanto a que se referiam aqueles dados brutos e, num segundo momento, pelo esboço do resultado do processamento daqueles dados, foi possível uma melhor compreensão do contexto em que estavam inseridos (ZIKMUND, 2000).

Vimos o quanto as organizações são dependentes das informações para seu processo de negócio; desse modo, para que esta seja considerada, de fato, como informação significativa, ela deve se enquadrar numa destas categorizações:

ATUAL	O valor da informação dependerá em grande parte da sua atualidade. Dado o dinamismo verificado em todos os setores da sociedade em geral e do ambiente empresarial em particular, o período de validade da informação é cada vez mais curto. Torna-se necessário dispor de fontes de informação que acompanhem continuamente essas modificações. Só com base em informação atualizada se podem tomar decisões acertadas.
CORRETA	Não basta que a informação seja atual, é também necessário que, na medida do possível, seja rigorosa. Só com informação correta se pode decidir com confiança.

RELEVANTE	<p>Dado o grande número de volume de informação envolvida, o processo de tomada de decisão, ao contrário de ser facilitado, pode ser dificultado pelo excesso de informação. A informação deve ser devidamente filtrada de tal forma que apenas aquela com relevância para cada situação seja considerada.</p>
DISPONÍVEL	<p>Ainda que a informação verifique os três requisitos anteriores, a sua utilidade poderá ser posta em causa se não puder ser disponibilizada de forma imediata, no momento em que é solicitada. As decisões muito ponderadas, com o longo período de gestação, são cada vez mais, situações do passado. Atualmente, considerando as características do meio envolvente, o processo de tomada de decisão tem que ser quase instantâneo. Para isso, a informação tem que ser disponibilizada rapidamente, caso contrário deixa de ser útil.</p>
LEGÍVEL	<p>Esta condição, apesar de apresentada em último lugar não é, por isso, menos importante. A informação só é informação se puder ser interpretada. De fato, de nada vale que a informação seja atual, precisa, relevante e disponibilizada em tempo oportuno se não puder ser entendida. A forma como é disponibilizada tem também grande importância informação só é informação se puder ser interpretada. A forma como é disponibilizada tem também grande importância.</p>

## 1.2 Conceituar Banco de Dados, Sistema Gerenciador de Bancos de Dados e Sistemas de Bancos de Dados

A programação de aplicações em computadores sofreu profundas modificações desde seus primórdios. No início, quando se usava linguagens como COBOL, Basic, C e outras, os programadores incorporavam em um programa



toda funcionalidade desejada. O programa continha as operações da interface de usuário, as transformações de dados e cálculos, as operações de armazenamento de dados, bem como as tarefas de comunicação com outros sistemas e programas. (HEUSER, 1998).

Um bando de dados nada mais é que uma coleção de dados persistentes, dados estes que são utilizados por aplicações empresariais (DATE, 2003).

Sabemos que para gerenciarmos dados de determinados segmentos necessitamos de um banco de dados computacional, este pode ser entendido como uma estrutura computacional compartilhada e integrada que armazena um conjunto de dados (ROB, 2005).

Um SGBD é um conjunto de programas que gerenciam a estrutura de um banco de dados e controlam o acesso aos dados armazenados. O SGBD é um intermediário entre o usuário e o banco de dados. Sua estrutura é armazenada como um conjunto de arquivos e o único modo de acessar os dados nesses arquivos é por meio do SGBD (ROB, 2005).

Observe que todas as definições em torno de banco de dados referem-se apenas como uma coleção de dados, ou seja, um agrupamento de informações. Devemos entender que banco de dados e Sistemas de Gerenciamento de Bancos de Dados (SGBD) são diferentes.

Isto é, podemos entender que um Sistema Gerenciador de Banco de Dados (SGBD) é uma coleção de programas que permite aos usuários criarem e manterem um banco de dados. Foi definido para ser um sistema de *software* de propósito geral que facilitasse os processos de definição, construção, manipulação e compartilhamento de bancos de dados entre vários usuários e aplicações.

Com as exigências e necessidades de novos conceitos e estruturas nos sistemas de arquivos, percebeu-se um aumento nos custos de equipamentos, manutenção excessiva e tempo de trabalho, solucionando assim, diversos problemas. Dessa forma, os Sistemas de Gerenciamento de Banco de Dados (SGBD) surgiram como uma evolução dos sistemas de arquivos, criando novas estruturas de dados com o objetivo de gerenciar todo o armazenamento de informações.

Assim, podemos entender por SGBD como sendo um sistema computado-rizado de manutenção de registro de dados composto por vários programas especializados em gerenciar dados. Atualmente há vários SGBDs comerciais sendo utilizados em várias aplicações e por várias empresas, a saber: PostgreSQL, Firebird, MySQL, Sybase, Oracle, SQL-Server entre outros.

O SGBD solucionou problemas dos sistemas de arquivos, tais como: integração de dados (mesmo local), redução de dados duplicados, independência de dados e aplicativos e representação das perspectivas do usuário de forma transparente.

Com a adoção dos SGBDs foi possível retirar o gerenciamento de dados do escopo dos programas de aplicação, centralizando essa tarefa em um único sistema especialista. Assim, qualquer programa novo que fosse desenvolvido e que precisasse acessar os dados do sistema, não teria que se preocupar em programar a estrutura ou a forma de escrita e leitura desses dados fisicamente. Os novos programas só deveriam se preocupar em “se conectar” nos SGBDs e, por meio de uma linguagem de acesso a dados não complexa, fazer as operações que fossem necessárias.

## 1.3 Analisar as Diferenças Entre Sistemas de Bancos de Dados e Sistemas Baseados em Arquivos

No passado, antes do Banco de Dados Relacional (conceito que discutiremos nos próximos temas), os próprios programas eram responsáveis pelo gerenciamento dos seus dados, ou seja, a estrutura dos dados estava incorporada aos programas que os acessavam.

Antigamente, (e até hoje em dia em sistemas legados) dados eram armazenados em arquivos do sistema operacional. Programas de aplicação tinham de ser desenvolvidos para manipular esses arquivos com o intuito de consultar, acrescentar novos dados, apagar, alterar, entre outros. A esses programas dava-se o nome de Sistema de Processamento de Arquivos (SILBERSCHATZ, KORTH, & SUDARSHAN, 2006).

As bases de dados eram formadas por inúmeros arquivos denominados Sistemas de Arquivos. Nesse sistema, os arquivos se relacionavam por meio de estruturas de ponteiros que apontavam para a localização física do dado nos arquivos, sendo que esses ponteiros eram gerenciados por programas de aplicação (IBM RESEARCH NEWS, 2008).

Sabemos que nos dias atuais este tipo de armazenamento já está ultrapassado, sendo assim, por que devemos estudar estes mecanismos?

Simples! Se compreendermos as características destes sistemas, que são relativamente simples, aprenderemos mais fácil a manipular complexos projetos de bancos de dados; ao entendermos os inúmeros problemas comuns nos sistemas de arquivos, conseguiremos evitar situações semelhantes; outro ponto importante é uma possível necessidade de migração dos sistemas de arquivos para um sistema de banco de dados.

Quando a utilização de sistemas de arquivos fazia-se presente na maioria dos casos, em pequenas organizações, por exemplo, era até rastrear-se dados necessários utilizando um sistema de arquivos manual. Esse sistema nada mais era que uma composição de pastas de arquivos, devidamente rotuladas. Assim todos os segmentos e departamentos das organizações procediam para gerirem seus arquivos.

Vamos pensar que, até onde o conjunto de dados rotulados por estas pastas físicas, dispostas em prateleiras ou armários, era relativamente pequeno, facilmente podia-se gerir as informações e de modo (talvez) até eficiente. Mas com o crescimento das organizações, as demandas por relatórios gerenciais também cresceram, não só a demanda quantitativa por relatórios, mas também o nível de complexidade das informações que neles deveriam estar contidas. Neste cenário de maior demanda, menores prazos e um volume crescente de arquivos a serem mantidos, a confecção desses relatórios tornara-se mais e mais trabalhosa, além de demorada. Imagine outro cenário, comumente as organizações devem apresentar algum tipo de relatório junto aos governos; imagine a confecção destes neste cenário, dramático, não?

Bem, na verdade, não é tão simples assim. Lembre-se que nesta fase de transição, os computadores eram muito menos amigáveis do que aqueles que conhecemos hoje! Logo, era necessário um profissional altamente especializado para a realização de tal trabalho, foi onde surgiram os especialistas em processamento de dados. Este profissional seria o criador das estruturas de arquivos computacionais necessárias, escrevendo o *software* que gerenciava os dados dentro dessas estruturas.

Num primeiro momento, os sistemas de arquivos computacionais eram muito similares àqueles manuais. Utilizando o meio computacional e em posse desses arquivos, o profissional em processamento de dados deveria escrever programas que permitissem, por exemplo, a emissão de resumos e verificações mensais na carteira de clientes, devedores, possibilidade de mala direta para contatar os clientes para divulgação de comunicados e/ou novos produtos.

Pois bem, até aí tudo parecia perfeito, mas com o passar do tempo as empresas começaram a necessitar de programas adicionais que emitissem relatórios. Embora muito esforço tenha sido empenhado na migração dos sistemas manuais para os computacionais, logo, os gestores começaram a perceber a eficiência destes sistemas, principalmente quanto a melhores e rápidas buscas.

### 1.3.1 Principais problemas de gerenciamento de dados do sistema de arquivos

Vimos que os sistemas de arquivos trouxeram grandes ganhos para as organizações, principalmente quando o gerenciamento de dados foi aperfeiçoado se comparado ao sistema manual. Os sistemas de gerenciamento de dados pós aos sistemas manuais permaneceram ativos por muito tempo, aproximadamente duas décadas, o que no meio tecnológico é muito tempo. Com o transcorrer dos anos, muitos problemas foram vivenciados.

Este sistema de armazenamento implica muitos problemas, por exemplo, tarefas de relativa simplicidade como a recuperação de dados, exige uma significativa quantidade de programação. Neste cenário, o programador deveria criar regras e métodos para quaisquer manipulações (ROB, 2005).

Esse tipo de abordagem, segundo (SILBERSCHATZ, KORTH, & SUDARSHAN, 2006), além de ser custosa para ser desenvolvida e mantida, também apresentava várias inconsistências, a saber:

- **Redundância e inconsistência de dados** – como os dados estão espalhados em vários arquivos do sistema operacional e são acessados e mantidos pelo sistema de processamento de arquivos, o qual é formado pelos mais diversos tipos de programas de aplicação, é muito difícil garantir que um determinado dado não estará sendo repetido nos arquivos. Informações repetidas, além de gerarem altos custos de armazenamento, também podem ocasionar a inconsistência de dados, pois como a informação está repetida, ela pode ser atualizada em um local e não ser atualizada no outro.

- **Dificuldade de acesso a dados** – imagine que você trabalhe em uma empresa que usa o Sistema de Processamento de Arquivo e o seu chefe peça uma lista com todos os *e-mails* das pessoas que fazem aniversário em março. Como o sistema de processamento de arquivo em que você está trabalhando não prevê originalmente este tipo de consulta, você tem duas opções: ou gerar uma lista com todos os clientes ou então pedir para um programador fazer um novo programa de aplicação que execute a pesquisa acima. Não há uma forma simples de se acessar dados, principalmente quando o acesso que se quer fazer é um que não havia sido pensado anteriormente.

- **Problemas de integridade** – como os dados estão espalhados em arquivos do sistema operacional, fica sob responsabilidade das aplicações fazer a gerência da integridade dos dados. Imagine uma aplicação escolar na qual os professores façam atribuição de notas para os alunos. “Nota” é um campo que deve permitir valores de 0 a 10. Esse tipo de restrição do dado é algo que não se consegue impor em um arquivo do sistema operacional, então quem trata essa restrição é a própria aplicação escolar. Imagine agora que a escola decidiu que as notas não são mais de 0 a 10 e sim de 0 a 100. Para implementar essa alteração, várias horas de desenvolvimento serão necessárias para identificar e alterar todos os programas do sistema de processamento de arquivos da aplicação acadêmica em questão que gerenciam essa restrição dos dados. Além disso, como são muitos sistemas postando dados no sistema de arquivos, ainda podemos incorrer na duplicidade de informação, ou seja, dois ou mais dados iguais colocados em lugares diferentes. Assim, para manter a integridade dos dados, qualquer alteração em um desses dados duplicados deve ser repetida em todos os outros.

- **Problemas de atomicidade** – pense na seguinte situação: você acorda em um sábado pela manhã e descobre que durante o *happy hour* de sexta-feira à noite você gastou todo o seu dinheiro que estava na sua carteira e não sobrou nada para o seu café da manhã. Então, imediatamente você pega o seu cartão de banco e dirige-se até a agência bancária mais próxima e vai até um caixa de autoatendimento. Chegando lá, você inicia uma operação de saque e quando a máquina iria começar a contar o dinheiro (ou seja, você já praticamente finalizou a operação, mas não retirou o dinheiro ainda) a energia acaba. O que acontece com o seu saldo? A resposta é: NADA. Porque a aplicação bancária de saque (ou transferência, entre outras) possui uma característica chamada: atomicidade de operação; em outras palavras o sistema da aplicação bancária

considera essas operações como única, não existindo a possibilidade dela ser executada pela metade. Se por qualquer motivo, em um dos passos da operação houver um problema, o sistema faz o estorno de todos os passos da operação executados até o momento. Assim fica parecendo que a operação nunca foi executada. Em sistemas de arquivos, gerenciar operações atômicas é algo extremamente complexo, uma vez que todo o gerenciamento fica no escopo das aplicações que normalmente são compostas por vários programas diferentes desenvolvidos em linguagens diferentes e por pessoas diferentes. Garantir que todos esses programas estejam de fato se preocupando com a atomicidade de uma operação é algo complicado.

- **Anomalias de acesso concorrente** – vamos pensar em outra situação, vamos imaginar que você conseguiu levantar um capital e com ele você acabou adquirindo uma franquia de vendas de passagens rodoviárias. Porém, essa franquia trabalha com um sistema de processamento de arquivos para gerenciar as operações de vendas de passagens entre todas as franqueadas e quando algum novo franqueado entra na organização, este deve contratar um desenvolvedor para implementar o *software* de aplicação que irá acessar os arquivos do sistema de processamento de arquivos da franquia para realizar as vendas de passagens da nova loja franqueada. Com o sistema já desenvolvido pelo seu programador, você inicia as suas vendas. Logo na primeira venda, a sua loja, que fica localizada na cidade de Araraquara-SP, recebe um grupo de 10 pessoas querendo comprar passagens para São Paulo. Ao mesmo tempo, a franquia da cidade de São Carlos recebe mais 13 pessoas também querendo ir para São Paulo. Acontece que o ônibus que faz a linha Araraquara-São Paulo é o mesmo que faz a linha São Carlos-São Paulo e no horário que esse grupo de pessoas quer comprar a passagem há somente 15 lugares. A loja de Araraquara começa o processo de vendas e recupera do sistema de arquivos da franquia a informação de que há 15 lugares vagos no ônibus; ao mesmo tempo, a loja de São Carlos também inicia o seu processo de venda e recupera a mesma informação de lugares livres igual a 15; em seguida, a loja de Araraquara realiza a venda e atualiza a nova quantidade de lugares livres para 5 no sistema de arquivo; posteriormente, a loja de São Carlos também finaliza o processo de venda e atualiza para 2 a quantidade de lugares livres no ônibus. Ao final das duas transações de compras, percebemos que foram vendidas 23 passagens de um ônibus cujo número de vagas disponíveis era igual a 15 e como se não fosse o bastante, ainda teremos 2 vagas livres para serem vendidas. Para evitar esse tipo de situação,

o sistema de arquivo precisaria manter algum tipo de supervisão de acesso simultâneo a dados, porém implementar esse sistema de supervisão é muito complicado em sistemas que utilizam arquivos, pois, mais uma vez, temos que lembrar que são vários programas de aplicações diferentes e não coordenados acessando e atualizando os dados.

- **Problemas de segurança** – há dados em uma organização que só podem ser vistos por determinadas pessoas/áreas da organização, esse é o caso, por exemplo, do dado salário, ou seja, apenas o departamento de Recursos Humanos deveria ter acesso ao dado salário. No entanto, quando trata-se de sistemas de processamento de arquivos, temos que lembrar, novamente, que os programas de aplicação são acrescentados de maneira não coordenada e em tempos diferentes, o que dificulta impor essa condição de acesso a todos eles.

## 1.4 Conhecer a História e a Evolução dos Bancos de Dados

Igualmente a muitas tecnologias na computação industrial, os fundamentos de bancos de dados relacionais surgiram na empresa IBM, nas décadas de 1960 e 1970, por meio de pesquisas de funções de automação das tarefas de escritório. Foi durante um período da história na qual empresas descobriram que estava muito custoso empregar um número grande de pessoas para fazer trabalhos manuais, relativos a administração de arquivos, pastas, e afins. Por este motivo, esforços e investimentos em pesquisas para um meio mais barato para realizar tais tarefas, e ter uma solução mais eficiente.

Muitas pesquisas foram conduzidas durante este período, cujos modelos hierárquicos, de rede e relacionais e outros modelos foram descobertos, bem como muita tecnologia utilizada hoje em dia.

Em 1970 um pesquisador da IBM – Ted Codd – publicou o primeiro artigo sobre bancos de dados relacionais. Este artigo tratava sobre o uso de cálculo e a álgebra relacional para permitir que usuários não técnicos armazenassem e recuperassem grande quantidade de informações. Codd visionava um sistema onde o usuário seria capaz de acessar as informações por meio de comandos em inglês, onde as informações estariam armazenadas em tabelas.

Devido ao caráter técnico do artigo publicado por Codd, bem como a complexidade matemática, o significado e o propósito do artigo não foram prontamente realizados. Entretanto, ele levou a IBM a montar um grupo de pesquisa conhecido como System R (Sistema R).



## CONEXÃO

Leia um pouco mais sobre a história e evolução dos SGBDs e da SQL:

[http://poleon.if.ufrgs.br/~leon/Livro\\_3\\_ed/node116.html](http://poleon.if.ufrgs.br/~leon/Livro_3_ed/node116.html)

[http://algot.dcc.ufla.br/~heitor/Artigos/Artigo\\_008.html](http://algot.dcc.ufla.br/~heitor/Artigos/Artigo_008.html)

O projeto do Sistema R era criar um sistema de banco de dados relacional o qual eventualmente se tornaria um produto. Os primeiros protótipos foram utilizados por muitas organizações. Novas versões foram testadas com empresas aviação para rastreamento do manufaturamento de estoque.

Eventualmente o Sistema R evoluiu para SQL/DS, o qual posteriormente tornou-se o DB2. A linguagem criada pelo grupo do Sistema R foi a Structured Query Language (SQL) – Linguagem de Consulta Estruturada). Esta linguagem tornou-se um padrão na indústria para bancos de dados relacionais e hoje em dia é um padrão ISO (*International Organization for Standardization*). A ISO é a Organização Internacional de Padronização.

O primeiro sistema de banco de dados construído baseado nos padrões SQL começaram a aparecer no início dos anos 80 com a empresa Oracle, com o Oracle 2 e depois com a IBM com o SQL/DS, servindo como sistema e repositório de informações para outras empresas.

Estes sistemas somente nasceram a partir da insistência de um jornal técnico em utilizar BNF para SQL e este jornal publicou tal artigo. BNF é o conjunto de sintaxes de linguagem de computador que explica exatamente como cada comando interage com os outros comandos e o que pode ou não ser realizado, como os comandos são formados em assim por diante. Por causa da publicação deste artigo, empresas puderam utilizá-lo para modelar seus próprios sistemas, os quais seriam 100%.

O *software* de banco de dados relacionais foi sendo refinado durante a década de 80. Neste período, os usuários destes sistemas faziam, devido ao desenvolvimento de sistemas para novas indústrias e ao aumento do uso de computadores pessoais e sistemas distribuídos.



Desde sua chegada, os bancos de dados têm tido aumento nos dados de armazenamento, desde os 8 MB (*Megabytes*) até centenas de *Terabytes* de dados em listas de *e-mail*, informações sobre consumidores, sobre produtos, vídeos, informações geográficas, etc. Com este aumento de volume de dados, os sistemas de bancos de dados em operação também sofreram aumento em seu tamanho. Um dos projetos mais ambiciosos de banco de dados está ainda em construção no CERN. A ideia é criar um banco de dados distribuído com a capacidade de armazenamento de *Hexabytes* ( $1 \text{ Hexabyte} = 1,000 \text{ Petabytes} = 1 * 10^{18} \text{ Bytes}$ ) de dados.

O padrão SQL passou da IBM para a ANSI (*American National Standards Institute*) – Instituto Nacional Americano para Padrões – e para a ISO, os quais formaram um grupo de trabalho para continuar o desenvolvimento. Este desenvolvimento ainda acontece com outras novas versões de padrões definidos.

Vamos ver a evolução ao longo das décadas:

#### • Década de 60

Os computadores se tornam parte efetiva do custo das empresas juntamente com o crescimento da capacidade de armazenamento. Foram desenvolvidos dois principais modelos de dados: modelo em rede (CODASYL – *Comitee for Data Systems Language*) e o modelo hierárquico (IMS – *Information Management System*). O acesso ao BD é feito por meio de operações de ponteiros de baixo nível, ou seja, uma espécie de “*link*” para os registros.

Detalhes de armazenamento dependiam do tipo de informação a ser armazenada, desta forma, a adição de um campo extra, necessitava de uma reescrita dos fundamentos de acesso/modificação do esquema. Os usuários precisavam conhecer a estrutura física do BD para poder realizar uma consulta.

#### • Década de 70

Edgar Frank Codd propõe o modelo de dados relacional, entre 1970 e 1972, que se tornou um marco em como pensar em banco de dados. Ele desconectou a estrutura lógica do banco de dados do método de armazenamento físico. Este sistema se tornou padrão desde então.

O Dr. Peter Chen propõe o modelo Entidade-Relacionamento (ER) para projetos de banco de dados dando uma nova e importante percepção dos conceitos de modelos de dados. Assim como as linguagens de alto nível, a modelagem ER possibilita ao projetista concentrar-se apenas na utilização dos dados, sem se preocupar com estrutura lógica de tabelas.

Muitas discussões a respeito do valor da competição entre os sistemas enquanto a teoria de banco de dados conduz ao objetivo final de projeto de pesquisa. Dois principais protótipos de sistema relacional foram desenvolvidos entre 1974 e 1977 e demonstram um ótimo exemplo de como a teoria conduz a boas práticas.

- **Década de 80**

No início dos anos 80, a comercialização de sistemas relacionais começa a ganhar adeptos e aumentar a demanda entre as organizações. Na metade dos anos 80, a Linguagem Estruturada de Consulta > SQL (*Structured Query Language*) se torna um padrão mundial. A IBM transforma o DB2 como carro chefe da empresa em produtos para BD. Os modelos em rede e hierárquico passam a ficar em segundo plano praticamente sem desenvolvimentos utilizando seus conceitos, porém muitos sistemas legados continuam em uso. O desenvolvimento do IBM PC desperta muitas empresas e produtos de BD como: RIM, RBASE 5000, PARADOX, OS/2Database Manager, Dbase III e IV (mais tarde transformado em FoxBase).

- **Década de 90**

No início dos anos 90, um indício de crise econômica nas indústrias e algumas empresas sobrevivem oferecendo alguns produtos a custos muito elevados. Muito desenvolvimento acontece em ferramentas de desenvolvimento para o *desktop* no desenvolvimento de aplicações (*clienttolls*), tais como: Power-Builder (Sybase), Oracle Developer, Visual Basic (Microsoft), entre outros.

O modelo cliente-servidor (*client-server*) passa a ser uma regra para futuras decisões de negócio e vemos o desenvolvimento de ferramentas de produtividade como Excel/Access (Microsoft) e ODBC, também são marcados como o início dos protótipos de *ObjectDatabase Management Systems* (ODBMS). Na metade dos anos 90, foi quando viu-se a explosão da Internet./WWW e uma louca corrida para prover acesso remoto a sistemas de computadores com dados legados.

Percebe-se um crescimento exponencial na tecnologia Web/BD. Aumentam o uso de soluções de código aberto (*open source*) através de gcc, cgi, Apache, MySQL, etc. Processos de transação em tempo real (OLTP - *On-Line TransactionProcess*) e processos analíticos em tempo real (OLAP >*On-Line AnaliticalProcess*) atingem maturidade por meio de muitos negócios utilizando

os PDVs (Ponto de Venda). No final dos anos 90, o grande investimento em empresas de Internet impulsiona as vendas de ferramentas para conexão Web/Internet/BD. Active Server Pages, Front Page, Java Servlets, JDBC, Enterprise Java Beans, ColdFusion, Dream Weaver, Oracle Developer 2000, são um exemplo dessas ferramentas (SILBERCHATZ, 1998).

- **SGBD's**

No início, as primeiras arquiteturas faziam uso de computadores de grande porte, os famosos *mainframes*, estes grandes computadores eram necessários para executar o processamento principal e todas as funções do sistema, incluindo os programas aplicativos, os programas de interface com o usuário, bem como a funcionalidade dos SGBDs. Por essa razão, a maioria dos usuários acessavam os sistemas via terminais que não possuíam poder de processamento, tinham apenas a capacidade de visualização. Todos os processamentos eram realizados remotamente, ou seja, apenas as informações a serem visualizadas e os controles eram enviados do *mainframe* para os terminais de visualização, conectados a ele por redes de comunicação.

Nos dias presentes, temos visto o barateamento de equipamentos de hardware e sua capacidade de processamento/armazenamento, aumentado. Assim, muitos usuários trocaram seus terminais por computadores pessoais. No começo, os SGBDs usavam esses computadores da mesma maneira que utilizavam os terminais, ou seja, o SGBD era centralizado e toda sua funcionalidade, execução de programas aplicativos e processamento da interface do usuário eram executados em apenas uma máquina (ROB, 2005).

Com o passar dos tempos os SGBDs começaram a explorar a disponibilidade do de processamento disponível do lado usuário; neste momento uma divisão no processamento foi aplicada para que aplicativos ficassem no cliente e o gerenciamento dos dados em um servidor, o que levou à arquitetura cliente-servidor.

Neste contexto, a arquitetura cliente-servidor foi incorporada aos SGBDs comerciais. E diferentes técnicas foram propostas para se implementar essa arquitetura. A mais adotada pelos Sistemas Gerenciadores de Banco de Dados Relacionais (SGBDR), que é a inclusão da funcionalidade de um SGBD centralizado no lado do servidor. As consultas e a funcionalidade transacional permanecem no servidor, o qual é denominado servidor de consulta ou servidor de transação. É dessa forma que um servidor SQL é visto por uma máquina cliente. Cada máquina cliente tem que formular suas consultas SQL, prover a interface do usuário e as funções de

interface utilizando uma linguagem de programação (ELMASRI, 2005).

Vamos resumir algumas das principais arquiteturas dos SGBDs:

- **Plataformas centralizadas:** na arquitetura centralizada, existe um computador com grande capacidade de processamento, o qual é o hospedeiro (host) do SGBD e emulador para os vários aplicativos. Essa arquitetura tem como principal vantagem permitir que muitos usuários manipulem grande volume de dados e sua principal desvantagem está no alto custo, pois exige ambiente especial para mainframes e soluções centralizadas.

- **Sistemas de Computador Pessoal – PC:** os computadores pessoais trabalham em sistema standalone, ou seja, realizam seus processamentos sozinhos. No início da implantação destes sistemas, o processamento era bastante limitado, porém, com a evolução do hardware, temos, atualmente, PCs com grande capacidade de processamento. Esta arquitetura é caracterizada pela simplicidade.

- **Banco de Dados Cliente-Servidor:** na arquitetura cliente-servidor, o cliente (front\_end) executa as tarefas do aplicativo, ou seja, fornece a interface do usuário (tela, e processamento de entrada e saída). O servidor (back\_end) executa as consultas no DBMS (Database Management System ou Sistema de Gerenciamento de Banco de Dados) e retorna os resultados ao cliente. Apesar de ser uma arquitetura bastante popular, são necessárias soluções sofisticadas de *software* para garantir todas as funcionalidades. A principal vantagem dessa arquitetura é a divisão do processamento entre dois sistemas, o que reduz o tráfego de dados na rede.

- **Bancos de dados Distribuídos:** com o volume crescente de dados e a evolução das redes de computadores surgiram os bancos de dados distribuídos. Banco de dados distribuídos é composto por diversas bases de dados, logicamente relacionadas e fisicamente distribuídas por uma rede de computadores.

- **Datawarehouse:** é considerado um banco de dados o qual armazena dados correntes e históricos de potencial interesse para os responsáveis pela tomada de decisão de toda a empresa (diretoria, executivos, gerentes funcionais, etc.). Normalmente, esses dados são extraídos de diversos sistemas de informação, como sistemas de vendas, contas de clientes e manufatura, podendo ainda incluir dados oriundos de transações web. O Data warehouse consegue padronizar informações extraídas de distintos bancos de dados existentes em diversos sistemas de informação, permitindo que essas informações sejam integradas,

e utilizadas por toda a empresa, possibilitando que os responsáveis apliquem análise gerenciais e tomada de decisões para obter excelência operacional.

- **Banco web:** os bancos de dados via Web integram os sistemas de gerenciamento de bancos de dados (SGBDs) com a Web, suprimindo desta forma o aumento da demanda por informação e formas mais eficiente de busca e armazenamento de dados.



## CONEXÃO

Recomendamos a leitura deste artigo para melhor compreensão das arquiteturas de um SGBD: <http://www.devmedia.com.br/arquitetura-de-um-sgbd/25007>

---

## 1.5 Partes que Compõem um Banco de Dados

A figura 1.2, ilustra as partes que compõe um *software* SGBD. As principais partes são:

- **Compilador DDL:** processa as definições do esquema feitas pelo usuário. Estas definições englobam, por exemplo, os nomes e tamanhos de arquivos do banco de dados nomes e tipos de itens de dados e seus armazenamento, mapeamentos entre esquemas e restrições, etc.

- **Processador do BD em tempo de execução:** executa comandos de atualização e recuperação de dados, como também o controle do acesso a estes dados em tempo de execução.

- **Compilador de consulta:** é responsável pela análise sintática dos comandos SQL feitas pelo usuário ou programa. Caso esteja tudo correto, então gera um código de acesso e o envia para ser executado ao processador do banco de dados em tempo de execução.

- **Catálogo:** é responsável por armazenar o resultado do processamento do compilador DDL e fornece-las aos demais módulos do SGBD quando forem solicitadas.

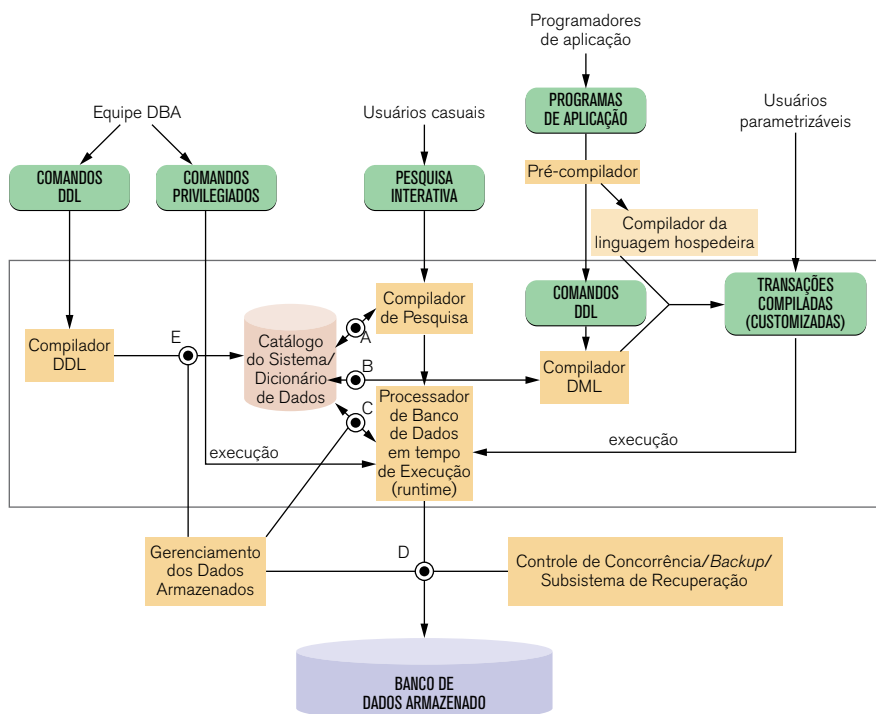


Figura 1.2 – Componentes de um SGBD Fonte: ELMASRI (2005).

## 1.6 Conhecer os Usuários de um Ambiente de Banco de Dados

Administrador de Banco de Dados (DBA) é o responsável pelo controle centralizado dos dados e das aplicações que acessam estes dados. Desta forma é responsabilidade do DBA efetuar as definições e atualização de esquemas do banco de dados (tabelas, índices, etc.) como também fornecer autorização de acesso para usuários e aplicações às diversas partes de um SGBD.

Os usuários de bancos de dados podem ser classificados em quatro tipos:

- **Usuários Programadores:** são desenvolvedores de sistemas computacionais que interagem com os SGBDs através de chamadas DML escritas na linguagem hospedeiras, tais como, VB.NET, Java, C, etc.

- **Usuários Sofisticados:** são usuários avançados com conhecimento em linguagem DML. Desta forma fazem acesso aos SGBDs sem a necessidade de escrever programas em uma linguagem hospedeira.
- **Usuários Especialistas:** são usuários que desenvolvem aplicações consideradas não tradicionais. Dentro deste tipo de aplicações estão sistemas de base de conhecimento, projetos auxiliados por computador, etc.
- **Usuários navegantes:** são usuários comuns que interagem com os SGBS através de aplicativos diversos.

## 1.7 Aprender as Principais Características dos SGBDS

O uso de um SGBD é altamente recomendável em qualquer ambiente que careça de manipulação de dados de forma confiável e íntegra. Dentre as características e vantagens do uso de um SGBD temos:

- **Aprimoramento e compartilhamento de dados:** o SGBD oferece um ambiente em que os usuários finais tenham melhor acesso aos dados, o que compreende a grandes volumes de dados e uma melhor gerência destes.
- **Aprimoramento da segurança de dados:** quanto maior for a quantidade de usuários que acessam os dados, maior será os riscos quanto a segurança. Com o uso de SGBDs é possível um modelo para melhor aplicar as políticas de segurança da organização, onde este cria um sistema de segurança que garante a segurança e nível de usuário e a privacidade dos dados. Regras de segurança podem ser criadas, definindo-se então, quais usuários podem acessar o banco de dados e, quais itens são possíveis serem acessados por este usuário. É possível restringir até as operações que determinado usuário pode realizar junto ao banco de dados.
- **Centralização dos dados:** a centralização dos dados é, por certo, um fator de significativa importância, pois esta permite que todos os dados sejam integrados a um único sistema, reduzindo-se assim as redundâncias e fazendo com que a administração dos dados seja mais eficiente (SUMATHI, 2010).
- **Flexibilidade:** possíveis alterações na estrutura de uma base de dados podem ser necessárias para atendimento de novos requisitos de negócio. Por exemplo, um novo grupo de usuários pode surgir com necessidade de informações adicionais, ainda não disponíveis na base de dados. Alguns SGBD's

permitem que tais mudanças na estrutura da base de dados sejam realizadas sem afetar a maioria dos programas de aplicações existentes;

- **Compartilhamento de Dados:** SGBD's multiusuários devem fornecer controle de concorrência para assegurar que atualizações simultâneas resultem em modificações corretas. Um outro mecanismo que permite a noção de compartilhamento de dados em um SGBD multiusuário é a facilidade de definir visões de usuário, que é usada para especificar a porção da base de dados que é de interesse para um grupo particular de usuários;

- **Fornecimento de Múltiplas Interfaces:** devido aos vários tipos de usuários, com variados níveis de conhecimento técnico, um SGBD deve fornecer uma variedade de interfaces atendê-los. Os tipos de interfaces incluem linguagens de consulta para usuários ocasionais, interfaces de linguagem de programação para programadores de aplicações, formulários e interfaces dirigidas por menus para usuários comuns.

