

Sistema Aberto de Educação



Guia de Estudo

Banco de Dados



Instituição Credenciada pelo MEC
Centro Universitário do Sul de Minas



SABE – Sistema Aberto de Educação

**Av. Cel. José Alves, 256 - Vila Pinto
Varginha - MG - 37010-540
Tele: (35) 3219-5204 - Fax - (35) 3219-5223**

Instituição Credenciada pelo MEC – Portaria 4.385/05

**Centro Universitário do Sul de Minas - UNIS/MG
Unidade de Gestão da Educação a Distância – GEaD**

**Mantida pela
Fundação de Ensino e Pesquisa do Sul de Minas - FEPESMIG**

Varginha/MG

Todos os direitos desta edição estão reservados ao Sistema Aberto de Educação – SABE.

É proibida a duplicação ou reprodução, total ou parcial, deste volume, sob qualquer meio, sem autorização expressa do SABE.

005.75

F676B FONSECA, Letícia Rodrigues.

Guia de Estudo – Banco de Dados.
Letícia Rodrigues da Fonseca. Varginha: GEaD-
UNIS/MG, 2008.

112p.

1. Banco de Dados 1. 2. Modelagem de
Dados 1. 3. Linguagem SQL 1. I. Título.

Revisado e Atualizado em Janeiro de 2009

REITOR

Prof. Ms. Stefano Barra Gazzola

GESTOR

Prof. Ms. Tomás Dias Sant' Ana

Supervisor Técnico

Prof. Ms. Wanderson Gomes de Souza

Coord. do Núcleo de Recursos Tecnológicos

Prof^a. Simone de Paula Teodoro Moreira

Coord. do Núcleo de Desenvolvimento Pedagógico

Prof^a. Vera Lúcia Oliveira Pereira

Revisão ortográfica / gramatical

Prof^a. Maria José Dias Lopes Grandchamp

Design/diagramação

Prof. César dos Santos Pereira

Equipe de Tecnologia Educacional

Prof^a. Débora Cristina Francisco Barbosa

Jacqueline Aparecida da Silva












Prof. Lázaro Eduardo da Silva

Autora

LETÍCIA RODRIGUES DA FONSECA – lety-fonseca@hotmail.com

Graduada em Ciência da Computação e Administração. Pós-Graduada em Gestão de Tecnologias da Informação e Mestre em Administração. Atua como docente na FACECA, UNIPAC-Lambari e UNIS, ministrando aulas em cursos de graduação e pós-graduação. Atua, também, como coordenadora do Núcleo de Pós-Graduação Lato Sensu da FACECA.

TABELA DE ÍCONES

	REALIZE. Determina a existência de atividade a ser realizada. Este ícone indica que há um exercício, uma tarefa ou uma prática para ser realizada. Fique atento a ele.
	PESQUISE. Indica a exigência de pesquisa a ser realizada na busca por mais informação.
	PENSE. Indica que você deve refletir sobre o assunto abordado para responder a um questionamento.
	CONCLUSÃO. Todas as conclusões, sejam de idéias, partes ou unidades do curso virão precedidas desse ícone.
	IMPORTANTE. Apona uma observação significativa. Pode ser encarado como um sinal de alerta que o orienta para prestar atenção à informação indicada.
	HIPERLINK. Indica um link (ligação), seja ele para outra página do módulo impresso ou endereço de Internet.
	EXEMPLO. Esse ícone será usado sempre que houver necessidade de exemplificar um caso, uma situação ou conceito que está sendo descrito ou estudado.
	SUGESTÃO DE LEITURA. Indica textos de referência utilizados no curso e também faz sugestões para leitura complementar.
	APLICAÇÃO PROFISSIONAL. Indica uma aplicação prática de uso profissional ligada ao que está sendo estudado.
	CHECKLIST ou PROCEDIMENTO. Indica um conjunto de ações para fins de verificação de uma rotina ou um procedimento (passo a passo) para a realização de uma tarefa.
	SAIBA MAIS. Apresenta informações adicionais sobre o tema abordado de forma a possibilitar a obtenção de novas informações ao que já foi referenciado.
	REVENDO. Indica a necessidade de rever conceitos estudados anteriormente.

SUMÁRIO

APRESENTAÇÃO.....	6
1. INTRODUÇÃO	8
1.1. Histórico	11
2. MODELAGEM DE DADOS: MODELOS CONCEITUAIS, MODELO E-R E SUAS VARIACÕES	15
2.2. Modelo E-R e suas Variações.....	17
2.2.1. Entidades.....	18
2.2.2. Atributos	19
2.2.2.1. Atributos Valorado e Multi Valorado	19
2.2.3. Atributo Chave.....	20
2.2.4. Chaves Candidatas e Chaves Estrangeiras	20
2.2.5. Entidades Fracas e Fortes.....	21
2.2.6. Relacionamentos.....	22
2.2.6.1. Mapeamento das Cardinalidades	22
2.2.6.2. Grau dos Relacionamentos	25
3. O MODELO RELACIONAL: NORMALIZAÇÃO E MANUTENÇÃO DA INTEGRIDADE.....	34
3.1. Domínios	35
3.2. Relações	35
3.3. Atributo Chave de uma Relação.....	36
3.4. Restrições de Integridade	37
3.5. Mapeamento do Modelo E-R para o Modelo Relacional	38
3.6. Normalização	44
Primeira Forma Normal	45
Segunda Forma Normal	46
Terceira Forma Normal.....	47
4. LINGUAGENS: ÁLGEBRA E CÁLCULO RELACIONAL.....	50
4.1. Álgebra Relacional	50
4.2. Cálculo Relacional.....	57
4.3. Linguagem SQL	63
5. ARQUITETURA DE SISTEMAS DE BANCO DE DADOS.....	84
5.1. Modelos de Dados	84
5.2. Esquemas e Instâncias	84
5.3. A Arquitetura Três Esquemas	85
5.4. Independência de Dados	86
5.5. As Linguagens para Manipulação de Dados	87
5.6. Os Módulos Componentes de um SGBD.....	88
5.7. Classificação dos SGBDs	89
5.8. Profissionais e Atividades Envolvidas em um SGBD	89

6. SEGURANÇA E MECANISMO DE PROTEÇÃO	92
6.1. Segurança e Violação de Integridade	92
6.2. Autorizações	94
6.3. Especificação de segurança em SQL	95
7. RECUPERAÇÃO.....	96
7.1. Falhas em Bancos de Dados	96
7.2. Recuperação e Atomicidade	97
8. NOÇÕES DE BANCOS DE DADOS DISTRIBUÍDOS	98
8.1 Definições	98
8.2 Visão geral	99
8.3. Funcionamento.....	100
8.4. Armazenamento distribuído	101
8.5. Arquitetura.....	102
8.6. Autonomia local.....	103
8.7. Independência de localização	103
8.8. Independência de fragmentação.....	103
8.9. Independência de replicação	105
8.10. Gerenciamento.....	106
8.11. Robustez	109
REFERÊNCIAS	111

APRESENTAÇÃO

Prezado(a) aluno(a):

É com muito prazer que irei trabalhar com você, neste semestre, a disciplina de Banco de Dados. Este é o seu Guia de Estudos, que visa auxiliá-lo, em seu aprendizado, por meio do oferecimento de uma noção geral sobre a construção de sistemas de banco de dados.

Para isso, dentre os diversos temas que serão abordados, iremos estudar a importância dos bancos de dados, conhecer os modelos para a construção de projetos lógicos e físicos, métodos de consultas, etc.

Sei que sou suspeita, mas, como profissional formada na área, posso afirmar que a disciplina de Banco de Dados, trata-se de uma das mais prazerosas de se trabalhar.

Os temas foram abordados de forma simples e clara. No entanto, não se esqueça: *“somos parceiros nesta nova caminhada”*.

Letícia Rodrigues da Fonseca

Professora de Banco de Dados – Bacharelado em Sistemas de Informação

1. INTRODUÇÃO

A tecnologia aplicada aos métodos de armazenamento de informações vem se aprimorando constantemente. Esse cenário é devido à necessidade, apresentada por diversas organizações, de armazenar grandiosas quantidades de informação de uma maneira rápida, simples e confiável e que, por sua vez, se possa acessá-las a qualquer momento, em qualquer lugar, sem se deslocar às salas dedicadas a arquivar documentos, como antigamente se fazia.

Devido a essa necessidade, surgiram os “Bancos de Dados”. Um **“Banco de Dados”** pode ser definido como um **conjunto de “dados” devidamente relacionados**.

Segundo O'brien (2004), **“Dados”** são **fatos ou observações crus**, normalmente sobre fenômenos físicos ou transações de negócios. Já, **“Informação”**, são **dados que foram convertidos em um contexto significativo e útil para usuários finais**. Portanto, podemos considerar a informação como dados processados e colocados em uma situação que lhes confere valor para usuários finais específicos.

Quando as organizações começaram a adquirir programas informatizados, começaram também a armazenar dados nos arquivos desses programas, o que era cômodo, porém, ainda assim existiam dificuldades no momento de se modificar registros, estruturas ou simplesmente buscar informações.



De acordo com Korth e Silberschatz (1999), antes do aparecimento dos “Bancos de Dados”, as organizações utilizavam esses programas para armazenar informações, que, por sua vez, apresentavam numerosas desvantagens:

- **Inconsistência e Redundância de Dados:** já que arquivos e aplicações são criados e mantidos por diferentes programadores, em geral, durante longos períodos

de tempo, é comum que os arquivos possuam formatos diferentes e os programas sejam escritos em diversas linguagens de programação. Além disso, a mesma informação poderá ser repetida em diversos lugares (arquivos). Por exemplo, em uma instituição bancária, o endereço e telefone de um cliente em particular poderá aparecer tanto no arquivo “Dados_Cliente” quanto no arquivo “Contas_Cliente”. Essa redundância aumenta os custos de armazenamento e acesso. Pode, ainda, originar inconsistências de dados, isto é, as várias cópias dos dados podem divergir ao longo do tempo. Por exemplo, a mudança de endereço de um cliente pode ser informada no arquivo “Contas_Cliente”, mas não ser informada no sistema como um todo.

- **Dificuldade de Acesso aos Dados:** suponha que um dos empregados da empresa necessite de uma relação com os nomes de todos os clientes que moram em determinada área da cidade cujo CEP é 37100000. O empregado solicita, então, ao departamento de processamento de dados que crie tal consulta. Esse tipo de consulta não foi prevista no projeto do sistema. No entanto, há uma consulta para gerar a relação de todos os clientes da empresa. Assim, o empregado tem duas alternativas: separar manualmente da lista de todos os clientes aqueles de que necessita ou requisitar ao departamento de processamento de dados um programador para criar a consulta necessária. Ambas as alternativas são, obviamente, insatisfatórias.
- **Isolamento de Dados:** como os dados estão dispersos em vários arquivos, e esses arquivos podem apresentar diferentes formatos, é difícil desenvolver novos programas para recuperação apropriada dos dados.
- **Problemas de Integridade:** os valores dos dados atribuídos e armazenados em um banco de dados devem satisfazer certas restrições para manutenção da consistência. Por exemplo, o balanço de uma conta bancária não pode ser inferior a um determinado valor (digamos, 50 reais). Os programadores determinam o cumprimento dessas restrições por meio da inclusão de código apropriado aos vários programas de aplicações. Entretanto, quando aparecem novas restrições

torna-se complicado alterar todos os programas para incrementá-las. O problema é ampliado quando as restrições atingem diversos itens de dados em diferentes arquivos.

- **Problemas de Atomicidade:** um sistema computacional, como qualquer outro dispositivo mecânico ou elétrico, está sujeito a falhas. Em muitas aplicações é crucial assegurar que, uma vez detectada uma falha, os dados sejam salvos em seu último estado consistente, anterior a ela. Considere um programa para transferir 50 reais da conta A para uma conta B. Se ocorrer falha no sistema durante sua execução, é possível que os 50 reais sejam debitados da conta A sem serem creditados na conta B, criando um estado inconsistente no banco de dados. Logicamente, é essencial para a consistência do banco de dados que ambos, débito e crédito, ocorram ou nenhum deles seja efetuado. Isto é, a transferência de fundos deve ser uma operação atômica, deve ocorrer por completo ou não ocorrer.

- **Anomalias no Acesso Concorrente:** muitos sistemas permitem atualizações simultâneas nos dados para aumento do desempenho do sistema como um todo e para melhores tempos de resposta. Nesses tipos de ambiente, a interação entre atualizações concorrentes pode resultar em inconsistência de dados. Suponha que o saldo de uma conta bancária A seja de 500 reais. Se dois clientes retiram uma determinada quantia da conta A (digamos 50 e 100 reais, respectivamente), essas operações, ocorrendo simultaneamente, podem resultar em erro (ou gerar inconsistência). Suponha que, na execução dos programas, ambos os clientes leiam o saldo antigo e retirem, cada um, seu valor correspondente, sendo o resultado armazenado. Os dois programas concorrendo, ambos lêem o valor 500 reais, resultando em 450 e 400 reais, respectivamente. Dependendo de qual deles registre seu resultado primeiro, o saldo da conta A será 450 ou 400 reais, em vez do valor correto de 350 reais. Para resguardar-se dessa possibilidade, o sistema deve manter algum tipo de supervisão. Como os dados podem sofrer acesso de diferentes programas, os quais não foram coordenados previamente, a supervisão é bastante dificultada.

- **Problemas de segurança:** não são todos os usuários de banco de dados que estão autorizados a acessar a todos os dados. Por exemplo, em um sistema bancário, os funcionários do departamento pessoal devem ter acesso apenas ao conjunto de dados de pessoas que trabalham no banco. Eles não deverão ter acesso às informações dos clientes do banco. Uma vez que os programas de aplicação são inseridos no sistema como um todo, é difícil garantir a efetividade das regras de segurança.

Essas dificuldades, entre outras, promoveram o desenvolvimento dos bancos de dados ao final dos anos sessenta.

1.1. Histórico

Igualmente a muitas tecnologias da computação industrial, os fundamentos de bancos de dados relacionais surgiram na empresa IBM, nas décadas de 1960 e 1970, durante um período da história em que as organizações identificaram um custo alto ao empregar um grande número de pessoas para fazer trabalhos como armazenar e organizar arquivos. Por esse motivo, concentraram-se esforços e investimentos em pesquisas que visavam identificar uma solução mecânica, com um custo menor, para a execução dessas tarefas.



Diversas pesquisas foram conduzidas durante esse período, cujos **modelos hierárquicos, de rede, relacionais entre outros** foram descobertos.

a) Banco de Dados Relacional: um banco de dados relacional consiste em uma coleção de tabelas, que podem ser relacionadas através de seus atributos, ou seja, uma linha de uma tabela pode estar sendo relacionada com outra linha em uma outra tabela.

b) Banco de Dados Rede: enquanto no modelo relacional os dados e os relacionamentos entre dados são representados por uma coleção de tabelas, o modelo de rede representa os dados por coleções de registros e os relacionamentos

entre dados são representados por ligações. Ou seja, um banco de dados de rede consiste em uma coleção de registros que são conectados uns aos outros por meio de ligações. Cada registro é uma coleção de campos (atributos), cada um desses campos contendo apenas um valor de dado. Uma ligação é uma associação entre precisamente dois registros.

E

Cliente			Conta	
Nome	Rua	Cidade	Número	Saldo
Paulo	Oliveira	Campinas	100-01	100,00

c) Banco de Dados Hierárquico: assim como no modelo de Redes o modelo Hierárquico trabalha com os dados e relacionamentos como uma coleção de registros relacionados por ligações. A única diferença entre os dois é que o modelo Hierárquico os registros são organizados como coleções de árvores em vez de grafos arbitrários.

E

Cliente		
Nome	Rua	Cidade
Paulo	Oliveira	Campinas

Conta	
Número	Saldo
100-01	100,00

Em 1970 um pesquisador da IBM - Ted Codd - publicou o primeiro artigo sobre bancos de dados relacionais. Esse artigo tratava sobre o uso de cálculo e álgebra

relacional para permitir que usuários não técnicos armazenassem e recuperassem grande quantidade de informações.

Codd visionava um sistema no qual o usuário seria capaz de acessar as informações por meio de comandos em inglês e essas informações estariam armazenadas em tabelas.

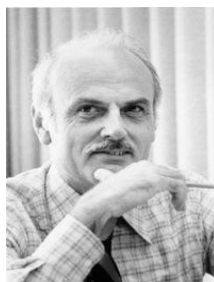


Figura 1 - Edgar Frank Codd - O Criador do Modelo Relacional.



Saiba mais sobre Codd em:

www.informatik.uni-trier.de/%7Eley/db/about/codd.html

Devido à natureza técnica do artigo e a relativa complicação matemática, a teoria de Codd não foi colocada em prática. Entretanto esse artigo levou a IBM a montar um grupo de pesquisa conhecido como *System R* (Sistema R).

O projeto do **Sistema R** tinha o **objetivo de criar um sistema de banco de dados relacional o qual, eventualmente, se tornaria um produto**. Os primeiros protótipos foram utilizados por muitas organizações, tais como MIT Sloan School of Management (uma escola renomada de negócios norte-americana). Novas versões foram testadas ainda em empresas de aviação para rastreamento do manufaturamento de estoque.

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. Essa linguagem tornou-se um padrão na indústria de bancos de dados relacionais e, hoje em dia, é um padrão ISO

(*International Organization for Standardization*). A ISO é a Organização Internacional de Padronização.

Mesmo a IBM sendo a companhia que inventou o conceito original e o padrão SQL, ela não produziu o primeiro sistema comercial de banco de dados. O feito foi realizado pela Honeywell Information Systems Inc., cujo sistema foi lançado em junho de 1976. Ele era baseado em muitos princípios do sistema que a IBM concebeu, mas foi modelado e implementado fora dela.

Os primeiros sistemas de banco de dados construídos baseados nos padrões SQL começaram a aparecer no início dos anos 80 com a empresa Oracle por meio do Oracle 2, e depois com a IBM por meio do SQL/DS.

O padrão SQL, criado pela IBM, hoje pertence à ANSI (*American National Standards Institute*) e à ISO, que formaram um grupo de pesquisa para aprimorá-lo.



Questões para Síntese:

- 1) Onde e quando você acredita que irá utilizar os conteúdos aprendidos nesta disciplina?
- 2) Defina dados e informação.
- 3) Quais são os principais diferenciais que você identifica entre o Processamento Tradicional de Arquivos e a Abordagem de Banco de Dados?

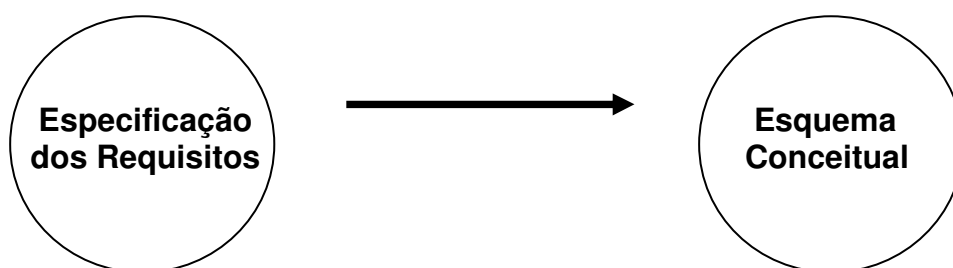
2. MODELAGEM DE DADOS: MODELOS CONCEITUAIS, MODELO E-R E SUAS VARIAÇÕES

2.1. Fases do Projeto de Base de Dados

O Projeto de Base de Dados pode ser decomposto em:

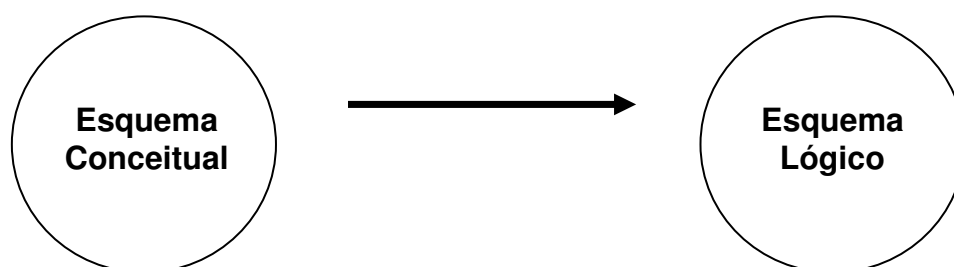
PROJETO CONCEITUAL

Modelo Conceitual: Linguagem usada para descrever esquemas conceituais. Independe do SGBD escolhido



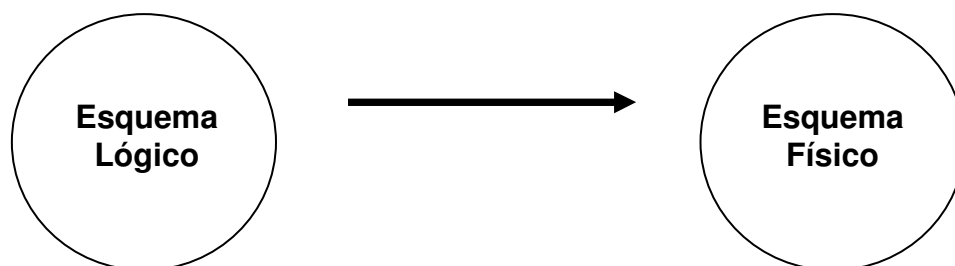
PROJETO LÓGICO

Modelo lógico: Linguagem usada para especificar esquemas lógicos. Pertencem a três classes: Relacional, Redes e Hierárquico.



PROJETO FÍSICO

Esquema físico: É a descrição da implementação da base de dados. Descreve estruturas de armazenamento e métodos de acesso. Tem forte ligação com o SGBD específico.



Conclusão

- **Projeto Conceitual:** Não possui dependência com a classe do SGBD (ferramenta a ser utilizada)



Exemplos de Ferramentas: Mysql, Firebird, Oracle, etc.

- **Projeto Lógico:** Tem dependência com a classe, mas não com o SGBD específico.
- **Projeto Físico:** Total dependência com SGBD específico.

Uma das vantagens em se trabalhar com o projeto conceitual está na possibilidade de se adiar a escolha do SGBD (mesmo a sua classe). O projetista deve concentrar o maior esforço nesta fase do projeto pois, a passagem para as outras fases é mais ou menos automática. Outra vantagem está na possibilidade de usuários não especialistas em bancos de dados darem diretamente a sua contribuição no projeto conceitual cuja maior exigência é a capacidade de abstração. A aproximação com o usuário final melhora bastante a qualidade do projeto.



Abstração: processo que consiste em mostrar as características e propriedades essenciais de um conjunto de objetos, ou esconder as características não essenciais.

Quando pensamos no objeto “bicicleta” de uma forma abstrata, normalmente “esquecemos” seus detalhes e as particularidades que as diferem entre si.

O Projeto Conceitual produz um esquema conceitual a partir de “requisitos” de um mundo real. Os Requisitos tratam-se do **esboço das necessidades do “cliente” (processo de coleta de requisitos)**.



Devem-se definir quais serão as informações de entrada, o processamento a elas atribuído e quais informações serão de saída, ou seja, que informações o usuário necessita ao final do processamento.



Figura 2 – Processo de Elaboração do Modelo de Dados Conceitual

2.2. Modelo E-R e suas Variações

Para Korth e Silberschatz (1999), o “**Modelo Entidade-Relacionamento (E-R)**” criado em 1976 por Peter Chen, é baseado na percepção do mundo real que consiste em **um conjunto de objetos básicos chamados entidades e nos relacionamentos entre esses objetos**. Ele foi desenvolvido para facilitar o projeto de banco de dados permitindo a especificação de um **esquema de empresa**. Tal esquema **representa a estrutura lógica geral do banco de dados**.



Figura 3 - Peter Chen - Criador do Modelo ER



Saiba mais sobre Chen em:

bit.csc.lsu.edu/~chen/chen.html

Suas principais características:

- Modelo simples (poucos conceitos)
- Representação gráfica de fácil compreensão (Diagrama E-R)

2.2.1. Entidades

O objeto básico tratado pelo modelo E-R é a “**Entidade**”, que pode ser definida como um **objeto do mundo real, concreto ou abstrato, que possui existência independente**.



Uma tabela na qual se cadastra “clientes” é uma entidade. Um cadastro de “produtos” é outra entidade

ENTIDADE



Tabela - Cadastro de Clientes

CLIENTES		
Código	Nome	Endereço



Pode-se dizer que **entidade é o mesmo que tabela**. **Entidades de um mesmo tipo são agrupadas em Classes de Entidade**. Assim, a classe de entidades **CLIENTES** é o conjunto de todas as instâncias de clientes. Cada ocorrência de um cliente dentro da classe **Clientes** (registros) é denominada **Instância de Entidade**.

2.2.2. Atributos

Cada entidade possui um **conjunto particular de propriedades** que a descreve chamado “**Atributos**” ou “**Campos**”.



Código é um atributo, nome é outro atributo, endereço também é outro atributo.

Tabela - Cadastro de Clientes

CLIENTE		
Código	Nome	Endereço

Diagram illustrating attributes (ATRIBUTO) pointing to the fields in the table: Código, Nome, and Endereço.

2.2.2.1. Atributos Valorado e Multi Valorado

- **Valorado:** possui apenas um determinado valor em uma determinada instância.



Data de Nascimento (cada pessoa possui apenas uma data de nascimento).

- **Multi Valorado:** possui diversos valores em uma mesma instância.



Um cliente que possui diversos números de telefone.

Conclusão**Qual é a estrutura básica de um Banco de Dados?**

- Um banco de dados é formado por um conjunto de entidades ou tabelas.
- Uma tabela ou entidade é formada por um conjunto de registros (de clientes, de alunos, etc).
- Um registro é formado por um conjunto de atributos ou campos que descrevem as entidades (código, nome, endereço, etc.).

2.2.3. Atributo Chave

Também conhecido como “**Chave Primária**”, é o campo ou o conjunto de campos que identificam um registro.



Cod_Cliente, Cod_Produto, CPF, RG.

- **Chave Primária Simples:** formada somente por um campo.



Campo “Código” da tabela Produtos.

- **Chave Primária Composta:** formada por mais de um atributo ou campo.

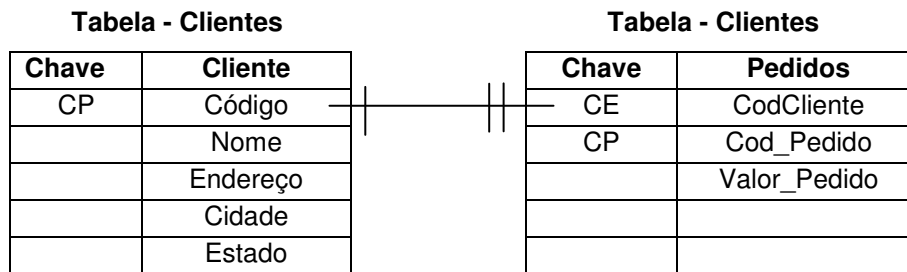


Campo Cod_Cliente + Cod_Pedido da tabela Pedidos.

2.2.4. Chaves Candidatas e Chaves Estrangeiras

- **Chave Candidata:** todo campo de uma entidade pode ser considerado chave candidata à chave primária. Após ser escolhido o campo chave, os demais campos são identificados como “campos normais”.

- **Chave Estrangeira:** é um campo em uma tabela que armazena o conteúdo da chave primária de outra tabela.



Em que: CP = Chave Primária CE = Chave Estrangeira

2.2.5. Entidades Fracas e Fortes

- **Entidades Fracas:** não existem sem a dependência de outra, ou seja, seus registros não existem por si só.



A entidade **DEPENDENTE** não poderá existir sem estar associada à entidade **EMPREGADO**.

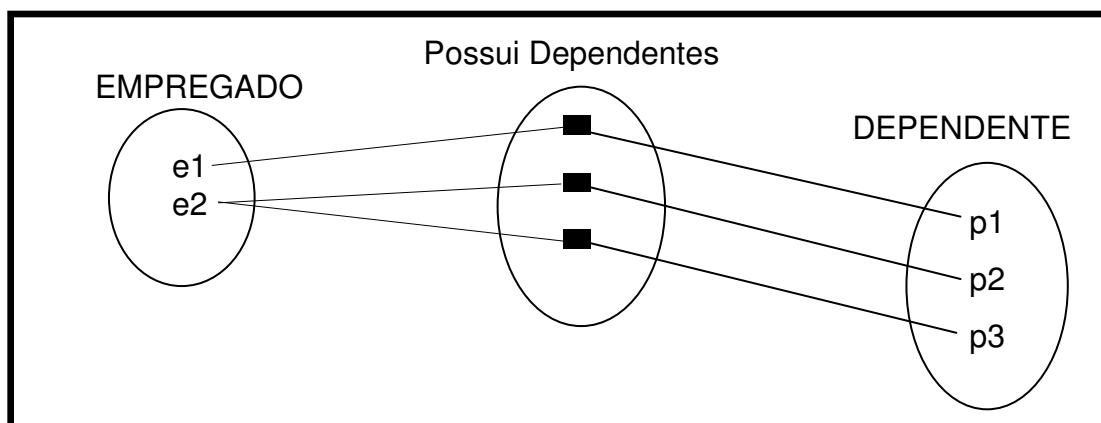


Figura 4 – Representação Entidades Fracas e Fortes

- **Entidades Fortes:** os registros das entidades fortes podem existir independentemente da existência de registros em outras entidades.



No relacionamento anterior, a entidade **EMPREGADO** é a entidade forte.

2.2.6. Relacionamentos

Um “**Relacionamento**”, como o próprio nome diz, é a relação que ocorre entre registros de duas ou mais entidades (tabelas).



Os relacionamentos existem para que as entidades possam compartilhar as informações, de forma que não haja repetição de informação sem necessidade (REDUNDÂNCIA).

2.2.6.1. Mapeamento das Cardinalidades

O “**Mapeamento das Cardinalidades**” expressa o **número de entidades às quais outra entidade pode estar associada via relacionamento**.



O Mapeamento das Cardinalidades deve seguir uma das seguintes instruções:

- **Um para Um:** um registro da entidade A está associado a somente um registro da entidade B.

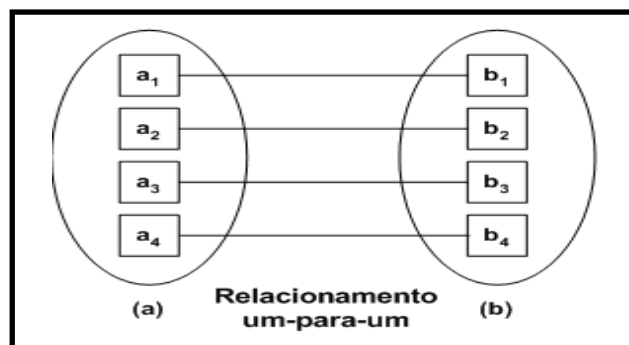
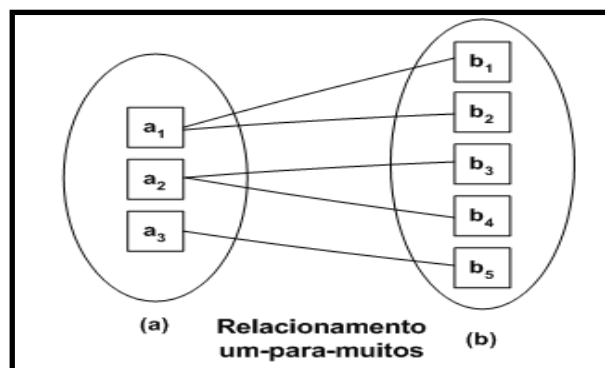


Figura 5 – Representação Relacionamento Um-Para-Um

E

Cada pessoa (A) possui apenas um número de CPF, e cada número de CPF(B) deve pertencer somente a uma pessoa.

- **Muitos para Um ou Um para Muitos:** um registro da entidade A está associado a \underline{n} registros da entidade B ou um registro da entidade B está associado a \underline{n} registros da entidade A.



**Figura 6 – Representação Relacionamento Um-Para-Muitos
ou Muitos para um**

E

Em uma faculdade, na qual os cursos funcionam no período noturno, cada Curso(A) possuirá vários alunos matriculados, e cada Aluno(B) poderá se matricular em apenas um Curso.

- **Muitos para Muitos:** vários registros da entidade A podem estar associados a vários registros da entidade B, e vários registros da entidade B podem estar associados a vários registros da entidade A.