- **Gerenciamento e armazenamento de dados:** o SGBD cria e gerencia as estruturas complexas necessárias para o armazenamento de dados, possibilitando que o usuário a focar nas reais necessidades do negócio e não com complexas tarefas e rotinas de armazenamento de dados em baixo nível. Este gerenciamento é importante para o desempenho do banco de dados.

- **Gerenciamento de transações:** uma transação é uma unidade lógica de trabalho; ela começa com a execução de uma operação `BEGIN TRANSACTION` e termina com a execução de uma operação `COMMIT` ou `ROLLBACK`; segundo este princípio, ou todos os registros correlatos são inseridos ou nada é registrado. Desta forma, a consistência e integridade dos dados são asseguradas pelo SGBD (DATE, 2003).

## 1.8 Aprender a Decidir Sobre o Emprego ou não de Bancos de Dados

Devido ao alto custo envolvido na aquisição, implantação e manutenção de um SGBD, é necessário avaliar se o investimento é realmente necessário. O desenvolvimento de uma aplicação envolvendo o armazenamento de dados, nem sempre tem a necessidade de utilizar um banco de dados. Estes casos podem ser:



Quando a aplicação é simples, bem definida e envolve pouca quantidade de dados.

Quando esta aplicação não terá mudanças significativas que podem impactar sobre os dados armazenados.

Quando não há a necessidade de múltiplos acessos concorrentes aos dados armazenados.

## 1.9 Conhecer Conceitos Fundamentais de um Ambiente com SGBD

Os SGBDs funcionam como servidores de dados e oferecem uma interface de comunicação para os programas de aplicação poderem “conversar” com ele.

Por meio dessa interface, os programas de aplicação podem solicitar a inclusão, alteração, busca, criação ou organização dos dados que tiverem sendo mantidos pelo SGBDs. Essas interfaces de comunicação normalmente são configuradas em drivers ou biblioteca de funções.

Então é correto dizer que o SGBD está no centro de todo o processo de negócio da organização?

Certamente que sim! As organizações atualmente vivem pautadas em suas informações e no processamento delas. O bom gerenciamento destas informações e tratamento das mesmas permitem boas tomadas de decisões. Assim, já que a informação é um bem de valor inestimável e todo este contingente informacional está sendo gerenciado pelo SGBD, logo, podemos dizer que o SGBD é o “coração” para a maioria das organizações.

Para melhor entendermos como um SGBD atua, observe a figura 1.3:

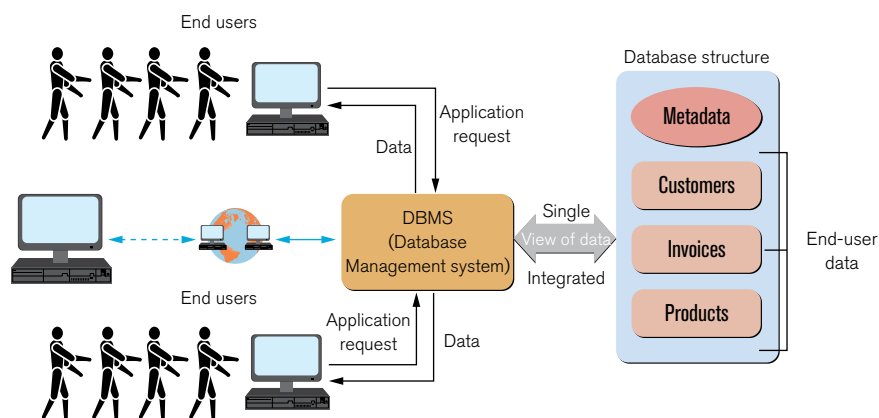


Figura 1.3 – Sistema de Gerenciamento de Banco de Dados. Fonte: (ROB, 2005 p. 8).

Observe que a Figura 4 demonstra que o SGBD é o elo central de acessos dos usuários e/ou aplicativos aos dados. Neste cenário, o SGBD recebe todas as solicitações das aplicações e as traduz nas operações complexas necessárias para o acesso. Dessa forma, grande parte da complexidade de acesso às fontes de dados são encapsuladas pelo SGBD, minimizando, assim, o esforço de implementação nas aplicações clientes.

### 1.9.1 Conhecer a Arquitetura de 3 Esquemas (Conceitual, Lógico e Físico)

Um problema para diversos usuários, principalmente aqueles que necessitam de informação para a tomada de decisão é o volume cada vez maior dos dados para serem analisados para efetuar uma tomada de decisão. Outros profissionais que interagem diretamente com o projeto e manutenção de banco de dados necessitam de forma que facilitem a visualização e manutenção dos dados. A solução para este cenário é a visão abstrata dos dados proporcionada por um SGBD. A abstração proporciona a visualização de determinados detalhes e oculta outras, sobre a forma como os dados são mantidos e armazenados. Quanto maior o nível de detalhamento menos abstrato é a representação dos dados. Diminuindo o nível de detalhamento, maior é a abstração dos dados.

Um SGBD proporciona a cada tipo de usuário uma representação conceitual dos dados permitindo uma melhor interação entre os usuários e o banco de dados. A abstração de dados pode ser classificada em três níveis:

- **Nível físico:** este nível representa como os dados são realmente armazenados no banco de dados. Devido ao grande detalhamento, este nível é considerado de baixa abstração.

- **Nível lógico:** este nível descreve como os dados se relacionam e são armazenados no banco de dados. Uma característica importante deste nível é proporcionar a organização dos dados pelos administradores de banco de dados.

- **Nível visão:** este nível proporciona ao usuário acesso as informações no banco de dados sem que tenha a necessidade de conhecer toda a estrutura do banco e sua complexidade.

## 1.9.2 Aprender o Conceito e o Processo de Abstração de Dados

Afinal, o que vem a ser abstração?

O dicionário nos traz algumas definições, vejamos:

1. Ato ou efeito de abstrair ou abstrair-se. 2. Consideração das qualidades independentemente dos objetos a que pertencem. 3. Ideia metafísica, teoria demasiado vaga que não pode receber aplicação.

Pareceu complicado? Mas não é. Basicamente, quando um projetista de banco de dados inicia seu trabalho, ele precisa abstrair os conceitos e necessidades sobre o ambiente de dados, de modo a acrescentar detalhes conforme o projeto esteja mais próximo de sua implementação.

A partir da análise dos requisitos é criado um projeto, chamado de conceitual, representado pelo Modelo Entidade Relacionamento, o qual não contém detalhes de implementação tratando-se, portanto, de um modelo de dados de alto nível e independente do SGBD a ser adotado.

A próxima etapa refere-se à criação do projeto lógico, que é realizada por meio do mapeamento do Modelo Entidade Relacionamento para o Modelo Relacional. A partir dessa fase, já se pensa em um modelo de dados de implementação do SGBD.

E, finalmente, a última etapa corresponde à fase do projeto físico, em que são definidas as estruturas de armazenamento interno, os índices, além de outras atividades que são desenvolvidas paralelamente, tal como a implementação dos programas de aplicação.

## 1.10 Modelagem de Dados

Até este ponto estudamos amplamente as temáticas dados e informação, suas discrepâncias e interdependências. Estudamos também algumas das características de um SGBD e as principais características e recursos que possibilitam um bom gerenciamento dos dados.

Verificamos também o modelo de sistemas de arquivos e muitos dos problemas envolvidos com este tipo de armazenamento de dados.

O SGBD mostrou-se como um ente que se posiciona de forma intermediária entre a base de dados e os programas que consumirão os dados ora armazenados; este possibilita meios para uma melhor gestão destes dados, provendo facilidades de administração e manipulação de sua estrutura.

Para que tenhamos uma boa administração dos dados geridos pelo SGBD escolhido, não basta apenas um bom SGBD, mas o entendimento do modelo de dados, bem como, um projeto de modelagem de dados correto. Logo, para constituirmos um bom projeto de banco de dados, devemos compreender a fundo a modelagem de dados.

Devemos compreender que a modelagem de dados – que é a primeira etapa envolvida no projeto – nada mais é que o processo de criação de um modelo de dados específico tendo como intenção a resolução de um problema cotidiano.

A modelagem de dados é um processo que envolve várias etapas, o que pode ser entendido como iterativo e progressivo. Inicia-se com uma compreensão simples do problema que desejamos sanar, e conforme o grau de entendimento do problema aumenta, o nível de detalhes que a modelagem compreende também (HEUSER, 2004).

## 1.11 Identificar os Principais Objetos Conceituais (Entidades, Relacionamentos e Atributos)

O modelo ER tem por objetivo representar de forma conceitual o banco de dados, de modo a possibilitar a visibilidade desse pelo usuário final. Esse representar objetos do mundo real que conhecemos e que estão envolvidos no contexto de negócio da organização. Ele deve representar elementos, tais como: entidade, atributos e relacionamentos. (ROB, 2011).

O modelo possui conceitos intuitivos que permitem aos projetistas de bancos de dados capturarem os conceitos inerentes aos dados da aplicação, independente de qualquer tecnologia utilizada para desenvolvimento de bancos de dados. O esquema conceitual criado utilizando-se os conceitos do Modelo Entidade Relacionamento é denominado de Diagrama Entidade Relacionamento (DER).

## 1.12 Aprender a Identificar e Conhecer as Representações Básicas Destes Objetos Conceituais

### 1.12.1 Entidade

Uma entidade representa, no modelo conceitual, um conjunto de objetos da realidade modelada. Como o objetivo de um modelo ER é modelar de forma abstrata um banco de dados, interessam-nos somente os objetos sobre os quais desejamos manter informações. Por exemplo, imagine um sistema de informações industrial, alguns exemplos de entidades poderiam ser os produtos, os tipos de produtos, as vendas ou as compras. Em outro cenário, imaginemos um sistema de controle de contas correntes, algumas entidades podem ser os clientes, as contas correntes, os cheques e as agências.

Segundo Heuser (1998, p.23), uma entidade é “um conjunto de objetos da realidade modelada sobre os quais deseja-se manter informações no banco de dados”.

Em resumo, devemos entender uma entidade, no contexto de banco de dados, como sendo uma tabela! Em ambos os modelos, Chen e “pé de galinha”, uma entidade é representada por um retângulo que contém seu nome. Aconselha-se que este nome seja um substantivo, pois, logo irá representar um objeto do mundo real, e, também, ser escrito em maiúsculas.

Em um DER, uma entidade é representada por um retângulo que contém o nome da entidade, conforme demonstra a figura 1.4:



Figura 1.4 – Representação gráfica de entidades.

Como a figura 1.4 nos mostra duas entidades: PESSOA e DEPARTAMENTO, ambas representadas graficamente por um retângulo.

### 1.12.2 Atributos

Antes de falarmos nos atributos, esperamos que você tenha compreendido plenamente o conceito de entidades e como estas devem ser representadas no processo de modelagem.

Vamos lá então!

Os atributos são as características das entidades. Imaginemos a entidade ALUNO, um aluno, no mundo real, tem características, tais como: nome, sobrenome, nascimento, entre outras. Estas características correspondem aos atributos que comporão a entidade ALUNO.

Na prática, atributos não são representados graficamente, para não sobrecarregar os diagramas, já que muitas vezes entidades possuem um grande número de atributos. Prefere-se usar uma representação textual que aparece separadamente do diagrama ER. No caso de ser usado um *software* para construção de modelos ER, o próprio *software* encarrega-se do armazenamento da lista de atributos de cada entidade em um dicionário de dados.

Para melhor visualizarmos a representação de um atributo junto a sua entidade, observe a figura 1.5:

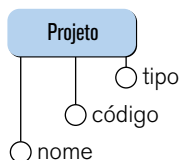


Figura 1.5 – Representação gráfica dos atributos de uma entidade.

Observamos que a Figura 6 é uma entidade que representa PROJETO, verificamos que as características, ou seja, os atributos do projeto são: nome, código e tipo.

Na maioria dos projetos de Banco de Dados implementados na prática, não é usual colocar todos os nomes dos atributos nos Conjuntos de Entidades para evitar poluição do diagrama, uma vez que as entidades costumam conter muitos atributos. Os atributos são representados textualmente em documentos separados como os dicionários de dados. (HEUSER, 2004). Quando modelados graficamente, os nomes dos atributos devem ser escritos em letras maiúsculas e no singular no diagrama. (SETZER & DA SILVA, 2005). No Modelo Entidade Relacionamento original, todo atributo de uma entidade deve conter apenas um único valor elementar, ou seja, uma entidade com atributo de nome igual a “Maria” do Conjunto de Entidades Pessoas só pode conter um CPF. E no caso do atributo Telefone, uma entidade pessoa pode conter apenas um telefone?

(SETZER & DA SILVA, 2005).

#### • Identificador de entidade

No modelo ER os identificadores podem ser entendidos como um atributo determinante. Por exemplo, imaginemos uma entidade Pessoa, que tem como identificador o CPF. Dessa forma, se soubermos o CPF de um indivíduo, poderemos localizar suas demais informações, tais como, nome, endereço, data de nascimento, entre outras. Dessa forma, dizemos que o CPF é determinante aos demais atributos, pois, por ele, podemos obter qualquer outra informação de uma pessoa, neste exemplo. É possível dizer que um identificador é exclusivo, no caso do CPF isto se confirma, haja vista que, não há dois indivíduos que possuem o mesmo número de CPF. Sempre que um modelo for constituído, pense com cautela quando for estabelecer o atributo determinante de sua relação. No

modelo Relacional estes identificadores são mapeados para chaves primárias (*primarykey*) da tabela. Usando a representação simplificada da estrutura e uma tabela, os atributos identificadores devem aparecer em sublinhados, conforme exemplo:

Nome\_tablea (Atributo\_chave1, Atributo\_1, Atributo\_n...)

Vamos imaginar a representação da entidade PESSOA, onde o código deve ser o atributo chave, assim teremos:

Pessoa (Código, Nome, Endereço)

**Refleta:** Como ficaria esta representação no modelo de Chen? É bem fácil também, a figura 1.6 nos apresenta:

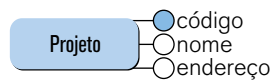


Figura 1.6 – Identificador de entidade. Fonte: Heuser (1998, p. 36).

Viu como é simples? Apenas preenchamos a bolinha do atributo que será o identificador.

#### • Identificadores compostos

**Refleta:** será que uma entidade pode ter seu identificador como sendo apenas um único atributo?

Num primeiro momento podemos pensar que sim, mas há situações que podemos ter identificadores compostos, ou seja, constituídos por mais de um atributo da entidade. Em outras palavras podemos utilizar um identificador composto, isto é, uma chave primária composta, de mais de um atributo.

Vamos imaginar um almoxarifado de uma empresa de ferragens organizado como segue. Os produtos ficam armazenados em prateleiras. Estas prateleiras encontram-se em armários organizados em corredores. Os corredores são numerados sequencialmente a partir de um e as prateleiras são numeradas sequencialmente a partir de um dentro de um corredor. Assim, para identificar uma prateleira é necessário conhecer o Número\_prateleira e o Número\_corredor em que se encontra. Para cada prateleira deseja-se saber sua capacidade em metros cúbicos. (HEUSER, 2005).



A figura 1.7 nos mostra como fica a representação da chave composta:



Figura 1.7 – Representação chave composta.

Observe que a representação não muda em nada com relação ao que já estudamos acerca dos identificadores, apenas devemos preencher as bolinhas dos atributos que compõem a chave composta. Lembre-se que isso teve poucas implicações no ER, mas, a nível conceitual de concepção do banco, haverá grandes mudanças.

#### • Atributos multivalorados

Até este ponto, você já estudou que há a possibilidade de atributos possuírem “sub-atributos”. Vimos, até o momento, que o atributo de uma entidade teve por finalidade representar apenas uma informação única, ou seja, em Pessoa, o atributo Nome tinha apenas um nome por entidade, o Código tinha apenas um por entidade.

Ainda não analisamos situações nas quais um atributo deve possuir mais do que uma informação.

Vamos voltar a uma pergunta que fizemos anteriormente:

Vamos imaginar que precisamos do atributo telefone na entidade Pessoa, deve conter apenas uma informação ou várias? Isto é, uma entidade pessoa qualquer de pessoas terá sempre apenas um telefone? Ou há a possibilidade de uma entidade pessoa ter mais do que um telefone?

Provavelmente você respondeu que sim. Pois uma pessoa no mundo real pode ter vários telefones mesmo, então uma entidade de pessoas também deveria poder ter vários telefones.

Podemos dizer que o atributo telefone é um Atributo Multivalorado e pode assumir vários valores diferentes para uma mesma entidade de Pessoa. Digamos que a pessoa “Luma” está cadastrada em nossa entidade, e esta possui os seguintes telefones: “6363636”, “6369898”, “3636655”.

**Refleta:** Da mesma forma, uma pessoa pode ter mais do que um endereço? Ora, se imaginarmos que uma pessoa possa residir em duas cidades, uma em que estuda durante a semana e outra em que reside aos finais de semana, então ela pode ter dois endereços.

Ou ainda, se imaginarmos que o endereço em questão é o comercial, poderíamos admitir que uma pessoa trabalhasse em mais de um lugar. Então podemos admitir que uma entidade de pessoas tivesse mais do que um endereço. Dessa forma vamos também considerar que o atributo endereço seja multivalorado.

Em um Diagrama Entidade Relacionamento, iremos representar atributos multivalorados por meio da adição do símbolo de asterisco (\*) antes do nome do atributo, conforme é mostrado na figura 1.8, conforme discutimos, chegamos a conclusão de que os atributos ENDEREÇO e TELEFONE devem ser multivalorados, veja como proceder:

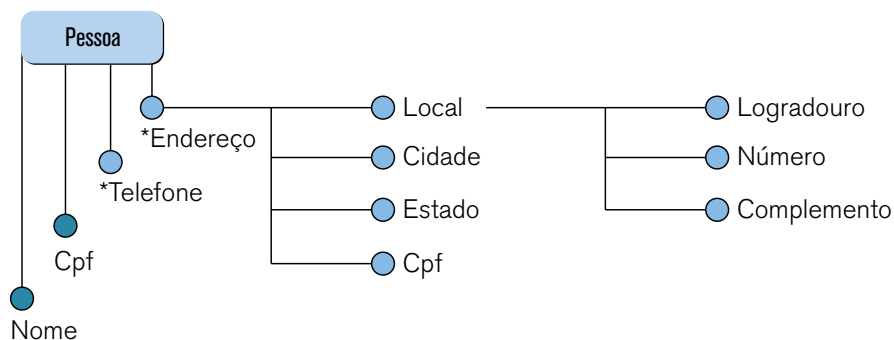


Figura 1.8 – Representação de atributos multivalorados.

#### • Atributos determinantes

Quando estudamos os índices, pudemos observar que há muito esforço envolvido para que as buscas dos dados ora armazenados sejam eficientes, certo? Vamos pensar no modelo computacional, temos que nos preocupar em como localizar uma determinada entidade dentro de um Conjunto de Entidades.

Para isso, devemos eleger um atributo, ou um conjunto de atributos, tal que dado um valor para esse atributo (ou para o conjunto de atributos) teremos apenas uma entidade dentro de um Conjunto de Entidades que o contém. Isto é, não haverá, para um mesmo Conjunto de Entidades, duas entidades que tenham o mesmo valor para esse atributo.

A esse atributo especial, ou conjunto de atributos, daremos o nome de Atributo(s) Determinante(s). (SETZER & DA SILVA, 2005).

No nosso exemplo da entidade PESSOA, que já exemplificamos anteriormente, poderíamos dizer que CPF seria o nosso atributo determinante, uma vez que o CPF não se repete para duas pessoas diferentes. Dessa forma, ao escolhermos um determinado valor de CPF encontraremos no máximo uma entidade com esse valor no Conjunto de Entidades em questão.

Vamos pensar um pouco, pode ser que haja situações que não iremos dispor do CPF, no caso de cadastro de crianças, o que fizermos? Para este caso é comum cadastrar o CPF da mãe ou do pai. Assim, o CPF deixaria de ser um atributo determinante, pois haveria mais de duas entidades diferentes (a mãe e o filho) com o mesmo CPF. O que fazer então?

**Refleta:** O que fazer na situação acima, quando não há um atributo que para um dado valor encontraremos no máximo uma entidade contendo esse valor?

Para resolvermos esse impasse a solução seria elegermos outro atributo, que em conjunto com o CPF, irá compor o atributo determinante. Esse atributo poderia ser Nome, por exemplo.

Ao darmos um valor para Nome e CPF, encontraremos no máximo uma entidade em Pessoas com esses valores. Mesmo que haja vários Nomes iguais ou vários CPFs iguais, não haverá duas entidades com Nome e CPF iguais ao mesmo tempo. Vejamos a figura 1.9:

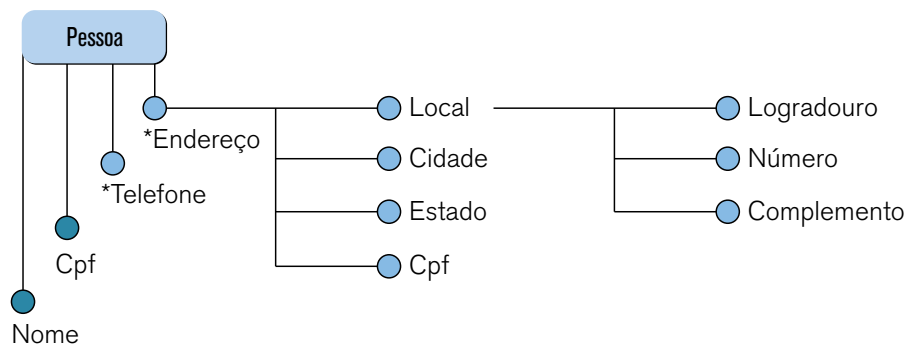


Figura 1.9 – Exemplo de atributos determinantes. Fonte: Setzer& da Silva, (2005).

### 1.12.3 Relacionamentos

Relacionamento é uma associação entre entidades. As entidades que participam de um relacionamento são também conhecidas como participantes.

O relacionamento deve ser nomeado por um verbo, por exemplo: um aluno frequenta uma turma, uma pessoa está vinculada a um departamento, são exemplos de relacionamentos.

Os relacionamentos entre entidades sempre operam em ambas as direções, como isto funciona? Por exemplo, vamos imaginar as entidades Cliente e Fatura, devemos especificar que:

- Um Cliente pode gerar muitas Faturas.
- Cada Fatura é gerada por apenas um Cliente.

Uma vez que conhecemos as direções do relacionamento entre Cliente e Fatura, é fácil verificarmos que este relacionamento é do tipo 1: N (um para muitos).

Antes de nos aprofundarmos nisto, vamos conferir como o relacionamento deve ser representado no modelo DER, vamos ilustrar o relacionamento de acordo com o exemplo que citamos, entre Pessoa e Departamento, conforme a figura 1.10:



Figura 1.10 – Representação gráfica de relacionamento.

E então, como foi sua primeira impressão, acho simples a representação e a identificação do relacionamento?

Acreditamos que sim! Vamos analisar melhor a figura 1.10, onde, a partir do relacionamento entre as entidades torna-se possível referirmos as associações específicas dentro de um conjunto. No caso do relacionamento Associação, uma ocorrência seria um par específico formado por uma determinada ocorrência da entidade Pessoa e por outra da entidade Departamento.

#### 1.12.4 Conhecer o Modelo Entidade Relacionamento

O modelo relacional foi apresentado em 1970 por E. F. Codd em um famoso artigo denominado: “Um modelo relacional de dados para grandes bancos de dados compartilhados”.

Observamos que este modelo foi criado, principalmente, para priorizar grandes volumes de dados, de modo a manter a integridade e consistência entre as mesmas. Segundo Date (2003, p. 93), “o modelo relacional se dedica ao exame de três aspectos principais dos dados: a estrutura dos dados, a manipulação dos dados e a integridade dos dados”, para tanto, faz uso central da matemática.

A constituição deste modelo assemelha-se em muitas das operações similares as financeiras, utilizadas dentro das corporações, são realizadas em forma de tabelas, sendo os dados dispostos em linhas e colunas. A vantagem de utilização deste mecanismo para organização das informações é a facilidade de comparação de uma informação com as demais (TURBAN, 2004).

Como dito, o fundamento desse modelo é dado por conceitos matemáticos, conhecidos como Relação. De modo mais simples, pense numa tabela, com linhas e colunas; cada linha desta tabela (relação) é denominada tupla; cada coluna da tabela é dita como sendo um atributo.

Na década de 70 o trabalho de Codd foi considerado inovador, mas também inviável! Inviável, pois a simplicidade desse modelo gerava uma sobrecarga do computador. Considere que as máquinas daquela época tinham baixíssimo poder computacional se comparado aos dias de hoje!

#### 1.12.5 Aprender a Criar um Modelo para o Negócio

A criação de um modelo para o negócio, ou o modelo conceitual passa por diversas etapas onde se pretende ter uma visão otimizada de como será a estrutura do banco de dados. Um modelo bem projetado permite uma melhor comunicação entre os integrantes da equipe de desenvolvimento do que se pretende fazer, diminuição da complexidade de um problema em partes menores e assim solucioná-los de forma eficiente e facilidade na manutenção do banco de dados. Antes da criação do modelo conceitual, é necessário efetuar o levantamento de requisitos junto ao cliente. Uma vez o levantamento finalizado, passa-se para a fase de criação do modelo conceitual. Com o modelo conceitual pronto, é feito um mapeamento do modelo conceitual para o lógico e em seguida do lógico para o físico.

Iniciamos a criação de um modelo para o negócio estudando o modelo entidade relacionamento. Todo o conhecimento adquirido neste e nos próximos capítulos irão compor uma base sólida para a criação de modelos para o negócio que com certeza serão de grande valia em sua carreira profissional.



## ATIVIDADES

01. Com os dados distribuídos em vários arquivos do sistema operacional é muito difícil garantir que um determinado dado não esteja sendo repetido nos arquivos. Este tipo de problema é classificado como:

- a) Problemas de Atomicidade;
- b) Redundância e Inconsistência de Dados;
- c) Anomalias de Acesso Concorrente;
- d) Integridade referencial;
- e) Chave primária.

02. Quais são as principais vantagens de uma arquitetura de bancos de dados Cliente-Servidor?

03. De acordo com o material e o que foi visto em aula o que é a modelagem de dados?



## REFLEXÃO

Parabéns! Você concluiu uma densa unidade de estudos! Neste capítulo aprendemos como os SGBDs funcionam e o quanto são complexos. Estudamos sobre o histórico e evolução dos SGBDs, as etapas e processos que foram necessários para estes chegarem ao patamar que conhecemos hoje, evoluindo desde os sistemas de arquivos. Pudemos compreender os modelos de dados existentes e como são importantes para a criação de um bom projeto de banco de dados.

Esperamos que você tenha ampliado sua visão acerca dos modelos de dados e suas profundas contribuições para o processo de constituição de um projeto de banco de dados.

Ainda aprendemos sobre as diferentes formas de se encarar um banco de dados de acordo com os seus níveis de abstrações.



## LEITURA

Para você incrementar mais o seu nível de aprendizagem sobre os modelos de dados:

- Projeto de Banco de Dados. HEUSER, Carlos Alberto. 4 ed. Instituto de Informática da UFRGS, Sagra DC Luzzatto, 1998.
- Sistemas de bancos de dados. KORTH, H.; SILBERCHATZ, A. 3. ed. São Paulo: Makron Books, 1998.

Nesse livro você encontra um bom conteúdo para complementar os estudos apresentados pela apostila. Aprofunde seus conhecimentos!

---



## REFERÊNCIAS BIBLIOGRÁFICAS

- DATE, C. J.; **Introdução a Sistemas de Banco de Dados**. 8ª ed. Trad. Daniel Vieira. Rio de Janeiro: Elsevier, 2003.
- ELMASRI, R.; NAVATHE, S. B. **Sistemas de bancos de dados**. São Paulo: Pearson (Addison Wesley), 2005.
- HEUSER, C. A. **Projeto de Bancos de Dados**. 2ª ed. Porto Alegre: Sagra Luzzatto, 1999.
- HEUSER, C. A. **Projeto de Banco de Dados**. 4ª ed. Instituto de Informática da UFRGS, Sagra DC Luzzatto, 1998.
- KORTH, H.; SILBERCHATZ, A. **Sistemas de bancos de dados**. 3ª ed. São Paulo: Makron Books, 1998.
- OLIVEIRA, A. R. F.; TAVEIRA L. M. P.; GILDA A. **Modelagem de Dados**. Rio de Janeiro. Ed. Senac Nacional, 2000.
- PRESSMAN, R. S. **Engenharia de software**. São Paulo: Makron Books, 1995.
- RAMAKRISHNAN, R; GEHRKE, J. **Database management systems**. 2ª ed. Boston: McGraw-Hill, 2000.
- ROB, P. CORONEL, C. **Sistemas de Banco de Dados – projeto, implementação e administração**. Cengage Learning: 2011.
- NETO, G. H.; ANDRADE, M. C.; MARTINS, M. D. C. **Gestão de Redes, Internet, Banco de Dados e Empreendedorismo** Módulo 4.2. Ribeirão Preto: UniSEB Interativo, 2013.
- SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistema de Banco de Dados**. 5ª ed. Rio de Janeiro: Elsevier, 2006.
- TURBAN, E. et al. **Tecnologia da Informação para a Gestão**. 3a. Ed. São Paulo, SP: Bookman, 2004.
-





# 2

## **Modelagem Conceitual**

Para a constituição de um bom projeto de banco de dados, o primeiro estágio é a compreensão apurada dos principais modelos de dados e sua evolução. Os modelos de dados ajudam a abstrair as necessidades organizacionais de forma a contribuir como projetista de banco de dados, bem como a fim de reunir todas as informações necessárias para a compreensão das regras de negócio e especificidades organizacionais.

Iniciamos nossos estudos dos modelos no capítulo anterior e iremos aprofundar nossos conhecimentos em torno dos modelos de dados que compõem os bancos de dados. Mais adiante, vamos estudar sobre os níveis de abstração de dados, saber que cada nível proposto é fundamental para constituirmos bons projetos de bancos de dados; você verá como estes níveis estão interconectados e os quesitos que carecem de atenção para a definição do projeto.



## OBJETIVOS

- Entender os conceitos de cardinalidade.
  - Estudar as extensões do modelo entidade relacionamento.
  - Aprender sobre os modelos lógicos de dados existentes.
  - Aprender a base conceitual para modelo relacional.
  - Conhecer os conceitos de chave.
  - Entender as restrições de integridade.
-

## 2.1 Definir e Exemplificar os Conceitos de Cardinalidade

Você já deve estar afiado quanto aos atributos de uma entidade e sobre como representar o relacionamento entre elas. Iremos nos aprofundar nos detalhes envolvidos nos relacionamentos, caso dúvidas tenham permanecido, retorne aos itens anteriores para reestudá-los.

## 2.2 Aprender Sobre Limites Mínimos e Máximos

Para fins de projeto de banco de dados, uma propriedade importante de um relacionamento é a de quantas ocorrências de uma entidade podem estar associadas a uma determinada ocorrência através do relacionamento. Esta propriedade é chamada de cardinalidade de uma entidade em um relacionamento. Há duas cardinalidades a considerar: a cardinalidade máxima e a cardinalidade mínima. (HEUSER, 2004).

Na vida real, vamos imaginar uma instituição escolar, onde temos Curso e Sala. Pense: concorda que um curso pode acontecer em mais de uma sala ao mesmo tempo? Além disso, uma sala pode ser usada para ministrar vários cursos, lógico que não simultaneamente.

Afinal, onde queremos chegar com isso? Simples, tudo se resume em ocorrências!



Figura 2.1 – Relacionamento Departamento-Pessoa.

A **cardinalidade máxima**: para compreendê-la, vamos tomar como referência o DER da figura 2.1, relacionamento entre Pessoa e Departamento, assim vamos considerar as cardinalidades máximas.

- Entidade Pessoa tem cardinalidade máxima 1 no relacionamento Associação: isso significa que uma ocorrência de PESSOA pode estar associada a no Máximo uma ocorrência de Departamento ou, em outras palavras, que um empregado pode estar associado em no máximo um departamento;
- Entidade Departamento tem cardinalidade máxima 120 no relacionamento Associação: isso significa que uma ocorrência de Departamento pode estar associada à no máximo 120 ocorrências de Pessoa, ou seja, um departamento pode associar apenas 120 pessoas;
- A cardinalidade máxima é 1;
- A cardinalidade máxima ilimitada, usualmente chamada de cardinalidade máxima “muitos” é referida pela letra n ou m.

## CONEXÃO

Leia mais sobre cardinalidades em:

<http://www.devmedia.com.br/tecnologias-de-banco-de-dados-e-modelagem-de-dados-parte-2/1871>

<http://www.gbdi.icmc.usp.br/system/files/Aula11-EstendendoMapeamento.pdf>

---

Resumindo: a quantidade de vezes que uma determinada entidade participa de um Conjunto de Relacionamentos chama-se cardinalidade. A cardinalidade pode ser representada no Diagrama ER por meio da adição de números ou letras que indiquem a quantidade de vezes que uma mesma entidade de um Conjunto de Entidades pode aparecer no Conjunto de Relacionamentos.

Veja como ficará o DER do relacionamento entre Pessoa e Departamento, na figura 2.2:



expressa que a uma ocorrência de EMPREGADO (entidade do lado oposto da anotação) pode estar associada ao máximo uma ("1") ocorrência de DEPARTAMENTO

expressa que a uma ocorrência de DEPARTAMENTO (entidade do lado oposto da anotação) pode estar associada ao máximo uma ("1") ocorrência de EMPREGADO

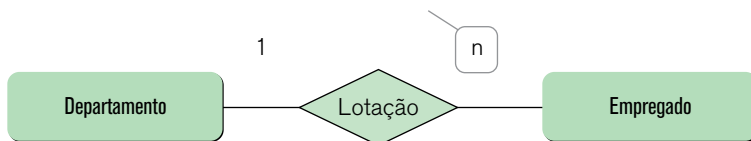


Figura 2.2 – Representação das cardinalidades no DER. (HEUSER, 1998).

Vamos verificar quais os tipos de relacionamentos poderão ocorrer entre duas entidades, o que chamamos de relacionamentos binários:

São os tipos de relacionamentos entre entidades binários:

- **Um-para-um:** uma entidade de EMPREGADO está associada a apenas uma entidade de MESA, conforme figura 2.3:



Figura 2.3 – Relacionamento 1:1.

- **Um-para-muitos:** uma entidade de Curso está associada a muitas entidades de Aluno, entretanto, uma entidade de Aluno está associada a apenas uma entidade de Curso, de acordo com a figura 2.4.



Figura 2.4– Relacionamento 1:N.

- **Muitos-para-muitos:** uma entidade de Médico está associada a qualquer quantidade de entidades de Paciente, e uma entidade de Paciente está associada a qualquer número de entidades de Médico, em conformidade com a figura 2.5



Figura 2.5 – Relacionamento N:N.

Por enquanto, em todos os exemplos que utilizamos, trabalhamos apenas com relacionamentos binários, ou seja, um Conjunto de Relacionamento entre dois Conjuntos de Entidades. Porém, há situações em que relacionamentos binários não serão suficientes.

Você verá que a grande maioria dos relacionamentos com os quais irá trabalhar serão binários, geralmente, a utilização de relacionamentos ternários ou de grau superior são oriundas de alguma liberdade que o projetista teve em relação à resolução de algum problema. Como o próprio nome diz, o relacionamento ternário indica a associação de três entidades diferentes. Vejamos um exemplo prática de relacionamento ternário, conforme a figura 2.6:

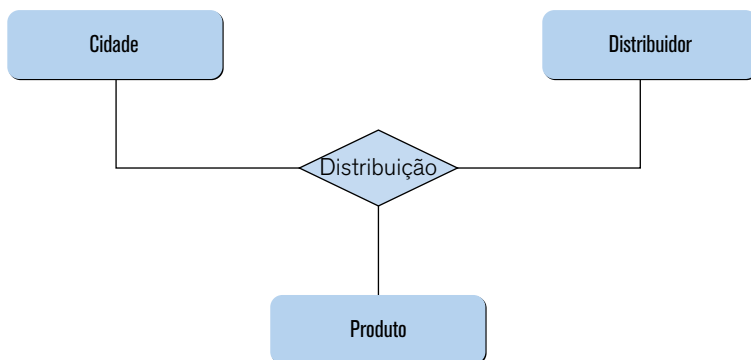


Figura 2.6 – Relacionamento ternário.

Conforme exibido na figura 2.6, cada ocorrência do relacionamento Distribuição associa três ocorrências de entidade: um produto a ser distribuída, uma cidade na qual é feita a distribuição e um distribuidor.

Quando se verificam relacionamentos de grau superiores a 2 (binários), o conceito de cardinalidade de relacionamento é algo não comum ao conceito

de cardinalidade de relacionamentos binários. Em um relacionamento ternário, a cardinalidade refere-se a pares de entidades. Suponha-se a existência de um relacionamento R entre entidade A, B e C, a cardinalidade máxima de A e B dentro de R indica quantas ocorrências de C podem estar associadas a um par de ocorrências de A e B.

Vamos nos aprofundar um pouquinho mais. Vamos observar a figura 2.7, o 1 na linha que liga o retângulo representativo da entidade Distribuidor ao losango representativo do relacionamento expressa que cada par de ocorrências (Cidade, Produto) está associado a no máximo um distribuidor.

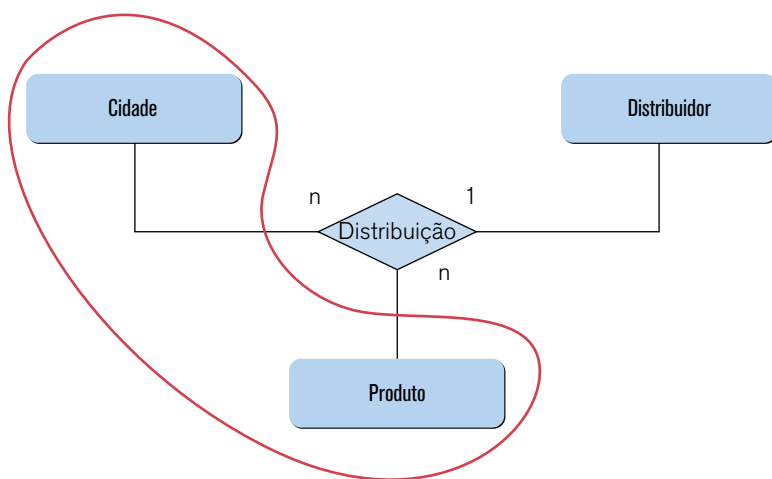


Figura 2.7 – Relacionamento ternário.

## 2.3 Conhecer as Possibilidades e Critérios para Nomear os Relacionamentos

Relações podem ser nomeadas por verbos ou palavras agregadas, como nos exemplos a seguir:

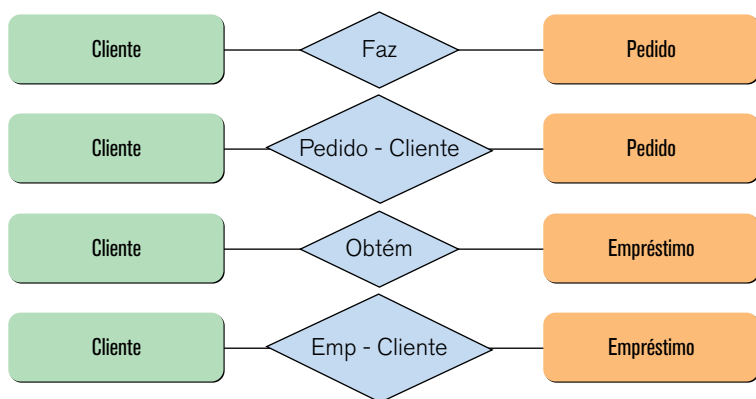


Figura 2.8 – Nome de relacionamentos.

## 2.4 Aprender Sobre Relacionamentos Recursivos

Vamos pensar um pouco, será que um relacionamento só pode ocorrer entre 2 ou mais entidades? Não, ocorre necessariamente entre entidades diferentes, verifica-se a existência do autor-relacionamento, conforme figura 2.9 demonstra, ou seja, é um relacionamento entre ocorrências de uma mesma entidade.

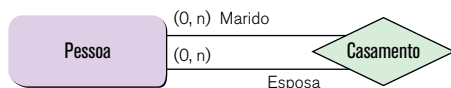


Figura 2.9 – Demonstração do autorrelacionamento.

Sabemos que a modelagem de dados visa atender ao modelo de negócios de uma determinada organização. Dessa forma, em muitas situações é necessário recorrer ao uso do auto-relacionamento; muitas modelagens o contemplam e o adotam para a resolução de determinados requisitos. Para que entendamos este tipo de relacionamento, vamos compreender, primeiramente, o conceito de papel da entidade no relacionamento.

Papel de entidade em relacionamento exerce a função que uma instância da entidade cumpre dentro de uma instância do relacionamento.

No exemplo do relacionamento de Casamento, uma ocorrência de pessoa exerce o papel de marido e a outra ocorrência de pessoa exerce o papel de esposa.



## 2.5 Aprender Sobre Atributos em Relacionamentos

Atributos também podem ser aplicados em relacionamentos. O relacionamento ilustrado na figura 2.10 define três atributos para o relacionamento Consulta. Notem que os atributos Receita e Data\_Consulta não podem ser considerados atributos da entidade Médico ou Paciente. No caso do Médico, não são atributos pelo fato do médico efetuar diversas consultas para pacientes distintos. Também não são de Paciente, já que paciente pode efetuar consultas com diferentes médicos.

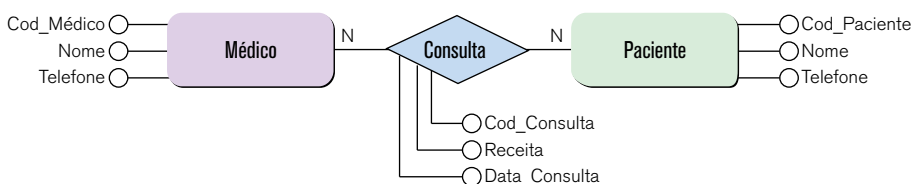


Figura 2.10 – Atributos em relacionamentos.

## 2.6 Conhecer as Extensões do Modelo Entidade Relacionamento: Generalizações e Agregações

À medida que a complexidade das estruturas de dados a serem modeladas aumenta, paralelo as exigências específicas de cada sistema, torna-se necessário o incremento de mais informações no modelo de dados.

O modelo de entidade relacionamento estendido (EERM – *Extended Entity Relationship Model*) adiciona estruturas semânticas ao modelo entidade relacionamento (ER).

## 2.6.1 Especialização

O modelo entidade relacionamento estendido abrange todos os conceitos de modelagem do modelo entidade relacionamento estudados até o presente momento, incluindo conceitos novos como, subclasse e superclasse e os conceitos relacionados de especialização e generalização. A subclasse (também conhecido como subtipo) é nosso primeiro conceito do modelo entidade relacionamento estendido, isto significa que, refere-se a um tipo de entidade específico utilizado para representar uma entidade em questão e ou até mesmo, uma coleção de entidades desse tipo que podemos encontrar em um banco de dados.

Para exemplificar, considere a entidade “funcionário” a qual descreve o tipo (atributos e relacionamento) de cada entidade de funcionário no banco de dados “empresa”.

Na maioria das vezes, um tipo de entidade pode possuir vários sub-grupos ou subtipos de suas entidades que podem ser importantes e necessitam de ser representados adequadamente.

Você consegue identificar as entidades (figura 2.11) que estão relacionadas ao tipo de entidade “funcionário”? A entidade do tipo “funcionário” pode ser caracterizada também por “secretária”, “engenheiro” e “técnico”. Você pode interpretar que, o conjunto de entidades em cada um desses agrupamentos é um subconjunto das entidades que estão associadas ao conjunto de entidades “funcionário”, ou seja, isso significa que cada entidade que, é membro de qualquer um desses subtipos também é um funcionário.

Por sua vez, o tipo de entidade “funcionário” é nomeado de superclasse ou supertipo de cada uma dessas subclasses.

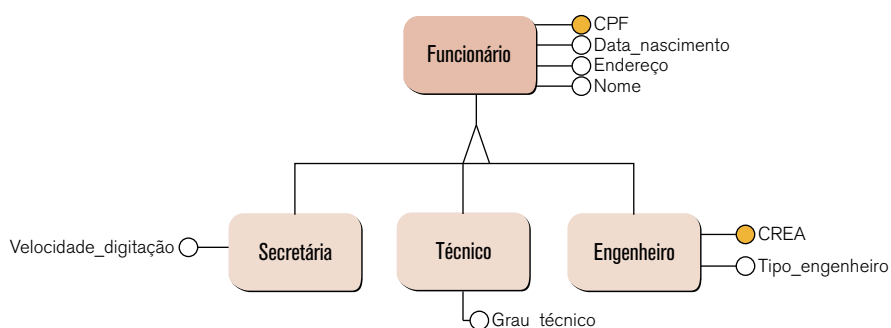


Figura 2.11 – Uso de Generalização/Especialização.

Podemos concluir que especialização como um processo de definir um conjunto de subclasses de um determinado tipo de entidade, isto é, superclasse da especialização. Esse conjunto de subclasses constitui uma especialização baseando-se nas diversas características da superclasse, por exemplo, “secretária”, “engenheiro” e “técnico” é, na verdade, uma especialização da superclasse “funcionário”, que por sua vez, diferencia as entidades do funcionário através do tipo de cargo de cada uma.

Dessa forma, o conjunto de entidades “secretária”, além do atributo considerado específico (atributo local) “velocidade\_digitação”, também conterá “CPF, nome, data de nascimento e endereço”, “herdados” de “funcionários”. (representamos neste diagrama apenas os atributos que nos interessam para não poluirmos demasiadamente o desenho. Também não nos preocupamos aqui com atributos identificadores).

Observe que uma entidade “secretária”, por exemplo, “Márcia Pereira” também é uma entidade “funcionário”, isto significa que, o membro da subclasse é o mesmo que a entidade na superclasse, porém, desempenhando papéis distintos.

Outro detalhe importante é que podemos possuir diversas especializações do mesmo tipo de entidade baseando-se exclusivamente em suas características particulares, por exemplo, outra especialização para o tipo de entidade “funcionário” poderia resultar nas subclasses “funcionário\_mensal” e “funcionário\_horista”, onde, o método de pagamento poderá representar a distinção dos funcionários.

Sabemos que um conjunto de entidades pode incluir sub-agrupamentos de entidades que, de alguma forma, são distintas uma das outras entidades no conjunto. Para exemplificar, imagine um subconjunto de entidades constituindo um conjunto de entidades, as quais podem possuir atributos não compartilhados por todas as entidades que fazem parte desse conjunto. Para representar esses agrupamentos de entidades distintos, o modelo-entidade relacionamento fornece recursos para adequar essa manipulação da melhor forma possível. Vamos imaginar a existência de um conjunto de entidades, ora nomeada de “pessoa”, formada pelos atributos “id\_pessoa, nome, rua e cidade”. Dessa maneira, uma “pessoa” pode ser subclassificada como “cliente” e “funcionário”.

Sendo assim, cada um desses tipos de “pessoa”, caracterizado pelo conjunto de atributos, os quais incluem todos os atributos do conjunto de entidades “pessoa”, mais, eventualmente, os atributos adicionais. Ou seja, as entidades

“cliente” podem também ser descritas incluindo o atributo “avaliação\_crédito”, da mesma maneira, as entidades “funcionário” podem incluir o atributo “salário”. Esse processo de especificar subgrupamentos dentro de um conjunto de entidades é nomeado de especialização

Na verdade, numa extensão do Modelo ER, o projetista pode dizer se há a necessidade de cada entidade do conjunto de entidades genérica aparecer em um dos Conjuntos de Entidades especializados ou não.

Quando cada entidade do conjunto de entidades genérico tiver obrigatoriamente que aparecer como uma entidade em um dos conjuntos de entidades especializados, dizemos que a especialização/generalização é TOTAL.

Ao contrário, quando uma entidade do conjunto de entidades genérico não tiver a obrigatoriedade de aparecer como uma entidade de um dos conjuntos de entidades especializados, dizemos que a especialização/generalização é PARCIAL.

Para representar uma especialização/generalização TOTAL no Diagrama ER basta acrescentar a letra “t” no lado superior direito do triângulo. Já para representar uma especialização/generalização PARCIAL no Diagrama ER basta acrescentar a letra “p” no lado superior direito do triângulo. (HEUSER, 2004).

No exemplo da figura 2.12, a nossa especialização/generalização é TOTAL, pois ou um funcionário é “secretária”, ou “técnico” ou deve ser “engenheiro”. Não existe uma situação na qual ele não seja nem secretária, nem técnico e nem engenheiro (isso decorre da especificação que o nosso cliente nos forneceu no momento da entrevista).

Também é prevista a possibilidade do projetista dizer se uma entidade do conjunto de entidades genérico poderá aparecer em mais de um conjunto de entidades especializados.

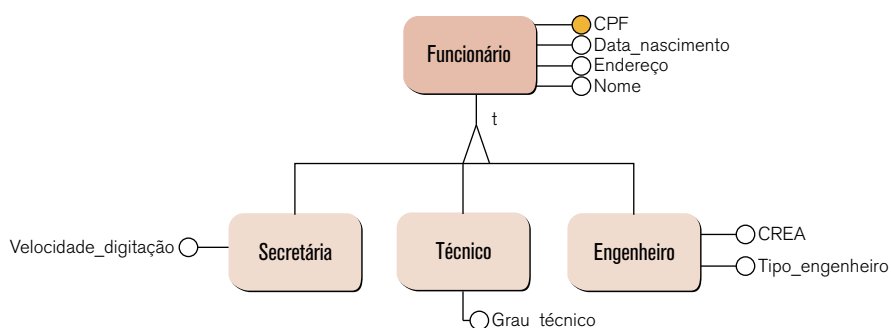


Figura 2.12 – Exemplo da representação de uma Especialização/Generalização TOTAL.

Quando cada entidade do conjunto de entidades genérico tiver obrigatoriamente que aparecer no máximo em uma entidade em um dos conjuntos de entidades especializados, dizemos que a especialização/generalização é EXCLUSIVA.

Ao contrário, quando uma entidade do conjunto de entidades genérico puder aparecer como uma entidade em mais de um dos conjuntos de entidades especializados, dizemos que a especialização/generalização é COMPARTILHADA.

Para representar uma especialização/generalização EXCLUSIVA no Diagrama ER basta acrescentar a letra “e” no lado superior direito do triângulo. Já para representar uma especialização/generalização COMPARTILHADA no Diagrama ER basta acrescentar a letra “c” no lado superior direito do triângulo.

É importante registrar que uma especialização/generalização pode ser ao mesmo tempo EXCLUSIVA e TOTAL ou EXCLUSIVA e PARCIAL, bem como, COMPARTILHADA e TOTAL ou COMPARTILHADA e PARCIAL. Porém, nunca teremos uma especialização/generalização que ao mesmo tempo seja COMPARTILHADA e EXCLUSIVA ou TOTAL e PARCIAL. (HEUSER, 2004).

No exemplo da Figura 23, nada é dito sobre a possibilidade de um funcionário técnico também poder ser um engenheiro. Então vamos assumir aqui que um funcionário técnico poderá sim ser um engenheiro.

Então, podemos dizer que a nossa especialização/generalização é COMPARTILHADA. Dessa forma, o Diagrama ER ficará como mostrado na figura 2.13.

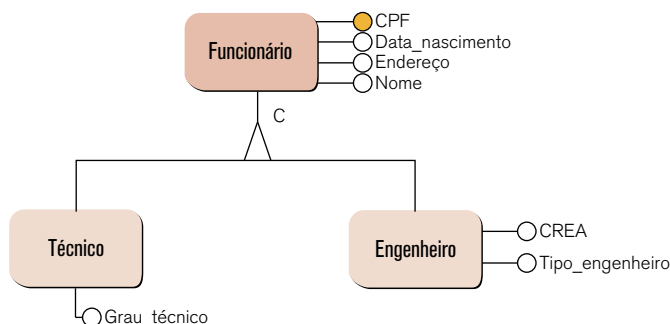


Figura 2.13 – Exemplo de representação de uma especialização/generalização COMPARTILHADA.

Concluindo, a especialização de uma entidade nos permite a definição de um conjunto de subclasses de um determinado tipo de entidade, como também, incluirmos atributos particulares, esses considerados específicos para essas subclasses, definir alguns tipos de relacionamentos exclusivos entre cada subclasse, e até, para outros tipos de entidade e outras subclasses.

## 2.6.2 Generalização

Generalização nada mais é que um processo reverso (de baixo para cima) da abstração em que eliminamos as diferenças existentes entre diversos tipos de entidade, ora identificando suas características comuns e, generalizando em uma única superclasse.

Você ficou confuso com esse conceito? Vamos apresentar um exemplo para tentar esclarecer melhor a generalização.

Imagine dois tipos de entidade, uma identificada de “carro” e outra de “caminhão” visualizada pela figura 2.14. Você pode perceber que existem vários atributos comuns entre as entidades “carro” e “caminhão”, podendo assim serem generalizados no tipo de entidade “veículo”. Agora, tanto “carro” quanto “caminhão” são subclasses da superclasse generalizada “veículo”.

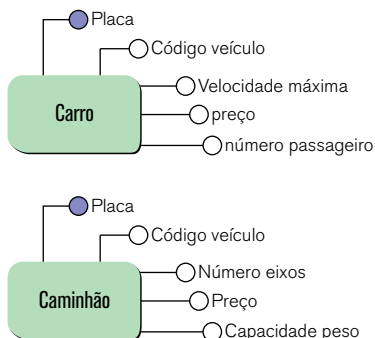


Figura 2.14 – Exemplo de generalização (dois tipos de entidade – CARRO e CAMINHÃO).

Perceba também que agora fizemos o processo inverso do processo de especialização, a fim de alcançarmos a generalização. Podemos ver na Figura 26 que “carro” e “caminhão” é uma especialização de “veículo”, ao invés de “veículo” como uma generalização de “carro” e “caminhão”. Análogo a essa observação, na figura 2.12, podemos ver que a entidade “funcionário” é uma generalização de “secretária”, “técnico” e “engenheiro”.

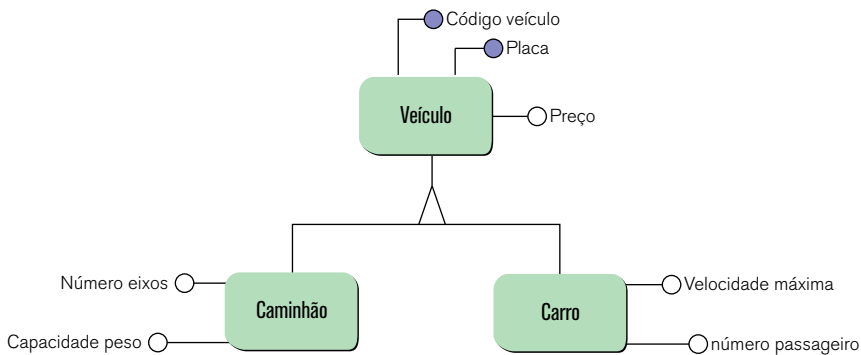


Figura 2.15 – Exemplo de generalização (generalizando CARRO e CAMINHÃO na superclasse VEÍCULO).

Dessa forma, conseguimos finalizar esse tópico utilizando adequadamente a propriedade de herança, a qual nos permite que uma subclasse herde os atributos e relacionamentos comuns da superclasse.

O detalhamento de um conjunto de entidades preliminar em consecutivos níveis de subagrupamentos de entidade caracteriza um processo chamado de “*top down*” (de cima para baixo). Esse processo considera que diversos conjuntos de entidade são resumidos em um conjunto de entidades superior, ora levando em consideração as estruturas básicas comuns.

Para elucidar a ideia, suponha que um DBA, inicialmente, identificou um conjunto de entidades “cliente”, formada pelos atributos “id\_cliente, nome\_cliente, rua\_cliente, cidade\_cliente e avaliação\_crédito”, e, na sequência, o mesmo DBA necessitou de um conjunto de entidades “funcionário”, essa constituída pelos atributos “id\_funcionário, nome\_funcionário, rua\_funcionário, cidade\_funcionário e salário\_funcionário”.

Podemos constatar que existem semelhanças entre o conjunto de entidades “cliente” e o conjunto de entidades “funcionário”, visto que, ambos possuem diversos atributos que representam características comuns entre si (nome, rua e cidade). Essa analogia pode ser representada pela “generalização”, que representa um vínculo existente entre um conjunto de entidades de nível superior e um ou mais conjuntos de entidades de nível inferior.

### 2.6.3 Agrupamento de entidades (entidade associativa)

No desenvolvimento de um projeto de banco de dados, a elaboração de um diagrama entidade relacionamento exige que realizemos descobertas de eventuais tipos de entidade e seus respectivos relacionamentos. O projetista do banco de dados desenvolve um DER inicial que, certamente, sofrerá alterações significativas até que o mesmo alcance os objetivos da regra de negócio sugerida. Ao final, tem-se uma quantidade expressiva de entidades e relacionamentos que se torna o DER de certa forma ilegível e ineficiente. Nesses casos, você poderá realizar o agrupamento de entidades para tentar reduzir o número de entidades apresentadas no DER.

Esse grupo de entidades pode ser considerado como um tipo de entidade “virtual” utilizado para representar diversas entidades e relacionamentos no DER. Tal grupo de entidades é constituído pela combinação de entidades ora relacionadas entre si, e considerado “virtual” ou “abstrato” por não representar efetivamente uma entidade no DER final.

Você já estudou em aulas anteriores que, em alguns casos, torna-se necessário associarmos uma entidade com a ocorrência de um relacionamento específico. Lembre-se de que o modelo entidade relacionamento não permite em hipótese alguma estabelecermos relacionamentos entre relacionamentos, apenas entre entidades. Dessa forma, o objetivo de uma entidade associativa é manipular um relacionamento como se ele fosse uma entidade, conforme você pode observar na figura 2.16 abaixo:



Figura 2.16 – Exemplo de agrupamento de entidades (agregação).

Se, eventualmente, você desejar controlar os medicamentos prescritos pelo médico provenientes de uma determinada consulta, você terá que relacionar a entidade “medicamento” com a hipótese de ter ocorrido uma consulta, vinculando a entidade “medicamento” com o relacionamento, ora identificado de “consulta”.



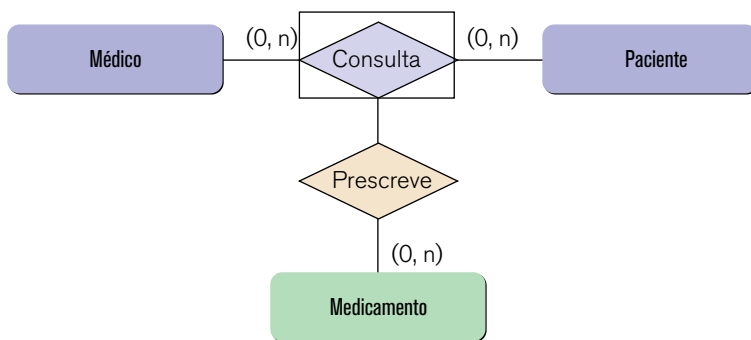


Figura 2.17 – Exemplo de agrupamento de entidades (Médico + Paciente = Consulta).

Você, provavelmente, deve responder que isso não é permitido. Exatamente, não podemos fazer isso diretamente. Dessa forma, para solucionar esse problema, indicamos que o relacionamento “consulta” é uma entidade “associativa”, representado graficamente por meio de um retângulo em volta do relacionamento (figura 2.17) e ou, um retângulo em torno das entidades “médico” e “paciente”, agora identificadas pela entidade associativa identificada de “consulta”, conforme podemos visualizar na figura 2.18.

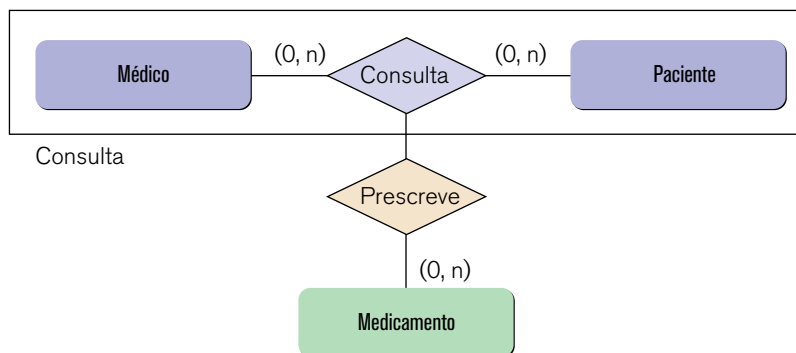


Figura 2.18 – Exemplo de entidade associativa.

Essa estratégia é também chamada de agregação, ou seja, agregamos o conjunto de relacionamentos a suas entidades relacionadas em uma nova entidade. (SETZER & DA SILVA, 2005).

Assim, no nosso exemplo, poderíamos primeiramente agregar o conjunto de relacionamento “consulta” às entidades “médico” e “paciente”, formando um novo Conjunto de entidades que chamaremos de “consulta” (nem sempre é necessário nomear a nova entidade criada na agregação).



Leia mais sobre Entidades Associativas em: <http://www.devmedia.com.br/tecnologias-de-banco-de-dados-e-modelagem-de-dados-parte-final/2106>

Leia mais sobre Entidades Especializadas em: <http://www-usr.inf.ufsm.br/~apereira/elc119/r1.php>

---

Na figura 2.18, representamos a Agregação no Diagrama ER por meio de um retângulo englobando os Conjuntos de entidades e o conjunto de relacionamentos envolvidos.

Também é possível utilizar cardinalidades mínimas e máximas no conjunto de relacionamentos da agregação. No nosso exemplo, uma “consulta” poderá não ter nenhum medicamento prescrito, em contra partida uma entidade “medicamento” pode não ter nenhuma prescrição.

1. Perceba que no nosso exemplo sempre há 0 (zero) na cardinalidade mínima entre a agregação e o conjunto de relacionamento com a outra entidade;
2. É possível fazermos uma agregação que envolva mais do que um conjunto de relacionamento. (SETZER & DA SILVA, 2005).
3. Os atributos de uma relação são todos os atributos dos conjuntos de entidades e dos conjuntos de relacionamentos envolvidos. (SETZER & DA SILVA, 2005).

## 2.7 Aprender Sobre a Modelagem Lógica dos Dados

Os modelos de dados tal como o conhecemos hoje, são frutos de um processo de evolução bastante significativa. Certamente, para que melhor entendamos como fazer uso do modelo de dados para um projeto de banco de dados é muito importante que compreendamos a evolução deste.

A evolução dos modelos de dados deu-se, mediante a busca por um melhor

gerenciamento de dados, o que gerou vários modelos que apresentam propostas para resolver as folhas fundamentais do sistema de arquivos.

Você verá que muitos dos novos conceitos e estrutura de bancos de dados, ainda mantém os conceitos dos modelos mais antigos. Como dito, os modelos foram evoluindo a medida que novos problemas surgiam. Os modelos podem ser classificados de acordo como os tipos de conceitos utilizados, podendo ser: modelo de dados representativos, modelo de dados conceituais e modelos de dados físicos.

Os modelos de dados representativos descrevem a estrutura do banco de dados da forma que o usuário possa entender como são os SGBD comerciais. São eles: o modelo hierárquico, modelo de rede, o modelo relacional, modelo de dados baseado em objetos, modelo de dados Semiestruturado entre outros.

## 2.8 Conhecer os Modelos Lógicos de Dados Existentes

- **Hierárquico**

O modelo hierárquico mostra-se como o primeiro a ser reconhecido como um modelo de dados. Este modelo foi desenvolvido na década de 1960 para gerenciar grandes quantidades de dados para projetos complexos de fabricação. Sua estrutura lógica básica é representada por uma estrutura de árvore “de cima para baixo”.

Seu desenvolvimento somente foi possível devido à consolidação dos discos de armazenamento endereçáveis, pois esses discos possibilitaram a exploração de sua estrutura de endereçamento físico para viabilizar a representação hierárquica das informações.

Conforme enuncia Fedeli e Polloni (2003, p. 56), “o modelo hierárquico é composto por um conjunto ordenado de árvores (ocorrências múltiplas de um único tipo de árvore)”. Deste modo, no modelo hierárquico os dados fazem-se

estruturados em hierarquias ou árvores. Os nós das hierarquias contêm ocorrências de registros, onde cada registro é uma coleção de campos (atributos), cada um contendo apenas uma informação.

O registro da hierarquia que precede a outros é o registro-pai, os outros são chamados de registros-filhos. Uma ligação é uma associação entre dois registros. Conforme verifica Oliveira et al (2000, p. 37), neste modelo “os dados e relacionamentos são representados por ligações, respectivamente”. A figura 2.20 exemplifica tal modelo:

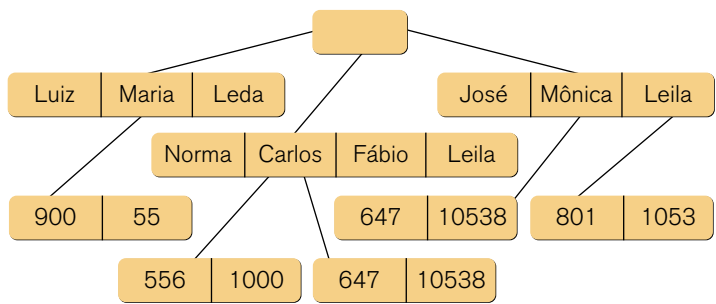


Figura 2.20 – Estrutura hierárquica.

• **Modelo em Rede**

O modelo em rede foi criado para representar relacionamentos de dados complexos com mais eficiência do que o modelo hierárquico, melhorar o desempenho dos bancos de dados, além, de estabelecer um padrão. Como discutimos no modelo anterior, a implementação e gerenciamento dos dados era um fator dificultoso na época, pois o gerenciamento era feito de modo programático, e sem o estabelecimento de um padrão para tal gerenciamento, muitos programadores criaram implementações peculiares a cada aplicação.

O gerenciador Data Base TaskGroup (DBTG) da CODASYL (*Committee on Data Systems and Languages*) estabeleceu uma norma para este modelo de banco de dados, com linguagem própria para definição e manipulação de dados. Os dados tinham uma forma limitada de independência física. A única garantia era que o sistema deveria recuperar os dados para as aplicações como se eles estivessem armazenados na maneira indicada nos esquemas. Os geradores de relatórios da CODASYL também definiram sintaxes para dois aspectos-chaves dos sistemas gerenciadores de dados: concorrência e segurança. O mecanismo de segurança fornecia uma facilidade na qual parte do banco de dados

(ou área) pudesse ser bloqueada para prevenir acessos simultâneos, quando necessário. A sintaxe da segurança permitia que uma senha fosse associada a cada objeto descrito no esquema. Ao contrário do Modelo Hierárquico, em que qualquer acesso aos dados passa pela raiz, o modelo em rede possibilita acesso a qualquer nó da rede sem passar pela raiz. No Modelo em Rede o sistema comercial mais divulgado é o CAIDMS da Computer Associates. O diagrama para representar os conceitos do modelo em redes consiste em dois componentes básicos: Caixas, que correspondem aos registros e Linhas, que correspondem às associações.

A figura 2.21 mostra a representação do modelo de dados em rede:

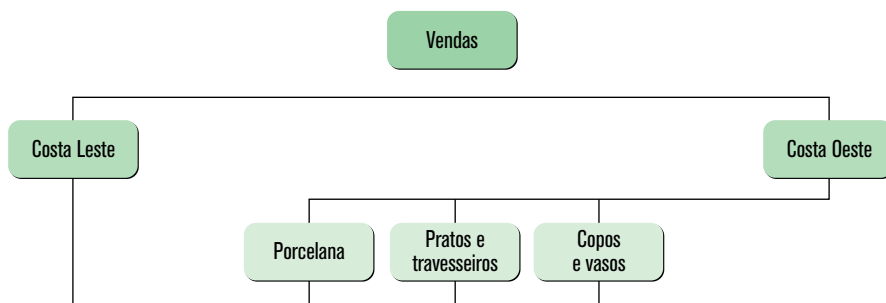


Figura 2.21- Representação do modelo em rede. Fonte: TURBAN et al. (2004, p. 595).

No modelo em rede, o usuário irá notar o banco de dados em rede de forma similar a uma coleção de registros em relacionamentos 1:M. Mas, de forma diferente do hierárquico, o modelo em rede permite que um registro tenha mais de um pai.

Em terminologia de bancos de dados de rede, o relacionamento é chamado de conjunto, onde cada conjunto é composto de, pelo menos, dois tipos de registros: proprietário e membro. Assim, observamos que um conjunto nada mais é que a representação de um relacionamento 1:M entre o proprietário e o(s) membro(s). Observando a figura 2.21, iremos visualizar que um registro membro (Costa Leste) pode num dado momento ser um membro (Vendas) e, também, ser proprietário de outros .

#### • Modelo Relacional

O modelo relacional foi apresentado em 1970 por E. F. Codd em um famoso artigo denominado: “Um modelo relacional de dados para grandes bancos de dados compartilhados”.

O modelo relacional é implementado por meio de um sistema de gerenciamento de banco de dados relacionais (SGBDR), muito sofisticado. O SGBDR executa as funções básicas fornecidas pelos sistemas hierárquicos e em rede, além de abrigar funções que facilitam sua gerência. A Figura 2.22, demonstra o uso deste modelo:

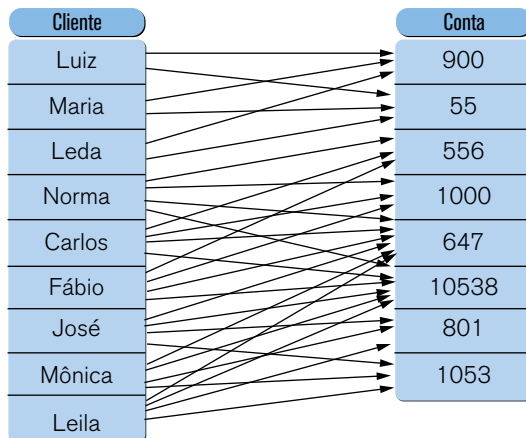


Figura 2.22 – Modelo relacional. Fonte: OLIVEIRA et al (2000, p. 39).

Conforme ilustra a figura 2.22, observa-se a existência de uma relação entre Cliente e Contas, onde um Cliente pode estar associado a mais de uma Conta.

Oliveira et al (2000, p.39), “o modelo relacional centra-se na ideia: organizar os dados em coleções de tabelas bidimensionais, também denominadas relações”, o termo relação empregado pelo autor é oriundo das relações matemáticas. Turban (2004, p. 595), explicita que o “modelo relacional é baseado neste conceito simples de tabelas para tirar vantagem das características das linhas e colunas de dados”.