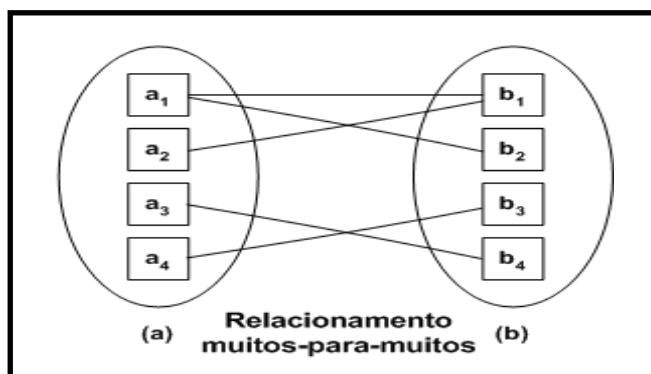


Figura 7 – Representação Relacionamento Muitos para Muitos



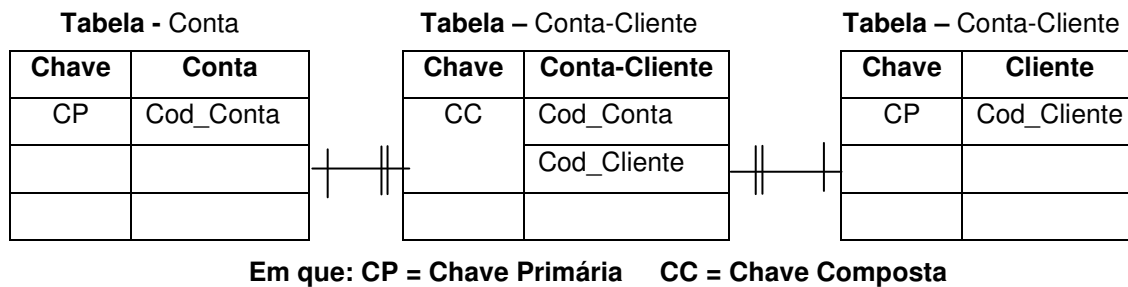
Em um banco, um cliente pode possuir diversas contas e uma conta pode pertencer a diversos clientes (conta conjunta).



Sempre que tivermos este tipo de relacionamento, teremos três entidades relacionadas, ou seja, utilizaremos uma terceira entidade (tabela) que relacionará as duas principais. No linguajar de banco de dados, costuma-se afirmar que este tipo de relacionamento “**Não Existe**”. A terceira entidade (do meio) se chamará “**Entidade associativa**” ou “**Relacionamento com Atributos**”.



A solução de um relacionamento muitos para muitos consiste na criação de uma terceira tabela, auxiliar, para interligar as duas partes que terá como chave primária composta as chaves primárias das outras duas tabelas. Assim, no caso do relacionamento Contas x Clientes, foi criada a tabela Conta_Cliente, a qual armazenará todas as combinações de Contas e Clientes. Sua chave composta é: (Cod_Conta e Cod_Cliente).



Representação dos Tipos de Cardinalidade:

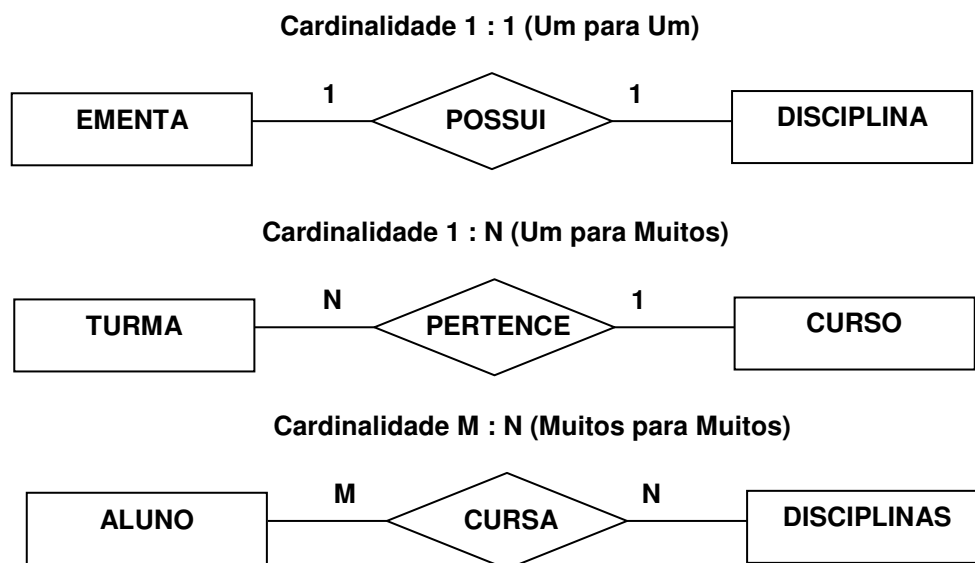


Figura 8 – Tipos de Cardinalidade

2.6.2.2. Grau dos Relacionamentos

Um Relacionamento pode envolver duas ou mais Entidades. O **“Grau do Relacionamento”** é o **número de Entidades envolvidas**. Desta forma podem-se categorizar os tipos de relacionamento em:

- **Relacionamentos Binários:** envolvem duas entidades.

E

As entidades “Cliente” e “Produtos” estão associadas por meio de um relacionamento binário.

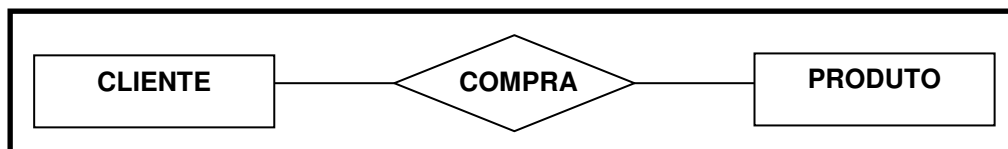


Figura 9– Representação de um Relacionamento Binário

- **Relacionamentos Ternários:** envolvem mais de duas entidades.

E

As entidades “Professor”, “Aluno” e “Disciplina” estão associadas por meio de um relacionamento ternário.

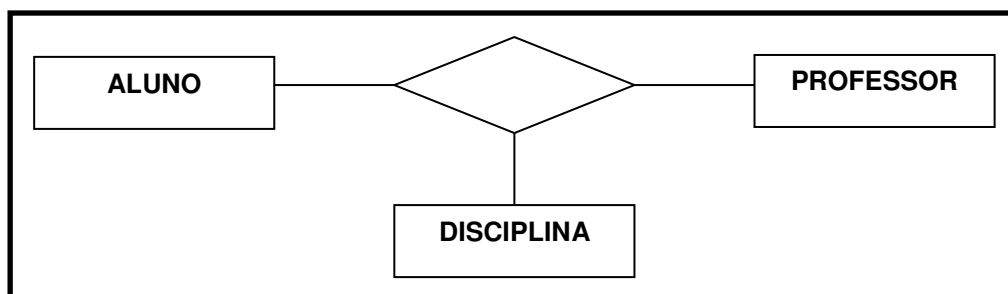


Figura 10 – Representação de um Relacionamento Ternário



Uma boa dica para se determinar a cardinalidade dos relacionamentos é observar de onde eles partem e de onde eles chegam, sob o ponto de vista das chaves das tabelas. Observe a tabela a seguir que irá nos ajudar a “errar menos” na hora de organizarmos a cardinalidade dos relacionamentos. É importante salientar

que esta tabela não é um padrão, mas algo que normalmente ocorre nos relacionamentos. Sendo assim, podem acontecer exceções.

Tabela – Cardinalidade dos Relacionamentos

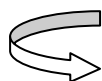
Origem	Destina	Tipo de Relacionamento
CPS	CPC	1-n
CPS	CNC	1-n
CPS	CPS	1-1
CPC	CPC	n-n (Indireto)
CPC	CNC	n-n (Indireto)
CPC	CPS	n-1
CNC	CPS	n-1

LEGENDA:

CPS – Chave Primária Simples

CPC – Chave Primária Composta

CNC – Campo não chave



Relembrando: as chaves primárias simples são chaves formadas somente por um campo e normalmente estão nas entidades fortes. As chaves primárias compostas são formadas por mais de um campo e são características das entidades fracas.

2.2.7. Diagrama Entidade – Relacionamento

O “**Diagrama Entidade-Relacionamento**” é composto por um **conjunto de objetos gráficos que visa representar todos os objetos do modelo Entidade Relacionamento** tais como: entidades, atributos, atributos chaves, relacionamentos, restrições estruturais, etc.

! O diagrama ER fornece uma visão lógica do banco de dados, oferecendo um conceito mais generalizado de como estão estruturados os dados de um sistema.

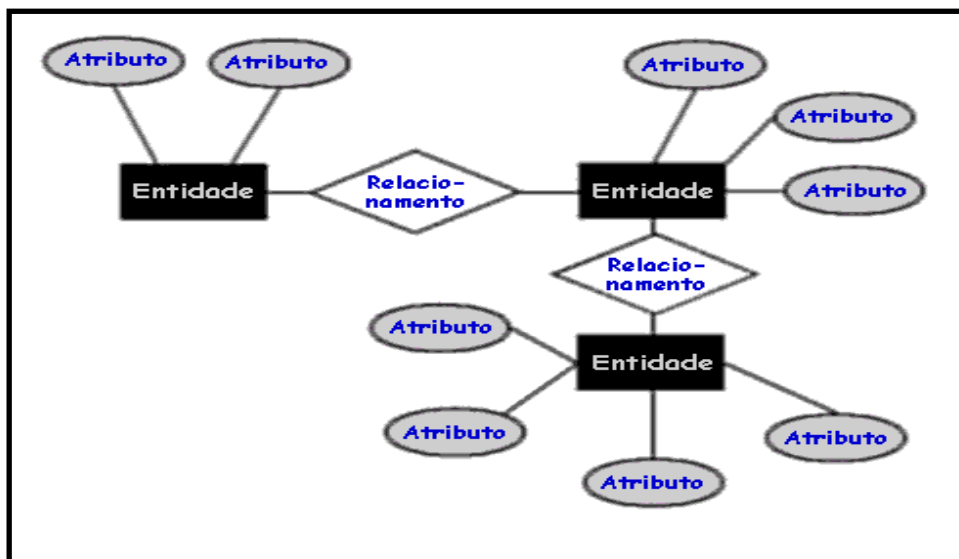


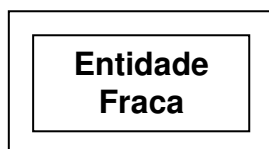
Figura 11 – Esquema de um Diagrama Entidade-Relacionamento

2.2.7.1. Notação e Símbolos do DER

! Os objetos que compõem o diagrama ER estão listados a seguir (exemplo de um dos tipos de metodologia utilizada):



Uma entidade é um conceito ou objeto sobre o qual desejamos armazenar informações.



Uma entidade fraca depende de outra entidade para poder existir.



Atributos são propriedades ou características de uma entidade.



Um atributo chave é único, característica identificadora da entidade. Por exemplo, o CNPJ de uma Empresa pode ser utilizado como atributo chave da Empresa.



Um atributo derivado é baseado em um ou mais atributos. Por exemplo, o imposto recolhido na fonte no pagamento de um funcionário.



Um atributo multivalorado pode ter mais do que um valor. Por exemplo, um motorista pode ser habilitado para dirigir vários tipos de veículos.



Os relacionamentos mostram como duas entidades compartilham informações na estrutura do banco de dados.



Para associar uma entidade fraca com outra, usa-se essa notação.



Notação e Símbolos quanto à Cardinalidade.

Losangos: representam conjuntos de relacionamentos. Segmentos de reta ligam atributos a conjuntos de entidades, e estes a conjuntos de relacionamentos. Cada componente é rotulado com seu nome correspondente. Se o atributo for chave primária, esta condição é demonstrada sublinhando seu nome. As retas que ligam conjuntos de entidades e de relacionamentos são rotulados segundo sua cardinalidade:



para cardinalidade um-para-um



para cardinalidade um-para-muitos

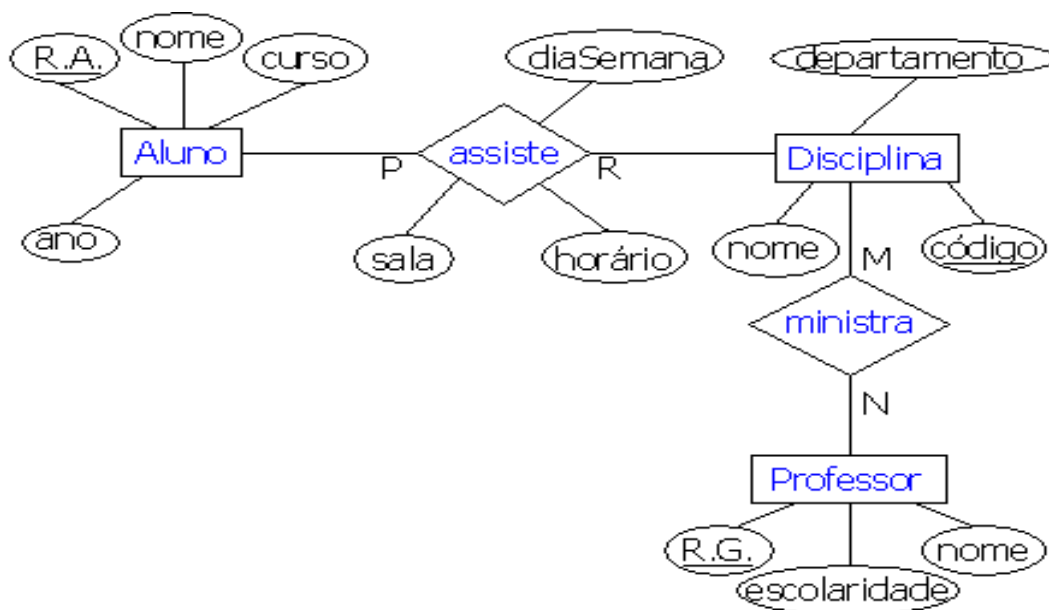


para cardinalidade muitos-para-muitos

A respeito de como nomear os vários componentes, como prática geral adota-se o emprego de substantivos para os conjuntos de entidades e verbos para os conjuntos de relacionamentos, de modo que o diagrama forneça uma descrição narrativa das requisições da base de dados.

E

O exemplo abaixo apresenta o diagrama entidade-relacionamento proposto para uma base de dados de uma universidade:



2.2.7.2. MER Estendido

Segundo Korth e Silberschatz (1999), com o passar do tempo, percebeu-se que o MER original, criado por Peter Chen, não modelava alguns tipos de problemas. Surgiu então, uma extensão do MER denominada MER Estendido ou MER-RX:

- **Generalização e Especialização:** algumas entidades contêm conjuntos de atributos específicos. Quando ocorre a situação em que uma entidade possui atributos que não fazem parte de todas as instâncias (registros) da entidade ou quando estas instâncias se relacionam de maneira diferente com outras entidades, temos aí o conceito de **Generalização/Especialização**.



Considere a entidade CLIENTE com os atributos NumeroCliente, NomeCliente e QuantiaDevida.

Suponha que um CLIENTE pode ser uma única pessoa individual, uma associação ou uma corporação e que serão armazenados dados adicionais dependendo do tipo.

Suponha ainda que estes dados adicionais são:

- **CLIENTE-INDIVIDUAL:** Endereco, NumeroPrevidenciaSocial
- **CLIENTE-ASSOCIACAO:** NomeAssociado, Endereco, Taxa
- **CLIENTE-CORPORACAO:** PessoaContato, Fone, NumeroIdentificacao

Para modelar esta situação, temos duas alternativas:

1. Alocar todos esses atributos na entidade CLIENTE. Nesse caso, alguns dos atributos (campos) não são aplicáveis a todas as entidades (ou seja, não serão preenchidos).
2. Definir três entidades para cada um dos tipos. As quais seriam: CLIENTE-INDIVIDUAL, CLIENTE-ASSOCIACAO e CLIENTE-CORPORACAO.

Uma vez que a entidade CLIENTE é uma entidade genérica, ela é denominada de “**Generalização**” (ou entidade de nível superior) e as entidades INDIVIDUAL, ASSOCIADO e CORPORAÇÃO são denominadas “**Especialização**” (ou entidade de nível inferior).