## CONEXÃO

Leia mais sobre os modelos de dados em:

<http://www.linhadecodigo.com.br/artigo/332/planeje-o-seu-modelo-de-dados.aspx>

- **Modelo de Dados Baseado em Objetos**

É um modelo que está ganhando muita atenção nos últimos anos e trata-se de uma extensão do Modelo Entidade Relacionamento incluindo algumas características da orientação ao objeto como encapsulamento, métodos e identidade de objeto.

- **Modelo Dados Semiestruturado**

São modelos que permitem que diferentes itens de dados de um mesmo tipo de dados possam ter diferentes conjuntos de atributos. Neste modelo, o uso de XML para a especificação dos tipos de dados é muito utilizada. (SILBERSCHATZ, KORTH, & SUDARSHAN, 2006).

## 2.9 Aprender a Base Conceitual para Modelo Relacional

O modelo relacional apresentado por E. F. Codd em 1970, este modelo foi construído baseado na lógica dos predicados e na teoria dos conjuntos. Este modelo representa os dados contidos em um Banco de Dados por meio de relações (tabelas).

The image shows a screenshot of a database application with three tables displayed in a windowed interface. The 'Clientes' table lists company and contact information. The 'Produtos' table lists product details including supplier and category. The 'Pedidos' table lists order details including customer, employee, and date.

Código do Cliente	Nome da Empresa	Nome do Contato	Representante
ALFKI	Alfreds Futterkiste	Maria Anders	Rep.
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Prod.
ANTON	Antonio Moreno Taquería	Antonio Moreno	Prod.
AROUT	Around the Horn	Thomas Hardy	Rep.
BERGS	Berglunds snabbköp	Christina Berglund	Adm.
BLAUS	Blaugårds skåpmat	Arne Blaugård	Rep.
BOLID	Bólido	Antonio Moreno	Prod.
BONAP	Bonaparte	Antonio Moreno	Prod.
BOTTM	Bottling	Antonio Moreno	Prod.
BSBEV	Beverly Hills	Antonio Moreno	Prod.
CACTU	Cactus	Antonio Moreno	Prod.
CENIC	Cenicienta	Antonio Moreno	Prod.
CHOPS	Chop Suey	Antonio Moreno	Prod.
COMM	Commodore	Antonio Moreno	Prod.
CONSH	Consolidated	Antonio Moreno	Prod.
DRACD	Drac	Antonio Moreno	Prod.
DUMON	Dumont	Antonio Moreno	Prod.
EASTC	East	Antonio Moreno	Prod.

Código do Produto	Nome do Produto	Fornecedor	Categoria	Quantidade por Unidade
17	Alice Mutton	Pavlova, Ltd.	Carnes/Aves	20 latas de 1kg
3	Aniseed Syrup	Exotic Liquids	Condimentos	12 garrafas de 550ml
40	Boston Crab Meat	New England Seafood Cannery	Frutos do Mar	24 latas de 4oz
60	Camembert Pierrot	Gai pâturage	Laticínios	15 unidades de 300g
18	Carnarvon Tigers			
1	Chai			
2	Chang			
39	Chartreuse verte			
4	Chef Anton's Cajun Seasoning			
5	Chef Anton's Gumbo Mix			
48	Chocolate			
38	Côte de Blaye			
58	Escargots de Bourgogne			
52	Filo Mix			
71	Filetmysost			
31	Geitost			
15	Genen Shouyu			
56	Gnocchi di nonna Alice			
31	Gorgonzola Telino			
6	Grandma's Boysenberry Spread			

Número do Pedido	Cliente	Funcionário	Data do Pedido
10249	Vins et alcools Chevalier	Buchanan, Steven	04-07-1996
10249	Toms Spezialitäten	Suyama, Michael	05-07-1996
10250	Hanari Carnes	Peacock, Margaret	08-07-1996
10251	Victuailles en stock	Leverling, Janet	08-07-1996
10252	Suprêmes délicies	Peacock, Margaret	09-07-1996
10253	Hanari Carnes	Leverling, Janet	10-07-1996
10254	Chop-suey Chinese	Buchanan, Steven	11-07-1996
10255	Richter Supermarkt	Dodsworth, Anne	12-07-1996
10256	Wellington Importadora	Leverling, Janet	15-07-1996
10257	HILARIÓN-Abastos	Peacock, Margaret	16-07-1996
10258	Ernst Handel	Davolio, Nancy	17-07-1996
10259	Centro comercial Moctezuma	Peacock, Margaret	18-07-1996
10260	Ottiles Käseladen	Peacock, Margaret	19-07-1996
10261	Que Delicia	Peacock, Margaret	19-07-1996
10262	Rattlesnake Canyon Grocery	Callahan, Laura	22-07-1996
10263	Ernst Handel	Dodsworth, Anne	23-07-1996

Figura 2.23 – Tabelas banco de dados. Fonte: elaborado pelo autor.

O modelo relacional é claramente baseado no conceito de matrizes, em que as chamadas linhas (das matrizes) seriam os registros e as colunas (das matrizes), os campos. Os nomes das tabelas e dos campos são de fundamental importância para sua compreensão entre o que você está armazenando, onde está armazenando e qual a relação existente entre os dados armazenados.

Um banco de dados é representado por uma coleção de relações em um banco de dados relacional. Informalmente, a relação é chamada de tabela. A relação é composta de atributos. Os atributos são conhecidos como colunas da tabela. Os valores são preenchidos em cada tupla (linha) da relação. (figura 2.24).

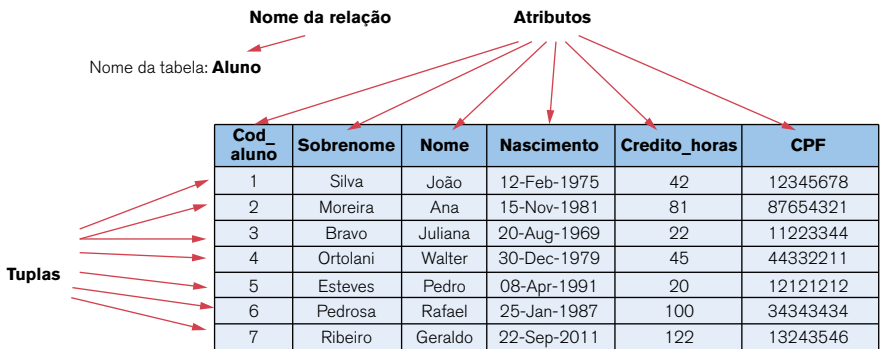


Figura 2.24 - Tabela aluno.

A qualquer momento, um Banco de Dados terá dados armazenados, estes dados serão inseridos de diversas fontes; podemos entender estas fontes como distintos usuários conectados simultaneamente gerando massa de dados. Dessa forma, imaginemos um determinado instante qualquer, vamos denominá-lo de T1; e um segundo instante, denominado por T2. Cada instante, representa uma ação de distintos usuários que podem estar inserindo ou consumindo dados do Banco de Dados. Seguindo este raciocínio, o conjunto de dados pode ser diferente de um determinado momento T2 se comparado ao momento T1. Ou seja, mais dados podem ter sido gerados, ou dados pré-existentes podem ter sofrido alteração, ou mesmo, a deleção de registros. Este cenário representa, o contexto real do ambiente gerenciado por Banco de Dados, onde usuários ou programas de aplicação a todo instante estão alterando e originando novos dados.

Ao conjunto de dados do banco de dados em um determinado momento dá-se o nome de Instância. Pela explicação acima, o aluno pode concluir que a



instância de um determinado banco de dados muda a todo o momento.

Apesar dos dados serem alterados constantemente em um banco de dados, a mesma coisa não acontece com a forma cujos dados são arranjados. A essa forma de arranjar os dados (lógica ou fisicamente) é dada o nome de estrutura do banco de dados.

Cod_aluno	Sobrenome	Nome	Nascimento	Credito_horas	CPF	Esquema
1	Silva	João	12-Feb-1975	42	12345678	Instância
2	Moreira	Ana	15-Nov-1981	81	87654321	
3	Bravo	Juliana	20-Aug-1969	22	11223344	
4	Ortolani	Walter	30-Dec-1979	45	44332211	
5	Esteves	Pedro	08-Apr-1991	20	12121212	
6	Pedrosa	Rafael	25-Jan-1987	100	34343434	
7	Ribeiro	Geraldo	22-Sep-2011	122	13243546	

Figura 2.25 – Esquema e instância.

Você pode estar pensando em como entender melhor esses conceitos, então vamos a uma analogia!

Imagine um hotel que você visita todo ano nas suas férias de verão. Quando você chega ao hotel é possível notar que a forma como os quartos estão dispostos nos corredores não se alteraram desde a sua última visita; poderíamos considerar esse projeto do hotel como sua estrutura.

Em contrapartida, a cada nova visita que você realiza no hotel você acaba por fazer novas amizades e conhecer novas pessoas, pois a cada dia, novas pessoas chegam e outras pessoas saem do hotel. Então, poderíamos considerar o conjunto de pessoas em um determinado dia no hotel como sendo a instância do hotel.

Tivemos que entender o conceito de Estrutura e Instância para termos uma base para entender o conceito de Modelo de Dados.

O conceito de modelo de dados é importante para que você possa responder a seguinte pergunta: como os dados são organizados no banco de dados?

O modelo de dados é quem apoia a estrutura de um banco de dados. Trata-se de um conjunto de ferramentas para a descrição, relações, semânticas de dados e restrições de consistência.

Há vários níveis de abstração com os quais pode-se modelar os dados, podemos citar algumas: Modelo Conceitual e o Modelo Computacional (DATE, 2004).

## 2.9.1 Conhecer os Conceitos de Chave Candidata, Primária e Estrangeira

No modelo relacional as chaves são importantes, pois sua utilização garante que cada linha da tabela seja identificável de modo exclusivo facilitando, assim, buscas posteriores, além de assegurar a consistência e integridade dos dados. Elas também são utilizadas para estabelecer relacionamentos entre tabelas e garantir integridade dos dados. Dessa maneira, a compreensão adequada do conceito de utilização de chaves no modelo relacional é muito importante.

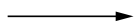
Veja que, para cada relação, deve existir uma chave, que vai ser constituída por um conjunto de um ou mais atributos e que identifica cada tupla (ou instância da relação) de um modo único, pois esta chave permitirá que seja estabelecido um relacionamento com outras tabelas.

Esta resposta requer um pouco de atenção. Uma chave baseia-se no conceito conhecido por determinação. No contexto de banco de dados temos a seguinte afirmação: “A determina B”, ou seja, uma vez que conhecemos o valor do atributo A podemos conhecer o atributo B.

Nome da tabela: **Aluno**

Cod_aluno	Sobrenome	Nome	Nascimento	Credito_horas	Classificacao_aluno
1	Silva	João	12-Feb-1975	42	Sr
2	Moreira	Ana	15-Nov-1981	81	Jr
3	Bravo	Juliana	20-Aug-1969	22	Sr
4	Ortolani	Walter	30-Dec-1979	45	Jr
5	Esteves	Pedro	08-Apr-1991	20	JR
6	Pedrosa	Rafael	25-Jan-1987	100	Sr.
7	Ribeiro	Geraldo	22-Sep-2011	122	Sr.

Continuação da  
tabela: **Aluno**



Media_notas	Transf	Cod_depto	Telefone	Cod_prof
2.87	NO	BIOL	2134	123
3.22	YES	CIS	4577	456
3.44	YES	ACCT	5422	876
3.09	NO	CID	2355	324
3.87	NO	ENGL	6788	683
3.12	NO	EDU	4577	567
2.07	YES	ACCT	5698	246

Figura 2.26 – Tabela Aluno.

Vimos que o Cod\_aluno determina o Nome, Sobrenome, mas não podemos dizer que Sobrenome determina o Cod\_aluno, uma vez que muitos alunos podem ter o mesmo sobrenome, logo, esta repetição descaracteriza o atributo como determinante.

O termo determinação, dito anteriormente, pode ser exemplificado por meio da figura 36 no esquema da tabela de ALUNOS, onde uma vez que conhecermos o COD\_ALUNO podemos determinar o NOME ou o NASCIMENTO. Para facilitar este termo, existe uma notação (símbolo) que representa a determinação, ou seja, dissemos que COD\_ALUNO determina o NOME, o que pode ser transcrito do seguinte modo:  $A \rightarrow B$  (HEUSER, 2004).

Este princípio de determinação é muito importante, pois é utilizado na definição de um conceito central de banco de dados relacionais, conhecido como dependência funcional. Este termo pode ser definido mais facilmente da seguinte maneira:

**O atributo B é funcionalmente dependente do atributo A se cada valor da coluna A determina um e somente um valor da coluna B.**

Observando os atributos da tabela Aluno, representada pela Figura 36, podemos dizer que o Telefone é funcionalmente dependente do Cod\_aluno, por exemplo, se temos Cod\_aluno 2, este determina que o Telefone será 4577. Mas não podemos dizer que o valor de Telefone 4577 é determinante de Cod\_aluno, pois observe que o referido número de telefone está associado aos Cod\_aluno 2 e 6.



## CONEXÃO

Recomendamos a leitura deste artigo, para aprofundar seus conhecimentos em chaves do modelo relacional:

[http://www.di.ubi.pt/~pprata/bd/BD\\_04\\_05\\_T3a.pdf](http://www.di.ubi.pt/~pprata/bd/BD_04_05_T3a.pdf)

<http://www.ic.unicamp.br/~geovane/mo410-091/Ch03-RM-Resumo.pdf>

Quando uma chave é composta apenas por um atributo, podemos dizer que se trata de uma chave simples, no exemplo da tabela aluno, demonstrada pela Figura 36, observa-se a existência de apenas uma chave. Uma chave constituída por mais de um atributo é denominada chave composta.

Para perceber melhor o que são as chaves e como funcionam no modelo relacional, é necessário compreender alguns conceitos, tais como: chave candidata, chave primária e chave estrangeira.

Chaves candidatas são todos os conjuntos de um ou mais atributos possíveis para identificar cada tupla de um modo único. No entanto, para proceder a esta seleção de chaves candidatas é necessário conhecer bem a realidade de cada um dos atributos da relação e qual o seu domínio (SILBERSCHATZ, 2006).

Dentre todas as chaves candidatas, apenas uma será escolhida para identificar cada tupla de forma única. A chave selecionada dentre as chaves candidatas é designada chave primária da relação. Em todas as tabelas deve existir sempre uma chave primária e os atributos que a constituem não podem conter valores nulos. Uma chave candidata pode ser descrita como uma superchave sem atributos desnecessários, ou seja, uma superchave mínima.

**Refleta:** Afinal, como devemos entender uma chave primária? Simples! Em uma tabela, o valor de chave primária deve ser único para garantir que todas as linhas possam ser identificadas por este valor, ou seja, esta chave! Quando uma tabela obedece este quesito de exclusividade, dizemos que a tabela apresenta integridade de entidade.

A superchave é qualquer chave que identifique cada linha exclusivamente. Em resumo, a superchave determina funcionalmente todos os atributos de uma linha. Uma superchave é um atributo ou uma combinação de atributos, cujos valores distinguem uma linha das demais dentro de uma mesma tabela. (HEUSER, 2004).

Resumidamente, uma superchave é um atributo (ou um conjunto de atributos) que quando valorado consegue encontrar apenas uma linha na tabela com uma célula contendo aquele valor.

Na tabela Aluno, a superchave poderia ser qualquer uma das seguintes colunas:

Cod\_aluno

Cod\_aluno, Sobrenome

Vejamos a tabela Curso. Podemos dizer que o conjunto de atributos CodCurso, Nome, Área e Carga Horária é uma superchave composta para a tabela Curso, isso porque não há duas ou mais linhas na tabela cujos valores dos atributos da superchave se repetem simultaneamente entre as linhas.

Curso			
CodCurso	Nome	Área	Carga Horária
1	Eng. Software	Computação	40
2	LDB	Pedagogia	40
3	Gerência de Projetos	Julho	50
4	LDB	Pedagogia	20
5	Eng. Software	Computação	20

A chave primária possui algumas características, vamos prestar bastante atenção nelas:

<b>SER UNÍVOCA</b>	Os atributos definidos para ser chave primária, por definição, têm de ter valor único para cada registro ou tupla na relação, de modo a garantir que todas as linhas sejam identificadas exclusivamente por essa chave. Nesse caso, diz-se que a tabela apresenta integridade de entidade.
<b>NÃO NULA</b>	Nenhum dos atributos que formam uma chave primária poderá conter um valor nulo em nenhum registro.
<b>NÃO REDUNDANTE:</b>	No caso de uma chave primária ser composta, não devem ser incluídos mais atributos do que os mínimos necessários para identificar os registros de modo unívoco.

Vamos estudar agora, sobre a chave estrangeira. Está associada à redundância controlada, esta redundância permite que o banco de dados relacional funcione.

Uma chave candidata é uma chave que apresenta duas características:

2. **Unicidade:** não há duas linhas diferentes na tabela com o mesmo valor para os atributos da chave (isso já era garantido na chave e foi explicado na seção anterior)
3. **Irredutibilidade:** não há um subconjunto de atributos da chave que apresentem a característica de Unicidade (DATE, 2004).

**Refleta:** o que é a redundância controlada?

A redundância controlada é uma das bases do funcionamento dos bancos de dados relacionais, ela contempla que diferentes tabelas compartilhem o mesmo atributo. Para entendermos melhor isto, observe a figura 3.7:

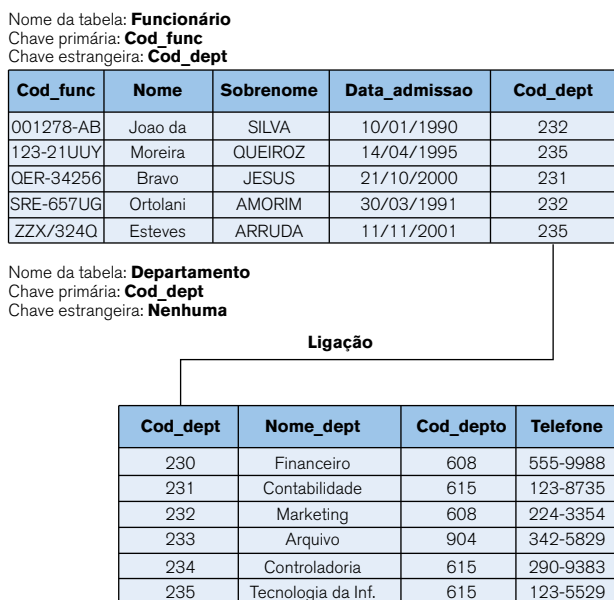


Figura 2.27– Exemplo de um banco de dados relacional simples. Fonte: adaptado de Rob, 2005, p. 70

Como podemos observar na figura 2.27, temos duas tabelas relacionadas, Funcionário e Departamento, relacionadas, pois compartilham um atributo em comum, o Cod\_dept. Observe que o atributo Cod\_dept 232 na tabela

Funcionário repete mais de uma vez. Verifique na tabela Departamento o departamento correspondente ao Cod\_dept 232. Essa ocorrência múltipla caracteriza o relacionamento 1:M entre as tabelas. Note que o Cod\_func é exclusivo na tabela Funcionário, assim como, Cod\_dept é exclusivo da tabela Departamento.

Ao examinar a figura 2.27, observe que o valor Cod\_dept em uma tabela pode ser utilizado para indicar o valor correspondente na outra tabela. Dessa forma, além de manter-se a integridade dos dados evitamos repetições desnecessárias. Por exemplo, o valor Cod\_dept 235 na tabela Departamento indica que a funcionária Maria Benedita da tabela Funcionário está situada no departamento de Tecnologia da Informação.

### 2.9.2 Compreender as Restrições de Integridade

Em relação às restrições de integridades mencionadas anteriormente, elas correspondem às regras a que toda relação de uma base de dados (relacional) deve obedecer. Tais regras são muito importantes para um projeto de bancos de dados. Muitos (mas certamente não todos) SGBDs aplicam as regras de integridade automaticamente.

Existem várias restrições que podem ser especificadas no modelo de dados relacional, conforme demonstra na tabela 2.1:

INTEGRIDADE DE ENTIDADES	DESCRIÇÃO
Exigência	Todas as entradas de chave primária são únicas e nenhuma parte dessa chave pode ser nula.
Finalidade	Cada linha terá uma identidade exclusiva e valores de chave estrangeira podem referenciar de modo adequado os valores de chave primária.
Exemplo	Nenhum vendedor pode ter número duplicado nem ser nulo. Em resumo, todos os vendedores são identificados de modo exclusivo por seu número

INTEGRIDADE REFERENCIAL	DESCRIÇÃO
Exigência	Uma chave estrangeira pode ter uma entrada nula, contanto que não faça parte de uma chave primária de suas tabelas ou uma entrada que coincida com o valor de chave primária de uma tabela que esteja relacionada (todo valor não nulo de chave estrangeira <b>deve</b> referenciar um valor de chave primária <b>existente</b> )

Finalidade	É possível que um atributo NÃO tenha valor correspondente, mas é impossível que tenha uma entrada inválida. A aplicação da regra de integridade referencial torna impossível a exclusão de uma linha de uma tabela cuja chave primária tenha valores obrigatórios de chave estrangeira em outra tabela.
Exemplo	Um cliente pode ainda não ter recebido a atribuição de um representante de vendas (número), mas é impossível que tenha um representante de vendas inválido (número).

Tabela 2.1 - Regras de integridade (ROB, 2011).

Uma definição formal da restrição de integridade referencial exige a definição de Chave Estrangeira (CE). Um conjunto de tributos CE em um esquema de relação R1 é uma chave estrangeira que referencia um esquema de relação R2 se ele satisfaz as seguintes propriedades:

- os atributos de CE possuem o mesmo domínio dos atributos da Chave Primária (CP) da outra relação de esquema R2. Diz-se que os atributos em CE referenciam ou referem-se a R2;
- uma CE na tupla t1 de r(R1) ou tem um valor que ocorre como CP de alguma tupla t2 de r(R2) ou tem o valor nulo. No primeiro caso, tem-se  $t1[CE] = t2[CP]$  e diz-se que t1 referencia ou refere-se a t2.

A restrição de integridade referencial é uma restrição que é especificada entre duas relações e é usada para manter a consistência entre tuplas de duas relações. Informalmente, a restrição de integridade referencial estabelece que uma tupla de uma relação que se refere à outra relação deve se referir a uma tupla existente naquela relação.

Normalmente, as restrições de integridade referencial derivam-se dos relacionamentos entre conjuntos de entidades representadas pelos esquemas de relação. Outras regras de integridade que podem ser aplicadas no modelo relacional são as restrições NOT NULL e UNIQUE. A restrição NOT NULL pode ser aplicada a uma coluna para garantir que todas as linhas da tabela apresentem um valor para essa coluna. A restrição UNIQUE é aplicada a uma coluna para garantir que não haja nenhum valor duplicado nela (DATE, 2003).



Uma base de dados tem muitas relações, e usualmente possuem muitas restrições de integridade referencial. Para especificar estas restrições, o projetista deve ter um claro entendimento do significado ou papel que os atributos desempenham nas diversas relações esquemas da base de dados. Normalmente, as restrições de integridade referencial são derivadas dos relacionamentos entre entidades representadas pelas relações esquemas.

Se observarmos a figura 2.27, que demonstra a relação entre Funcionário e Departamento, verifica-se que na tabela Funcionário há o atributo Cod\_dept que é a chave estrangeira daquela tabela, que indica sua relação com a tabela Departamento. Em outras palavras, o código do departamento contido no atributo Cod\_dept da tabela Funcionário está relacionado aos departamentos contidos na tabela Departamento, logo, espera-se encontrar o código comum compartilhado entre as tabelas.

### 2.9.3 Aprender o Método de Conversão do Modelo Conceitual para o Modelo Relacional

Dois pontos são fundamentais e devem ser observados na transformação do modelo conceitual para o modelo relacional: desempenho e simplicidade.

O desempenho está relacionado em criar um banco de dados onde as consultas SQL para alterar ou obter as informações desejadas sejam feitas com um mínimo de acesso ao banco de dados. Isto pelo fato de que, o acesso ao banco de dados equivale ao acesso ao disco rígido, e esta tarefa é a que mais consome tempo em uma execução de uma consulta SQL.

Já a simplicidade está relacionada ao desenvolvimento de aplicativos que utilizam o banco de dados que será projetado. Um banco de dados com estrutura simples facilita o desenvolvimento de aplicações. A conversão consiste em transformar o modelo conceitual em modelo lógico.

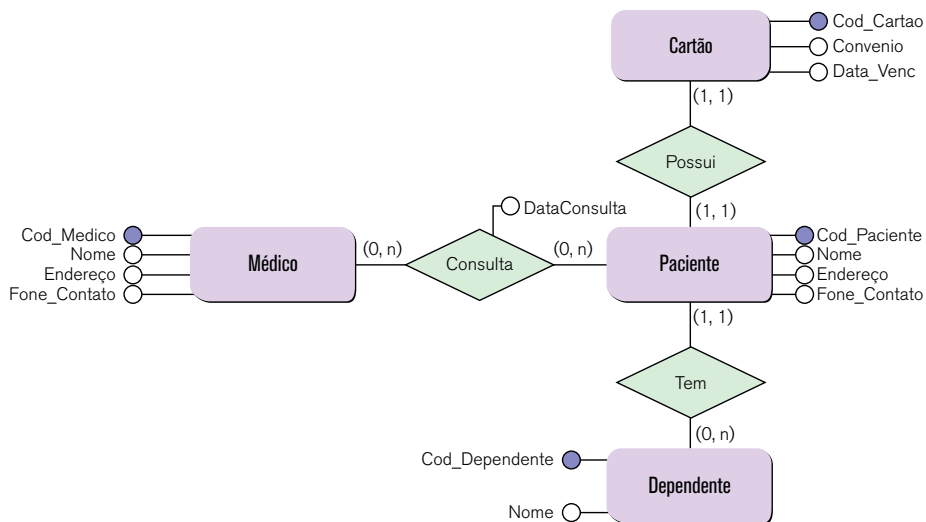


Figura 2.28 – Modelo conceitual.

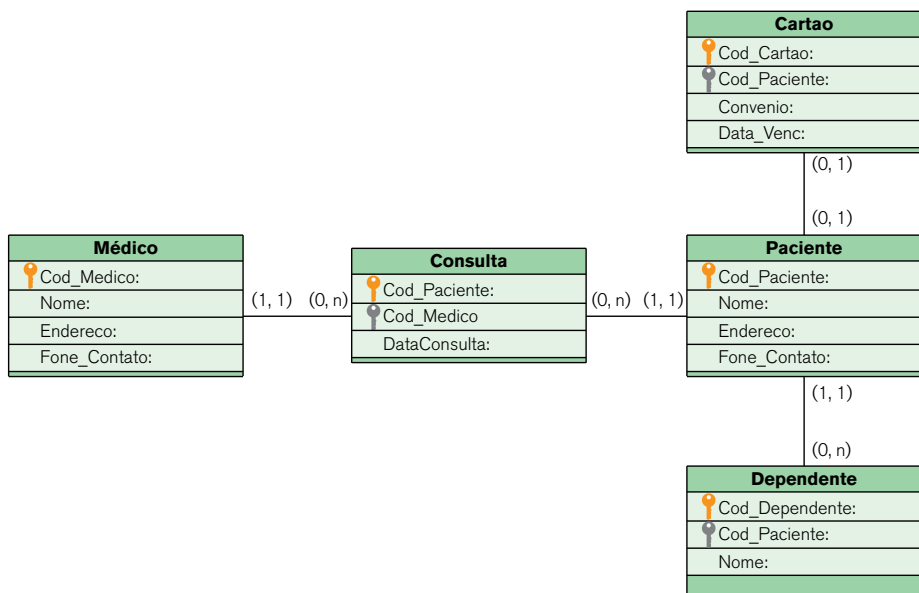


Figura 2.29 – Modelo lógico.

## • Entidades e Atributos

As entidades de um modelo conceitual serão transformadas em tabelas e os atributos em colunas destas tabelas. No exemplo abaixo, tem-se a entidade Médico que foi transformada na tabela Médico. Os atributos em colunas da tabela Médico. Observe o atributo identificador Cod\_Médico. Este atributo ao ser convertido se transformou na chave primária da tabela.



Figura 2.30 – Conversão de entidades.

## • Relacionamentos

O relacionamento no modelo lógico é representado pela interligação entre as tabelas através da chave estrangeira como vimos anteriormente. Dessa forma, há alguns pontos que devem ser observados nos relacionamentos de cardinalidade 1:1, 1:N e N:N.

No relacionamento 1:1 não há nenhuma regra com relação a chave estrangeira. No exemplo abaixo, a chave primária da tabela Paciente é chave estrangeira na tabela Cartão. Poderia ser ao contrário que não haveria nenhum problema, ou seja, a chave primária da tabela Cartão estar como chave estrangeira na tabela Paciente.

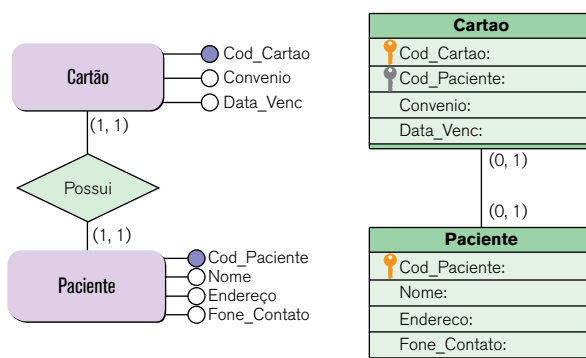


Figura 2.31 – Conversão 1:1.

Já nos relacionamentos 1:N a chave primária da tabela representada pela cardinalidade “1” deve aparecer como chave estrangeira na tabela com cardinalidade “N”.

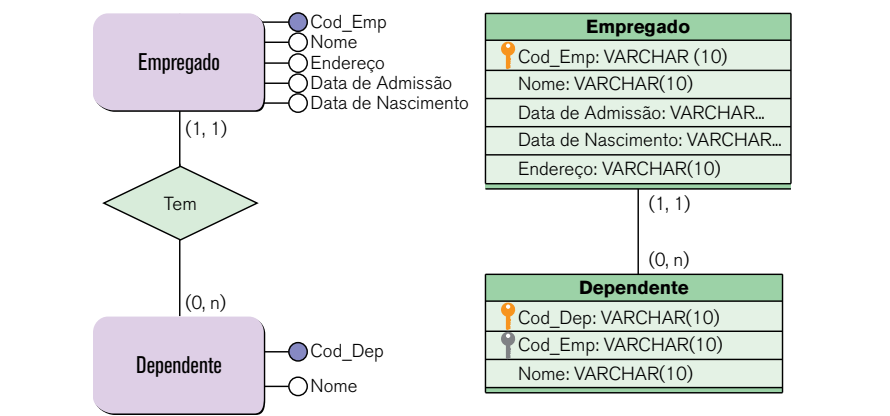


Figura 2.32 – Conversão 1:N.

Os relacionamentos N:N tem o relacionamento transformado em uma terceira tabela. Além disso, as chaves primárias das duas tabelas principais aparecem com chave estrangeira na tabela referente ao relacionamento.

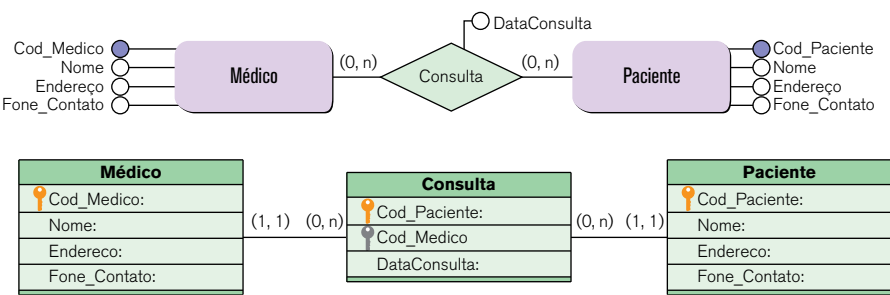


Figura 2.33 – Conversão N:N.



## ATIVIDADES

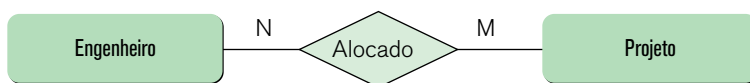
01. No modelo relacional as chaves são importantes, pois sua utilização garante que:

- a) Cada linha da tabela seja identificável de modo exclusivo, além de assegurar a consistência e integridade dos dados;
- b) Cada linha da tabela não seja identificável de modo exclusivo, além de assegurar a consistência e integridade dos dados;
- c) Cada linha da tabela seja identificável de modo exclusivo, além de não assegurar a consistência e integridade dos dados;
- d) Cada linha da tabela seja identificável de modo exclusivo, além de assegurar a consistência e a não integridade dos dados;
- e) Cada linha da tabela não seja identificável de modo exclusivo, além de assegurar a consistência e a não integridade dos dados.

02. Modelo de dados cuja estrutura lógica básica é representada por uma estrutura de árvore “de cima para baixo”:

- a) Modelo em Redes
- b) Modelo Relacional
- c) Modelo Hierárquico
- d) Modelo Conceitual
- e) Modelo Lógico

03. Dado o conjunto de relacionamentos R binário entre os conjuntos de entidades Engenheiro e Projeto, é correto afirmar que, em um mapeamento de cardinalidade muitos para muitos, a entidade:



- a) Engenheiro está associada a qualquer número de entidades em Projeto.
- b) Engenheiro não está associada a qualquer número de entidades em Projeto.
- c) Projeto não está associada a qualquer número de entidades em Engenheiro.
- d) Engenheiro está associada a apenas uma entidade em Projeto e Projeto está associada a apenas uma entidade Engenheiro.
- e) Engenheiro está associada a qualquer número de entidades em Projeto, mas Projeto não está associada a qualquer número de entidades em Engenheiro.



## REFLEXÃO

Parabéns! Você está se tornando um *expert* em banco de dados!

Neste capítulo compreenderemos componentes para a constituição de um DER estudamos os principais tipos de relacionamentos. Como já discutimos os bancos de dados, em sua maioria, adotam a abordagem relacional, portanto, compreender de forma satisfatória os principais tipos de relacionamentos, bem como, saber aplicá-los nas circunstâncias adequadas.

Vimos que há várias abordagens e possibilidades para compormos o relacionamento do nosso banco de dados, de forma a melhor atender as exigências do usuário final. Aprofunde seus conhecimentos nestes quesitos, pois, certamente serão imprescindíveis para sua vivência como profissional.



## LEITURA

Artigos on-line: para você incrementar mais o seu nível de aprendizagem de modelagem ER:

<http://www.devmedia.com.br/mer-e-der/14332> <http://www.devmedia.com.br/articles/viewcomp.asp?comp=2007> (Vaz)

<http://gbdi.icmc.sc.usp.br/documentacao/apostilas/me-r/> (Gomes, 1998)

<http://www.ime.usp.br/~andrrs/aulas/bd2005-1/aula7.html>



## REFERÊNCIAS BIBLIOGRÁFICAS

DATE, C. J.; **Introdução a Sistemas de Banco de Dados**. 8ª ed. Trad. Daniel Vieira. Rio de Janeiro: Elsevier, 2003.

ELMASRI, R.; NAVATHE, S. B. **Sistemas de bancos de dados**. São Paulo: Pearson (Addison Wesley), 2005.

HEUSER, C. A. **Projeto de Bancos de Dados**. 2ª ed. Porto Alegre: Sagra Luzzatto, 1999.

HEUSER, C. A. **Projeto de Banco de Dados**. 4ª ed. Instituto de Informática da UFRGS, Sagra DC Luzzatto, 1998.

KORTH, H.; SILBERCHATZ, A. **Sistemas de bancos de dados**. 3ª ed. São Paulo: Makron Books, 1998.

NETO, G. H.; ANDRADE, M. C.; MARTINS, M. D. C. **Gestão de Redes, Internet, Banco de Dados e Empreendedorismo** Módulo 4.2. Ribeirão Preto: UniSEB Interativo, 2013

OLIVEIRA, A. R. F.; TAVEIRA L. M. P.; GILDA A. **Modelagem de Dados**. Rio de Janeiro. Ed. Senac Nacional, 2000.

PRESSMAN, R. S. **Engenharia de software**. São Paulo: Makron Books, 1995.

RAMAKRISHNAN, R; GEHRKE, J. **Database management systems**. 2ª ed. Boston: McGraw-Hill, 2000.

ROB, P. CORONEL, C. **Sistemas de Banco de Dados – projeto, implementação e administração**. Cengage Learning: 2011.

SETZER, V. W. **Dado, Informação, Conhecimento e Competência**. Acesso em: set. de 2008, disponível em Dado, Informação, Conhecimento e Competência: <http://www.ime.usp.br/~vwsetzer/dado-info.html>

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistema de Banco de Dados**. 5ª ed. Rio de Janeiro: Elsevier, 2006.

---





3

# Normalização

Vimos nos temas anteriores que um bom projeto de Banco de Dados deve ser tal capaz de manter a integridade dos dados, não ter redundância, ser eficiente e de fácil desenvolvimento e manutenção.

Porém, o que aprendemos e aplicamos até agora ainda não é suficiente para garantir todas as características acima mencionadas.

Por isso, iremos aprender sobre a Normalização do Modelo Relacional, o que nos irá permitir projetar um Banco de Dados que não tenha redundâncias, garantindo assim a integridade do Banco de Dados além de possibilitar a recuperação de informações de forma fácil e eficiente.

O que é dependência funcional? Quais são os principais tipos de dependência funcional? O que evitamos quando aplicamos a normalização em um banco de dados?



## OBJETIVOS

- Preservar a informação e reduzir a redundância de dados.
  - Manter a consistência entre as cópias de dados.
  - Entender o conceito de dependência funcional, conceito fundamental para adequarmos os agrupamentos de atributo em um esquema relacional consistente.
  - Analisar e discutir as principais formas normais, juntamente com os processos envolvidos para normalização de um banco de dados.
-

## 3.1 Aprender o Conceito de Normalização

Frequentemente, encontramos sistemas com vários problemas de desempenho oriundos de uma modelagem de dados inadequada, resultando em uma grande quantidade de redundância de informações, prejudicial para um negócio bem planejado.

Erros na estruturação de dados dificultam consideravelmente a resolução desses problemas. Uma possível solução seria a criação de um novo sistema ou, até mesmo, adquirir algum sistema pronto, disponível no mercado (sistemas de prateleira), deixando de lado o atual.

Você consegue imaginar quais são os principais fatores negativos que cooperam para tamanho retrabalho?

Infelizmente, a maioria dos sistemas inicia seu planejamento pela codificação, ou seja, não é criado o modelo de dados da regra de negócio, dando ênfase no desenvolvimento ao invés de planejar a criação adequada das tabelas, colunas, relacionamentos etc, onde esses objetos são criados exclusivamente para atender o código fonte.

Com o objetivo de incrementar a qualidade do sistema, torna-se necessário a criação de um modelo de dados que atenda inteiramente a regra de negócio aplicando paralelamente, a normalização.

A normalização tem como objetivo reestruturar as informações a fim de eliminar redundâncias de dados indesejáveis que, eventualmente, possam existir nos arquivos.

É verdade dizer que existem diversas regras de normalização, todavia, aplicando a 1FN, 2FN e 3FN, é possível alcançarmos satisfatoriamente um modelo de dados bem estruturado.



### CONEXÃO

Recomendações 6.1 leia mais sobre normalização de dados em: <http://support.microsoft.com/kb/283878/pt-br>

Leia mais sobre vantagens e desvantagens de normalização de dados em: <http://www.aprendercomastics.net/tic/materiaisapoio/Normalizacao.pdf>

### 3.1.1 Fases da normalização (Dependência funcional)

Em 1972, Ted Codd, apresentou as primeiras publicações sobre os conceitos de normalização. Mas afinal, você sabe me dizer o que realmente seria a normalização?

Digamos que a normalização é um processo que analisa e adequa as estruturas e tabelas de maneira a minimizar as redundâncias de dados, eliminando assim, satisfatoriamente, as anomalias de inserção, alteração e exclusão.

Uma das funções principais da normalização é agir como um filtro sobre as entidades e seus relacionamentos, eliminando elementos indesejáveis sem produzir perda de qualquer tipo de informação nessas entidades, como também em seus relacionamentos. Como mencionado anteriormente, isso apenas é possível utilizando as formas normais, ora introduzidas para atuar onde a informação necessite de tratamento, resultando em dados bem estruturados dentro de um banco de dados.

Você pode observar abaixo, o que devemos evitar ao aplicarmos a normalização:

- dados repetitivos (atributos multivalorados);
- dependências funcionais totais e ou parciais em relação a uma chave primária composta;
- redundâncias de dados desnecessárias;
- perdas de informação;
- dificuldade de representar a realidade de fatos do negócio;
- dependências transitivas entre atributos.

Os primeiros estágios do processo de normalização são respectivamente, primeira forma normal (1FN), segunda forma normal (2FN) e terceira forma normal (3FN). Sob um ponto de vista estrutural, esses estágios iniciais, ou seja, a forma normal superior apresenta-se melhor sobre sua antecessora, sendo assim, a 3FN é melhor que a 2FN, que por sua vez é melhor que a 1FN.

Podemos dizer que um atributo A2 depende funcionalmente de um atributo A1, ou que o atributo A1 determina o atributo A2, quando para cada valor de A1 que aparecer na tabela, aparecerá o mesmo valor de A2 na mesma linha. (HEUSER, 2004).

Para entender as formas normais que serão apresentadas na próxima seção, é de extrema importância que você, gestor de TI, tenha o conceito do que vem a ser uma dependência funcional.

Bem, não podemos deixar de mencionar dependência funcional quando estudamos a normalização. Uma dependência funcional é um relacionamento entre dois ou mais atributos de maneira que o valor de um determinado atributo identifique o valor para cada um dos demais atributos, ou seja, um atributo está vinculado a outro. Para exemplificar, analise essa dependência funcional, ora representada pela expressão:  $A \rightarrow B$ .

Você deverá interpretar que o atributo B é dependente funcional-mente do atributo A, ou seja, para identificar o valor de B, precisamos conhecer o valor de A. Você também deve se atentar que o inverso não é verdadeiro, por exemplo:

Código do Funcionário  $\rightarrow$  Nome do Funcionário

Nesse caso, para identificarmos o nome do funcionário, inicialmente precisamos conhecer o código do mesmo, tornando assim o atributo “nome do funcionário” dependente do atributo “código do funcionário”. Sendo assim, não poderíamos aplicar a regra inversa, entretanto, podemos conhecer o nome do funcionário e não o seu código. Agora, você deve estar se questionando, “mas nem sempre é necessário conhecer o código do funcionário para obter o seu nome”. Porém, esse pensamento é inadequado, pois, imagine que você tenha funcionários com o mesmo nome.

Outro detalhe relevante diz respeito que, eventualmente, você pode encontrar mais de uma dependência funcional, por exemplo:

Código do Funcionário  $\rightarrow$  Nome do Funcionário

Código do Funcionário  $\rightarrow$  UF do Funcionário

Esse mesmo exemplo poderia ser afirmado da seguinte forma: Código do Funcionário  $\rightarrow$  Nome do Funcionário, UF do Funcionário.

Ficou confuso? Não se preocupe, vamos explorar um exemplo mais detalhado. Imagine que em uma tabela exista os seguintes atributos: código do funcionário, nome do funcionário, tipo de logradouro, logradouro, número, complemento, bairro, cidade e UF. Nessa tabela, para cada código do funcionário teremos apenas um valor para nome do funcionário, tipo de logradouro, logradouro, número, complemento, bairro, cidade e UF. Dessa forma você pode dizer que os atributos nome do funcionário, tipo de logradouro, logradouro, número, complemento, bairro, cidade e UF são funcionalmente dependentes

do código do funcionário. Nesse caso específico, você já deve ter percebido que estamos considerando apenas o endereço residencial do funcionário, ou seja, caso contrário, a dependência funcional seria inexistente. Agora você é capaz de exemplificar essa dependência funcional da seguinte forma:

Código do Funcionário → nome, tipo logradouro, logradouro, número, complemento, bairro, cidade, UF

Sendo assim, podemos afirmar que o valor de um atributo determina o valor dos demais atributos, por exemplo, o valor de uma chave primária determina o valor dos outros atributos do mesmo registro.

Agora você está confortável no que se refere à dependência funcional, certo? Vamos agora esmiuçar outro tipo de dependência funcional, a chamada dependência funcional parcial.

A dependência funcional parcial é determinada quando os atributos não chave, por sua vez, não dependem funcionalmente de toda a chave primária, obviamente, quando essa for composta. Entretanto, em tabelas onde a chave primária é composta, todos os atributos, obrigatoriamente devem depender de toda a chave primária, caso contrário, caracterizamos a dependência funcional parcial. Para exemplificar a ocorrência de uma dependência funcional parcial, considere que o atributo C dependa funcionalmente de A, mas, por sua vez, o atributo A não depende de B. Ficou confuso com esse exemplo? Então vamos utilizar um exemplo mais próximo de nossa realidade.

Imagine que a tabela ora identificada de “avaliação” seja constituída dos seguintes campos: código do aluno, código da disciplina, período, nome da disciplina, nota (conforme tabela 3.1). Considere que a chave primária dessa tabela seja constituída pelos atributos “código do aluno, período e código da disciplina”, evidenciando uma chave primária composta. Você pode perceber que o atributo “nome da disciplina” depende apenas do “código da disciplina”, ou seja, não depende do “código do aluno”, muito menos do “período”, formando assim uma dependência funcional parcial.

CÓDIGO_ALUNO	CÓDIGO_DICIPLINA	PERÍODO	NOME_DICIPLINA	NOTA
34213	401	1	Banco de Dados I	3,75
34213	408	1	Análise e Projeto de Sistema	8,15
34213	409	1	Engenharia de Software	7,00
34213	401	2	Banco de Dados I	8,00

Tabela 3.1 – Exemplo de dependência funcional parcial.

Isso é um fator negativo para um projeto de banco de dados bem estruturado, promovendo redundância de informação.

Suponha que o aluno ora identificado através do código 34.213 tenha, equivocadamente, reprovado na disciplina “Banco de Dados I” no primeiro período, tendo que cursá-la novamente no segundo período. Você pode perceber na tabela 3.2 a duplicidade do campo “nome da disciplina”. Uma possível solução seria a criação de outra tabela contendo exclusivamente os dados da disciplina, aplicando assim a 2FN (tópico estudado ainda nessa aula).

Além da dependência funcional parcial, temos também a dependência funcional transitiva, onde podemos nos deparar com a situação em que um campo não seja dependente da chave primária ou de parte dela, mas sim, dependente de outro campo que não é, e nem faz parte da chave primária. Você deve perceber a diferença existente entre dependência funcional parcial e a transitiva, ou seja, ambas não são análogas, visto que, na dependência funcional parcial, no mínimo um campo da tabela depende de parte da chave primária composta e, na dependência funcional transitiva, pelo menos um campo da tabela depende de outro campo o qual não faz parte da chave primária.

Está confuso com tantos conceitos? Não se preocupe, vamos agora utilizar um exemplo real onde você identificará uma dependência funcional transitiva. Suponha a constituição de uma tabela nomeada de “funcionário” formada pelos campos “código do funcionário (chave primária), nome do funcionário, código do cargo, descrição do cargo e salário” conforme você pode visualizar na Tabela 3 a seguir:

CÓDIGO_FUNCIONÁRIO	NOME_FUNCIONÁRIO	CODIGO_CARGO	DESCRIÇÃO_CARGO	SALÁRIO
21	Venilton Pereira	10	Administrador de BD	7.300,00
32	Hamilton Mendes	12	Desenvolvedor C#	5.500,00
43	Marcelo Cunha	13	Escriturário	2.100,00
54	Rodrigo Ricardo	14	Analista de Sistema	5.900,00
65	Rogério Nunes	10	Administrador de BD	7.300,00

Tabela 3.2 - Exemplo de dependência funcional transitiva.

Vamos analisar a tabela 3 cujo objetivo é identificar a dependência funcional transitiva. Você pode identificar que o “código do funcionário” determina exclusivamente o “nome do funcionário e o código do cargo”. Todavia, podemos reconhecer que o “código do cargo” não é e muito menos faz parte da chave primária da tabela “funcionário”, mas, determina a “descrição do cargo e o salário”, ou seja, esses campos não dependem do campo “código do funcionário”.

Código do Funcionário → Nome do Funcionário, Código do Cargo. Código do Cargo → Descrição do Cargo, Salário do Cargo.

Notamos também a existência de informações redundantes, pois, o funcionário “Rogério Nunes” também é um “Administrador de BD”, possuindo os campos “Descrição do Cargo” e “Salário” duplicados em relação ao funcionário “Venilton Pereira”. Esse tipo de anomalia não é desejável em um projeto de banco de dados bem estruturado, então, uma possível solução seria a criação de uma nova tabela constituída pelos campos “código do cargo, descrição do cargo e salário”, pois, basta apenas conhecermos o “código do cargo” para extrair as demais informações (descrição do cargo e salário), evitando assim a redundância de informação.

Agora que você já absorveu os conceitos sobre dependência funcional, dependência funcional parcial e transitiva, você já se encontra preparado para abstrair conceitos de mais um tipo de dependência funcional, a dependência funcional multivalorada. Esse tipo de dependência ocorre quando para cada valor do campo A, por exemplo, existe um conjunto de valores para os campos B e C, ora, vinculados a ele. Salientamos que B e C estão associados a A, todavia são independentes entre si.

LIVRO	AUTOR	REVISOR
Oracle Database 11g	Jason Price Michael Jackson	Fernando Ribeiro Alex de Souza Aquinaldo Trevisan
BI2 - Business Inteligence	Carlos Barbieri José Barbieri	Milton Cruz Luiz Inácio
Sistema de Informação Gerenciais	Kenneth Laudon Jane Laudon	Renato Violin Juliana Cardoso

Tabela 3.3 – Tabela Livros.

Vamos utilizar outro tipo de exemplo, para elucidar uma dependência funcional multivalorada. Imagine a necessidade de criarmos uma tabela chamada de “livros”, conforme podemos verificar na tabela 3.3. Para cada “livro”, existe um conjunto de autores associados a um livro e também, existe um conjunto de “revisores” vinculados a este “livro”. Sendo assim, autores e revisores são independentes um do outro.

As dependências funcionais multivaloradas ora identificadas na tabela 3.3 podem ser representadas como:

Livro →→ Autor

Livro →→ Revisor



Você pode notar que esse tipo de dependência acarreta em uma quantidade relevante de redundância. Perceba que a tabela 3.3 possui campos multivalorados, promovendo um cenário indesejável no que tange o armazenamento e recuperação de informações no banco de dados. Se, por ventura, tratássemos cada campo de maneira a possuir apenas um valor, teríamos uma tabela idêntica a da tabela 3.4, isto é, com uma enorme quantidade de dados duplicados, ocasionando sérios problemas, principalmente para a manutenção da consistência e integridade das informações.

LIVRO	AUTOR	REVISOR
Oracle Database 11g	Jason Price	Fernado Ribeiro
Oracle Database 11g	Jason Price	Alex de Souza
Oracle Database 11g	Jason Price	Aquinaldo Trevisan
Oracle Database 11g	Michael Jackson	Fernado Ribeiro
Oracle Database 11g	Michael Jackson	Alex de Souza
Oracle Database 11g	Michael Jackson	Aquinaldo Trevisan
BI2 - Business Intelligence	Carlos Barbieri	Milton Cruz
BI2 - Business Intelligence	Carlos Barbieri	Luiz Inácio
BI2 - Business Intelligence	José Barbieri	Milton Cruz
BI2 - Business Intelligence	José Barbieri	Luiz Inácio
Sistema de Informação Gerenciais	Kenneth Laudon	Renato Violin
Sistema de Informação Gerenciais	Kenneth Laudon	Juliana Cardoso
Sistema de Informação Gerenciais	Jane Laudon	Renato Violin
Sistema de Informação Gerenciais	Jane Laudon	Juliana Cardoso

Tabela 3.4 – Exemplo de dependência funcional multivalorada.

Podemos perceber que é necessário não apenas um campo multivalorado, mas no mínimo dois (autor e revisor). Você pode verificar que para cada dado do campo “livro”, existe um conjunto de valores para o campo “autor” e, ainda, um conjunto de valores para o campo “revisor”. Note também que os campos “autor” e “revisor” não possuem relação alguma, ou seja, a inexistência de um dos dois campos (autor e revisor) implica na eliminação de uma dependência funcional multivalorada, isto é, teríamos apenas um campo multivalorado.

Por fim, a dependência funcional cíclica encerra o tópico o qual discorre sobre dependências funcionais. Esse tipo de dependência ocorre exclusivamente quando possuímos dependências como:

$$A \rightarrow B, B \rightarrow C, C \rightarrow A$$

Não ficou claro? Novamente, vamos utilizar um exemplo mais próximo de nossa realidade. Imagine uma instituição de ensino, onde podemos ter um professor ministrando diversas disciplinas, representando a seguinte dependência:

**Professor → Disciplina**

Agora iremos representar a tabela “professor” com alguns dados inclusos nela, auxiliando ainda mais nosso entendimento no que se refere à dependência funcional cíclica, conforme você pode visualizar na tabela 3.5.

PROFESSOR	DICIPLINA
Márcio junqueira	Inteligência Artificial 1 Inteligência Artificial2
Fernando Siqueira	Engenharia de Software 1 Análise e Projeto de Sistemas

Tabela 3.5 – Tabela Professor.

Nesse contexto, podemos ainda ter a seguinte situação, um professor também pode escrever uma ou várias apostilas, conforme representado na tabela 3.6, ilustrando a seguinte dependência funcional:

**Professor → Apostila**

1,058	APOSTILA
Márcio junqueira Rafael Duarte Joaquim Nunes	Inteligência Artificial 1 - Básico ao Avançado
Fernando Siqueira Gilberto Wong	Análise e Projeto de Sistemas

Tabela 3.6 – Tabela Professor – Apostila.

Como não bastasse, cada disciplina pode possuir várias apostilas, conforme você pode visualizar a partir da tabela 8, caracterizando a seguinte dependência funcional:

**Disciplina → Apostila**

DICIPLINA	APOSTILA
Inteligência Artificial 1 Inteligência Artificial 2	Inteligência Artificial 1 - Básico ao Avançado
Engenharia de Software 1	Análise e Projeto de Sistemas

Tabela 3.7 – Tabela Disciplina – Apostila.

Perceba que agora temos uma relação cíclica, ou seja, disciplina é dependente funcional de professor, apostila, por sua vez, é dependente funcional de disciplina e, por fim, apostila é também dependente funcional de professor, formando assim a dependência funcional cíclica, ora representada como segue abaixo:

Professor → Disciplina, Disciplina → Apostila e Professor → Apostila

Espero que você tenha realizado um excelente estudo sobre os principais tipos de dependência funcional, pois, a partir de agora você deverá utilizar esses conceitos para aplicar as principais formas normais.

### 3.1.2 Problemas de Redundância

A redundância é o armazenamento dos mesmos dados em mais de um lugar dentro do banco de dados. Este é um dos maiores problemas do modelo relacional.

Tomemos como exemplo a tabela 3.8. Há vários dados redundantes que podem causar os seguintes problemas:

- **Anomalia De Inclusão:**

- Foi contratado um novo professor para o curso de Tecnologia da Informação. Terá que incluir o novo professor, e alterar todos os valores de QTDE TURMAS para o Curso de Tecnologia da Informação.

- **Anomalia De Alteração**

- O nome do curso Tecnologia da Informação mudou para T.I. Você vai ter que alterar o nome do Curso em todas as linhas da tabela.

- **Anomalia De Exclusão**

O que acontece se você excluir o professor com o código igual a 4?

Resposta: o curso Eng. Produção será excluído também.

COD	NOME	CARGO	CURSO	QTDE TURMA
1	Evandro	Professor	Tecnologia da Informação	5
2	Tamara	Assessora	Gestão Comercial	10
3	Maria	Coordenador	Tecnologia da Informação	5
4	Rodrigo	Professor	Eng. Produção	1
5	Miriam	Coordenadora	RH	7

Tabela 3.8 - Tabela Professores.

## 3.2 Conhecer as Principais Formas Normais: 1ª Forma Normal (1Fn), 2ª Forma Normal (2Fn), 3ª Forma Normal (3Fn) E Forma Normal De Boyce-Codd

O processo de normalização se faz necessário para a otimização das tabelas do Modelo Relacional objetivando que estas fiquem “melhores projetadas” principalmente no sentido da eliminação de redundâncias.

A normalização é baseada em diversas formas normais, que são regras que devem ser impostas às tabelas para que estas fiquem normalizadas.



### CONEXÃO

Recomendações 6.3 Leia mais sobre Forma Normal de

Boyce-Codd (FNBC) em: <http://www.oocities.org/br/lavcm/bcnf.pdf>

Leia mais sobre Dependências Funcionais e Formas Normais em: <http://www.ime.usp.br/~andrers/aulas/bd2005-1/aula11.html>

Temos diversas formas normais que podemos considerar, por exemplo: a primeira forma normal (1FN), a segunda forma normal (2FN), a terceira forma normal (3FN), a forma normal de Boyce-Codd (FNBC), a quarta forma normal (4FN) e a quinta forma normal (5FN).

Um ponto importante a ser observado é que essas formas normais são hierárquicas, ou seja, na sequência acima citada elas vão aumentando o seu grau de rigidez e uma acaba tendo como pré-requisito que a anterior tenha sido cumprida, por exemplo, a 2FN tem como primeira regra que a 1FN tenha sido cumprida.

Nesta aula iremos tratar a 1FN, a 2FN, 3FN e a Boyce-Codd (FNBC).

Dizemos que uma tabela está na 1FN quando o domínio de todos os seus atributos é atômico. Ser atômico significa que a tabela não terá subestruturas em seus atributos, ou seja, o domínio de cada atributo não é subdividido em vários outros domínios de outros atributos. (HEUSER, 2004).

### 3.2.1 Primeira Forma Normal (1FN)

Consideramos que uma tabela se encontra na 1FN somente se todos os seus campos possuírem apenas dados atômicos (dados simples e indivisíveis) e os dados vinculados a cada campo também devem ser um valor simples (não composto). Entretanto, caso existam campos compostos, estes devem ser segmentados em campos atômicos. Outro detalhe é que, se eventualmente existir campos multivalorados, estes devem, obrigatoriamente, constituírem outra tabela, a qual será relacionada com a tabela original.

Vamos utilizar a tabela identificada de “pessoa” constituída pelos campos “código da pessoa, nome, telefone, endereço”, a qual não atende a 1FN, representada pela tabela 3.9.

CÓDIGO	NOME	TELEFONE	ENDEREÇO
12	Patrícia Campos	(16) 3822-1928 (16) 3822-1929 (16) 3822-1930	Av. Nove de Julho, 1334 - Apto 708 - Higienópolis Ribeirão Preto - SP
13	Juliana Aparecida Cunha	(16) 3511-0002 (16) 9144-8712	Rua Martins Pena, 345 - Centro - Ituverava - SP
14	Rogério Martins	(31) 7123-8200	Av. Do Brasil, 98 - Apto 1003 Nova Aliança - Recife - PE
15	Paula de Souzam Pinho	(11) 54231-0023	Largo do Arouche, 51 Centro - São Paulo - SP

Tabela 3.9 – abela Pessoa (não atende a 1FN).

Para atender a 1FN, a partir da tabela “pessoa”, poderíamos constituir duas novas tabelas, ora identificada de “pessoa” e formada pelos campos “código da pessoa, nome, tipo do logradouro, número, complemento, bairro, cidade e estado” representada pela tabela 3.10.

CÓDIGO	NOME	TIPO LAGRA-DOURO	LAGRA-DOURO	NÚMERO	COMPLE-MENTO	BAIRRO	CIDADE	ESTADO
12	Patrícia Campos	Avenida	Nove de Julho	1334	Apto 708	Higienópolis	Ribeirão Preto	SP
13	Juliana Aparecida Cunha	Rua	Martins Pena	345		Centro	Ituverava	SP

CÓDIGO	NOME	TIPO LAGRA-DOURO	LAGRA-DOURO	NÚMERO	COMPLE- MENTO	BAIRRO	CIDADE	ESTADO
14	Rogério Martins	Avenida	Brasil	98	Apto 1003	Nova Aliança	Recife	PE
15	Paula de Souza Pinho	Vila	Largo do Arouche	51		Centro	São Paulo	SP

Tabela 3.10 – Tabela Pessoa agora sem campos compostos.

E, a tabela identificada de “telefone”, essa outra, constituída pelos campos “código do telefone, código do ddd, número do telefone e código da pessoa”, visualizada a partir da tabela 3.11, onde, código da pessoa é uma chave estrangeira que referencia a tabela “pessoa”. Agora sim, após essa reestruturação, podemos concluir que ambas as tabelas atendem as regras aplicadas à 1FN.

CÓDIGO TELEFONE	DDD TELEFONE	NÚMERO TELEFONE	CÓDIGO PESSOAL
1	16	3822-1920	12
2	16	3822-1929	12
3	16	3822-1930	12
4	16	3511-0002	13
5	16	9144-8712	13
6	31	7123-8200	14
7	17	54231-0023	15

Tabela 3.11 – Tabela telefone com campos multivalorados.

### 3.2.2 Segunda Forma Normal (2FN)

Antes mesmo de iniciarmos a 2FN, devemos recapitular o conceito de dependência funcional parcial, vista nessa aula. Caso você ainda tenha algum tipo de dúvida, retorne no tema destinado às dependências funcionais.

Digamos que uma tabela se encontra na 2FN se estiver, obviamente, na 1FN e não possuir em hipótese alguma dependência funcional parcial. Se, eventualmente, existir campos que não dependam de toda a chave primária, você deverá extraí-los da tabela, formando assim uma nova tabela.

Vamos imaginar a existência de uma tabela “vendas” formada pelos campos “código da venda, código do produto, descrição do produto, valor unitário do produto, quantidade do produto e valor total da venda”, conforme podemos visualizar na tabela 3.12.

CÓDIGO_VENDA	CÓDIGO_PRODUTO	DESCRIÇÃO_PRODUTO	VALOR_PRODUTO	QUANTIDADE	VALOR_TORAL
1	712	Detergente Veja	R\$ 7,50	2	R\$ 15,00
2	715	Sabonete Líquido	R\$ 4,80	5	R\$ 24,00
3	712	Detergente Veja	R\$ 7,50	3	R\$ 22,50
3	715	Sabonete Líquido	R\$ 4,80	2	R\$ 9,60

Tabela 3.12 – Tabela Vendas.

A chave primária da tabela “vendas” é constituída pelos campos “código da venda e código do produto”, ou seja, você pode perceber que se trata de uma chave primária composta. Agora sim, podemos verificar se esta tabela atende as especificações da 2FN. Inicialmente, verificamos se a tabela encontra-se na 1FN. Nesse caso, não encontramos nenhum campo composto ou multivalorado nesta tabela, logo, podemos afirmar que a mesma está na 1FN.

Na sequência, torna-se necessário verificarmos a existência de dependência funcional parcial. Podemos constatar que na tabela “vendas” encontramos essa dependência, pelo simples fato de que os campos “descrição do produto e valor unitário” dependem funcionalmente apenas de parte da chave primária, ou seja, são determinados pelo “código do produto”, apresentando indícios de que essa tabela não se encontra na 2FN.

Uma dependência funcional parcial ocorre em uma tabela quando uma coluna qualquer da tabela depende apenas de parte da Chave Primária. (HEUSER, 2004).

Para adequarmos a tabela “vendas” à 2FN, devemos segmentar essa tabela em duas tabelas, isto é, tabela “vendas” e tabela “produtos”, onde os campos que constituem a dependência funcional parcial farão parte da tabela “produtos”.

Posteriormente essa divisão, você terá duas tabelas, a tabela “produtos” formada pelos campos “código do produto, descrição do produto e valor do produto” onde “código do produto” é chave primária, e, a tabela “vendas”, onde os campos “código da venda, código do produto, quantidade e valor total” fazem parte da mesma, sendo que, os campos “código da venda e código do produto” constituem uma chave primária composta e, “código do produto” é também chave estrangeira que referencia a tabela “produtos”.

Agora sim, após a criação dessas duas novas tabelas (produtos e vendas) podemos constatar a adequação das mesmas à 2FN. As tabelas “produtos” e “vendas” podem ser visualizadas, respectivamente, a partir das tabelas 3.13 e 3.14.

CÓDIGO_PRODUTO	DESCRIÇÃO_PRODUTO	VALOR_UNITÁRIO
712	Detergente Veja	R\$ 7,50
715	Sabonete Líquido	R\$ 4,80

Tabela 3.13 – Tabela Produtos.

CÓDIGO_VENDA	CÓDIGO_PRODUTO	QUANTIDADE	VALOR_TOTAL
1	712	2	R\$ 15,00
2	715	5	R\$ 24,00
3	712	3	R\$ 22,50
3	715	2	R\$ 9,60

Tabela 3.14 – Tabela Vendas.

### 3.2.3 Terceira Forma Normal (3FN)

A 3FN busca normalizar a tabela atuando diretamente em outro tipo de redundância, isto significa que, uma tabela está na 3FN se estiver, obrigatoriamente, na 2FN e não possuir nenhuma dependência funcional transitiva. Caso você ainda tenha algum tipo de dúvida na identificação de uma dependência funcional transitiva, retome novamente esse item de estudo para elucidar suas dúvidas, pois agora você irá aplicar esse conceito.

Uma tabela está na 2FN quando está na 1FN e não contém nenhuma dependência funcional parcial. (HEUSER, 2004).

Na tentativa de facilitar nosso entendimento, vamos utilizar a tabela “funcionário” ora formada pelos campos “código do funcionário, nome do funcionário, código do cargo, descrição do cargo, salário), onde o campo “código do funcionário” é nossa chave primária, conforme podemos visualizar na tabela 3.15.



CÓDIGO_FUNCIONÁRIO	NOME_FUNCIONÁRIO	CÓDIGO_CARGO	DESCRIÇÃO_CARGO	SALÁRIO
10	José Geraldo Neto	1	Analista de Sistema 1	R\$ 5.900,00
20	Armando Silva	2	Administrador de BD	R\$ 7.200,00
30	Joaquim Ferreira	1	Analista de Sistema 1	R\$ 5.900,00
40	Thiago Mendes	2	Administrador de BD	R\$ 7.200,00

Tabela 3.15 – Tabela Funcionário.

Para resolvermos esse problema, devemos verificar se esta tabela atende as especificações da 3FN, isto significa que, ela deve estar na 2FN e não possuir dependência funcional transitiva. Quanto à 2FN, podemos constatar que a tabela “funcionário” atende as regras, pois existem apenas campos atômicos (indivisíveis) e também está ausente a dependência funcional parcial, por simplesmente inexistir uma chave primária composta nessa tabela.

Após checarmos se a tabela “funcionário” encontra-se na 2FN, devemos verificar se existe a dependência funcional transitiva (conceito já estudado anteriormente). Analisando, podemos constatar a existência da seguinte dependência:

Código\_Cargo → Descrição\_Cargo, Salário

Você deve estar afirmando: “Código do cargo não é chave primária e os campos descrição do cargo e salário estão dependendo dele!”. Exatamente, isto significa que a tabela “funcionários” não se encontra na 3FN.

Bem, agora que você identificou a dependência funcional transitiva, vamos eliminá-la segmentando a tabela em duas ou mais tabelas. Em nosso cenário, dividir a tabela “funcionário” em duas novas tabelas, já é suficiente, ou seja, iremos trabalhar com as tabelas “funcionário” e “cargo” com a seguinte estrutura:

Funcionário (Código\_Funcionário, Nome, Código\_Cargo) onde Código\_Cargoreferencia a tabela “funcionário” Cargo (Código\_Cargo, Descrição\_Cargo, Salário)

Você pode visualizar o resultado final das tabelas “funcionário” e “cargo”, respectivamente representadas pelas tabelas 3.16 e 3.17 apresentadas a seguir:

CÓDIGO_FUNCIONARIO	NOME_FUNCIONÁRIO	CÓDIGO_CARGO
10	José Geraldo Neto	1
20	Armando Silva	2
30	Joaquim Ferreira	1
40	Thiago Mendes	2

Tabela 3.16 – Tabela Funcionário (atendente à 3FN).

CÓDIGO_CARGO	DESCRIÇÃO_CARGO	SALÁRIO
1	Analista de Sistema 1	R\$ 5.900,00
2	Administrador de BD	R\$ 7.200,00
3	Analista de Sistema 2	R\$ 6.050,00
4	Aquiteto Java	R\$ 6.900,00

Tabela 3.17 – Tabela Cargo.

Com a nova tabela “cargo”, se quisermos alterar o salário de uma categoria qualquer, basta irmos à tabela “cargo” e alterar apenas uma linha, inibindo assim a chance de termos inconsistência dos dados.

Vamos definir agora algumas regras que quando não respeitadas nos indicarão que determinadas tabelas não estão na 3FN.

Em nosso exemplo, percebemos que o salário dependia de “código do cargo”, porém “código do cargo” não era um atributo pertencente ao conjunto de atributos da chave primária, então o separamos em outra tabela “importando” a sua chave primária.

Então, ao retirarmos as dependências funcionais transitivas da tabela “funcionário” já conseguimos deixar essa tabela na 3FN, pois ela já se encontrava na 2FN.

Uma dependência funcional transitiva ocorre em uma tabela quando qualquer atributo desta tabela, além de depender funcionalmente da chave primária, também depende de outro atributo que não faz parte da chave primária. Uma tabela esta na 3FN quando esta na 2FN e não contém nenhuma dependência transitiva. (HEUSER, 2004).

### 3.2.4 Forma Normal de Boyce-Codd

Consideramos que uma tabela está na Forma Normal de Boyce-Codd se todas as dependências funcionais existentes na relação  $X \rightarrow Y$ , X é chave ou superchave da relação.

**Relembrando:**

Superchave – é um atributo (ou um conjunto de atributos) que quando valorado consegue encontrar apenas uma linha na tabela com uma célula contendo aquele valor.

Ex: se Código\_Cargo é chave da tabela Cargo, Código\_Cargo + Descrição\_Cargo é uma superchave da tabela Cargo.

CÓDIGO_CARGO	DESCRIÇÃO_CARGO	SALÁRIO
1	Analista de Sistema 1	R\$ 5.900,00
2	Administrador de BD	R\$ 7.200,00
3	Analista de Sistema 2	R\$ 6.050,00
4	Aquiteto Java	R\$ 6.900,00

Tabela 3.18 – Tabela Cargo.

Observe a tabela Fornecedor a seguir. O modo como esta representada, ainda não foi definida a chave primária, o neste caso, deve-se definir a(s) superchave(s) da tabela.

COD_FORNECEDOR	NOME_FORNECEDOR	COD_FORNECEDOR	QUANTIDADE

Tabela 3.19 – Tabela Fornecedor.