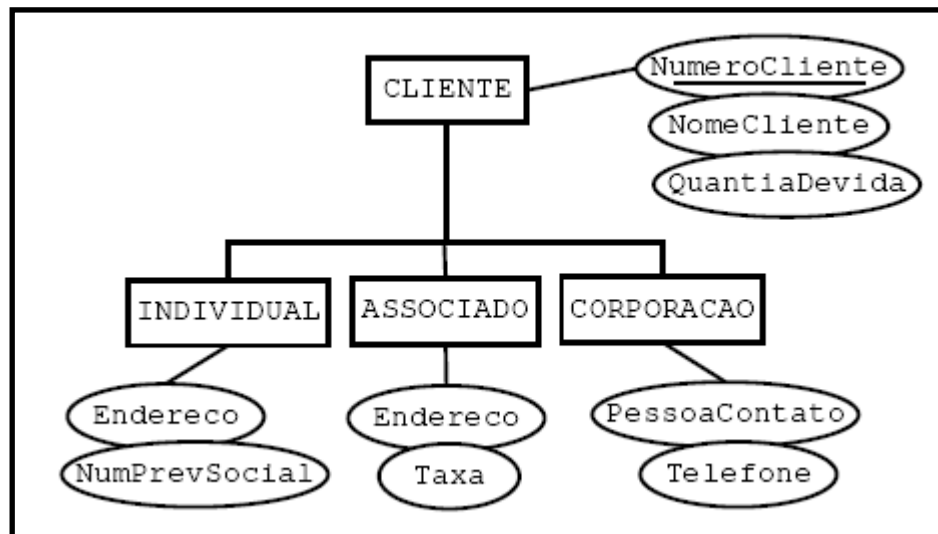


Figura 12 – Exemplo de Situação que Apresenta Generalização

- **Agregação:** uma limitação do modelo E-R é que ele não permite expressar relacionamentos entre relacionamentos.

E

Considere um banco de dados que contém informações sobre **Funcionários** que trabalham em um determinado **Projeto** e utilizam uma série de **diferentes Máquinas** em seus trabalhos.

A solução é usar a agregação. A “**Agregação**” é uma **abstração por meio da qual relacionamentos são tratados como entidades de nível superior**. Assim, para nosso exemplo, o relacionamento *Trabalho* e as entidades **FUNCIONÁRIO** e **PROJETO**, tornar-se-ão uma única entidade. Isso permite relacionar o conjunto **FUNCIONARIO**, **TRABALHO** e **PROJETO** com a entidade **MÁQUINA**.

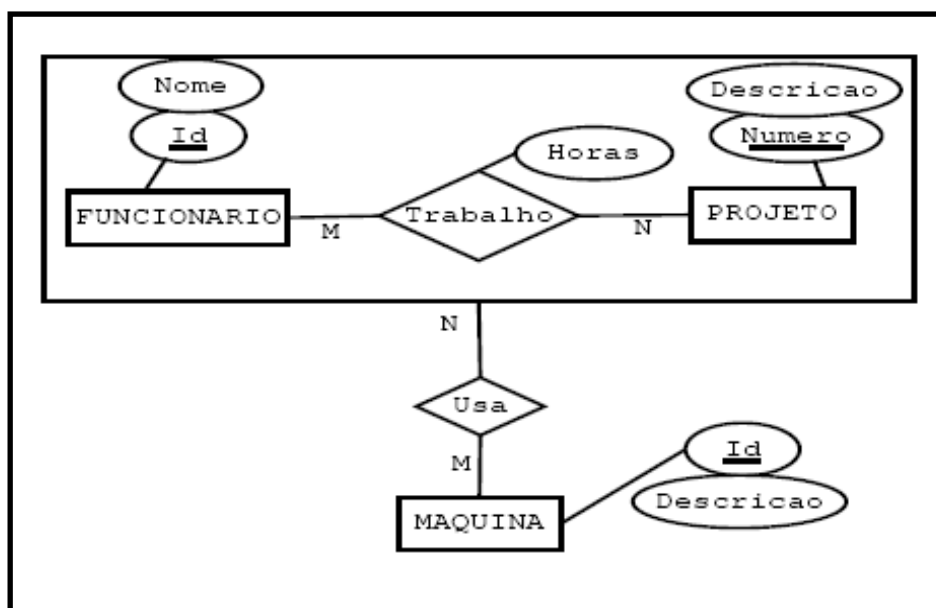


Figura 13 – Exemplo de Situação que Apresenta Agregação



Questões para a Síntese:

- 1) Se você ainda não decidiu qual SGBD utilizar, pode, mesmo assim, iniciar o projeto do modelo lógico (MER)? Por quê?
- 2) Diferencie entidade, registro e campo. Exemplifique.
- 3) Por que as chaves primárias são importantes?
- 4) Quais os tipos de relacionamentos sob o ponto de vista de sua cardinalidade? Exemplifique.
- 5) Qual procedimento deve ser adotado quando nos depararmos com um tipo de relacionamento muitos para muitos? Exemplifique.
- 6) Diferencie generalização e agregação.

3. O MODELO RELACIONAL: NORMALIZAÇÃO E MANUTENÇÃO DA INTEGRIDADE.

Criado por Edgar Codd, nos anos 70, começou a ser realmente utilizado nas empresas a partir de 1987, por meio do SGBDs. De acordo com Heuser (2001), a **abordagem relacional** está **baseada no princípio de que as informações, em uma base de dados, podem ser consideradas como relações matemáticas e representadas de maneira uniforme, por meio do uso de tabelas.**

Esse princípio coloca os **dados (entidades e relacionamentos)** dirigidos para **estruturas mais simples de armazenar, que são as tabelas.**

O conceito principal vem da teoria dos conjuntos (álgebra relacional) atrelado à idéia de que não é relevante ao usuário saber onde os dados estão e nem como os dados estão (princípio da transparência). Os usuários manipulam objetos dispostos em linhas e colunas das tabelas. O usuário pode lidar com esses objetos, além de contar com um conjunto de operadores e funções de alto nível que fazem parte da álgebra relacional.



Terminologia do modelo relacional:

- **Tabela** é chamada de **RELAÇÃO**;
- **Linha** de uma tabela é chamada de **TUPLA**;
- **Coluna** é chamada de **ATRIBUTO**;
- O tipo de dado que descreve cada coluna é chamado de **DOMÍNIO**.

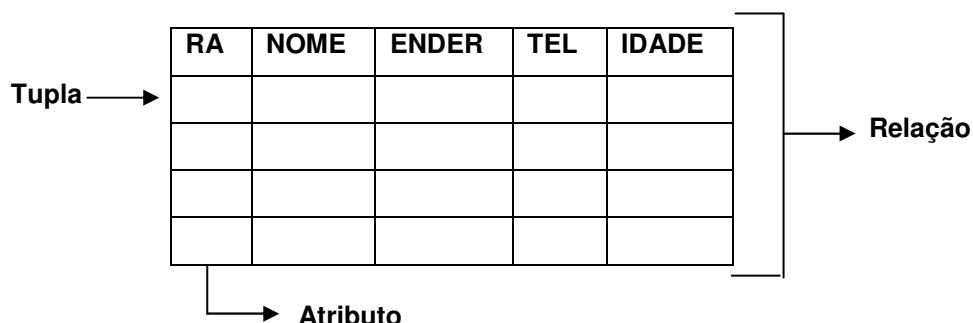


Figura 19 - Representação de uma tabela no Modelo Relacional



Uma relação é representada da seguinte maneira:

$Aluno = \{Ra, Nome, Ender, Tel, Idade\}$

3.1. Domínios

O **Domínio** de um atributo é, em geral, um tipo de dado que especifica o que o atributo pode receber.



- **Número de salas de aula:** conjunto dos números de 1 a 150, inteiros no formato 999.
- **Nomes de alunos:** conjunto de todos os nomes possíveis para pessoas no formato String[60].
- **Códigos de Disciplinas:** conjunto de três letras seguidas de um traço e de três dígitos: AAA-999.



Domínio: é um conjunto de valores **atômicos**.

Valor Atômico significa um valor **indivisível** e **monovalorado**.

3.2. Relações

Um **esquema de relação R**, denotado por $R(A_1, A_2, \dots, A_n)$, em que cada atributo A_i é o nome do papel desempenhado por um domínio **D** no esquema relação **R**, em que **D** é chamado domínio de A_i e é denotado por $dom(A_i)$. O grau de uma relação **R** é o número de atributos presentes em seu esquema de relação.



De acordo com o seguinte esquema de uma Relação de Alunos:

$Aluno = \{Nome, RG, Idade\}$

Uma possível instanciação para esse esquema é a Relação:

$R(Aluno) = \{ \langle José, 12345, 21 \rangle, \langle Pedro, 54321, 18 \rangle, \langle Paulo, 32123, 22 \rangle \}$

Como o esquema de uma Relação **R** é definido como um conjunto, não existe a idéia de *ordem*. Assim, desde que se indique que cada valor **Vi** corresponde a um atributo **Ai**, a ordem dos atributos em esquemas de relações é apenas uma questão de disposição física.

3.3 Atributo Chave de uma Relação

Chave é uma **Superchave** da qual não se pode retirar nenhum atributo para que assim seja preservada a sua propriedade de identificação única.

E O conjunto: *(RA, Nome, Endereço)* é uma superchave para estudante, porém, não é uma chave, pois, se tirarmos o campo *Endereço*, continuaremos a ter uma superchave. Já o conjunto *(Nome da Revista, Volume, N da Revista)* é uma superchave e uma chave, pois, se retirarmos qualquer um dos atributos, deixaremos de ter uma superchave, ou seja, *(Nome da Revista, Volume)* não identificam de maneira única uma tupla.

Em outras palavras, uma superchave é uma **Chave Composta**, ou seja, uma chave formada por mais que um atributo.

E

Tabela DEPENDENTES				
<u>RG</u> <u>Responsável</u>	<u>Nome Dependente</u>	Dt. Nascimento	Relação	Sexo
10101010	Jorge	27/12/86	Filho	Masculino
10101010	Luiz	18/11/79	Filho	Masculino
20202020	Fernanda	14/02/69	Conjuge	Feminino
20202020	Angelo	10/02/95	Filho	Masculino
30303030	Fernanda	01/05/90	Filho	Feminino



Quando uma relação possui mais que uma chave (não confundir com chave composta) como, por exemplo, RG e CIC para empregados, cada uma dessas chaves são chamadas de **Chaves Candidatas**. Uma delas deve ser escolhida como **Chave Primária**.



Uma **chave estrangeira CE** de uma tabela **R1** em **R2**, ou vice-versa, especifica um relacionamento entre as tabelas **R1** e **R2**.



Tabela DEPARTAMENTO		
Nome	<u>Número</u>	RG Gerente
Contabilidade	1	10101010
Engenharia Civil	2	30303030
Engenharia Mecânica	3	20202020

Tabela EMPREGADO					
Nome	<u>RG</u>	CIC	Depto.	RG Supervisor	Salário
João Luiz	10101010	11111111	1	NULO	3.000,00
Fernando	20202020	22222222	2	10101010	2.500,00
Ricardo	30303030	33333333	2	10101010	2.300,00



3.4. Restrições de Integridade

São regras a respeito dos valores que podem ser armazenados nas relações; regras, às quais, devem ser sempre satisfeitas. Existem três regras que são consideradas necessárias para uma base de dados relacional:

- **Restrição de Integridade da Chave:** uma chave candidata qualquer não poderá ter o mesmo valor em duas tuplas distintas da mesma relação.
- **Restrição de Integridade da Entidade:** a chave primária de qualquer relação não poderá ser nula em nenhuma tupla (linha da tabela) dessa relação.
- **Restrição de Integridade Referencial:** informalmente, a restrição de integridade referencial declara que uma tupla (linha da tabela) em uma relação, que faz referência a outra relação, deve se referir a uma tupla (linha da tabela) existente nessa relação. O conceito de Integridade Referencial está ligado ao conceito de **Chave Estrangeira**.

3.5. Mapeamento do Modelo E-R para o Modelo Relacional

O **Modelo Entidade-Relacionamento** é um **Modelo Conceitual** utilizado para especificar a estrutura de dados de uma aplicação.

O **Modelo Relacional** é um **Modelo Lógico (porém bem próximo ao Físico)**, trata-se de uma descrição de um banco de dados no nível de abstração visto pelo usuário do SGBD. Assim, o modelo lógico é dependente do tipo particular de SGBD que está sendo usado.

O **Mapeamento** permite que se **traduzam os esquemas concebidos com um modelo de conteúdo semântico mais alto** para uma implementação (MER), utilizando um modelo lógico ou físico (MRel).



O Mapeamento do MER para o Modelo Relacional (MRel) é um procedimento executado em **seis passos** consecutivos apresentados a seguir.

Passo 1: cada **Conjunto de Entidades é mapeado** como uma relação (tabela) que envolve todos os atributos (campos) do conjunto de entidades. Os atributos Chave irão compor a chave da relação

E

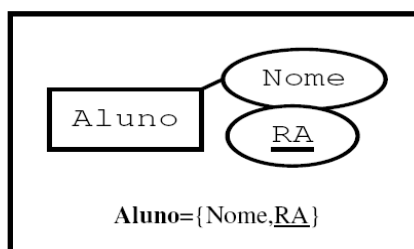


Figura 12 – Mapeamento de Entidades

Passo 2: os **Conjuntos de Entidades Fracas (CEF)** são mapeadas numa relação formada por todos os atributos do CEF, mais os atributos que são chave da relação do qual o CEF depende.

E

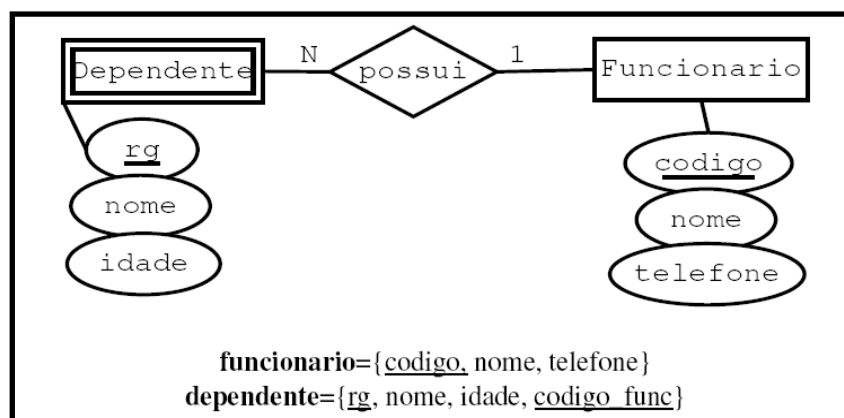


Figura 13 – Mapeamento de Entidades Fracas

Passo 3: os **Conjuntos de Relacionamentos Binários (CRB) de Cardinalidade 1:1 não são representados como novas relações**. Seus atributos são acrescentados em uma das relações que mapeiam os Conjuntos de Entidades (CE)

envolvidos (qualquer uma). Nessa mesma relação, incluem-se também os atributos chave da relação que mapeia o outro CE.

E

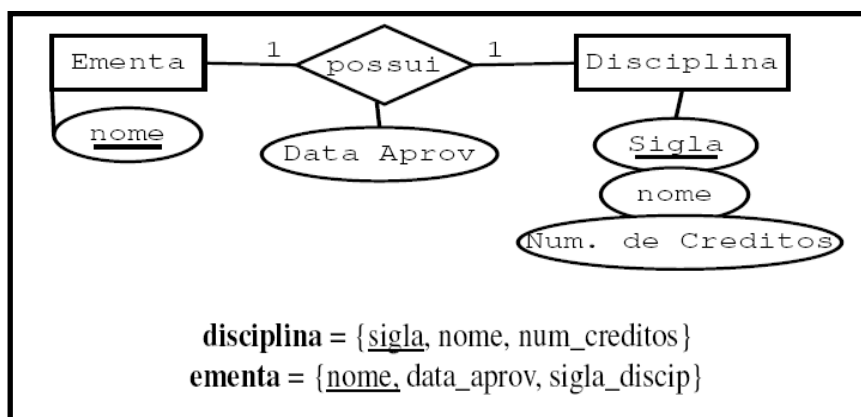


Figura 14 – Mapeamento de Relacionamentos 1:1

Passo 4: os Conjuntos de Relacionamento Binário com Cardinalidade (CRB) 1:N também não são representados como novas relações. Seus atributos são acrescentados na relação que mapeia o CE que ocupa o papel de cardinalidade **N**. Os atributos chave da relação que mapeia o CE que participa com cardinalidade 1 são também acrescentados no CE com papel de cardinalidade **N**.

E

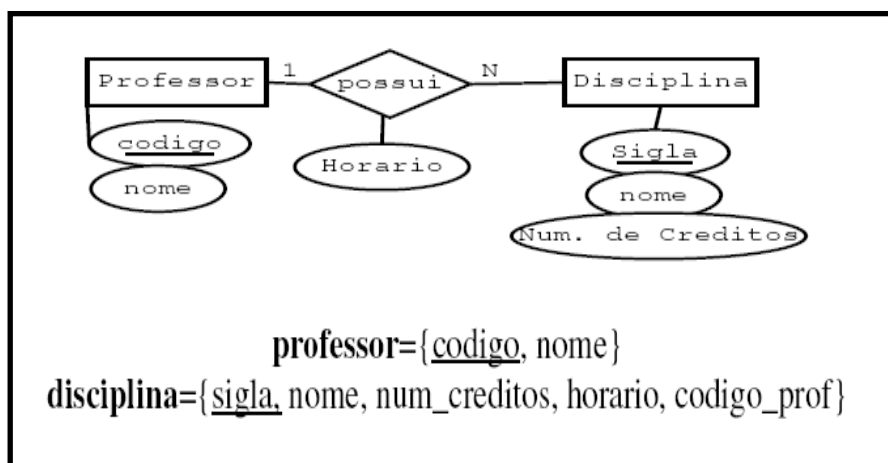


Figura 15 - Relacionamento Binário com Cardinalidade 1:N

Passo 5: cada **Conjunto de Relacionamento Binário (CRB) de Cardinalidade M:N** é representado como uma nova relação. Os atributos da relação são os do CRB mais os atributos chave das relações que mapeiam os CEs envolvidos. A **Chave da Relação** é a concatenação dos atributos chaves das relações que mapeiam os CEs envolvidos.

E

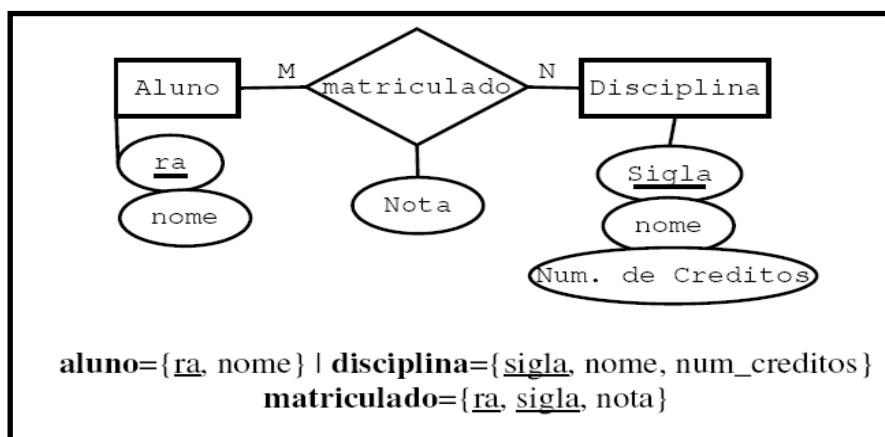


Figura 16 – Relacionamento Binário com Cardinalidade M:N

Passo 6: os **Conjuntos de Relacionamentos de ordem maior do que dois com cardinalidade diferente de M:N:P** têm um mapeamento complexo. Assim, usualmente, se mapeiam os conjuntos de Relacionamentos ternários, quaternários, etc., como se todos fossem de cardinalidade vários para vários para vários.

F

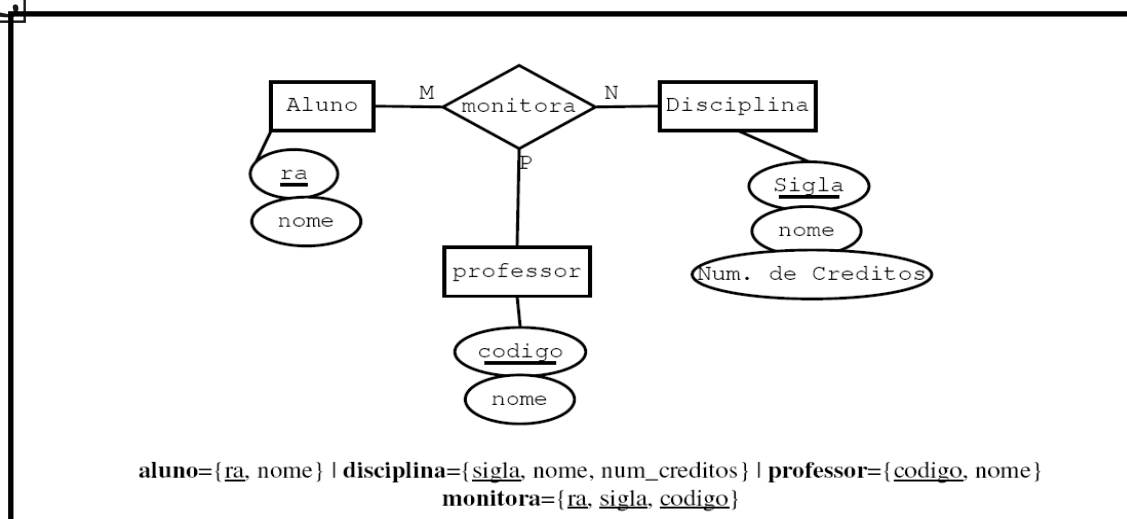


Figura 17 - Relacionamento Ternário com Cardinalidade M:N:P



Exemplo de Mapeamento de um DER Completo para o Modelo Relacional.

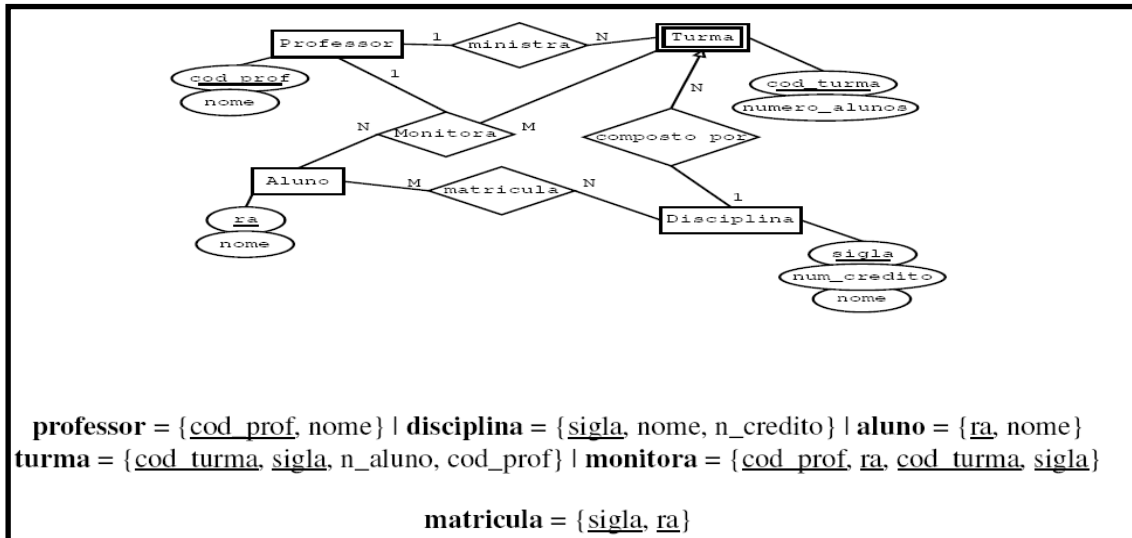


Figura 18 - DER Completo Mapeado para o Relacional

3.5.1. Atributos Compostos e Multivalorados

Os atributos compostos serão decompostos nas relações e os multivalorados tornar-se-ão relações cuja chave primária será composta pela chave da entidade possuidora do atributo mais o atributo multivalorado.

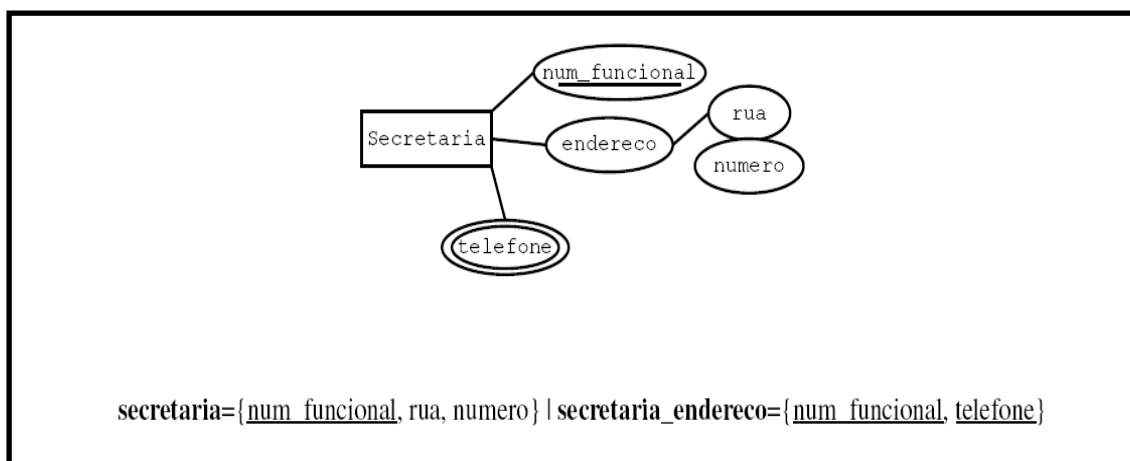


Figura 19 - Mapeamento de Atributos Compostos e Multivalorados

3.5.2. Generalização

E Quanto à generalização:

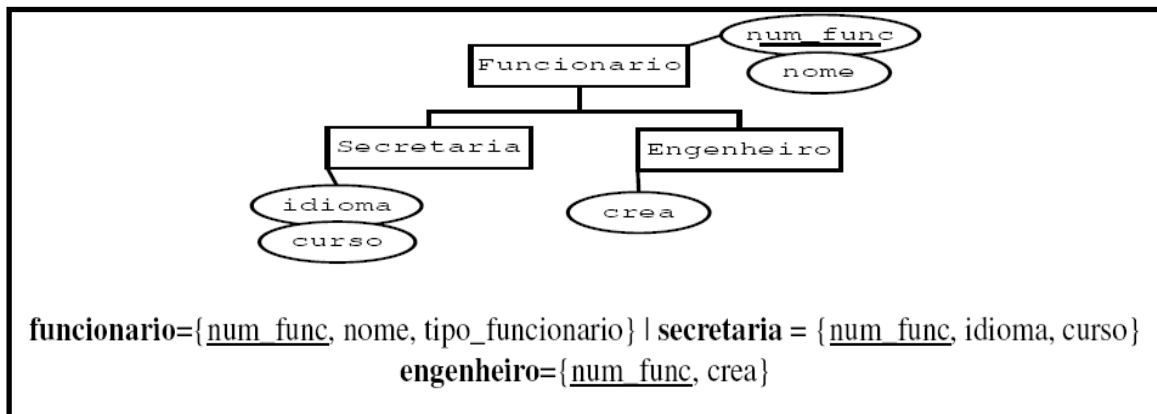


Figura 20 - Mapeamento Generalização/Especialização

3.5.3. Agregação

E Quanto à agregação:

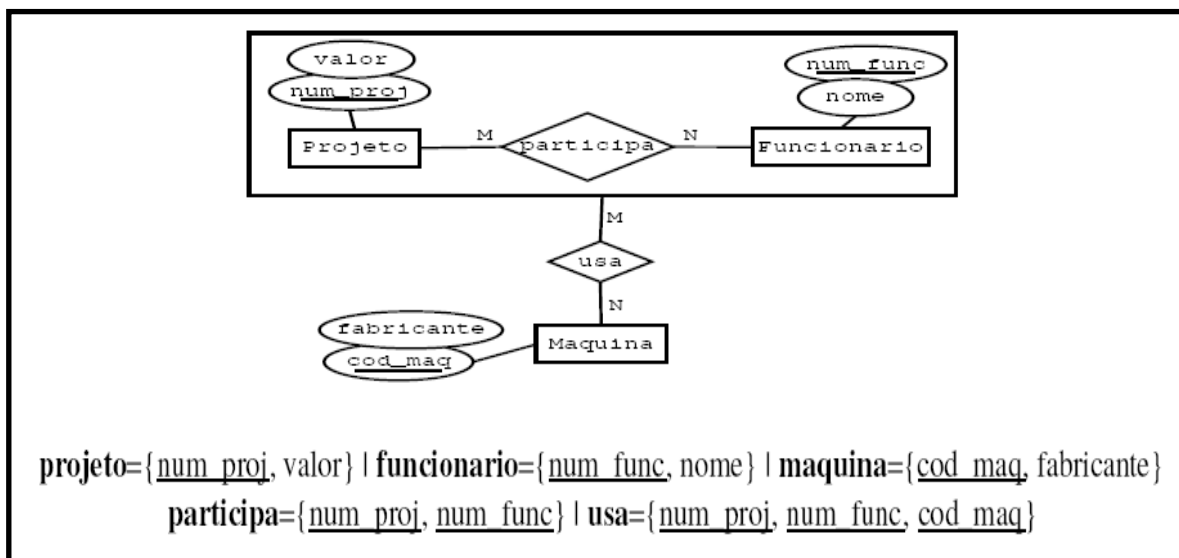


Figura 21 – Mapeamento Agregação

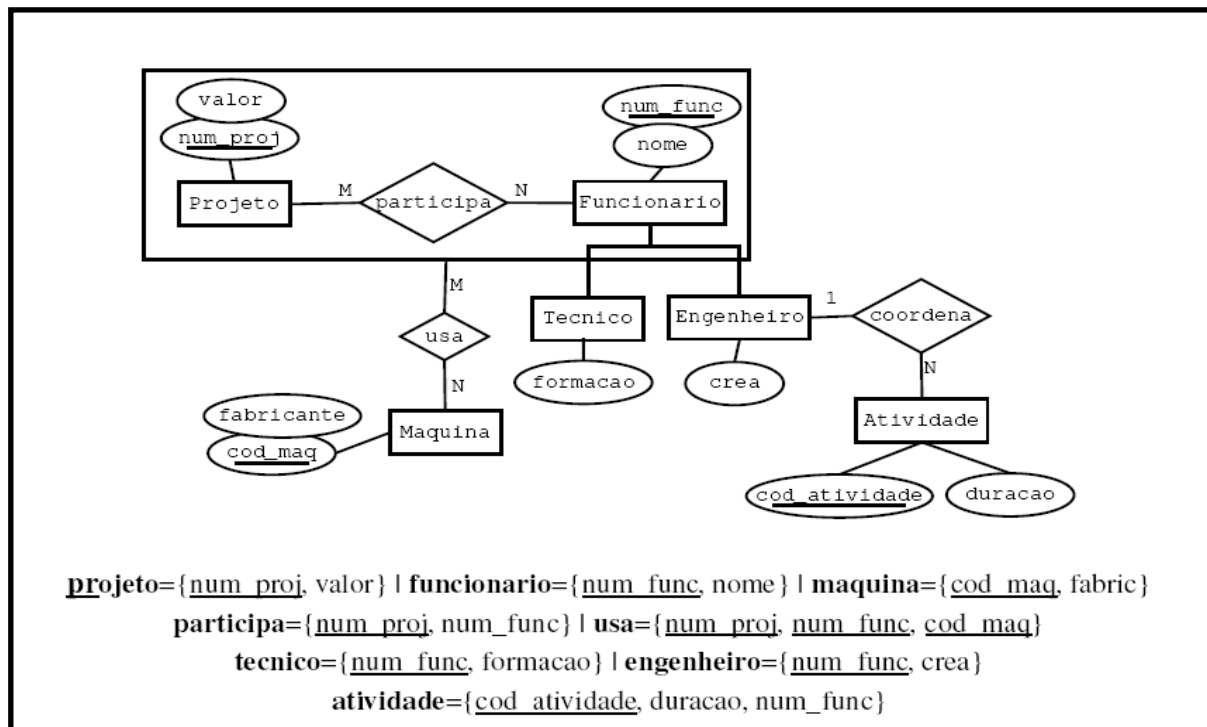


Figura 22 - DER Completo Mapeado para o Relacional

3.6. Normalização

O conceito de normalização foi introduzido por E. F. Codd em 1972. Inicialmente Codd criou as três primeiras formas de normalização chamando-as de: primeira forma normal (1NF), segunda forma normal (2NF) e terceira forma normal (3NF). Uma definição mais forte da 3NF foi proposta depois por Boyce-Codd, e é conhecida como forma normal de Boyce-Codd (FNBC).

Através do processo de normalização pode-se, gradativamente, substituir um conjunto de entidades e relacionamentos por um outro, o qual se apresenta "purificado" em relação às anomalias de atualização (inclusão, alteração e exclusão) as quais podem causar certos problemas, tais como:

- grupos repetitivos (atributos multivalorados) de dados;

- variação temporal de certos atributos, dependências funcionais totais ou parciais em relação a uma chave concatenada;
- redundâncias de dados desnecessárias;
- perdas acidentais de informação;
- dificuldade na representação de fatos da realidade observada;
- dependências transitivas entre atributos.