Inicialmente podemos definir as seguintes superchaves, analisando as dependências funcionais:

Cod\_Fornecedor, Cod\_Produto → Nome\_Fornecedor, Qtde

Cod\_Produto, Nome\_Fornecedor → Cod\_Fornecedor, Qtde

Mas, se preencheremos com dados, veremos que a tabela não esta na FNBC porque nem Cod\_Fornecedor e nem Cod\_Produto são realmente chaves ou superchaves da tabela.

COD_FORNECEDOR	NOME_FORNECEDOR	COD_PRODUTO	QTDE
1	Pereira S/A	20	18
1	Pereira S/A	30	150
1	Pereira S/A	15	22

Tabela 3.20 – Tabela Fornecedor.

Note que os dados do fornecedor ficaram repetidos, tanto na coluna Cod\_Fornecedor como em Nome\_Fornecedor. Para que a tabela fique normalizada, é necessário dividi-la em duas, Fornecedor (Cod\_Fornecedor, Nome\_Fornecedor) e Pedido (Cod\_Fornecedor, Cod\_Produto, Qtde).

COD_FORNECEDOR	NOME_FORNECEDOR
1	Pereira S/A

Tabela: Fornecedor.

COD_FORNECEDOR	COD-PRODUTO	QTDE
1	25	18
1	30	150
1	15	22

Tabela: Pedido.

3.2.5 Problemas de Decomposições

Vimos que para efetuar a normalização, uma tabela é decomposta em duas ou mais tabelas. A forma como a tabela é decomposta pode acarretar perda de informação. A figura abaixo mostra a decomposição da tabela A em Tabela B e C. Ao tentar a junção da Tabela B e C para recompor os dados da Tabela A, o resultado final foi que a Tabela A ficou diferente da original, assim houve perda de informação.

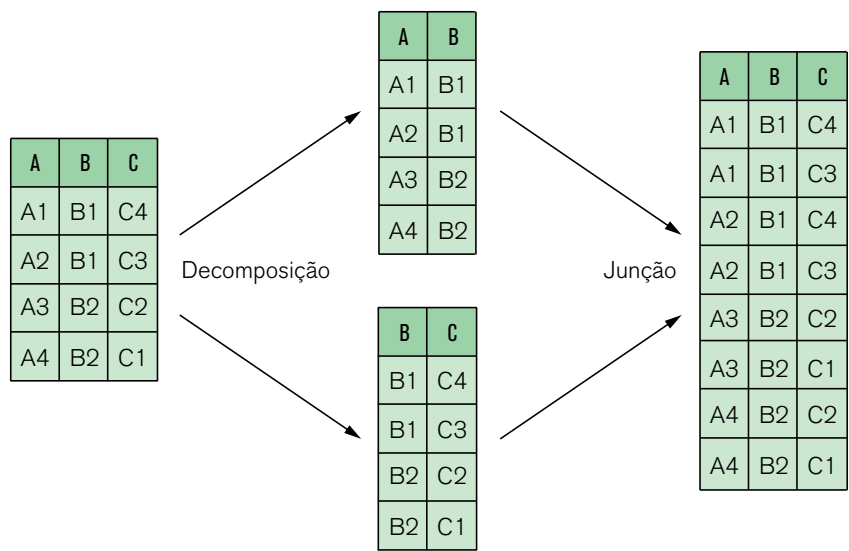


Figura 3.1 – Decomposição com perda de informação.

O exemplo a seguir ilustra a decomposição da mesma tabela mas de forma a não gerar perdas de informação.

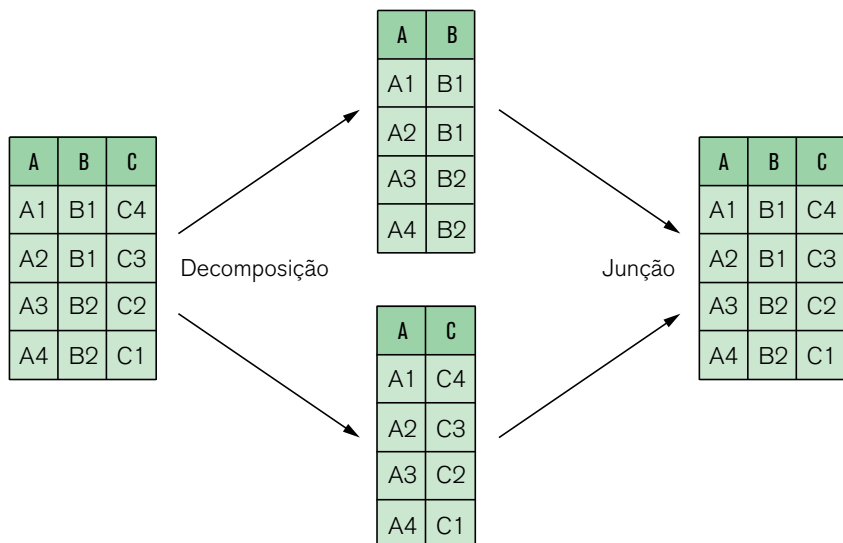


Figura 3.2 – Decomposição sem perda de informação.

### 3.3 Aprender Sobre as Linguagens de Manipulação de um Banco de Dados Relacional

As linguagens de manipulação de banco de dados relacional são fundamentais para podermos acessar os dados de um banco de dados. Para tanto utilizamos como linguagem de manipulação a linguagem SQL. A linguagem SQL possibilita, não somente a manipulação dos dados, como também a criação da estrutura do banco de dados, permissões etc. Antes de estudarmos a linguagem SQL, veremos a álgebra relacional.

## 3.4 Conhecer Conceitos Básicos de Álgebra Relacional

As operações da álgebra relacional são usadas para selecionar tuplas de uma determinada relação ou para combinar tuplas relacionadas a diversas relações com o propósito de especificar uma consulta – uma requisição de recuperação – sobre a base de dados.

A álgebra relacional é uma linguagem teórica que trabalha com operações que contenham uma ou mais relações, onde a partir destas, outra relação é criada de modo a não alterar a relação originária.

A álgebra relacional é uma coleção de operadores que tomam relações com seus operandos e retornam uma relação como um resultado, dessa forma, por meio de seus operadores específicos e adequados, pode-se fazer a combinação de tuplas, relacionadas ou não e obter-se a resultante ora desejada (ROB, 2005).

É importante mencionar que a álgebra relacional é uma linguagem abstrata, o que significa que as queries formuladas pela álgebra relacional não são destinadas à execução em um computador, portanto, o uso desta linguagem permite a compreensão da execução das queries no banco de dados. Os operadores relacionais são oito: *select*, *project*, *join*, *intersect*, *union*, *difference*, *product* e *divide*.

- **Union**

Este operador combina todas as linhas de duas tabelas, excluindo as duplicadas. As tabelas devem apresentar as mesmas características de atributos (as colunas e os domínios devem ser idênticos) para serem utilizadas no operador Union. O exemplo da utilização da Union é apresentado na figura 3.3 a seguir:

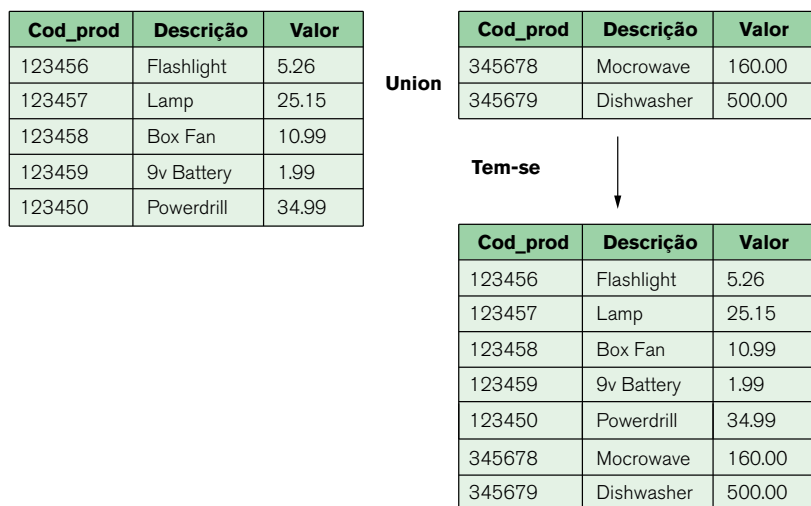


Figura 3.3 – Union. Fonte: Rob, 2005, p. 78.

#### • Intersect

O operador *intersect* resulta apenas nas linhas que aparecem em ambas as tabelas. Como no caso de *union*, só se produzirão resultados se as tabelas forem compatíveis para união. Por exemplo, não é possível utilizar *intersect* se um dos tributos for numérico e outro for baseado em caracteres. O efeito do *intersect* é apresentado na figura 3.4:

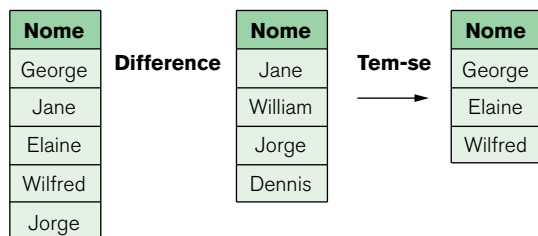


Figura 3.4 – Intersect. Fonte: Rob, 2005, p. 79.

#### • Difference

O resultado deste operador será todas as linhas de uma tabela que não se encontram na outra tabela, ou seja, uma tabela é subtraída da outra. Como no caso de *union*, só produzirão resultados válidos se as tabelas forem compatíveis para união; o efeito do *difference* é apresentado na figura 3.5. No entanto, observe que subtrair a primeira tabela da segunda não é igual a subtrair a segunda da primeira.

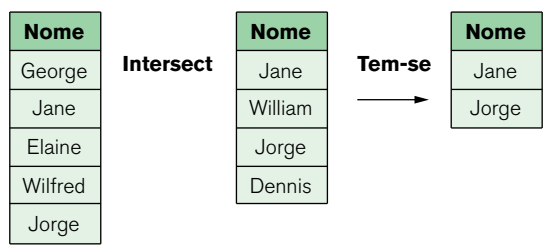


Figura 3.5 – Difference. Fonte: Rob, 2005, p. 7+

• Product

Resultará em todos os pares de linhas possíveis a partir de duas tabelas (também conhecido como produto cartesiano). Portanto, se uma tabela tiver seis linhas e a outra tiver três, o *product* resulta em uma lista composta de 6 x 3 = 18 linhas. O efeito do *product* é apresentado na figura 3.6.

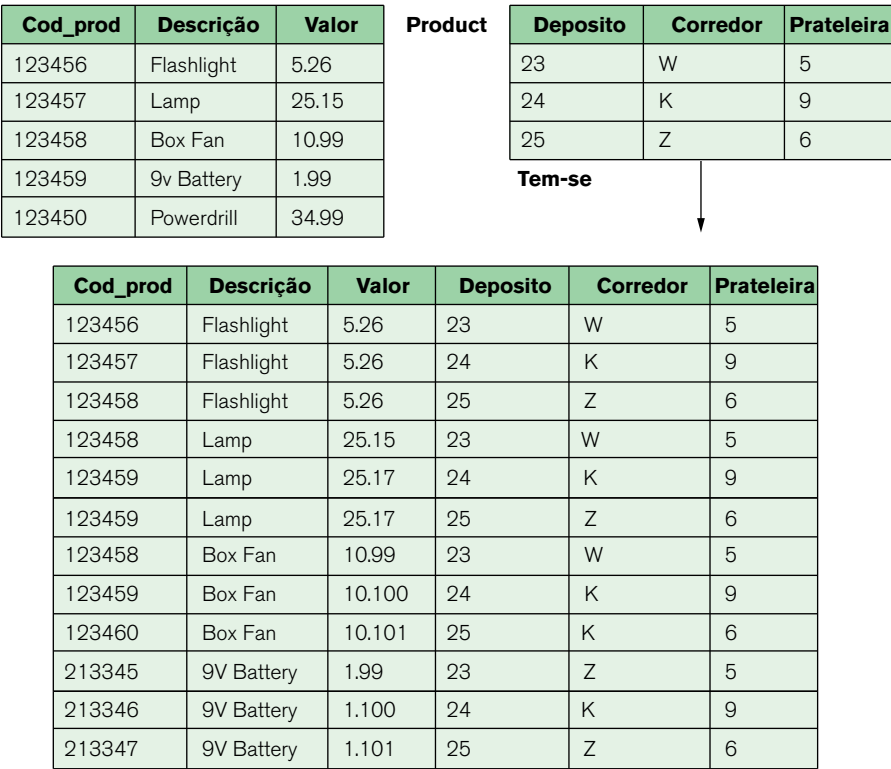


Figura 3.6 – Product. Fonte: Rob, 2005, p.80.



- **Select**

Este operador também é conhecido como restrict, resulta nos valores de todas as linhas de uma tabela que satisfaçam uma condição dada. select pode ser utilizada para listar todos os valores de linha ou apenas aqueles que atenderem a um critério especificado. Em outras palavras, Select produz um subconjunto horizontal de uma tabela. O efeito do select é apresentado na figura 3.7.

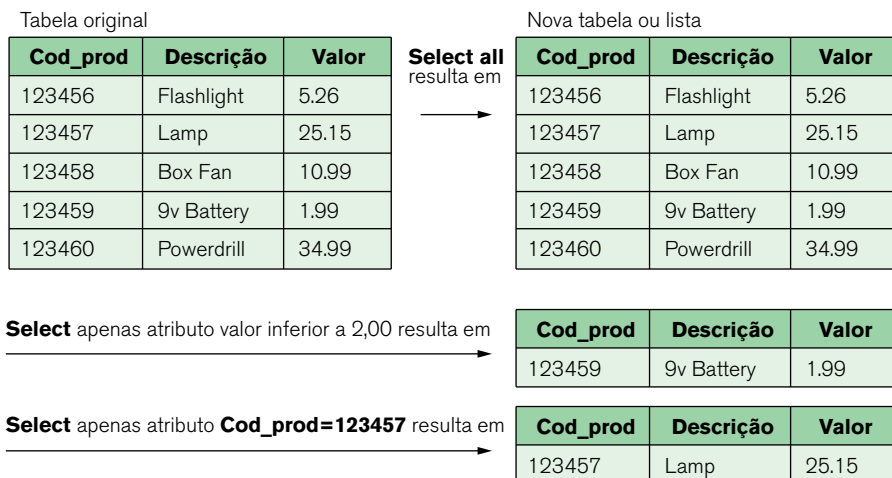


Figura 3.7 – Operador Select. Fonte: Rob, 2005, p. 80

- **Project**

A resultante deste operador será todos os valores de atributos selecionados. Em outras palavras, project produz subconjunto vertical de uma tabela. O efeito do product é apresentado na figura 3.8:

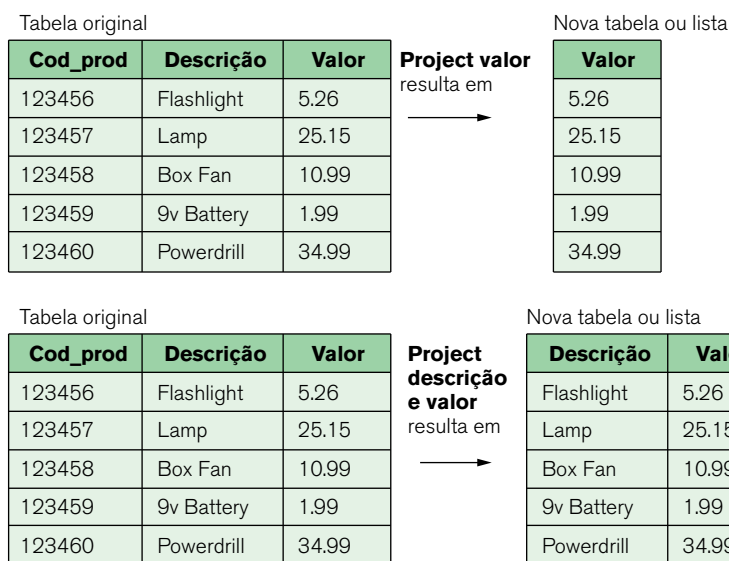


Figura 3.8 – Operador Project. Fonte: Rob, 2005, p. 81.

## • Join

O operador *join* possibilita a combinação de informações de duas ou mais tabelas. Trata-se da potência real por trás dos bancos de dados relacionais, permitindo a utilização de tabelas independentes ligadas por atributos comuns. As tabelas Cliente e Corretor apresentadas na figura 3.9 serão utilizadas para ilustrar vários tipos de utilização do operador *join* (junções).

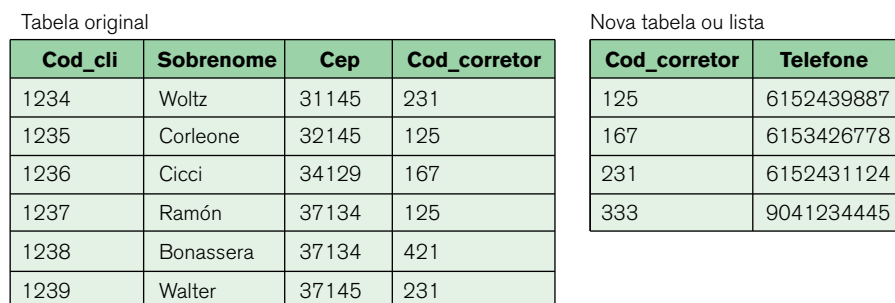


Figura 3.9 – Tabelas para demonstrar o operador JOIN. Fonte: Rob, 2005, p. 81.

A junção natural (natural *join*) liga tabelas selecionando apenas as linhas com valores comuns em seu(s) atributo(s) comum(ns). É o resultado de um processo em três estágios:

- I. primeiro, aplica-se o operador *product* junto às tabelas;
- II. em segundo lugar, executa-se uma operação *select* sobre o resultado da Etapa “i” para se obter apenas as linhas para as quais os valores Cod\_corretor são iguais. As colunas comuns são chamadas de colunas de junção;
- III. executa-se uma operação *project* sobre os resultados da Etapa (ii) para produzir uma única cópia de cada tributo, eliminando-se assim as colunas duplicadas.

O resultante da operação *join* entre Cliente e Corretor, é demonstrado pela figura 3.10:

COD_CLI	SOBRENOME	CEP	COD_CORRETOR	TELEFONE
1234	Wolts	31145	231	6152431124
1235	Corleone	32145	125	6152439887
123.	Ramóm	34129	167	6153426778

Figura 3.10 – Tabela resultante do join entre Cliente e Corretor. Fonte: Rob, 2005, p. 81.

O resultado de uma junção natural produz uma tabela que não inclui pares sem correspondência e fornece apenas as cópias das correspondências.

## CONEXÃO

Aprofunde seus conhecimentos sobre os operadores relacionais,

<http://www.carlosbezerra.com/Banco1.pdf>

<http://www.devmedia.com.br/algebra-relacional/9229>

## ATIVIDADES

01. Para que o banco de dados tenha desempenho, o mesmo deve ter suas tabelas normalizadas. Quando uma tabela tem uma dependência funcional parcial é necessário efetuar a normalização da mesma. Neste cenário deve se aplicar:

- a) Primeira forma normal (1FN).
- b) Segunda forma normal (2FN).
- c) Terceira forma normal (3FN),
- d) Forma normal de Boyce-Codd (FNBC).
- e) Quarta forma normal (4FN).

02. Tendo a normalização como contexto, uma atenção especial deve ser dada a dependência funcional parcial. De acordo com o material e o que foi visto em aula, pode-se afirmar que a dependência funcional parcial é:

- a) determinada quando os atributos não chave, por sua vez, não dependem funcionalmente de toda a chave estrangeira.
  - b) determinada quando os atributos chave, por sua vez, não dependem funcionalmente de toda a chave não primária.
  - c) determinada quando os atributos chave, por sua vez, não dependem funcionalmente de toda a chave não estrangeira.
  - d) determinada quando os atributos não chave, por sua vez, não dependem funcionalmente de toda a chave primária.
  - e) determinada quando os atributos não chave, por sua vez, não dependem funcionalmente da integridade referencial.
- 



## REFLEXÃO

Neste capítulo estudamos sobre como melhorar um projeto computacional relacional, eliminando as redundâncias e prevenindo algumas possíveis falhas que poderiam gerar inconsistência de dados. Você deve aplicar todos os conceitos estudados nessa aula para criar modelos de dados bem estruturados e sem anomalias de inserção, alteração e remoção de dados.

---



## LEITURA

Artigos **on-line**: para você incrementar mais o seu nível de aprendizagem acerca do tema normalização de dados, consulte as sugestões de **links** a seguir:

<[http://pt.wikipedia.org/wiki/Normaliza%C3%A7%C3%A3o\\_de\\_dados](http://pt.wikipedia.org/wiki/Normaliza%C3%A7%C3%A3o_de_dados)

<http://www.database-normalization.com/>    <http://imasters.com.br/artigo/7020/banco-de-dados/modelagem-de-dados-final-normalizacao>>

Livro: Sistemas de Banco de Dados

Autor: Ramez Elmasri e Shamkant B. Navathe.

Editora: Pearson

Ano: 2011

Esse livro possui conceitos específicos de banco de dados, explicando os conceitos de normalização como também a necessidade de aplicarmos, pelos menos, os três níveis de normalização (1FN, 2FN e 3FN).

---



## REFERÊNCIAS BIBLIOGRÁFICAS

- DATE, C. J.; **Introdução a Sistemas de Banco de Dados**. 8ª ed. Trad. Daniel Vieira. Rio de Janeiro: Elsevier, 2003.
- ELMASRI, R.; NAVATHE, S. B. **Sistemas de bancos de dados**. São Paulo: Pearson (Addison Wesley), 2005.
- HEUSER, C. A. **Projeto de Bancos de Dados**. 2ª ed. Porto Alegre: Sagra Luzzatto, 1999.
- HEUSER, C. A. **Projeto de Banco de Dados**. 4ª ed. Instituto de Informática da UFRGS, Sagra DC Luzzatto, 1998.
- KORTH, H.; SILBERCHATZ, A. **Sistemas de bancos de dados**. 3ª ed. São Paulo: Makron Books, 1998.
- OLIVEIRA, A. R. F.; TAVEIRA L. M. P.; GILDA A. **Modelagem de Dados**. Rio de Janeiro. Ed. Senac Nacional, 2000.
- PRESSMAN, R. S. **Engenharia de software**. São Paulo: Makron Books, 1995.
- RAMAKRISHNAN, R; GEHRKE, J. **Database management systems**. 2ª ed. Boston: McGraw-Hill, 2000.
- ROB, P. CORONEL, C. **Sistemas de Banco de Dados – projeto, implementação e administração**. Cengage Learning: 2011.
- SETZER, V. W. **Dado, Informação, Conhecimento e Competência**. Acesso em: set. de 2008, disponível em Dado, Informação, Conhecimento e Competência: <<http://www.ime.usp.br/~vwsetzer/dado-info.html>>
- SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistema de Banco de Dados**. 5ª ed. Rio de Janeiro: Elsevier, 2006.
- TURBAN, E. et al. **Tecnologia da Informação para a Gestão**. 3a. Ed. São Paulo, SP: Bookman, 2004.
-



# 4

## **Linguagem SQL (Parte 1)**

Nos capítulos anteriores discutimos sobre como transformar os requisitos de dados de um *stakeholder* do sistema em um modelo formal, porém o modelo conceitual é impossível de ser implementado no computador, no caso o Modelo Entidade Relacionamento (MER).

Na sequência, estudamos o Modelo Relacional, que é um modelo capaz de ser implementado computacionalmente, e por isso o chamamos de Modelo Computacional. Porém, era muito difícil partir dos requisitos do usuário e ir direto para a abstração computacional. Então, estudamos uma forma de mapear o Modelo Entidade Relacionamento em Modelo Relacional, conseguindo, com uma facilidade maior, ter um modelo computacional partindo dos requisitos dos *stakeholders*, utilizando o Modelo Entidade Relacionamento como ponte.

Agora, só falta descobrirmos uma forma de inserirmos esse Modelo Relacional dentro do SGBD, para isso é que existem as linguagens de definição de dados (DDL – *Data Definition Language*), as linguagens de manipulação de dados (DML – *Data Manipulation Language*) e a linguagem de consulta de dados (DQL – *Data Query Language*).

Quais são os comandos que constituem a linguagem de definição de dados (DDL)? Quais são os comandos que fazem parte da linguagem de manipulação de dados (DML)? Como realizar uma consulta envolvendo mais de uma tabela (relação)? Quais são as funções de agregação mais utilizadas?



## OBJETIVOS

- Compreender conceitos da linguagem SQL, seu funcionamento e aplicações práticas.
- Aprender a construir aplicações utilizando a linguagem SQL, por exemplo: criação dos objetos, manipulação dos dados e consultas aos dados de forma clara e objetiva.
- Realizar consultas consideradas complexas, as quais utilizam mais de uma tabela na linguagem SQL.



## 4.1 Conhecer a História da Linguagem da Linguagem SQL e suas Principais Características

A linguagem SQL (*Structured Query Language*) é uma linguagem cuja finalidade é facilitar o acesso às informações através de consultas, atualizações e manipulações de dados, esses armazenados em bancos de dados relacionais.

Não vamos nesta aula, cobrir totalmente todos os aspectos dessa linguagem, sendo que o principal objetivo aqui será dar um embasamento teórico sobre a linguagem de forma que você, futuro gestor de TI, tenha um alicerce o qual permitirá construir conhecimentos mais aprofundados sobre essa aula.

A linguagem SQL foi criada inicialmente pela IBM (*International Business Machine*) e desenvolvida pelo PhD Donald D. Chamberlin, na década de 70, com o nome *StructuredEnglish Query Language* (SEQUEL). A linguagem de consulta estruturada SEQUEL foi disponibilizada para uso de um banco de dados relacional da própria IBM, originando em um novo nome, agora SEQUEL-XRM (1974-1975).

Em 1979, integrantes do projeto de desenvolvimento do SYSTEM/R fundaram uma empresa, por ocasião, denominada Relational Software Inc., que disponibilizou o primeiro sistema de gerenciamento de banco de dados relacional comercial, totalmente baseado na linguagem SQL. Esse SGBD, intitulado de Oracle (o mesmo Oracle atualmente conhecido), foi o pioneiro na forte concorrência junto a IBM.

Infelizmente, devido a sua boa aceitação no mercado, surgiram os primeiros problemas operacionais, pois cada empresa passou a incorporar funcionalidades e comandos próprios à linguagem SQL, diferenciando-a da sua forma original. Para resolver tais problemas, o instituto ANSI (*American National Standards Institute*) passou a estabelecer através de um Comitê, normativas e critérios para definição de padrões para a linguagem SQL, que a partir desse período passou a ser referenciada ANSI/SQL.

É oportuno mencionar que, apesar dos esforços de entidades e institutos em estabelecer padrões para fornecer um trabalho adequado, infelizmente, ainda existem empresas implementando rotinas, funções e comandos totalmente fora do padrão estabelecido (ANSI/SQL), dificultando o cotidiano dos profissionais da área de Tecnologia da Informação (TI), que, em alguns casos

específicos, utilizam mais de um sistema de gerenciamento de banco de dados em um mesmo ambiente operacional.

A linguagem SQL utilizada no SGBD PostgreSQL é subdividida em basicamente quatro subconjuntos que constituem a base das instruções: DML (*Data ManipulationLanguage*), DDL (*Data DefinitionLanguage*), DQL (*Data Query Language*) e DCL (*Data ControlLanguage*). Em algumas bibliografias, podemos encontrar ainda, os subconjuntos SRC (*StoredRoutinesCommands*) e DTC (*Data TypeCommands*).

## CONEXÃO

Leia mais sobre Banco de Dados Orientados a Objetos em: <[http://www.malima.com.br/article\\_read.asp?id=40](http://www.malima.com.br/article_read.asp?id=40)>

## 4.2 Aprender os Comandos de Criação de Esquemas

A linguagem de definição de dados (DDL – *Data DefinitionLanguage*) é um subconjunto que promove recursos para definir objetos e controlar os dados, ou seja, são comandos responsáveis pela estruturação do banco de dados, como a própria criação do banco de dados (*database*), criação das tabelas, dos índices, dentre demais possibilidades.

O PostgreSQL, MySQL, Oracle e Microsoft SQL Server são exemplos de SGBDs relacionais que suportam um banco de dados relacional. O SGBD desonera o programador e ou usuário final da complexidade do armazenamento dos dados, separando as visões lógica e física dos dados.

LAUDON, K.; LAUDON, J.; Sistemas de Informações Gerenciais. 9ª ed. São Paulo: Pearson, 2011.

Primeiramente, para interagirmos com as principais operações, torna-se necessário iniciar a ferramenta, ora nomeada de pgAdminIII, estabelecer a

conexão com o sistema gerenciador de banco de dados através da autenticação (login), utilizando o usuário “*root*” nomeado de “*postgres*”, informando sua respectiva “*password*”, ora configurada na instalação do SGBD.

O comando CREATE DATABASE nos proporciona a definição do esquema do banco de dados, o qual iremos utilizar durante todo este capítulo.

```
CREATE DATABASE nome  
[argumentos];
```

Veja abaixo o exemplo da criação de um banco de dados chamado de “ESTÂCIO” utilizando o comando CREATE DATABASE. Como vimos anteriormente, devemos evitar acentuação nos nomes de bancos, tabelas, colunas, etc.

```
CREATE DATABASE “ESTACIO”;
```

Após a execução desse comando, o banco de dados nomeado de ESTACIO é criado. Para continuarmos o estudo dessa unidade, é necessário trocar de banco de dados (*database*), pois, o banco de dados “*postgres*”, foi utilizado apenas para criar nosso novo banco de dados, chamado de ESTACIO, não podendo conter nenhum objeto, isto é, esse banco de dados é utilizado pelos DBA (*Database Administrator*) para desempenhar funções administrativas, por exemplo, criação de novos bancos de dados, adequação de desempenho do banco de dados (*tuning*), entre outras possibilidades.

Se, eventualmente, necessitarmos de excluir um banco de dados (*database*), podemos optar em realizar o comando DROP DATABASE através da interface pgAdminIII e ou via interface de linha de comando (psql), por meio do comando “dropdb”. Abaixo podemos visualizar um exemplo de comando o qual remove o banco de dados (*database*) identificado de ESTACIO, via interface pgAdminIII:

```
DROP DATABASE “ESTACIO”;
```

## 4.3 Aprender os Comandos para Criação e Destruição de Tabelas

Agora que já criamos nosso banco de dados nomeado de ESTACIO e, já nos encontramos conectado nele, podemos dar sequência em nossos estudos.

Todos nós sabemos que um banco de dados é constituído de tabelas, as quais são criadas através do comando `CREATE TABLE`. Para explorarmos esse comando, utilizaremos um modelo relacional simples e já normalizado. Dessa maneira, cada uma das relações é, na verdade, uma tabela do banco de dados.

A sintaxe básica do comando `CREATE TABLE` é apresentada abaixo:

```
CREATE TABLE nome da tabela (  
    COLUNA1 - TIPO DE DADOS [restrição],  
    COLUNA2 - TIPO DE DADOS [restrição],  
    PRIMARY KEY (coluna1, coluna2),  
    FOREIGN KEY (coluna1) REFERENCES nome da tabela (colunas)  
    CONSTRAINT restrição);
```

Onde “nome da tabela” indica o nome da tabela a ser criada no banco de dados, “coluna1” e “coluna2” sinalizam os nomes dos campos a serem definidos e “tipo de dados” define o tipo de dado a partir de uma lista de tipos padrão.

O PostgreSQL trabalha com diversos tipos de dados, classificados de acordo com o conteúdo que será utilizado em uma determinada coluna (campo). A seguir, podemos visualizar os principais tipos de dados suportados pelo nosso SGBD, ou seja, tipos de dados (atributos) mais comuns.

- *bigint*: representa valores inteiros da faixa de -9.223.372.036.854.775.808 até 9.223.372.036.854.775.807;
- *bigserial*: gera um valor único sequencial para um novo registro entre 1 e 9.223.372.036.854.775.807;
- *char*(tamanho) ou *character*(tamanho): são sequências de caracteres de tamanho fixo limitados a 255 caracteres de comprimento. O parâmetro tamanho determina o valor máximo em caracteres que pode conter a sequência. Esse tipo de dado, quando definido, preenche o campo com espaços em branco até

completar o total de caracteres definidos, quando a totalidade do tamanho do campo não é preenchida;

- *date*: data de calendário no formato AAAA-MM-DD (formato ANSI) com intervalo de tempo de 4.713 AC a 5.874.897 DC;
- *decimal*: determina a precisão do valor de casas decimais;
- *double*: determina a precisão do valor de até 15 casas decimais;
- *integer ou int*: representa valores inteiros na faixa de -2.147.483.648 até 2.147.483.647;
- *money*: define valores monetários na faixa numérica de -92.233.720.368.547.758.08 até 92.233.720.368.547.758.07;
- *numeric*: determina a precisão do valor de casas decimais;
- *real*: determina a precisão do valor de até seis casas decimais;
- *serial*: gera um valor único inteiro sequencial para o novo registro entre 1 e 2.147.483.647;
- *smallint*: representa valores inteiros na faixa de 32.768 até 32.767;
- *time*: representa um determinado horário de relógio no intervalo de tempo entre 00:00:00 e 24:00:00;
- *varchar(tamanho)*: permite o armazenamento de uma sequência de dados de caracteres com comprimento variável. Ao contrário do tipo CHAR, não armazena espaços não utilizados para o armazenamento do *string* (em branco) em seu lado direito, os seja, caso não sejam utilizados, os mesmo são descartados automaticamente. Como exemplo, a designação VARCHAR(25) possibilita o armazenamento de até 25 caracteres de comprimento, descartando eventuais espaços em branco não utilizados, caso eles ocorram.
- *blob*: campo do tipo BLOB com tamanho máximo de 65535.

## CONEXÃO

Leia mais sobre Tipos de Dados suportados pelo PostgreSQL em:

<[http:// www.webtuts.com.br/cotidiano/curso-postgresql-%E2%80%93-par-te-ii-tipos-de-dados](http://www.webtuts.com.br/cotidiano/curso-postgresql-%E2%80%93-par-te-ii-tipos-de-dados)>

---

Para remover uma tabela, o SQL oferece o comando DROP TABLE. Esse comando tem a seguinte sintaxe:

```
DROP TABLE <nome da tabela>;
```

Onde:

<nome da tabela>: é o nome da tabela que se deseja apagar

Ao apagar uma tabela você deve ter muito cuidado, pois pode haver alguma outra tabela que usa a chave primária da primeira como chave estrangeira. Além disso, ao se apagar uma tabela, todos os dados que estão nela são apagados também. (SETZER & DA SILVA, 2005).

Para editar uma tabela o SQL oferece o comando `altertable`. Quando desejasse adicionar um novo campo à tabela o comando tem a seguinte sintaxe:

```
ALTER TABLE <nome da tabela> ADD <nome do atributo> <tipo do atributo>;
```

Onde:

<nome da tabela>: é o nome da tabela na qual se deseja incluir o novo atributo;

<nome do atributo>: é o nome do novo atributo;

<tipo do atributo>: é o domínio do novo atributo. (semelhante ao do `createtable`)

Quando se deseja excluir um campo da tabela, o comando tem a seguinte sintaxe:

```
ALTER TABLE <nome da tabela> DROP <nome do atributo> CASCADE;
```

A cláusula *cascade* utilizada faz com que qualquer referência ao <nome atributo> que esta sendo apagado seja também eliminada preservando a integridade do banco de dados. Você pode utilizar, ao invés do *cascade*, a cláusula *restrict*, que só produzirá a eliminação do atributo em questão.

Uma tabela, que já existe no banco de dados, possivelmente já terá algumas linhas de dados armazenadas. Então, ao se criar um novo atributo para esta tabela, ele deverá receber o valor *null*. Por isso, na sintaxe do comando *altertable* não aparece a opção de *notnull*. (SETZER& DA SILVA, 2005).