Normalização de relações é portanto uma técnica que permite depurar um projeto de banco de dados, através da identificação de inconsistências (informações em duplicidade, dependências funcionais mal resolvidas, etc).

À medida que um conjunto de relações passa para uma forma normal, vamos construindo um banco de dados mais confiável.

O objetivo da normalização não é eliminar todas as inconsistências, e sim controlá-las.

Primeira Forma Normal

Uma relação está na primeira forma normal se todos os seus atributos são monovalorados e atômicos.

Quando encontrarmos um atributo multivalorado, deve-se criar um novo atributo que individualize a informação que esta multivalorada:

BOLETIM = {matricula-aluno, materia, notas}

No caso acima, cada nota seria individualizada identificando a prova a qual aquela nota se refere:

BOLETIM = {matricula-aluno, materia, numero-prova, nota}

Quando encontrarmos um atributo não atômico, deve-se dividi-lo em outros atributos que sejam atômicos:

PESSOA = {CPE, nome-completo}

Vamos supor que, para a aplicação que utilizará esta relação, o atributo nome-completo não é atômico, a solução então será:

PESSOA = {CPE, nome, sobrenome}

Segunda Forma Normal

Uma relação está na segunda forma normal quando duas condições são satisfeitas:

1. a relação estiver na primeira forma normal;
2. todos os atributos primos dependerem funcionalmente de toda a chave primária.

Observe a relação abaixo:

BOLETIM = {matricula-aluno, codigo-materia, numero-prova, nota, data-da-prova, nome-aluno, endereço-aluno, nome-materia}

Fazendo a análise da dependência funcional de cada atributo primo, chegamos às seguintes dependências funcionais:

- matricula-aluno, codigo-materia, numero-prova -> nota
- codigo-materia, numero-prova -> data-da-prova
- matricula-aluno -> nome-aluno, endereço-aluno
- codigo-materia -> nome-materia

Concluimos então que apenas o atributo primo nota depende totalmente de toda chave primária. Para que toda a relação seja passada para a segunda forma normal,

deve-se criar novas relações, agrupando os atributos de acordo com suas dependências funcionais:

BOLETIM = {matricula-aluno, codigo-materia, numero-prova, nota}

PROVA = {codigo-materia, numero-prova, data-da-prova}

ALUNO = {matricula-aluno, nome-aluno, endereço-aluno}

MATERIA = {codigo-materia, nome-materia}

O nome das novas relações deve ser escolhido de acordo com a chave.

Terceira Forma Normal

Uma relação está na terceira forma normal quando duas condições forem satisfeitas:

1. a relação estiver na segunda forma normal;
2. todos os atributos primos dependerem não transitivamente de toda a chave primária.

Observe a relação abaixo:

PEDIDO = {numero-pedido, codigo-cliente, data-pedido, nome-cliente, codigo-cidade-cliente, nome-cidade-cliente}

Fazendo a análise da dependência funcional de cada atributo primo, chegamos às seguintes dependências funcionais:

- numero-pedido -> codigo-cliente
- numero-pedido -> data-pedido
- codigo-cliente -> nome-cliente
- codigo-cliente -> codigo-cidade-cliente

- codigo-cidade-cliente -> nome-cidade-cliente

Concluimos então que apenas os atributos primos codigo-cliente e data-pedido dependem não transitivamente totalmente de toda chave primária.

Observe que:

- numero-pedido -> codigo-cliente -> nome-cliente
- numero-pedido -> codigo-cliente -> codigo-cidade-cliente
- numero-pedido -> codigo-cliente -> codigo-cidade-cliente -> nome-cidade-cliente

Isto é dependência transitiva, devemos resolver inicialmente as dependências mais simples, criando uma nova relação onde codigo-cliente é a chave, o codigo-cliente continuará na relação PEDIDO como atributo primo, porém, os atributos que dependem dele devem ser transferidos para a nova relação:

PEDIDO = {numero-pedido, codigo-cliente, data-pedido}

CLIENTE = {codigo-cliente, nome-cliente, codigo-cidade-cliente, nome-cidade-cliente}

As dependências transitivas da relação PEDIDO foram eliminadas, porém ainda devemos analisar a nova relação CLIENTE:

codigo-cliente -> codigo-cidade-cliente -> nome-cidade-cliente

Observe que o nome-cidade-cliente continua com uma dependência transitiva, vamos resolvê-la da mesma maneira :

PEDIDO = {numero-pedido, codigo-cliente, data-pedido}

CLIENTE = {codigo-cliente, nome-cliente, codigo-cidade-cliente}

CIDADE = {codigo-cidade-cliente, nome-cidade-cliente}

O nome das novas relações deve ser escolhido de acordo com a chave.



Questões para a Síntese:

1. Identifique, nas relações abaixo, as dependências funcionais e, se houver violação de alguma forma normal (1, 2 e 3), normalize-as. **Observação:** Os campos que estão entre parênteses indicam campos multivalorados.

(a) Funcionario={RG, nome, endereco, dada_nasc.}

(b) Projeto={Codigo, descricao, (cidade),ano}

(c) Atleta={ID_Atleta, Nome, numero_entidade_patrocinadora, ender_entidade_patrocinadora, sexo, (numero_documento, orgao_expedidor_documento, data_expedicao_documento)}

(d) Pedido= {Numero, data, item, descricao_item, quantidade, valor_do_pedido}

(e) Aluno={ RA, nome_aluno, Endereco, cod_professor, nome_professor, cod_disciplina, descricao_disciplina}

(f) Emprestimo={ Cod_agencia, cidade_agencia, cic_cliente, nome_cliente, (endereco_cliente), valor_empréstimo}

(g) Salario={RG_Func, nome_funcion, (data_salario, valor)}

(h) Gerencia={Cod_Depart, RG_Gerente, descricao_depart}

4. LINGUAGENS: ÁLGEBRA E CÁLCULO RELACIONAL

4.1. Álgebra Relacional

Conforme Korth e Silberschatz (1999), a **Álgebra Relacional** é uma linguagem de **consulta procedural**. Consiste em um conjunto de operações que utilizam uma ou mais relações como entrada e produz uma nova relação como saída.



Todas as operações da álgebra sobre uma relação produzem uma outra relação. Mesmo que a relação resultante seja vazia.

As principais operações são:

- Seleção
- Projeção
- União
- Diferença
- Intersecção
- Produto Cartesiano
- Junção
- Divisão

Antes de estudarmos cada uma dessas operações, considere as instâncias das relações que compõem o banco de dados abaixo:

Relação Cliente

cod_cli	nome_cli	rua	cidade
c1	Joao	Rua TT	Rio Preto
c3	Pedro	Rua A4	Sao Paulo
c2	Marcos	Rua XY	Sao Paulo
c4	Maria	Rua A	Belo Horizonte

Relação Agência

cod_agenc	nome_agenc	gerente	cidade
ag1	Sao Jose	Pedro	Sao Paulo
ag4	Botafogo	Manoel	Rio de Janeiro
ag3	Praiana	Dario	São Paulo
ag2	Bom Retiro	Pedro	Belo Horizonte

Relação Conta

cod_agenc	cod_cli	num_conta	saldo
ag1	c3	101	500
ag2	c2	215	780
ag1	c1	102	400
ag3	c4	305	350
ag3	c3	105	230

Relação Empréstimo

cod_agenc	cod_cli	num_emprest	valor
ag3	c4	e100	1500
ag2	c2	e500	800
ag2	c2	e550	100
ag1	c1	e230	1200
ag3	c1	e150	150

4.1.1. Operação Selecionar (σ)

A operação “Selecionar” seleciona tuplas (linhas), que satisfazem um dado predicado (condição). É representada pela letra grega Sigma (σ).

E

Considere a consulta:

Encontre os empréstimos feitos na agência ag2:

A resposta em álgebra relacional seria:

$\sigma_{cod_agenc='ag2'}(empréstimo);$

O resultado dessa consulta apresentaria as seguintes tuplas:

cod_agenc	cod_cli	num_emprest	valor
ag2	c2	e500	800
ag2	c2	e550	100

E

Se quisermos saber todas as contas que possuem saldo maior do que 400:

$\sigma_{saldo > 400}(conta);$

cod_agenc	cod_cli	num_conta	saldo
ag1	c3	101	500
ag2	c2	215	780

Geralmente, permitimos as comparações =, \neq , <, \leq , >, \geq .

Além disso, diversos predicados (condições) poderão ser combinados em um predicado maior, utilizando os conectivos *AND* (\wedge) e *OR* (\vee).

E

Para encontrarmos os empréstimos maiores do que 500 e realizados na agência “ag2”.

$\sigma_{cod_agenc="ag2" \wedge valor > 500} (emprestimo);$

cod_agenc	cod_cli	num_emprest	valor
ag2	c2	e500	800

4.1.2. A Operação Projetar (π)

Representada pela letra grega pi (π), a operação “Projetar” seleciona (projeta) atributos específicos de uma relação. O resultado da projeção é uma nova relação com os atributos selecionados.

E

Queremos saber o nome de todos os clientes de um banco:

$\pi_{nome_cli}(cliente);$

Teremos como resposta:

nome_cli
Joao
Pedro
Marcos
Maria

E

Agora queremos saber o nome dos gerentes e suas respectivas agências:

$\pi_{gerente, nome_agenc}(agencia);$

gerente	nome_agenc
Pedro	Sao Jose
Manoel	Botafogo
Dario	Praiana
Pedro	Bom Retiro

E

Se desejarmos saber o nome de todos os gerentes, sem saber suas agências:

$\Pi_{\text{gerente}}(\text{agencia});$

gerente
Pedro
Manoel
Dario

Note que apareceu apenas um gerente de nome Pedro. Isto se dá ao fato de que, para a álgebra, uma relação é um conjunto de elementos e, em um conjunto, não existem valores repetidos.

Podemos combinar as operações de projeção com seleção.

E

Considere a consulta em que desejamos saber os nomes dos clientes que residem em São Paulo:

$\Pi_{\text{nome_cli}}(\sigma_{\text{cidade} = \text{"Sao Paulo"}}(\text{cliente}));$

nome_cli
Pedro
Marcos

E

Ou o código da agência e o código do cliente cujo saldo da conta seja inferior ou igual a 400:

$\Pi_{\text{cod_agenc}, \text{cod_cli}}(\sigma_{\text{saldo} \leq 400}(\text{saldo}));$

cod_agenc	cod_cli
ag1	c1
ag3	c4
ag3	c3



As operações “Projetar” e “Selecionar” são consideradas **unárias**. Pois operam sobre apenas uma única relação.

4.1.3. Operação União (U)

A união de duas relações é formada pela adição das tuplas de uma relação às tuplas de uma segunda relação, para que seja produzida uma terceira. É representada, como na teoria dos conjuntos, pelo símbolo (U).



Queremos saber todos os clientes (cod_cli) que possuem contas ou que fizeram empréstimos na agência “ag3”. Existem duas maneiras de realizar essa consulta:

Maneira 1:

$R1 = \Pi_{cod_cli}(\sigma_{cod_agenc="ag3"}(conta));$

$R2 = \Pi_{cod_cli}(\sigma_{cod_agenc="ag3"}(emprestimo));$

$R3 = R1 \cup R2;$

Maneira 2:

$\Pi_{cod_cli}(\sigma_{cod_agenc="ag3"}(conta)) \cup \Pi_{cod_cli}(\sigma_{cod_agenc="ag3"}(emprestimo));$

Resultado:

cod_cli
c4
c1
c3

4.1.4. A Operação Diferença de Conjuntos (-)

A diferença de duas relações é uma terceira relação contendo as tuplas que ocorrem na primeira relação, mas não ocorrem na segunda. É representada pelo símbolo (-).



Desejamos encontrar todos os clientes (cod_cli) que possuem contas, mas não fizeram empréstimos:

$$R1 = \Pi_{cod_cli}(conta);$$

$$R2 = \Pi_{cod_cli}(emprestimo);$$

$$R3 = R1 - R2;$$

cod_cli
c3

4.1.5. A Operação Produto Cartesiano (X)

O produto de duas relações é a concatenação de todas as tuplas de uma relação com todas as tuplas de uma segunda relação. O produto da relação A (tendo m tuplas) com a relação B (tendo n tuplas) é uma relação contendo $m \times n$ tuplas. A representação do produto cartesiano é feita pelo símbolo X.



Encontre o nome dos clientes e o número de suas respectivas contas bancárias que possuem saldo maior do que 450:

$$R1 = \sigma_{saldo > 450}(conta);$$

$$R2 = cliente \times R1;$$

cliente. cod_cli	cliente. nome_cli	cliente. rua	cliente. cidade	conta. cod_agenc	conta. cod_cli	conta. num_conta	conta. saldo
c1	Joao	Rua TT	Rio Preto	ag1	c3	101	500
c1	Joao	Rua TT	Rio Preto	ag2	c2	215	780
c3	Pedro	Rua A4	Sao Paulo	ag1	c3	101	500
c3	Pedro	Rua A4	Sao Paulo	ag2	c2	215	780
c2	Marcos	Rua XY	Sao Paulo	ag1	c3	101	500
c2	Marcos	Rua XY	Sao Paulo	ag2	c2	215	780
c4	Maria	Rua A	Belo Horizonte	ag1	c3	101	500
c4	Maria	Rua A	Belo Horizonte	ag2	c2	215	780

$$R3 = \Pi_{nome_cli, num_conta}(\sigma_{cliente.cod_cli = conta.cod_cli}(R2));$$

nome_cli	num_conta
c3	101
c2	215

4.1.6. A Operação Junção Natural (\bowtie)

A operação de “Junção Natural” forma um produto cartesiano. É representada pelo símbolo \bowtie . Essa operação faz uma seleção forçando uma igualdade sobre os atributos que aparecem em ambas as relações e finalmente remove colunas duplicadas.

Um problema que ocorre no “Produto Cartesiano” é o consumo excessivo de memória. Imagine duas relações em que uma contém 3000 tuplas de 100 bytes cada, e outra com 5000 tuplas de 200 bytes cada. Seria necessário 4.5 mb de memória para armazenar a relação resultante. Sendo assim, a operação “Junção Natural” é vista como um meio mais eficiente de se realizar esse tipo de operação.

Vamos considerar novamente a consulta na qual desejamos saber o nome dos clientes e suas respectivas contas bancárias que possuem saldo maior que 450. Ela poderá ser realizada de duas maneiras:

Maneira 1:

$R1 = \sigma_{saldo > 450}(conta);$

$R2 = \Pi_{nome_cli, num_conta}(cliente \bowtie_{num_cli = num_cli(R1)} R1);$

Maneira 2:

$R1 = \sigma_{saldo > 450}(conta);$

$R2 = \Pi_{nome_cli, num_conta}(cliente \bowtie R1)$

//É importante frisar que desta maneira, os atributos de junção estão implícitos no símbolo de junção;

nome_cli	num_conta
c3	101
c2	215

4.1.7. A Operação Divisão (\div)

A operação divisão, representada por (\div), retorna as tuplas da relação A que se relacionam com todas as tuplas da relação B, ao mesmo tempo. Não é uma operação importante como as anteriores.

E

Considere agora que queremos encontrar todos os clientes (cod_cli) que possuem pelo menos uma conta em todas as agências de “São Paulo”:

$$R1 = \Pi_{cod_agenc}(\sigma_{cidade="Sao\ Paulo"}(agencia));$$

cod_agenc
ag1
ag3

$$R2 = \Pi_{cod_cli, cod_agenc}(conta);$$

cod_cli	cod_agenc
c3	ag1
c2	ag2
c1	ag1
c4	ag3
c3	ag3

$$R3 = \Pi_{cod_cli, cod_agenc}(R2) \div \Pi_{cod_agenc}(R1);$$

cod_cli	cod_agenc
c3	ag1
c3	ag3

4.2. Cálculo Relacional

O **cálculo relacional** é uma **linguagem de consulta formal baseada em expressões declarativas**. Estas são utilizadas para especificar a busca por informações de interesse.

Ao contrário da **Álgebra Relacional**, considerada uma **linguagem procedural**, o **cálculo relacional** é considerado uma **linguagem não-procedural**. Em outras palavras, uma expressão do cálculo relacional especifica “o que” está sendo buscado, e não “como” a busca será feita.

No cálculo relacional, existem variáveis, constantes, operadores lógicos, de comparação, e quantificadores. As expressões do cálculo são chamadas fórmulas.

Uma tupla-resposta (linha-resposta) é essencialmente uma atribuição de constantes às variáveis que leva a fórmula a um estado verdadeiro.

Existem dois tipos básicos de cálculo relacional: o **Cálculo Relacional de Tuplas** (CRT) e o **Cálculo Relacional de domínios (CRD)**, descritos a seguir.

4.2.1. Cálculo Relacional de Tuplas

Para Korth e Silberschatz (1999), o cálculo relacional de tuplas está baseado na especificação de um número de variáveis do tipo tupla. Qualquer variável desse tipo pode assumir como valor qualquer tupla da relação especificada.

Uma consulta em CRT é especificada da seguinte forma:

$$\{t|COND(t)\}$$

em que t é uma variável do tipo tupla e $COND(t)$ é uma expressão condicional envolvendo t . O resultado de uma consulta desse tipo corresponde ao conjunto de todas as tuplas t que satisfazem a condição em $COND(t)$.



Para encontrar todos os funcionários cujo salário é maior que R\$5.000,00, a seguinte expressão pode ser utilizada:

$$\{t|FUNCIONARIO(t) \text{ E } t.Salario > 5.000,00\}$$

A condição $FUNCIONARIO(t)$ determina que a variável t pode receber como valor qualquer tupla da relação $FUNCIONARIO$. Cada tupla t de $FUNCIONARIO$ que satisfaz a condição $t:Salario > 5.000,00$ é recuperada.



A expressão abaixo:

$$\{t.NomeFunc|FUNCIONARIO(t) \text{ E } t.Salario > 5.000,00\}$$

é utilizada para recuperar o nome dos funcionários que recebem mais do que 5.000,00.

Com base nos exemplos acima, as seguintes informações precisam ser especificadas durante a construção de uma expressão do cálculo relacional de tuplas:

1. para cada variável t do tipo tupla, especificar a relação R cujas tuplas podem ser atribuídas a t ;
2. uma condição para selecionar uma combinação particular de tuplas;
3. um conjunto de atributos de interesse.



Na expressão:

$\{t.NomeFunc, t.DataNasc | FUNCIONARIO(t) \text{ E } t.NomeFunc = 'José'\}$

$FUNCIONARIO$ corresponde à relação R cujas tuplas podem ser atribuídas a t , $FUNCIONARIO(t)$ e $t.NomeFunc = 'José'$, corresponde à condição para a seleção das tuplas e ' $t.NomeFunc, t.DataNasc$ ' são os atributos de interesse.

Genericamente, uma expressão do cálculo relacional de tuplas é representada da seguinte forma:

$\{t1.A1, t2.A2, \dots, tn.An | COND (t1, t2, \dots, tn, tn+1, tn+2, \dots, tn+m)\}$

em que $t1, t2, \dots, tn, tn+1, tn+2, \dots, tn+m$ são variáveis do tipo tuplas, cada Ai é um atributo da relação sobre a qual ti encontra-se definida e $COND$ é uma fórmula ou condição do cálculo relacional de tuplas.

Uma fórmula do cálculo relacional é formada de átomos, que podem ser:

1. um átomo da forma $R(ti)$, em que R é o nome de uma relação e ti é uma variável do tipo tupla.

2. um átomo da forma $ti.A \text{ op } tj.B$ em que **op** é um dos operadores de comparação, ti e tj são duas variáveis do tipo tupla e $A(B)$ é um atributo da relação sobre a qual $ti(tj)$ está definida.

3. um átomo da forma $ti.A \text{ op } c$ em que **op** é um dos operadores de comparação, c é uma constante, ti e tj são duas variáveis do tipo tupla e $A(B)$ é um atributo da relação sobre a qual $ti(tj)$ está definida.

Uma fórmula é composta de um ou mais átomos conectados via os operadores lógicos E (AND), OU (OR) e NÃO (NOT). Além disso, alguns símbolos especiais podem aparecer em fórmulas do cálculo relacional.

E

É o caso do símbolo de quantificador existencial (\exists) que permite consultas do tipo:

$\{t.NomeFunc | FUNCIONARIO(t) \text{ E } (\exists d)(DEPARTAMENTO(d) \text{ E } d.Nome = 'Pesquisa \text{ E } d.NumDep = t.NumDep)\}$

A consulta acima retorna o nome e endereço de todos os funcionários que trabalham no departamento de pesquisa.

E

Abaixo seguem exemplos de consultas em CRT.

1) Encontre todos os empregados cujos salários estejam acima de R\$3.500,00.

$\{t | EMPREGADO(t) \text{ AND } t.SALARIO > 3500\}$

2) Dê apenas os nomes e sobrenomes dos empregados cujos salários estejam acima de R\$3.500,00.

$\{t.NOME, t.SOBRENOME | EMPREGADO(t) \text{ AND } t.SALARIO > 3500\}$

3) Selecione o nome e o endereço dos empregados que trabalham para o departamento de 'Informática'.

$\{t.NOME, t.SOBRENOME, t.ENDERECO \mid EMPREGADO(t) \text{ AND } (\exists d) (DEPARTAMENTO(d) \text{ AND } d.NOMED = 'Informática' \text{ AND } d.NUMERODEP = t.NUD)\}$

4) Para cada projeto localizado em 'São Paulo', liste o número dele, o nome do departamento proponente, bem como sobrenome, data de nascimento e endereço do gerente responsável.

$\{p.NUMEROP, p.NUMD, m.SOBRENOME, m.DATANASCIMENTO, m.ENDERECO \mid PROJETO(p) \text{ AND } EMPREGADO(m) \text{ AND } p.LOCALIZACAO = 'São Paulo' \text{ AND } ((\exists d) (DEPARTAMENTO(d) \text{ AND } d.NUMD = d.NUMERODEP \text{ AND } d.NSSGER = m.NSS))\}$

4.2.2. Cálculo Relacional de Domínio

A **diferença básica entre CRT e CRD** é que, **neste último, as variáveis estendem-se sobre valores únicos de domínios de atributos**. Para formar uma relação de grau n para um resultado de consulta, faz-se necessário criar n variáveis de domínio, uma para cada atributo.

Uma expressão genérica do cálculo relacional de domínios tem a seguinte forma:

$\{x_1, x_2, \dots, x_n \mid COND(x_1, x_2, \dots, x_n, x_{n+1}, x_{n+2}, \dots, x_{n+m})\}$

em que $x_1, x_2, \dots, x_n, x_{n+1}, x_{n+2}, \dots, x_{n+m}$ são variáveis de domínio aplicadas sobre o domínio dos atributos requeridos na consulta, e COND é **uma condição fórmula do cálculo relacional de domínios, que pode ser especificada utilizando os seguintes átomos**:

1. Uma fórmula atômica $R(x_1, x_2, \dots, x_n)$, em que R é o nome de uma relação de grau j e cada $x_i, 1 \leq i \leq j$, é uma variável de domínio. Isso implica que uma lista de

valores de $\langle x_1, x_2, \dots, x_j \rangle$ deve ser uma tupla na relação R, em que x_i é o valor do i -ésimo valor de atributo da tupla.

2. Uma fórmula atômica $x_i \text{ op } x_j$, em que op é um operador de comparação $\{=, <, >, \dots\}$ e x_i e x_j são variáveis de domínio.

3. Uma fórmula atômica $x_i \text{ op } c$ ou $c \text{ op } x_j$, em que op é um operador de comparação $\{=, <, >, \dots\}$, x_i e x_j são variáveis de domínio, e c é um valor constante qualquer.

E

A expressão abaixo é um exemplo de consulta que pode ser feita utilizando o cálculo relacional de domínios:

$$\{s | (\exists q)(\exists r)(\exists t)(\exists u)(\text{FUNCIONARIO}(qrst) \wedge q = \text{'José'})\}$$

A expressão acima retorna a data de nascimento de todos os funcionários que se chamam José. Para uma melhor compreensão dessa expressão, considere o seguinte esquema para uma tabela de funcionários.

FUNCIONARIO

NOMEFUNC	<u>CPE</u>	DATANASC	NUMDEP	CPFSUPER
----------	------------	----------	--------	----------

Na expressão em questão, escolhe-se primeiramente o atributo de interesse. No caso, a data de nascimento, associada à variável s . Determina-se então a condição para seleção das tuplas. Ela exige que os valores associados às variáveis $qrst$ sejam uma tupla do esquema FUNCIONÁRIO e que o valor de q seja José.

E

Uma forma mais compacta de escrever essa consulta é:

$$\{s | \text{FUNCIONARIO}(\text{'José'}, r, s, t, u)\}$$

E

Abaixo, para fins de comparação, seguem em CRD os mesmos exemplos de consultas já escritos em CRT.

1) Encontre todos os empregados cujos salários estejam acima de R\$3.500,00.

$\{qrstuvwxyz \mid (\exists x) \text{EMPREGADO}(qrstuvwxyz) \text{ AND } x > 3500\}$

2) Dê apenas os nomes e sobrenomes dos empregados cujos salários estejam acima de R\$3.500,00.

$\{qs \mid (\exists x) \text{EMPREGADO}(qrstuvwxyz) \text{ AND } x > 3500\}$

3) Selecione o nome e o endereço dos empregados que trabalham para o departamento de 'Informática'.

$\{qsv \mid (\exists z) (\exists l) (\exists m) (\text{EMPREGADO}(qrstuvwxyz) \text{ AND DEPARTAMENTO}(lmno) \text{ AND } l = \text{'Informática'} \text{ AND } m = z)\}$

4) Para cada projeto localizado em 'São Paulo', liste o número dele, o nome do departamento proponente, bem como sobrenome, data de nascimento e endereço do gerente responsável.

$\{iksuv \mid (\exists j) (\exists m) (\exists n) (\exists t) (\text{PROJETO}(hijk) \text{ AND EMPREGADO}(qrstuvwxyz) \text{ AND DEPARTAMENTO}(lmno) \text{ AND } k = m \text{ AND } n = t \text{ AND } j = \text{'São Paulo'})\}$

4.3. Linguagem SQL

A linguagem SQL é um padrão de linguagem de consulta comercial que usa uma combinação de construtores em Álgebra e Cálculo Relacional e possui as seguintes partes:

- Linguagem de definição de dados (DDL)
- Linguagem interativa de manipulação de dados (DML)
- Incorporação DML (*Embedded SQL*)
- Definição de Visões
- Autorização
- Integridade
- Controle de Transações

A SQL permite que uma tabela (relação) tenha duas ou mais tuplas (linhas) idênticas em todos os seus valores de atributos. Assim, em geral, uma relação SQL não é um conjunto de tuplas porque um conjunto não permite elementos idênticos. Ao invés disso, SQL é um multi-conjunto de tuplas.

4.3.1. Estrutura Básica

A estrutura básica de uma expressão SQL consiste em três cláusulas: **select**, **from** e **where**.

- A cláusula **select** corresponde à operação projeção da álgebra relacional. É usada para listar os atributos desejados no resultado de uma consulta.
- A cláusula **from** corresponde à operação produto cartesiano da álgebra relacional. Ela lista as relações a serem examinadas na avaliação da expressão.
- A cláusula **where** corresponde ao predicado de seleção da álgebra relacional.

Consiste em um predicado envolvendo atributos de relações que aparecem na cláusula from.

Uma típica consulta SQL segue a seguinte ordem:

select a1, a2, ..., an β 3ª

from T1, T2, ..., Tm β 1ª

where P β 2ª

Cada ai representa um atributo, cada Ti é uma relação, e P é um predicado. Essa consulta é equivalente à expressão da álgebra relacional:

$$\pi_{a1, a2, \dots, an} (\sigma_P (T1 \times T2 \times \dots \times Tm))$$

A SQL forma o produto cartesiano das relações chamadas na cláusula from, executa uma seleção da álgebra relacional usando o predicado da cláusula where e, então, projeta o resultado para os atributos da cláusula select. Na prática, a SQL pode converter esta expressão em uma forma equivalente que pode ser processada mais eficientemente.

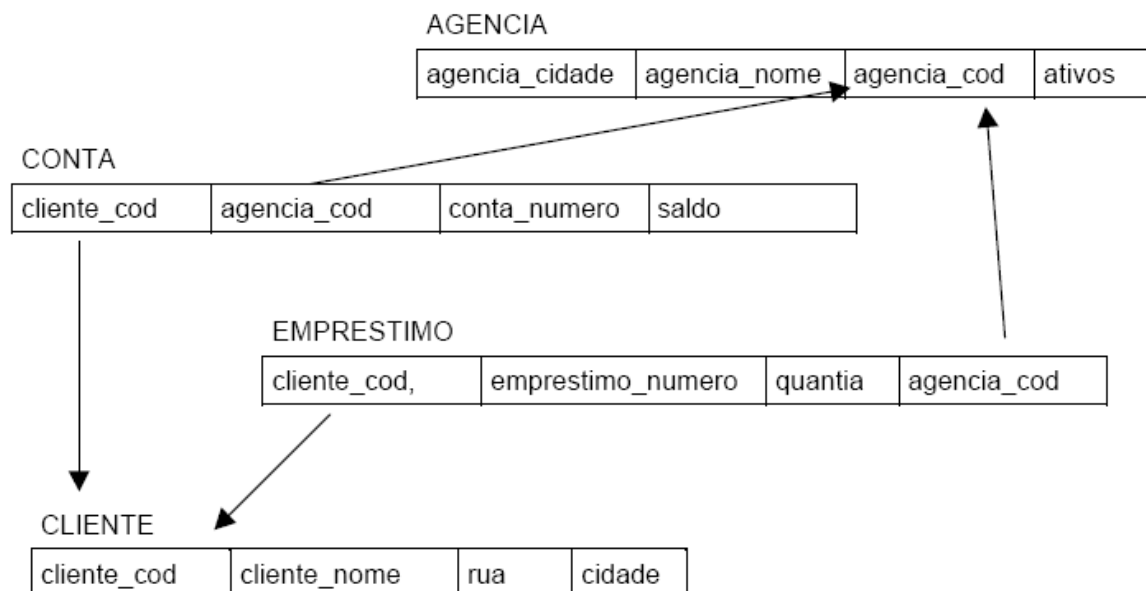


Considere as bases de dados a seguir para as consultas apresentadas como exemplo:

BASE DE DADOS – BANCO

AGENCIA = (agencia_cod, agencia_nome, agencia_cidade, ativos)
 CLIENTE = (cliente_cod, cliente_nome, rua, cidade)
 CONTA = (agencia_cod, conta_numero, cliente_cod, saldo)
 EMPRESTIMO = (agencia_cod, cliente_cod, emprestimo_numero, quantia)

De uma forma visual:



BASE DE DADOS – EMPRESA

Tabela EMPREGADO					
Nome	RG	CPF	Depto	RG Supervisor	Salario
João Luiz	10101010	11111111	1	NULO	3.000,00
Fernando	20202020	22222222	2	10101010	2.500,00
Ricardo	30303030	33333333	2	10101010	2.300,00
Jorge	40404040	44444444	2	20202020	4.200,00
Renato	50505050	55555555	3	20202020	1.300,00

Tabela DEPARTAMENTO		
Nome	Numero	RG Gerente
Contabilidade	1	10101010
Engenharia Civil	2	30303030
Engenharia Mecânica	3	20202020

Tabela PROJETO		
Nome	Numero	Localizacao
Financeiro 1	5	São Paulo
Motor 3	10	Rio Claro
Prédio Central	20	Campinas

Tabela DEPENDENTES				
RG Responsavel	Nome Dependente	Nascimento	Relacao	Sexo
10101010	Jorge	27/12/86	Filho	Masculino
10101010	Luiz	18/11/79	Filho	Masculino
20202020	Fernanda	14/02/69	Cônjuge	Feminino
20202020	Ângelo	10/02/95	Filho	Masculino
30303030	Adreia	01/05/90	Filho	Feminino

Tabela DEPARTAMENTO_PROJETO	
Numero Depto.	Numero Projeto
2	5
3	10
2	20

Tabela EMPREGADO_PROJETO		
RGEmpregado	Numero Projeto	Horas
20202020	5	10
20202020	10	25
30303030	5	35
40404040	20	50
50505050	20	35



No esquema BANCO, encontre os nomes de todos os clientes de 'Vitória'.

```
select cliente_nome
from CLIENTE
where cidade = 'Vitória'
```



Encontre os nomes de todos os clientes.

```
select cliente_nome  
from CLIENTE
```



No esquema EMPRESA, selecionar o nome e o RG dos funcionários que trabalham no departamento número 2 na tabela EMPREGADO

```
select nome, rg  
from EMPREGADOS
```

```
where depto = 2;
```

obteremos o seguinte resultado:

Nome	RG
Fernando	20202020
Ricardo	30303030
Jorge	40404040

A consulta acima é originária da seguinte função em álgebra relacional:

$$\pi_{\text{nome, rg}} (\sigma_{\text{depto} = 2} (\text{EMPREGADOS})) ;$$


Em SQL também é permitido o uso de condições múltiplas. Veja o exemplo a seguir:

```
select nome, rg, salario
```

```
from EMPREGADOS
```

```
where depto = 2 AND salario > 2500.00;
```

que fornece o seguinte resultado:

Nome	RG	Salário
Jorge	40404040	4.200,00

e que é originária da seguinte função em álgebra relacional:

$$\pi_{\text{nome, rg, salario}} (\sigma_{\text{depto} = 2 \text{ .and. } \text{salario} > 3500.00} (\text{EMPREGADOS})) ;$$

O operador * dentro do especificador *select* seleciona todos os atributos de uma tabela, enquanto que a exclusão do especificador *where* faz com que todas as tuplas de uma tabela sejam selecionadas.