Depois que conhecemos como criar tabelas e os tipos de dados mais usuais do PostgreSQL, já estamos preparadas para definir uma grande variedade de campos para a construção das tabelas utilizadas nessa aula. Em nosso banco de dados, ora identificado como ESTACIO, criaremos um conjunto de tabelas, conforme podemos visualizar no modelo abaixo representado pela figura 4.1:

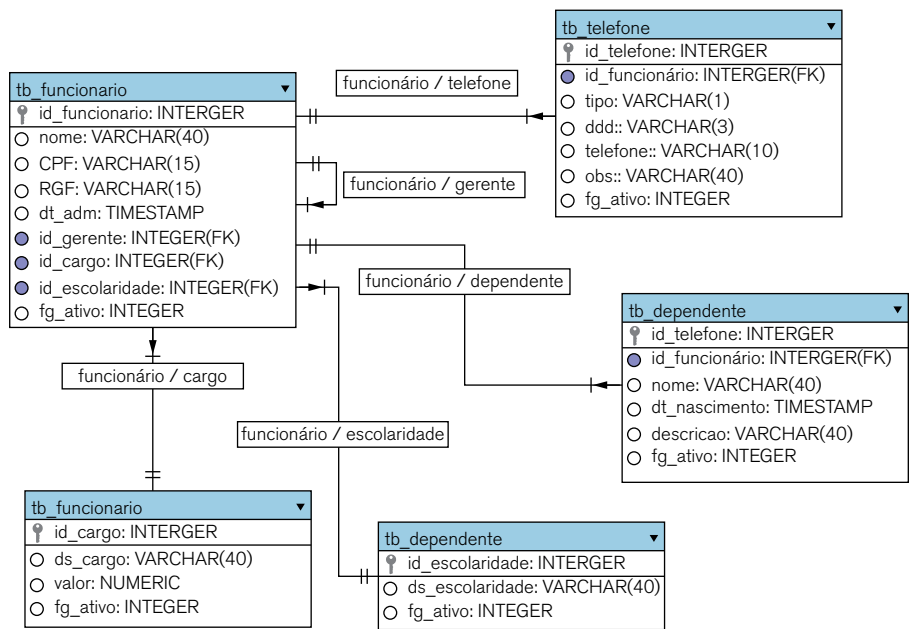


Figura 4.1 – Modelo físico.

Levando em consideração que você atendeu a sugestão e instalou o PostgreSQL conforme apresentado no Anexo 1, em seu *desktop* deverá existir um ícone com a imagem de um “elefante azul”, representando o “pgAdminIII”, conforme podemos visualizar na figura 4.2 a seguir:



Figura 4.2 – Acessando a interface pgAdmin III.

Dê um duplo clique com o botão esquerdo do “*mouse*” sobre qualquer um dos ícones em destaque. Esse procedimento exibirá a interface do “pgAdminIII”, conforme você pode visualizar na figura 4.3 a seguir:



Figura 4.3 – Tela principal do pgAdmin III.

Existe a necessidade de conectarmos no SGBD, aplicando um clique com o botão direito do mouse sobre o servidor, nomeado de “PostgreSQL 9.0 (localhost:5432)”, selecionando a opção “*connect/conectar*” ou apenas dê um duplo clique com o botão esquerdo do *mouse*. Na sequência, informe a senha que você utilizou na instalação do PostgreSQL, conforme apresentado na figura 4.4:





Figura 4.4 – Conectando-se ao SGBD PostgreSQL.

Agora que você encontra-se conectado ao nosso SGBD, uma tela semelhante à apresentada na Figura 58 será exibida. Você pode perceber que existe apenas um banco de dados (*database*) em nosso servidor, nomeado de “*postgres*”. Selecione o banco de dados “*postgres*” e clique com o botão esquerdo do mouse sobre o ícone “Query Tool” (Ctrl + E).

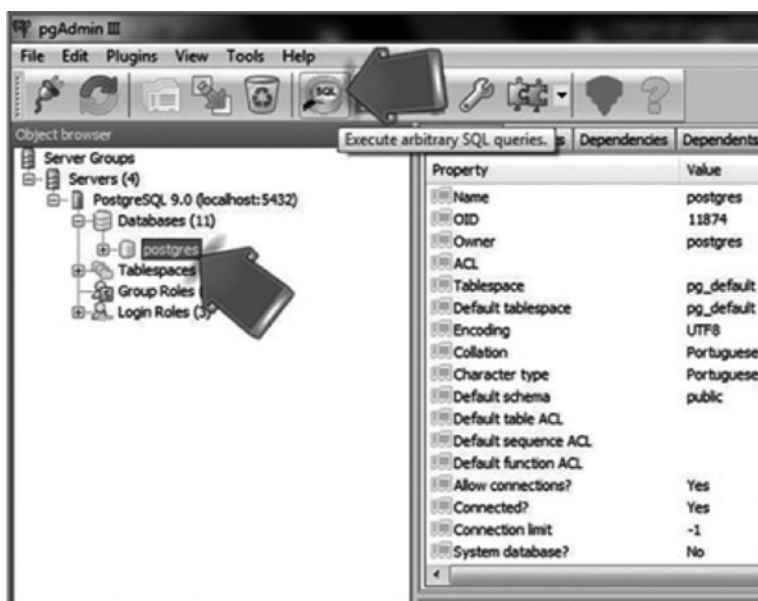


Figura 4.5 – Abrindo uma sessão SQL para o banco de dados postgres.

Bem, agora você acabou de criar uma sessão SQL para o banco de dados “postgres”, podemos criar um novo Banco de Dados (*database*), indicando o nome e seus principais argumentos, como visto anteriormente. Na sequência, clique no botão/ícone “Execute Query” (F5), e o seu banco de dados, ora identificado de “ESTACIO” será criado, conforme figura 4.6 a seguir:

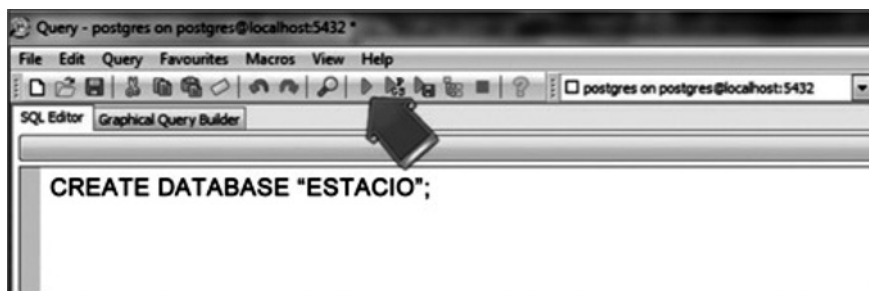


Figura 4.6 – Criação de um novo Banco de Dados.

Para criar os objetos (tabelas), selecione o banco de dados ESTACIO e clique com o botão esquerdo do *mouse* sobre o ícone Query Tool (Ctrl + E), criando uma nova sessão SQL, agora para o banco de dados ESTACIO. A partir de agora, podemos inserir os comandos SQL a serem aplicados para o banco de dados corrente (atual).

Nossa primeira tabela a ser criada no banco de dados ESTACIO, será a tabela responsável pelo armazenamento de dados pertinentes aos cargos dos funcionários, que é identificada de “tb\_cargo”. Veja abaixo os detalhes da estrutura da tabela “tb\_cargo”:

CAMPO	TIPO	DESCRIÇÃO
ID_CARGO	INTEGER	Identificador do cargo (não nulo)
DS_CARGO	VARCHAR(40)	Descrição do cargo (não nulo)
VALOR	NUMERIC(15,2)	Salário do funcionário (não nulo)
FG_ATIVO	INTEGER	Status do cargo (ativo/inativo)
Chave Primário	Campo/Atributo ID_CARGO	

O campo/atributo ID\_CARGO será definido como chave primária, ou seja, tornando proibido inserir dois cargos com o mesmo código. Caso deseje definir em uma tabela, um ou mais campo/atributo contendo registros exclusivos (únicos), utilize a cláusula UNIQUE.

Na sequência criaremos a tabela TB\_CARGO através a interface do “pgAdminIII”, utilizada anteriormente para a criação do banco de dados “ESTACIO”. Escreva as instruções como segue abaixo, dentro da sessão SQL do banco de dados ESTACIO. Para concluir a criação da tabela, clique no botão/ícone “Execute Query” (F5), como você pode ver na figura 4.7:

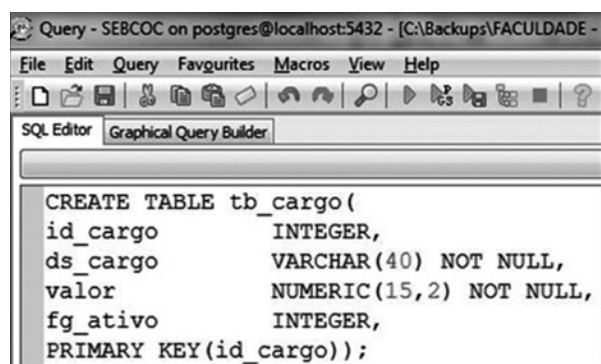


Figura 4.7 – Criação da tabela TB\_CARGO.

Quando criamos uma tabela, o PostgreSQL automaticamente cria um índice para cada chave primária (PRIMARY KEY), porém, em algumas situações faz-se necessário a criação de um outro índice, permitindo que o SGBD realize pesquisas no banco de dados de maneira eficiente, como por exemplo, você, eventualmente, poderá identificar a necessidade de criar um índice para a coluna NOME na tabela TB\_FUNCIONARIO, visto que, certamente, serão realizadas pesquisas pelo nome dos funcionários.

Uma tabela pode conter vários índices e os índices podem ser compostos.

Um SGBD normalmente permite a criação de um índice no momento em que uma tabela é criada ou então posterior a sua criação. (SETZER & DA SILVA, 2005).

Nossa segunda tabela a ser criada em nosso banco de dados será a tabela responsável pelo armazenamento de dados oriundos da escolaridade dos funcionários, nomeada de TB\_ESCOLARIDADE. Observe a seguir a estrutura da tabela TB\_ESCOLARIDADE:

CAMPO	TIPO	DESCRIÇÃO
ID_ESCOLARIDADE	INTEGER	Identificador da escolaridade (não nulo)
DS_ESCOLARIDADE	VARCHAR(40)	Descrição da escolaridade (não nulo)
FG_ATIVO	INTEGER	Status da escolaridade (ativo/inativo)
Chave Primário	Campo/Atributo ID_ESCOLARIDADE	

Agora que identificamos com detalhes o conteúdo da tabela TB\_ESCOLARIDADE, escreva a sequência de instruções, conforme você pode visualizar na figura 4.8, dentro da sessão SQL do banco de dados ESTACIO. Clique no botão/ícone “Execute Query” (F5) para concluir a criação da tabela.

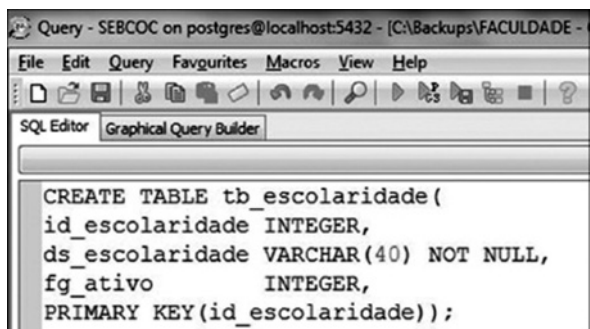
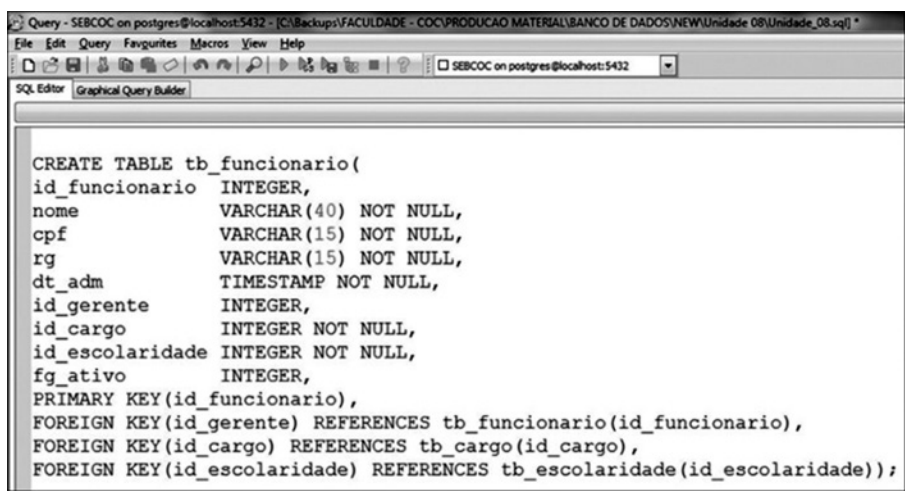


Figura 4.8 – Criação da tabela TB\_ESCOLARIDADE

A terceira tabela a ser criada em nosso banco de dados será responsável pelo armazenamento dos dados referente aos funcionários da nossa empresa fictícia. Vamos identificar esta tabela como TB\_FUNCIONARIO. Observe a seguir a estrutura da tabela TB\_FUNCIONARIO.

CAMPO	TIPO	DESCRIÇÃO
ID_FUNCIONÁRIO	INTEGER	Identificador do funcionário (não nulo)
NOME	VARCHAR(40)	Nome do funcionário (não nulo)
CPF	VARCHAR(15)	CPF do funcionário (não nulo)
RG	VARCHAR(15)	RG do funcionário (não nulo)
DT_ADM	TIMESTAMP	Data e hora de admissão do funcionário
ID_GERENTE	INTEGER	Identificador do gerente do funcionario
ID_CARGO	INTEGER	Identificador do cargo do funcionário (não nulo)
ID_ESCOLARIDADE	INTEGER	Identificador da escolaridade do funcionário (não nulo)
FG_ATIVO	INTEGER	Status do funcionário (ativo/inativo)
Chave Primária	Campo/Atributos: ID_FUNCIONÁRIO	
Chave Estrangeira	Campo/Atributos: ID_GERENTE, ID_CARGO e ID_ESCOLARIDADE	

Posteriormente, insira as instruções, dentro da sessão SQL do banco de dados ESTACIO. Clique no botão/ícone “Execute Query” (F5) para efetivar a criação da tabela, conforme você pode ver na figura 4.9:



```
CREATE TABLE tb_funcionario(  
id_funcionario    INTEGER,  
nome              VARCHAR(40) NOT NULL,  
cpf               VARCHAR(15) NOT NULL,  
rg                VARCHAR(15) NOT NULL,  
dt_adm            TIMESTAMP NOT NULL,  
id_gerente        INTEGER,  
id_cargo           INTEGER NOT NULL,  
id_escolaridade   INTEGER NOT NULL,  
fg_ativo          INTEGER,  
PRIMARY KEY(id_funcionario),  
FOREIGN KEY(id_gerente) REFERENCES tb_funcionario(id_funcionario),  
FOREIGN KEY(id_cargo) REFERENCES tb_cargo(id_cargo),  
FOREIGN KEY(id_escolaridade) REFERENCES tb_escolaridade(id_escolaridade));
```

Figura 4.9 – Criação da tabela TB\_FUNCIONARIO.

Veja que nessa tabela foram indicadas: uma chave primária e três chaves estrangeiras. A chave primária é uma chave primária simples, ou seja, composta por apenas um campo/atributo, ID\_FUNCIONARIO. Os campos ID GERENTE, ID\_CARGO e ID\_ESCOLARIDADE são chaves estrangeiras, definida por meio da cláusula FOREIGN KEY.

A fim de especificar para qual tabela a chave estrangeira foi criada, utilizamos REFERENCES, permitindo que o SGBD apenas aceite valores pré-cadastrados no campo indicado da tabela referenciada.

A tabela TB\_DEPENDENTE será nossa quarta tabela a ser criada em nosso banco de dados. Essa tabela será responsável por armazenar dados de todos os dependentes, ora vinculados a um funcionário específico.

Observe a estrutura utilizada para a tabela TB\_DEPENDENTE:

CAMPO	TIPO	DESCRIÇÃO
ID_DEPENDENTE	INTEGER	Identificador do dependente (não nulo)
ID_FUNCIONÁRIO	INTEGER	Identificador do funcionário
NOME	VARCHAR(40)	Nome do dependente (não nulo)
DT_NASCIMENTO	TIMESTAMP	Data de nascimento do dependente (não nulo)
DESCRIÇÃO	VARCHAR(40)	Descrição do dependente (não nulo)
FG_ATIVO	INTEGER	Status do funcionário (ativo/inativo)
Chave Primária	Campo/Atributos: ID_DEPENDENTE	
Chave Estrangeira	Campo/Atributos: ID_FUNCIONÁRIO	

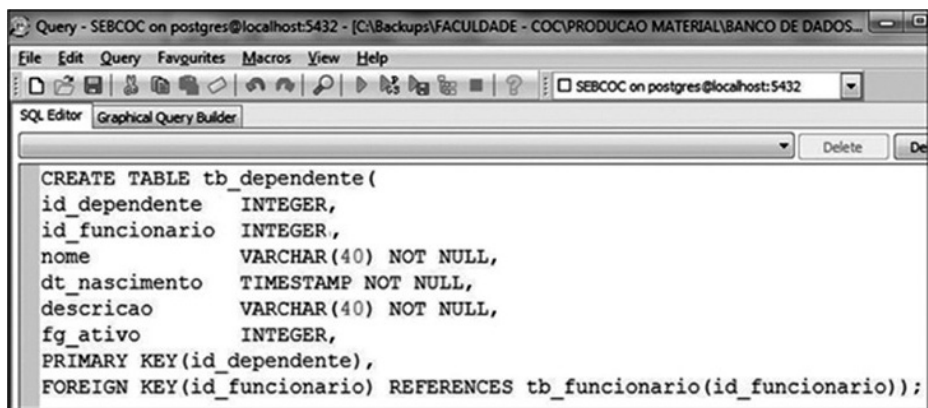


Figura 4.10 – Criação da TB\_DEPENDENTE.

O campo ID\_FUNCIONARIO é chave estrangeira, definida pela cláusula FOREIGN KEY. A palavra-chave REFERENCES é utilizada para indicar que uma referencia (vínculo) foi criada, fazendo com que o SGBD somente aceite valores previamente cadastrados no campo indicado da tabela referenciada.

A quinta e última tabela pertencente ao nosso banco de dados armazenará os telefones dos funcionários. Vamos identificar esta tabela como TB\_TELEFONE. A estrutura da tabela pode ser visualizada a seguir:

CAMPO	TIPO	DESCRIÇÃO
ID_TELEFONE	INTEGER	Identificador do telefone (não nulo)
ID_FUNCIONÁRIO	INTEGER	Identificador do funcionário (não nulo)
TIPO	VARCHAR(1)	Tipo de telefone (não nulo)
DDD	VARCHAR(3)	DDD do telefone (não nulo)
TELEFONE	VARCHAR(10)	Telefone do funcionário (não nulo)
OBS	VARCHAR(40)	Observação do telefone
FG_ATIVO	INTEGER	Status do funcionário (ativo/inativo)
Chave Primário	Campo/Atributo ID_TELEFONE	
Chave Estrangeira	Campo/Atributo ID_FUNCIONÁRIO	

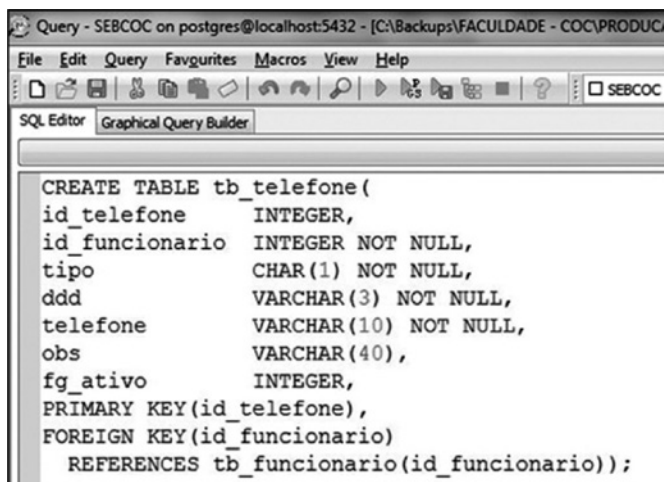


Figura 4.11– Criação da TB\_TELEFONE.

## 4.4 Inserir, Modificar e Excluir Registros nas Tabelas

A linguagem DML é utilizada para manipular os dados em um banco de dados. Deve ficar claro para você que manipular dados refere-se às três operações básicas possíveis: inserir novos dados; alterar dados existentes e remover os dados. Os comandos responsáveis por essas três operações são, respectivamente, INSERT, UPDATE e DELETE.

- **Inserindo registros (INSERT)**

A instrução INSERT é responsável por incluir novas linhas (registros) nas tabelas existentes no banco de dados. Sua sintaxe é bastante simples, veja a seguir:

```
INSERT INTO <TABELA> (CAMPO1, CAMPO2, ..., CAMPOn)  
VALUES (VALOR1, VALOR2, ..., VALORn);
```

Os valores informados no comando INSERT, obrigatoriamente, deverão ser respectivamente do mesmo tipo do campo, isto é, se o primeiro campo do comando for do tipo INTEGER, o primeiro valor deverá também ser do tipo INTEGER. Se, eventualmente, essa regra não for contemplada, o SGBD emitirá uma mensagem de erro, e como consequência, não efetuará a inserção.

Você acabou de obter algumas informações acerca da instrução INSERT, agora você já é capaz de adicionar algumas linhas para as tabelas TB\_CARGO, TB\_ESCOLARIDADE, TB\_FUNCIONARIO, TB\_DEPENDENTE e TB\_TELEFONE.

Veja abaixo o exemplo do uso da instrução INSERT para adicionar algumas linhas em cada tabela. Você deverá perceber que existe uma sequência para adicionar essas linhas, principalmente quando algumas tabelas possuem integridade referencial. Isso significa que, por exemplo, não podemos inserir uma linha na TB\_FUNCIONARIO, sem antes adicionar linhas nas tabelas TB\_ESCOLARIDADE e TB\_CARGO, visto que ambas são referenciadas na TB\_FUNCIONARIO pela chave estrangeira ID\_CARGO e ID\_ESCOLARIDADE.

Inserindo registro na tabela "tb\_cargo"

```
INSERT INTO tb_cargo(id_cargo,  
                    ds_cargo,  
                    valor,  
                    fg_ativo)
```

**VALUES**

```
(1, 'Analista de Sistema', 42000.00,1),  
(2 'Administrador de BD', 7900.00,1),  
(3, 'Desenvolvedor C#', 5740.00,1);
```

Inserindo registro na tabela "tb\_escolaridade"

```
INSERT INTO tb_escolaridade(id_escolaridade,  
                            ds_escolaridade,  
                            fg_ativo)
```

**VALUES**

```
(1, 'Ensino Médio',1),  
(2 'Graduação',1),  
(3, 'Pós-graduado,1);
```



Inserindo registro na tabela "tb\_funcionario"

```
INSERT INTO tb_funcionario(id_funcionario,
                           nome,
                           cdp,
                           rg,
                           dt_adm,
                           id_gerente,
                           id_cargo,
                           id_escolaridade,
                           fg_ativo)

VALUES
(1, 'João Roberto da Silva', '12332145699', '122344344320', '31/05/2001', null,
1, 2, 1),
(2 'Miguel dos Santos', '87612354677', '876523333', '01/12/2003', 1,1, 2 1),
(3, 'Maria dos Santos', '34576512344', '765465230', '28/02/2004', 1,2, 2, 1);
```

Inserindo registro na tabela "tb\_dependente"

```
INSERT INTO tb_dependente(id_dependente,
                           id_funcionario,
                           nome,
                           dt_nascimento,
                           descrição,
                           fg_ativo)

VALUES
(1, 1, 'Roger da Silva Mendes', '02/03/2003', 'Filho',1),
(2, 1, 'Bruna da Silva', '31/12/2011', 'Filha',1),
(3, 3,'Rafael dos Santos', '01/05/2006', 'Filho',1);
(4, null, 'Jéssica da Cruz', '21/04/2010', 'Filha',1);
```

Achou estranho o campo/atributo DT\_NASCIMENTO da tabela TB\_DEPENDENTE ter recebido uma data em um formato considerado não convencional e entre aspas, como se fosse um texto. Bem, aqui vai à explicação: quando criamos a tabela TB\_DEPENDENTE, definimos o campo/atributo

DT\_NASCIMENTO com o tipo `TIMESTAMP`, o qual armazena datas no formato DD/MM/AAA, ou seja, ano com quatro dígitos, mês e dia com dois dígitos. A respeito das aspas, o PostgreSQL automaticamente converte o texto, que está no formato de data e hora, para uma data.

Inserindo registro na tabela "tb\_telefone"

```
INSERT INTO tb_telefone(id_telefone,
                        id_funcionario,
                        tipo,
                        ddd,
                        telefone,
                        obs
                        fg_ativo)
VALUES
(1, 1, 'M', '016', '99886565', 'Celular Empresa',1),
(2, 1, 'M', '016', '78762233', 'Celular Particular', 1),
(3, 2, '016', '34565656', 'Residencial',1);
(4, 3, 'C', '019', '56451234',1);
```

- **Modificando registros (UPDATE)**

A instrução `UPDATE` permite alterar registros existentes em tabelas do banco de dados. É importante que você não altere os valores dos campos/atributos que façam parte da chave primária, evitando assim, erros graves de inconsistência de dados. Podemos especificar as seguintes informações em uma instrução `UPDATE`:

- A própria tabela que contém os registros a serem alteradas;
- a cláusula `WHERE`, a qual irá especificar quais registros deverão ser alterados;
- uma lista de nomes de colunas, associados com seus novos valores, especificado na cláusula `SET`, conforme você pode visualizar na sintaxe abaixo:

```
UPDATE <TABLE> SET  
    CAMPO1 = VALOR1,  
    CAMPO2 = VALOR2,  
    (...)  
    CAMPOn = VALORn  
WHERE <CONDIÇÃO LÓGICA>
```

Para exemplificar melhor o uso da instrução UPDATE, vamos atualizar o nome para “João Roberto Junqueira” do funcionário cujo ID\_FUNCIONARIO corresponda ao número 1. Tome cuidado para não se esquecer de adicionar a cláusula WHERE, caso contrário, todos os registros serão atualizados.

```
UPDATE tb_funcionario  
SET nome = 'João Roberto Junqueira'  
WHERE id_funcionario = 1;
```

- **Apagando registros (DELETE)**

DELETE é a instrução responsável por remover registros de uma determinada tabela do banco de dados. Semelhante à instrução UPDATE, a cláusula WHERE deverá ser utilizada a fim de limitar os registros que se deseja excluir, caso contrário, todos os registros serão excluídos da tabela. Você pode observar abaixo a simplicidade da sintaxe:

```
DELETE FROM <TABELA>  
WHERE <CONDIÇÃO LÓGICA>;
```

Bem, agora você já está preparado para remover um telefone armazenado na tabela TB\_TELEFONE, onde seu identificador (ID\_TELEFONE) corresponde ao número 2, telefone associado ao funcionário “João Roberto Junqueira”. Muita atenção para não se esquecer de incluir a cláusula WHERE da instrução DELETE, caso contrário, todos os registros serão removidos de sua tabela.

```
DELETE
```

```
FROM tb_telefone
```

```
WHERE id_telefone = 2;
```

## 4.5 Aprender Comandos Básicos para a Recuperação dos Dados Contidos nas Tabelas

A linguagem SQL inclui uma linguagem que permite recuperar os dados inseridos em um banco de dados através de consultas baseada na álgebra relacional e cálculo relacional de tupla. Para tanto, utiliza-se o comando **SELECT** que veremos em detalhes nos tópicos a seguir.

## 4.6 Conhecer um Pouco mais Sobre a Linguagem SQL

Para consultarmos os dados de uma tabela, a SQL nos fornece a instrução **SELECT**. Por meio dessa instrução é possível selecionar certos atributos de (FROM) determinadas tabelas as quais (WHERE) certas condições são obedecidas.

A sintaxe da instrução de seleção segue a seguir.

```
SELECT <lista de atributos>
```

```
FROM <lista de tabelas>
```

```
WHERE <lista de condições>;
```

Para iniciar vamos pensar em consultas simples nas quais não aparece a parte de *where* da seleção. Vamos recuperar as colunas **ID\_FUNCIONARIO**, **NOME**, **CPF**, **RG** e **DT\_ADM** (data de admissão).

```
SELECT id_funcionario, nome, cpf, dt_adm  
FROM tb_funcionario;
```

Agora você irá recuperar todas as colunas (atributos) de uma determinada tabela utilizando o caractere especial asterisco (\*). Por exemplo, recuperar todas as colunas da tabela TB\_FUNCIONARIO.

```
SELECT*  
FROM tb_funcionario;
```

Você pode identificar que alguns funcionários possuem mais de um dependente e, portanto, aparecem duplicado. Como você pode eliminar os registros duplicados? **DISTINCT** é utilizado para eliminar os registros duplicados. Em nosso próximo exemplo, repetiremos a consulta anterior, agora utilizando o **DISTINCT** cujo objetivo é eliminar os dados duplicados.

```
SELECT DISTINCT (id_funcionario)  
FROM tb_dependente;
```

## 4.7 Aprender Sobre Outros Operadores da Cláusula Where

Podemos ainda, por meio da instrução **SELECT**, especificar as linhas a serem recuperadas por meio da cláusula **WHERE**. Esse tipo de recurso é importante visto que, eventualmente, você estará interessado apenas em um subconjunto muito pequeno de linhas, mesmo sabendo que a maioria dos sistemas gerenciadores de bancos de dados possui uma boa capacidade de armazenar grandes quantidades de linhas.

Para exemplificar, mesclaremos a instrução **SELECT** com a cláusula **WHERE**, onde a cláusula **WHERE** recuperará registro da tabela TB\_FUNCIONARIO onde a coluna ID\_FUNCIONARIO seja equivalente ao número 2.

```
SELECT*  
FROM tb_funcionario  
WHERE id_funcionario = 2;
```

Um valor nulo não é um *string* em branco, e sim um valor único cujo significado do valor da coluna é desconhecido. O funcionário correspondente ao ID\_FUNCIONARIO igual a 1 possui um valor nulo na coluna ID\_GERENTE. Isso simboliza que esse funcionário não é subordinado a ninguém, ele possui a maior hierarquia dentro de nossa empresa fictícia. Para que você possa verificar a existência de valores nulos, faça uso de IS NULL. No exemplo a seguir, o funcionário que possui um valor nulo na coluna ID\_GERENTE será recuperado:

```
SELECT id_funcionario,  
        nome,  
        dt_adm  
FROM tb_funcionario  
WHERE id_gerente IS NULL;
```

Nosso próximo exemplo, recuperaremos as colunas ID\_CARGO, DS\_CARGO e VALOR da tabela TB\_CARGO, onde o valor correspondente à coluna VALOR seja superior a R\$ 4.500,00 e inferior a R\$ 6.000,00:

```
SELECT id_cargo,  
        ds_cargo,  
        valor  
FROM tb_cargo  
WHERE valor > 4500 And valor< 6000;
```

Você pode ainda utilizar o NOT para inverter o significado de um operador, por exemplo: NOT LIKE, NOT IN, NOT BETWEEN e IS NOT NULL.

O operador LIKE é utilizado na cláusula WHERE para procurar um padrão em um determinado *string*. Os padrões podem ser especificados usando uma combinação de caracteres normais e os dois caracteres considerados curinga:

- Sublinhado (\_): corresponde a um caractere em uma posição específica;
- Porcentagem (%): corresponde a qualquer número de caracteres a partir da posição especificada.

- Exemplo do padrão (\_a%):

- Sublinhado corresponde a qualquer caractere na primeira posição;
- “a” corresponde a um caractere “a” na segunda posição;
- Porcentagem corresponde a todos os caracteres após o caractere “a”

Vamos colocar em prática o uso do operador LIKE, procurando o padrão ‘\_o%’ na coluna NOME da tabela TB\_FUNCIONARIO:

```
SELECT nome
FROM tb_funcionario
WHERE nome LIKE '_o%';
```

Podemos ainda recuperar registros cujo valor da coluna esteja em uma lista de valores, por meio do operador IN em uma cláusula WHERE. Em nosso próximo exemplo, recuperaremos os registros da tabela TB\_FUNCIONARIO onde os valores vinculados à coluna ID\_CARGO corresponda aos números 1, 2 ou 3:

```
SELECT*
FROM tb_funcionario
WHERE id_cargo IN (1,2,3);
```

Semelhante ao operador IN, o operador BETWEEN também é utilizado na cláusula WHERE, permitindo recuperar os registros onde o valor correspondente a uma determinada coluna encontre-se em um determinado intervalo. É importante observar que o intervalo inclui valores das duas extremidades. Para o próximo exemplo, o operador BETWEEN recuperará registros da tabela TB\_DEPENDENTE onde valores associadas à DT\_NASCIMENTO encontre-se no intervalo de ‘01/01/2000’ a ‘31/12/2005’:

```
SELECT nome, dt_nascimento, descricao  
FROM tb_dependente  
WHERE dt_nascimento BETWEEN '01/01/200' AND '31/12/2005';
```

## 4.8 Aprender Sobre Cláusulas de Ordenação, Cálculos e Apresentação de Dados

A cláusula **ORDER BY** é utilizada para classificar os registros oriundos de uma determinada consulta. Você poderá especificar uma ou mais colunas para a classificação dos dados.

Para exemplificar, usaremos a cláusula **ORDER BY** para classificar por **NOME** os registros pertinentes à tabela **TB\_DEPENDENTE**:

```
SELECT*  
FROM tb_dependente  
ORDER BY nome;
```

Você pode utilizar a palavra-chave **DESC** para classificar as colunas em ordem decrescente, e, a palavra-chave **ASC** poderá ser utilizada para especificar explicitamente uma classificação crescente. Vale a pena lembrar que, por padrão, os registros serão classificados de ordem crescente (*default*).

Utilize esse próximo exemplo para explorar o uso da cláusula **ORDER BY** para classificar os registros provenientes da tabela **TB\_FUNCIONARIO** por **NOME** em ordem crescente e **DT\_ADM** em ordem decrescente:

```
SELECT*  
FROM tb_funcionario  
ORDER BY nome ASC, dt_adm DESC;
```



A cláusula ORDER BY ainda nos permite informar o número da posição da coluna, indicando assim qual coluna deve ser classificada. Você deverá utilizar 1 (um) para classificar pela primeira coluna, 2 para classificar pela segunda coluna e assim sucessivamente.

Em nosso exemplo a seguir, utilizaremos ORDER BY posicional. A coluna ID\_FUNCIONARIO, que em nosso caso corresponde à primeira coluna é utilizada para classificar os registros pertinentes à tabela TB\_FUNCIONARIO:

```
SELECT id_funcionario,  
        nome  
        dt_adm  
FROM tb_funcionario  
ORDER BY 1;
```

Grande parte dos SGBDs permite a realização de cálculos em instruções SQL por meio de expressões aritméticas. As expressões aritméticas consistem em basicamente dois operandos (números ou data) e um operador aritmético. Os operadores aritméticos são: adição (+), subtração (-), multiplicação (\*) e divisão (/).

Como exemplo, a expressão  $13 * 2.75 / 3 - 1$  será executada por meio da instrução SELECT.

```
SELECT 13*2,75/3 - 1;
```

Outro detalhe importante é que, podemos utilizar operadores de adição e subtração com datas. No exemplo a seguir, você irá perceber que foram adicionados dois dias para data especificada como 25 de junho de 2012:

```
SELECT DATE ('25-JUN-2012') + 2;
```

Além das operações aritméticas efetuadas até o presente momento com números literais ou datas, você pode utilizar os valores armazenados em colunas na realização dos cálculos aritméticos. Em nosso exemplo a seguir, vamos utilizar a coluna VALOR da tabela TB\_CARGO, onde, valores associados à coluna VALOR serão somados a R\$ 350,00:

```
SELECT ds_cargo  
       valor + 350,00  
WHERE tb_cargo
```

As funções agregadas operam em um grupo de linhas e retornam uma linha de saída. Essas funções de agregação também são conhecidas como funções de grupo, pelo simples fato de trabalhar em grupo de linhas. As funções agregadas podem ser utilizadas em qualquer expressão válida, por exemplo, COUNT(), MAX() e MIN() com números, strings e data/ hora. Outro detalhe importante, valores nulos são ignorados pelas funções agregadas (nulo indica um valor desconhecido). É permitido também o uso da palavra DISTINCT em uma função agregada para excluir entradas duplicadas.

A função COUNT(x) obtém o número de linhas retornadas por uma consulta. Vamos utilizar a função COUNT(x), para recuperar o número de linhas na tabela TB\_DEPENDENTE:

```
SELECT COUNT(id_dependente)  
FROM tb_dependente;
```

As funções MAX(x) e MIN(x) obtêm respectivamente o valor máximo e mínimo de “x”. Você pode colocar em prática o uso de ambas as funções para obter o valor máximo e mínimo da coluna VALOR da tabela TB\_CARGO:

```
SELECT MAX(valor), MIN(valor)  
FROM tb_cargo;
```

Um detalhe importante é que as funções MAX(x) e MIN(x) possibilitam a utilização de qualquer tipo de dados, inclusive *strings* e datas. Quando você utilizar MAX(x) com strings, as mesmas serão classificadas em ordem alfabética, com o *string* “máximo” no final da lista e o *string* “mínimo” no início. Vamos colocar em prática extraíndo os *strings* máximos e mínimos referentes à coluna NOME da tabela TB\_FUNCIONARIO:

```
SELECT MAX(nome), MIN(nome)
FROM tb_funcionario;
```

Você também pode obter a data máxima (mais recente) e a data mínima (mais antiga). Vamos utilizar novamente as funções agregadas MAX(x) e MIN(x) para manipular datas que serão recuperadas a partir do valor mínimo e máximo dos valores associados à coluna DT\_NASCIMENTO da tabela TB\_DEPENDENTE:

```
SELECT MAX(dt_nascimento), MIN(dt_nascimento)
FROM tb_dependente;
```



## ATIVIDADES

01. O PostgreSQL trabalha com diversos tipos de dados, classificados de acordo com o conteúdo que será utilizado em uma determinada coluna (campo). De acordo com o material e o que foi visto em aula, cite alguns dos tipos de dados possíveis para uma coluna (campo) de tabela de um PostgreSQL.

02. Utilizando os tipos de dados do banco PostgreSQL defina os tipos de dados para a criação de uma tabela CLIENTE que tenha os seguintes campos CLIENTE, RUA, NRO, DATA\_NASCIMENTO, SALARIO.



## REFLEXÃO

Neste capítulo, vimos alguns comandos da linguagem SQL. Para executar esses comandos, a maioria dos SGBDs modernos oferecem uma interface que se conecta a ele por meio da qual o usuário pode digitar as instruções SQL, executá-la e visualizar a resposta.

Os desenvolvedores de sistemas computacionais também utilizam a linguagem SQL para manipular os dados armazenados nos bancos de dados.



## LEITURA

Artigos *on-line*: para você aprimorar sua habilidade e conhecimento sobre a linguagem SQL, consulte as sugestões de *links* abaixo:

<<http://pgdocptbr.sourceforge.net/pg80/tutorial-sql.html>>

<<http://www.linhadecodigo.com.br/artigo/165/noco-es-da-linguagem-sql-para-consul-tas.aspx>>

<<http://www.devmedia.com.br/sql-e-programacao-de-banco-de-dados/3139>>

<<http://www.devmedia.com.br/o-guia-de-referencia-da-linguagem-sql/3211>>



## REFERÊNCIAS BIBLIOGRÁFICAS

DATE, C. J.; **Introdução a Sistemas de Banco de Dados**. 8ª ed. Trad. Daniel Vieira. Rio de Janeiro: Elsevier, 2003.

ELMASRI, R.; NAVATHE, S. B. **Sistemas de bancos de dados**. São Paulo: Pearson (Addison Wesley), 2005.

HEUSER, C. A. **Projeto de Bancos de Dados**. 2ª ed. Porto Alegre: Sagra Luzzatto, 1999.

HEUSER, C. A. **Projeto de Bancos de Dados**. 4ª ed. Instituto de Informática da UFRGS, Sagra DC Luzzatto, 1998.

KORTH, H.; SILBERCHATZ, A. **Sistemas de bancos de dados**. 3ª ed. São Paulo: Makron Books, 1998.

OLIVEIRA, A. R. F.; TAVEIRA L. M. P.; GILDA A. **Modelagem de Dados**. Rio de Janeiro. Ed. Senac Nacional, 2000.

PRESSMAN, R. S. **Engenharia de software**. São Paulo: Makron Books, 1995.

RAMAKRISHNAN, R; GEHRKE, J. **Database management systems**. 2ª ed. Boston: McGraw-Hill, 2000.

ROB, P. CORONEL, C. **Sistemas de Banco de Dados – projeto, implementação e administração**. Cengage Learning: 2011.

SETZER, V. W. **Dado, Informação, Conhecimento e Competência**. Acesso em: set. de 2008, disponível em Dado, Informação, Conhecimento e Competência: <http://www.ime.usp.br/~vwsetzer/dado-info.html>

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistema de Banco de Dados**. 5ª ed. Rio de Janeiro: Elsevier, 2006.

TEOREY, T.; LIGHTSTONE, S.; NADEAU, T.; JAGADISH, H. V.; **Database Modeling and Design: logical design**. 5ª ed., Burlington – USA: Elsevier, 2011.

TURBAN, E. et al. **Tecnologia da Informação para a Gestão**. 3a. Ed. São Paulo, SP: Bookman, 2004.

# 5

## **Linguagem SQL (Parte 2)**

No capítulo anterior descobrimos como é feito a inserção do Modelo Relacional dentro do SGBD, utilizando a linguagem de definição de dados (DDL – *Data Definition Language*). Em seguida utilizamos as linguagens de manipulação de dados (DML – *Data Manipulation Language*) e a linguagem de consulta de dados (DQL – *Data Query Language*) para manipular e recuperar os dados do banco de dados.

Neste capítulo avançaremos nossos estudos sobre a linguagem SQL. Veremos como realizar uma consulta envolvendo mais de uma tabela (relação)? Quais são as funções de agregação mais utilizadas?

Efetuiremos consultas aninhadas e aprenderemos o conceito e as propriedades das transações em um ambiente de banco de dados.



## OBJETIVOS

- Aprender como recuperar dados de mais de uma tabela simultaneamente.
  - Aprender sobre sinônimos e qualificadores.
  - Aprender sobre a utilização e construção de consultas aninhadas.
  - Aprender sobre o conceito e as propriedades das transações em um ambiente de banco de dados.
  - Aprender a criar índices e visões.
  - Conhecer os comandos que tratam da segurança do banco de dados.
-

## 5.1 Aprender Sobre Sinônimos e Qualificadores

Você não está limitado a utilizar apenas o cabeçalho padrão gerado pelo SGBD. É possível criar o seu próprio cabeçalho, usando um sinônimo (alias). Em nosso próximo exemplo, a expressão `VALOR*2`, a qual dobra o salário dos funcionários, receberá o apelido `DOBRO_SALÁRIO`:

```
SELECT ds_cargo,  
       valor * 2 DOBRO_SALÁRIO  
FROM tb_cargo;
```

O que houve de estranho? Você deve ter percebido que a caixa de texto não foi preservada (maiusculo). Para manter a caixa de texto (maiusculo e minusculo) e possibilitar o uso de espaços em nomes compostos do seu nome alternativo (apelido), simplesmente envolva-o entre aspas duplas, conforme o exemplo abaixo:

```
SELECT ds_cargo,  
       valor * 2 DobrO_SaLÁrIO  
FROM tb_cargo;
```

A palavra-chave `AS` antes do apelido é opcional. Você pode visualizar o resultado do exemplo a seguir, o qual utilizou a palavra-chave `AS` em seu nome alternativo (apelido):

```
SELECT 3.78 * (12-3/0.43) AS "Resultado da Expressão";
```

Veremos no próximo tópico a junção de duas ou mais tabelas em uma consulta SQL. Em muitas situações, de consultas com junções de tabelas, temos colunas com o mesmo nome, o que daria problema para o compilador identificar qual coluna a consulta esta se referindo. Para resolver este problema, temos

o qualificador. O qualificador consiste em acrescentar o nome da tabela a sua coluna, separado por um ponto. Desta forma o compilador tem a definição correta de qual coluna o usuário está se referindo em uma consulta SQL

```
SELECT tb_funcionario.nome  
FROM tb_funcionario
```



## CONEXÃO

Leia sobre ALIAS em uma Instrução SQL em: <[https://technet.microsoft.com/pt-br/library/ms187455\(v=sql.105\).aspx](https://technet.microsoft.com/pt-br/library/ms187455(v=sql.105).aspx)>.

## 5.2 Aprender como Recuperar Dados de mais de uma Tabela Simultaneamente

Quando mapeamos os relacionamentos do Modelo ER para o Modelo Relacional, começamos a posicionar os dados em tabelas diferentes, porém mantendo uma referência de uma tabela a outra por meio das chaves estrangeiras. Isso também pode acontecer quando trabalhamos em um banco de dados normalizado, onde se torna necessário manipularmos dados armazenados em diversas tabelas de maneira simultânea. Como exemplo, podemos citar a necessidade de obter a descrição do cargo e o nome do funcionário. As tabelas TB\_FUNCIONARIO e TB\_CARGO são relacionadas entre si através da coluna de chave estrangeira ID\_CARGO. A coluna ID\_CARGO (chave estrangeira – *foreignkey*) da tabela TB\_FUNCIONARIO, por sua vez, aponta para a coluna ID\_CARGO (chave primária – *primarykey*) da tabela TB\_CARGO.

Agora vamos aprender a escrever comandos SQL que relacionam essas tabelas.

Entendeu a necessidade de realizar a junção entre as duas tabelas (TB\_FUNCIONARIO e TB\_CARGO)? Agora você já está pronto para recuperar o nome do funcionário e a descrição do seu respectivo cargo usando uma única



consulta, através de uma junção (JOIN). É simples, apenas incluímos as duas tabelas na cláusula FROM da consulta e incluímos as colunas relacionadas de cada tabela na cláusula WHERE.

```
SELECT tb_funcionario.nome, tb_cargo.ds_cargo
FROM tb_funcionario, tb_cargo.
WHERE tb_funcionario.id_cargo = tb_cargo.id_cargo;
```

Analisando o exemplo, você pode perceber que a junção é a primeira condição da cláusula WHERE (`tb_funcionario.id_cargo = tb_cargo.id_cargo`). Normalmente as duas colunas na junção são chave primária (*primarykey*) de uma tabela e uma chave estrangeira (*foreignkey*) de outra tabela. Como utilizamos o operador de igualdade (=) na condição de junção, esta junção é referenciada de EQUIJOIN.

Em nosso exemplo, existe uma coluna ID\_CARGO na tabela TB\_FUNCIONARIO e outra na tabela TB\_CARGO, necessitando assim informar ao banco de dados em qual tabela está a coluna que se deseja utilizar. Se, por acaso, as colunas possuísem nomes distintos, não seria necessário informar os nomes das tabelas em questão.

Para nosso próximo exemplo de junção entre tabelas, realizaremos a junção entre as tabelas TB\_FUNCIONARIO e TB\_DEPENDENTE, obtendo todos os dependentes ordenados pela coluna NOME do funcionário.

```
SELECT tb_funcionario.nome,
        tb_dependente.name
FROM tb_funcionario, tb_dependente
WHERE tb_funcionario.id_funcionario = tb_dependente.id_funcionario;
ORDER BY 1;
```

Você pode observar com mais detalhes e, identificar que a dependente “Jéssica da Cruz” encontra-se ausente entre as linhas resultantes. Isso se deve ao fato de que o valor correspondente ao ID\_FUNCIONARIO para essa linha de dependente corresponde a um valor nulo e a condição da junção não retorna o registro. Como solução, torna-se necessário a utilização de uma junção externa (OUTER JOIN), recurso explorado ainda nessa aula.

A sintaxe de junção vista até o presente momento utiliza o padrão SQL/86 ANSI. Com objetivo de facilitar nosso trabalho, principalmente no que se refere ao uso de junção, utilizando apelidos (alias) para as tabelas, permitindo que seja possível inserirmos nomes alternativos para as tabelas. Para exemplificar o uso de nomes alternativos para as tabelas, a consulta a seguir utiliza o apelido “f” para referenciar a tabela TB\_FUNCIONARIO e “d” para referenciar a tabela TB\_DEPENDENTE:

```
SELECT d.nome,  
        d.name  
FROM tb_funcionario f, tb_dependente d  
WHERE f.id_funcionario = d.id_funcionario  
ORDER BY f.name;
```

A junção externa (OUTER JOIN) permite recuperar linhas mesmo quando uma de suas colunas contém um valor nulo. Vimos no exemplo anterior que “Jéssica da Cruz” cujo ID\_FUNCIONARIO da tabela TB\_DEPENDENTE é nulo, é recuperado apenas por meio da junção externa:

```
SELECT d.nome,  
        f.name  
FROM tb_dependente d  
LEFT OUTER JOIN tb_funcionario f ON (f.id_funcionario = d.id_funcionario);
```

As junções externas (OUTER JOIN) podem ser fragmentadas em dois tipos: junção externa esquerda (LEFT OUTER JOIN) e junção externa direita (RIGHT OUTER JOIN). Sua sintaxe pode ser compreendida como:

```
SELECT ...  
FROM tabela_x  
...
```

Imagine que as tabelas sejam unidas da seguinte maneira: tabela\_a.coluna\_a e tabela\_b.coluna\_b, e que a tabela\_a contenha uma linha com um valor nulo na coluna\_a, então, para realizar uma junção externa esquerda (LEFT OUTER JOIN), a instrução SQL seria constituída da seguinte maneira:

```
LEFT OUTER JOIN tabela_b ON (tabela_a.coluna_a = tabela_b.coluna_b)
```

Sendo assim, por outro lado, imagine que a tabela\_b contenha uma linha com um valor nulo na coluna\_b. A instrução SQL seria modificada para realizar uma junção externa direita (RIGHT OUTER JOIN):

```
RIGHT OUTER JOIN tabela_b ON (tabela_a.coluna_a = tabela_b.coluna_b)
```

## 5.3 Aprender Sobre a Utilização e Construção de Consultas Aninhadas

Podemos dividir as consultas aninhadas ou subconsultas em dois tipos: única linha, a qual retorna zero ou uma linha para a instrução SQL externa e, várias linhas, a qual retorna uma ou mais linhas para a instrução SQL externa.

Além disso, outros três tipos de subconsultas que retornam uma ou várias linhas, identificadas como correlacionadas, aninhadas e várias colunas. As correlacionadas referenciam uma ou mais colunas na instrução SQL externa. Elas são nomeadas de subconsultas “correlacionadas” porque são relacionadas à instrução SQL externa por meio das mesmas colunas. Já as aninhadas, por sua vez, são inseridas dentro de outra subconsulta. É permitido aninhar subconsultas até uma profundidade de 255. E, por fim, temos a subconsulta de várias colunas, a qual retorna mais de uma coluna para a instrução SQL externa.

Imagine a necessidade de recuperar os registros de NOME e VALOR salarial do funcionário que possui o maior salário. Você poderia utilizar a instrução SQL a seguir:

```

SELECT f.nome, c.valor
FROM tb_funcionario f
JOIN tb_cargo c ON(f.id_cargo = c.id_cargo)
WHERE c.valor = (SELECT MAX(c.valor)
                FROM tb_funcionario f
                JOIN tb_cargo c ON(f.id_cargo = c.id_cargo));

```

Ainda é possível utilizarmos outros operadores de comparação (<>, <, >, <= e >=) em uma subconsulta de uma única linha, por exemplo, recuperar os funcionários que possuem o valor salarial superior à média salarial, o qual será passado para a cláusula WHERE da consulta externa. A consulta interna retornará os valores de NOME, DS\_CARGO e VALOR salarial dos funcionários cujo valor salarial seja maior que a média de salários:

```

SELECT f.nome, c.valor
FROM tb_funcionario f
JOIN tb_cargo c ON(f.id_cargo = c.id_cargo)
WHERE c.valor = (SELECT AVG(valor)
                FROM tb_cargo);

```

Uma consulta de várias linhas retorna uma ou mais linhas para uma instrução SQL externa. Os operadores IN, ANY e ALL são utilizados para realizar o tratamento de uma subconsulta que retorna várias linhas, permitindo verificar se o valor de uma coluna está contido em uma lista de valores. Essa lista de valores pode ser proveniente dos resultados retornados por uma subconsulta. Vamos utilizar o operador IN, verificando se um valor de NOME da tabela TB\_FUNCIONARIO está incluído na lista de valores retorna da pela subconsulta:

```

SELECT id_funcionario
       nome
FROM tb_funcionario
WHERE nome IN (SELECT nome
                FROM tb_funcionario
                WHERE nome LIKE '%e%');

```

Você deve estar se perguntando: Por que uma consulta é considerada correlacionada? A resposta é simples, quando a mesma referência uma ou mais colunas na instrução SQL, caracterizando um vínculo à instrução SQL externo por meio das mesmas colunas.

Os operadores EXISTS e NOT EXISTS podem ser utilizados em uma subconsulta correlacionada. EXISTS verifica a existência de linhas retornadas por uma subconsulta. Você pode utilizar EXISTS em subconsultas não correlacionadas, todavia, normalmente esse operador é utilizado em subconsultas correlacionadas. Já NOT EXISTS executa a lógica inversa do EXISTS. Para exemplificar, vamos recuperar os funcionários que gerenciam outros funcionários:

```
SELECT id_funcionario  
       nome  
FROM tb_funcionario externa  
WHERE EXISTS (SELECT id_funcionario interna  
              FROM tb_funcionario interna  
              WHERE interna.idgerente = externa.id_funcionario);
```

O GROUP BY permite agrupar linhas em uma tabela e obter alguma informação sobre esses grupos de linhas.

Em nosso exemplo, a cláusula GROUP BY é utilizada para agrupar linhas em blocos com um valor comum de coluna, agrupando as linhas da tabela TB\_FUNCIONARIO com o mesmo valor de ID\_CARGO.

```
SELECT f.id_cargo, AVG(c.valor)  
FROM tb_funcionario f  
JOIN tb_cargo c ON(f.id_cargo = c.id_cargo)  
GROUP BY f.id_cargo;
```

Em nosso próximo exemplo, a cláusula GROUP BY é mesclada com a função COUNT(x), obtendo o cálculo no grupo de linhas em cada bloco, retornando um valor por bloco. O resultado desse exemplo é o número de linhas com o mesmo valor de DS\_CARGO da tabela TB\_CARGO:

```
SELECT c.ds_cargo, COUNT(f.id_cargo) AS "Total de Cargos"  
FRON tb_funcionario f  
JOIN tb_cargo c ON(f.id_cargo = c.id_cargo)  
GROUP BY c.id_cargo;
```

Existe algumas consultas em que se torna necessário filtrar grupos de linhas. A cláusula **HAVING** nos permite tal recurso. Ela deve ser inserida após a cláusula **GROUP BY**. Um detalhe importante que você deve observar é que, **GROUP BY** pode ser utilizado sem **HAVING**, mas **HAVING** sempre deverá ser usada em conjunto com **GROUP BY**.

Imagine que você tenha a necessidade de recuperar os salários médios superiores a R\$ 4.500,00. Para isso, a cláusula **GROUP BY** deverá ser utilizada para agrupar linhas em blocos com o mesmo valor de **DS\_CARGO** e **HAVING** será utilizado para limitar as linhas resultantes retornando apenas os grupos que possuem valores médios superiores à R\$4.500,00:

```
SELECT c.ds_cargo, AVG(c.valor)  
FRON tb_funcionario f  
JOIN tb_cargo c ON(f.id_cargo = c.id_cargo)  
GROUP BY c.id_cargo  
HAVING AVG(c.valor) > 4.500,00;
```

## 5.4 Aprender Sobre o Conceito e as Propriedades das Transações em um Ambiente de Banco de Dados

Como estudamos anteriormente uma transação é uma unidade lógica de trabalho; ela começa com a execução de uma operação **BEGIN TRANSACTION** e termina com a execução de uma operação **COMMIT** ou **ROLLBACK**; segundo este princípio, ou todos os registros correlatos são inseridos ou nada é registrado.

Desta forma, a consistência e integridade dos dados são asseguradas pelo SGBD (DATE, 2003).

No PostgreSQL um bloco de transação inicia-se com o comando `BEGIN`. Em seguida são definidos os comandos que serão realizados dentro do bloco. Por fim, utiliza-se o comando `COMMIT` ou `ROLLBACK`. O comando `COMMIT` confirma que todos os comandos foram realizados com sucesso e assim pode ser persistidos definitivamente no banco de dados. Já o comando `ROLLBACK` indica que houve algum erro e que todos os comandos realizados dentro do bloco de transação devem se descartados, ou seja, não devem ser persistido no banco de dados.

No exemplo a seguir, estamos inserindo telefones na tabela `tb_telefone` conforme fizemos no capítulo anterior. A diferença aqui, esta no fato de utilizarmos transação. Notem o comando `BEGIN` iniciando o bloco e o comando `COMMIT` finalizando o bloco. Após o comando se executado, podemos efetuar um comando `SELECT` para confirmarmos que os dados foram realmente inseridos no banco de dados.

```
BEGIN;

INSERT INTO tb_telefone(id_telefone, id_funcionario, tipo,
ddd, telefone, obs, fg_ativo)
VALUES
(1, 1, 'M', '016', '99886565', 'Celular Empresa', 1),
(2, 1, 'M', '016', '78762233', 'Celular Particular', 1),
(3, 2, 'R', '016', '34565656', 'Residencial', 1),
(4, 3, 'C', '019', '56451234', 'Comercial', 1);

COMMIT;
```

Neste outro exemplo, no final do bloco está sendo executado o comando `ROLLBACK`. Isto faz com que os `INSERTs` realizados anteriormente não sejam efetivados no banco de dados. Após a execução do bloco da transação, podemos executar uma consulta SQL para verificar que os dados não foram inseridos na tabela `tb_telefone`.

```
BEGIN;

INSERT INTO tb_telefone(id_telefone, id_funcionario, tipo,
ddd, telefone, obs, fg_ativo)
VALUES
(1, 1, 'M', '016', '99886565', 'Celular Empresa', 1),
(2, 1, 'M', '016', '78762233', 'Celular Particular', 1),
(3, 2, 'R', '016', '34565656', 'Residencial', 1),
(4, 3, 'C', '019', '56451234', 'Comercial', 1);

ROLLBACK;
```

## 5.5 Aprender a Criar Índices e Visões

Os índices agregam muito na resultante de consultas junto aos bancos de dados. Vamos supor que seja necessário localizarmos um livro específico em uma biblioteca. Este é o nosso problema!

Certamente, vasculhar a biblioteca olhando livro a livro, se aquele é o que desejamos, não será o meio mais viável, embora você deva concordar que o livro seria encontrado em algum momento.

É evidente que ninguém, principalmente nos dias atuais, têm tempo para esperar uma busca tão penosa quanto esta, necessita-se de informação rápida e disponível. O método mais adequado seria simplesmente fazer uso do catálogo da biblioteca. Certamente este catálogo irá apresentar índices por assunto, autor, título, entre outros.

Observe que o índice desta biblioteca apenas aponta para um local que o livro provavelmente será encontrado. Este mecanismo funciona igualmente no meio computacional.

Um índice é uma disposição ordenada utilizada para acessar logicamente as linhas de uma tabela.

Vamos imaginar que este livro que desejamos encontrar esteja compreendido no assunto “*tunning* de banco de dados”, faria sentido lermos todas as páginas do catálogo para encontrar este assunto? Obviamente que não, concorda? Seria muito mais fácil, conveniente e rápido irmos pelo índice de assunto até



encontrarmos a referência a *tunning*. De forma geral, verifica-se que o índice é utilizado para localizar de forma mais rápida um item requerido.

No contexto dos bancos de dados, os índices funcionam de forma muito similar a que descrevemos acima. Dentro de uma visão conceitual, o índice é composto de uma chave de índice e de um conjunto de ponteiros. A chave de índice nada mais é que o ponto de referência do índice.



## CONEXÃO

Entenda um pouco mais sobre índices consultando uma abordagem prática com o SGBD PostgreSQL,

<<http://www.inf.unioeste.br/~clodis/BDI/A11.pdf>>

Para ilustrar melhor o funcionamento dos índices dentro do domínio dos bancos de dados, vamos pensar na seguinte situação: necessita-se procurar todas as pinturas de um determinado pintor no banco de dados. Se nosso banco não estiver provido de índice, será necessário ler todas as linhas da tabela que representa pintura e verificar se o atributo `Cod_pintor` corresponde ao pintor ora solicitado. Neste mesmo contexto, se indexarmos a tabela pintor e utilizarmos a chave de índice de `Cod_pintor`, basta que procuremos apenas o valor do atributo no índice e encontrar os ponteiros correspondentes.

Os SGBD's utilizam índices para finalidades distintas. No exemplo que expomos você verificou que os índices auxiliariam na recuperação de registros. Mas eles vão além disso, por exemplo, é possível que registros sejam recuperados, ordenados por vários ou um determinado atributo. Pesquise mais sobre os tipos de índices comumente utilizados nos SGBD's (ROB, 2005).

Os índices executam um papel importante nos SGBDs para a implantação das chaves primárias. Toda vez que definimos a chave primária de uma tabela, automaticamente um índice é criado para aquele atributo. É importante salientar que uma tabela pode conter vários índices, sendo que cada um pode estar associado a apenas uma tabela, seja a um ou mais atributos, o que denomina-se por índice composto.

Para criar um índice utiliza-se o comando:

```
CREATE [UNIQUE] INDEX <nome_índice>  
ON <nome_tabela> (<lista_de_colunas>);
```

A cláusula UNIQUE é opcional, e a sua inclusão assegura a não existência de valores duplicados no índice a ser criado. O exemplo a seguir cria um índice para a coluna CPF da tabela funcionario. O nome do índice será idx\_CPF.

```
CREATE INDEX idx_CPF ON tb_funcionario(CPF);
```

Caso seja necessário excluir o índice, basta usar o commando DROP INDEX

```
DROP INDEX idx_CPF;
```

Utilizar junções com várias tabelas, durante o acesso há um banco de dados ou quando se esta desenvolvendo um aplicativo, pode ser bem trabalhoso dependendo da complexidade da junção.

Uma forma de facilitar a utilização de junções, e transformá-las em visões. As visões permite acessar uma junção como se fosse uma simples tabela.

Para criar uma visão utilizamos o comando CREATE VIEW como ilustrado abaixo:

```
CREATE VIEW <nome_visão> (<nomes das colunas>)  
AS <expressão_de_consulta>;
```

O exemplo a seguir cria uma visão chamada FuncDependente, referente a junção entre as tabelas tb\_funcionario e tb\_dependente. Notem que apenas algumas colunas de cada tabela foram utilizadas.

```
CREATE VIEW FuncDependente AS
SELECT tb_funcionario.nome As Funcionario, CPF, RG, tb_dependente.nome As De-
pendente, dt_nascimento
FROM tb_funcionario, tb_dependente
WHERE tb_funcionario.id_funcionario = tb_dependente.id_funcionario;
```

Para consultar a visão, utiliza-se o comando SELECT e no nome da tabela, preenche-se com o nome da visão.

```
SELECT * FROM FuncDependente;
```

Para excluir a visão utiliza-se o comando DROP VIEW.

```
DROP VIEW FuncDependente;
```

## 5.6 Conhecer os Comandos que Tratam da Segurança do Banco de Dados

A informação é um dos ativos mais preciosos de uma empresa. Dessa forma é importante que as informações contidas em um banco de dados sejam acessadas de forma segura e controlada. Os bancos de dados nos fornecem diversas ferramentas para que possamos garantir a segurança dos dados. Entre estas ferramentas, estão os comandos para conceder e retirar privilégios de usuários sobre os dados contidos em um SGBD.

Inicialmente veremos os comandos para criar e excluir usuários do banco de dados. Para que um usuário possa ter privilégios a objetos do banco de dados, tais como, tabelas, visões, etc., este usuário deve estar cadastrado no banco de dados. O comando CREATE USER permite a criação de um usuário como descrito no exemplo a seguir:

```
CREATE USER jose WITH PASSWORD 'maria';
```

Caso seja necessário excluir o usuário do banco de dados, basta utilizar o comando **DROP USER**.

```
DROP USER jose;
```

O privilégio de execução de comandos sobre um objeto do banco de dados é concedido através do comando **GRANT**. No exemplo a seguir, o usuário José esta recebendo permissão de consulta na tabela **tb\_funcionario**.

```
GRANT SELECT ON tb_funcionario TO Jose;
```

Os privilégios podem ser revogados com o comando **REVOKE**.

```
REVOKE ALL PRIVILEGES ON tb_funcionario FROM Jose;
```



## ATIVIDADES

01. Em uma tabela do banco de dados chamada **TB\_FUNCIONARIO** tem-se todos os dados referentes aos funcionários de uma determinada empresa. Qual das funções abaixo pode ser utilizada para retornar a quantidade de funcionários cadastrados.

- a) **LENGTH (ID\_Func)**
- b) **COUNT(ID\_Func)**
- c) **MIN(ID\_Func)**
- d) **AVG(ID\_Func)**
- e) **SUM(ID\_Func)**

02. Um analista de sistemas esta criando uma consulta SQL para um determinado relatório. Só falta a ordenação dos resultados para finalizar a consulta. De acordo com o material e o que foi visto em aula, qual o comando para SQL que o mesmo deve efetuar para esta tarefa?

- a) ORDER BY
- b) GROUP BY
- c) HAVING
- d) BETWEEN
- e) AND

03. Em uma empresa, para controlar os dados dos funcionários, tem-se uma tabela, no banco de dados, chamada TB\_FUNCIONARIO. Algumas das colunas desta tabela são COD\_FUNCIONARIO, NOME e SALARIO. O funcionário de código 345 recebeu um aumento salarial passando a ganhar R\$ 2.000,00. Escreva uma consulta SQL que atualize o salário deste funcionário.

---



## REFLEXÃO

Neste capítulo finalizamos nossos estudos sobre a linguagem SQL. Aprendemos a recuperar dados de mais de uma tabela simultaneamente. Vimos como utilizar os sinônimos, qualificadores e visões para construir consultas SQL que permitam leituras e compreensão mais simples. Estudamos índices e transações dentro do PostgreSQL.

Não pare os seus estudos, mantenha-se sempre lendo e pesquisando sobre os diversos temas relacionados aos sistemas de gerenciamento de banco de dados. Esperamos que todos estes conhecimentos adquiridos sejam um diferencial em sua vida profissional. Desejamos a você um grande sucesso!

---



## LEITURA

**Livro:** Sistemas de Banco de Dados: projeto, implementação e administração.

**Autor:** Peter Rob e Carlos Coronel.

**Editora:** Cengage Learning

**Ano:** 2010

Este livro trata da abordagem geral de sistemas de banco de dados, desde a constituição do projeto, implementação e administração. Existe um capítulo bem estruturado dedicado exclusivamente a linguagem SQL.



## REFERÊNCIAS BIBLIOGRÁFICAS

- DATE, C. J.; **Introdução a Sistemas de Banco de Dados**. 8ª ed. Trad. Daniel Vieira. Rio de Janeiro: Elsevier, 2003.
- ELMASRI, R.; NAVATHE, S. B. **Sistemas de bancos de dados**. São Paulo: Pearson (Addison Wesley), 2005.
- HEUSER, C. A. **Projeto de Bancos de Dados**. 2ª ed. Porto Alegre: Sagra Luzzatto, 1999.
- HEUSER, C. A. **Projeto de Banco de Dados**. 4ª ed. Instituto de Informática da UFRGS, Sagra DC Luzzatto, 1998.
- KORTH, H.; SILBERCHATZ, A. **Sistemas de bancos de dados**. 3ª ed. São Paulo: Makron Books, 1998.
- OLIVEIRA, A. R. F.; TAVEIRA L. M. P.; GILDA A. **Modelagem de Dados**. Rio de Janeiro. Ed. Senac Nacional, 2000.
- PRESSMAN, R. S. **Engenharia de software**. São Paulo: Makron Books, 1995.
- RAMAKRISHNAN, R; GEHRKE, J. **Database management systems**. 2ª ed. Boston: McGraw-Hill, 2000.
- ROB, P. CORONEL, C. **Sistemas de Banco de Dados – projeto, implementação e administração**. Cengage Learning: 2011.
- SETZER, V. W. **Dado, Informação, Conhecimento e Competência**. Acesso em: set. de 2008, disponível em Dado, Informação, Conhecimento e Competência: <http://www.ime.usp.br/~vwsetzer/dado-info.html>
- SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistema de Banco de Dados**. 5ª ed. Rio de Janeiro: Elsevier, 2006.
- TEOREY, T.; LIGHTSTONE, S.; NADEAU, T.; JAGADISH, H. V.; **Database Modeling and Design: logical design**. 5ª ed., Burlington – USA: Elsevier, 2011.
-



## GABARITO

### Capítulo 1

01. A resposta correta é a B. Dados espalhados em diversos arquivos podem acarretar o que nem todos sejam atualizados quando necessário causando a sua inconsistência. A própria repetição dos dados em diversos é um problema de redundância.

02. Podemos citar como principais vantagens a divisão de tarefas entre o cliente e o servidor. A máquina cliente executa a tarefa do aplicativo fornecendo a interface com o usuário como também o processamento de entrada e saída enquanto a máquina servidor executa as consultas SQL e retorna o resultado para a máquina cliente. Esta divisão de tarefas entre dois sistemas, reduz o tráfego de dados na rede.

03. A modelagem de dados é um processo que envolve várias etapas, o que pode ser entendido como iterativo e progressivo. Inicia-se com uma compreensão simples do problema que desejamos sanar, e conforme o grau de entendimento do problema aumenta, o nível de detalhes que a modelagem compreende também. Aprenderemos como converter o modelo conceitual em modelo lógico nos próximos capítulos.

### Capítulo 2

01. A resposta correta é a A.

02. A resposta correta é a C.

03. A resposta correta é a A

### Capítulo 3

01. A resposta correta é a B.

02. A resposta correta é a D.

### Capítulo 4

01. Podemos citar os seguintes tipos de dados do PostgreSQL: *bigint*, *bigserial*, *char* (tamanho) ou *character* (tamanho), *date*, *decimal*, *double*, *integer* ou *int*, *money*, *numeric*, *real*, *serial*, *smallint*, *time*, *varchar* (tamanho), *blob*, etc.

02. Os tipos de dados para a tabela CLIENTE são os seguintes: CLIENTE = *varchar*(tamanho), RUA = *varchar*(tamanho), NRO = integer ou int, DATANASCIMENTO = *date* e SALARIO = *Money*.

## Capítulo 5

01. A resposta correta é a B.

02. A resposta correta é a A.

03. Update TB\_FUNCIONARIO Set SALARIO = 2000 Where COD\_FUNCIONARIO = 345

---