Dessa forma, a expressão:

```
select *
```

```
from empregado;
```

gera o seguinte resultado:

Nome	RG	CPF	Depto	RG Supervisor	Salário
João Luiz	10101010	11111111	1	NULO	3.000,00
Fernando	20202020	22222222	2	10101010	2.500,00
Ricardo	30303030	33333333	2	10101010	2.300,00
Jorge	40404040	44444444	2	20202020	4.200,00
Renato	50505050	55555555	3	20202020	1.300,00

4.3.1.1. Cláusulas Distinct e All

Diferente de álgebra relacional, a operação *select* em SQL permite a geração de tuplas duplicadas como resultado de uma expressão. Para evitar isso, devemos utilizar o cláusula **distinct**.



Sejam as seguintes consultas no esquema EMPRESA.

```
select depto
from EMPREGADO;
```

```
select distinct depto
from EMPREGADO;
```

que geram respectivamente os seguintes resultados:

Depto
1
2
2
2
3

Depto
1
2
3

A cláusula All é o default para o select, ou seja: Select All indica para obter todas as tuplas. Logo, esta cláusula não precisa ser colocada (a não ser, talvez, por motivos de documentação).

4.3.1.2. Predicados e ligações

A SQL não tem uma representação da operação ligação natural. No entanto, uma vez que a ligação natural é definida em termos de um produto cartesiano, uma seleção e uma projeção, é relativamente simples escrever uma expressão SQL para uma ligação natural.



Encontre os nomes e cidades de clientes que possuam empréstimos em alguma agência.

Select distinct cliente_nome, cidade

From Cliente, Empréstimo

Where Cliente.cliente_cod=Empréstimo.cliente_cod

A SQL inclui os conectores **and**, **or** e **not**; caracteres especiais: (,), ., :, _, %<, >, <=, >=, =, <>, +, -, * e /; operador para comparação: **between**, como mostra o exemplo a seguir.



Selecionar todas as contas que possuam saldo entre 10000 e 20000.

Select conta_numero

From CONTA

Where saldo >= 10000 **and** saldo <= 20000

Que equivale à consulta

Select conta_numero

From CONTA

Where saldo **between** 10000 **and** 20000

A SQL inclui também um operador para comparações de cadeias de caracteres, o **like**. Ele é usado em conjunto com dois caracteres especiais:

- Por cento (%). Substitui qualquer subcadeia de caracteres.
- Sublinhado (_). Substitui qualquer caractere.



Encontre os nomes de todos os clientes cujas ruas incluem a subcadeia 'na'

Select distinct cliente_nome

From CLIENTE

Where rua **like** '%na%'



Encontre os nomes de todos os clientes cujas ruas finalizem com a subcadeia 'na', seguido de um caractere.

Select distinct cliente_nome

From CLIENTE

Where rua **like** '%na_'

4.3.1.3. Variáveis Tuplas (Renomeação)



No esquema EMPRESA, selecione o número do departamento que controla projetos localizados em Rio Claro;

select t1.numero_depto

```
from departamento_projeto as t1, projeto as t2
where t1.numero_projeto = t2.numero;
```

Na expressão SQL acima, *t1* e *t2* são chamados “*alias*” (apelidos) e representam a mesma tabela a qual estão referenciando. Um “*alias*” é muito importante quando há redundância nos nomes das colunas de duas ou mais tabelas que estão envolvidas em uma expressão. Ao invés de utilizar o “*alias*”, é possível utilizar o nome da tabela, mas isso pode ficar cansativo em consultas muito complexas além do que, impossibilitaria a utilização da mesma tabela mais que uma vez em uma expressão SQL. A palavra chave **as** é opcional.



No esquema EMPRESA, selecione o nome e o RG de todos os funcionários que são supervisores.

```
select e1.nome as "Nome do Supervisor", e1.rg as "RG do Supervisor"
from empregado e1, empregado e2
where e1.rg = e2.rg_supervisor;
```

que gera o seguinte resultado:

Nome do Supervisor	RG do Supervisor
João Luiz	10101010
Fernando	20202020



A consulta acima é decorrente da seguinte expressão em álgebra relacional:

$$\pi_{\text{nome, rg}} (\text{EMPREGADOS} \mid X \mid \text{tg_t1} = \text{rg_supervisor_t2} \text{ EMPREGADOS})$$


Encontre o nome e a cidade de todos os clientes com uma conta em qualquer agência.

```
Select distinct C.cliente_nome, C.cidade
from CLIENTE C, CONTA S
where C.cliente_cod = S.cliente_cod
```

4.3.1.4. Operações de Conjuntos

A SQL inclui as operações de conjunto **union**, **intersect** e **minus** que operam em relações e correspondem às operações \cup , \cap e $-$ da álgebra relacional. Uma vez que as relações são conjuntos, na união destas, as linhas duplicadas são eliminadas.

Para que uma operação $R \cup S$, $R \cap S$ ou $R - S$ seja válida, precisamos que duas condições sejam cumpridas:

- as relações R e S devem ter o mesmo número de atributos;
- os domínios do i-ésimo atributo de R e do i-ésimo atributo de S devem ser os mesmos.



Nem todos os interpretadores SQL suportam todas as operações de conjunto. Embora a operação union seja relativamente comum, são raros os que suportam intersect ou minus.



Mostrar o nome dos clientes que possuem conta, empréstimo ou ambos na agência de código '051':

Empréstimo na agência '051':

```
Select distinct cliente_nome  
From CLIENTE, EMPRESTIMO  
Where CLIENTE.cliente_cod= EMPRESTIMO.cliente_cod and  
agencia_cod = '051'
```

Conta na agência '051':

```
Select distinct cliente_nome  
From CLIENTE, CONTA  
Where CLIENTE.cliente_cod= CONTA.cliente_cod and  
CONTA.agencia_cod = '051'
```

Fazendo a união dos dois:

```
(Select distinct cliente_nome  
From CLIENTE, EMPRESTIMO  
Where CLIENTE.cliente_cod= EMPRESTIMO.cliente_cod and  
agencia_cod = '051' )  
Union  
(Select distinct cliente_nome  
From CLIENTE, CONTA  
Where CLIENTE.cliente_cod= CONTA.cliente_cod and  
CONTA.agencia_cod = '051' )
```




Achar todos os clientes que possuam uma conta e um empréstimo na agência de código '051'.

```
(Select distinct cliente_nome
From CLIENTE, EMPRESTIMO
Where CLIENTE.cliente_cod= EMPRESTIMO.cliente_cod and
agencia_cod = '051' )
intersect
(Select distinct cliente_nome
From CLIENTE, CONTA
Where CLIENTE.cliente_cod= CONTA.cliente_cod and
CONTA.agencia_cod = '051' )
```



Achar todos os clientes que possuem uma conta, mas não possuem um empréstimo na agência de código '051'.

```
(Select distinct cliente_nome
From CLIENTE, EMPRESTIMO
Where CLIENTE.cliente_cod=Emprestimos.cliente_cod and
EMPRESTIMO.agencia_cod = '051' )
minus
(Select distinct cliente_nome
From CLIENTE, CONTA
Where CLIENTE.cliente_cod= CONTA.cliente_cod and
CONTA.agencia_cod = '051' )
```

4.3.1.5. Ordenando a Exibição de Tuplas (ORDER BY)

A cláusula **order by** ocasiona o aparecimento de tuplas no resultado de uma consulta em uma ordem determinada.



Para listar em ordem alfabética todos os clientes do banco, fazemos:

```
Select distinct cliente_nome
From CLIENTE
Order by cliente_nome
```

Como padrão, SQL lista tuplas na ordem ascendente. Para especificar a ordem de classificação, podemos especificar **asc** para ordem ascendente e **desc** para descendente.



Podemos ordenar uma relação por mais de um elemento. Como se segue:

Select *

From EMPRESTIMO

Order by quantia **desc**, agencia_cod **asc**

4.3.1.6. Membros de Conjuntos

O conectivo **in** testa os membros de conjunto, em que o conjunto é uma coleção de valores produzidos por uma cláusula **select**. Da mesma forma, pode ser usada a expressão **not in**.



Selecione todas as agências com código 1, 2 ou 3.

select *

from agencia

where agencia_cod **in** (1,2,3)



Selecione o nome de todos os funcionários que trabalham em projetos localizados em Rio Claro.

select e1.nome, e1.rg, e1.depto

from empregado e1, empregado_projeto e2

where e1.rg = e2.rg_empregado **and**

e2.numero_projeto **in**

(**select** numero

from projeto

where localizacao = 'Rio Claro');



Encontre todos os clientes que possuem uma conta e um empréstimo na agência 'Princesa Isabel'.

Select distinct cliente_nome

From CLIENTE

Where CLIENTE.cliente_cod **in**

```
(select cliente_cod
from CONTA, AGENCIA
where CONTA.agencia_cod = AGENCIA.agencia_cod and
agencia_nome = 'Princesa Isabel')
and CLIENTE.cliente_cod in
(select cliente_cod
from EMPRESTIMO, AGENCIA
where EMPRESTIMO.agencia_cod= AGENCIA.agencia_cod and
agencia_nome = 'Princesa Isabel')
```



Encontre todas as agências que possuem ativos maiores que alguma agência de Vitória.

```
select distinct t.agencia_nome
from AGENCIA t, AGENCIA s
where t.ativos > s.ativos and s.cidade = 'Vitória'
```



Uma vez que isto é uma comparação “maior que”, não podemos escrever a expressão usando a construção **in**.



A SQL oferece o operador **some** (equivalente ao operador **any**), usado para construir a consulta anterior. São aceitos pela linguagem: **>some**, **<some**, **>=some**, **<=some**, **=some**

```
select agencia_nome
from AGENCIA
where ativos > some
(select ativos
from AGENCIA
where agencia_cidade = 'Vitória')
```

Como o operador **some**, o operador **all** pode ser usado como: **>all**, **<all**, **>=all**, **<=all**, **=all** e **<>all**. A construção **> all** corresponde à frase “maior que todos”.



Encontrar as agências que possuem ativos maiores do que todas as agências de Vitória.

```
select agencia_nome
from AGENCIA
where ativos > all
```

```
(select ativos  
from AGENCIA  
where agencia_cidade = 'Vitória')
```

A SQL possui um recurso para testar se uma subconsulta tem alguma tupla em seus resultados. A construção **exists** retorna **true** se o argumento subconsulta está não-vazio. Podemos usar também a expressão **not exists**.



No esquema EMPRESA, liste o nome dos gerentes que têm ao menos um dependente.

```
Select nome  
from EMPREGADO  
where exists  
(select *  
from DEPENDENTES  
where DEPENDENTES.rg_responsavel = EMPREGADO.rg)  
and exists  
(select *  
from DEPARTAMENTO  
where DEPARTAMENTO.rg_gerente = EMPREGADO.rg)
```



Usando a construção **exists**, encontre todos os clientes que possuem uma conta e um empréstimo na agência 'Princesa Isabel'.

```
Select cliente_nome  
from CLIENTE  
where exists  
(select *  
from CONTA, AGENCIA  
where CONTA.cliente_cod= CLIENTE.cliente_cod and  
AGENCIA.agencia_cod = CONTA.agencia_cod and  
agencia_nome = 'Princesa Isabel')  
and exists  
(select *  
from EMPRESTIMO, AGENCIA  
where EMPRESTIMO.cliente_cod= CLIENTE.cliente_cod and  
AGENCIA.agencia_cod = EMPRESTIMO.agencia_cod and
```

agencia_nome = 'Princesa Isabel')

4.3.1.7. Funções Agregadas

A SQL oferece a habilidade para computar funções em grupos de tuplas usando a cláusula **group by**. O(s) atributo(s) dado(s) na cláusula group by é (são) usado(s) para formar grupos. Tuplas com o mesmo valor em todos os atributos na cláusula group by são colocados em um grupo. A SQL inclui funções para computar:

Média: **avg** Mínimo: **min** Máximo: **max** Soma: **sum** Contar: **count**



Encontre o saldo médio de conta em cada agência.

```
Select agencia_nome, avg(saldo)
From CONTA, AGENCIA
where CONTA.agencia_cod = AGENCIA.agencia_cod
Group by agencia_nome
```



Encontre o número de depositantes de cada agência.

```
Select agencia_nome, count(distinct cliente_nome)
From CONTA, AGENCIA
where CONTA.agencia_cod = AGENCIA.agencia_cod
Group by agencia_nome
```



Note que, nesta última consulta, é importante a existência da cláusula distinct, pois um cliente pode ter mais de uma conta em uma agência e deverá ser contado uma única vez.



Encontre o maior saldo de cada agência.

```
Select agencia_nome, max(saldo)
From CONTA, AGENCIA
Where CONTA.agencia_cod= AGENCIA.agencias_cod
Group by agencia_nome
```

Às vezes, é útil definir uma condição que se aplique a grupos em vez de tuplas. Por exemplo, poderíamos estar interessados apenas em agências nas quais a média dos saldos é maior que 1200. Essa condição será aplicada a cada grupo e não às tuplas simples e é definida por meio da cláusula **having**.



Expressamos esta consulta em SQL assim:

```
Select agencia_nome, avg(saldo)
From CONTA, AGENCIA
Where CONTA.agencia_cod= AGENCIA.agencias_cod
Group by agencia_nome Having avg(saldo)>1200
```

Às vezes, desejamos tratar a relação inteira como um grupo simples. Nesses casos, não usamos a cláusula **group by**.



Encontre a média de saldos de todas as contas.

```
Select avg(saldo)
From CONTA
```

4.3.1.8. Modificando o Banco de Dados

a) Remoção: Podemos remover somente tuplas inteiras, não podemos remover valores apenas em atributos particulares.

Sintaxe:

Delete R

Where P

em que R representa uma relação e P um predicado.

Note que o comando **delete** opera em apenas uma relação. O predicado da cláusula **where** pode ser tão complexo como o predicado **where** do comando **select**.



Remover todas as tuplas de empréstimo.

```
delete EMPRESTIMO
```



Remover todos os registros da conta de 'João'.

```
delete CONTA
where cliente_cod in
```

```
(select cliente_cod  
from CLIENTE  
where cliente_nome = 'João')
```



Remover todos os empréstimos com números entre 1300 e 1500.

```
delete EMPRESTIMO  
where emprestimo_numero between 1300 and 1500
```



Remova todas as contas de agências localizadas em 'Vitória'.

```
Delete CONTA  
where agencia_cod in  
(select agencia_cod  
from AGENCIA  
where agencia_cidade='Vitoria')
```

b) Inserção: Para inserir um dado em uma relação, especificamos uma tupla para ser inserida ou escrevemos uma consulta cujo resultado seja um conjunto de tuplas a serem inseridas. Os valores dos atributos para tuplas inseridas precisam necessariamente ser membros do mesmo domínio do atributo.



Inserir uma nova conta para João (código = 1), número 9000, na agência de código=2 cujo valor seja 1200.

```
insert into CONTA  
values (2,9000,1,1200)
```

Na inserção acima, é considerada a ordem na qual os atributos correspondentes estão listados no esquema relação.



Caso o usuário não se lembre da ordem desses atributos, pode fazer o mesmo comando da seguinte forma:

```
insert into CONTA (agencia_cod, conta_numero, cliente_cod, saldo)  
values (2,9000,1,1200)
```

Podemos querer também inserir tuplas baseadas no resultado de uma consulta.



Inserir todos os clientes que possuam empréstimos na agência 'Princesa Isabel' na relação CONTA com um saldo de 200. O número da nova conta é o número do empréstimo * 10000.

insert into CONTA (agencia_cod, conta_numero, cliente_cod, saldo)

select AGENCIA.agencia_cod, emprestimo_numero*10000, cliente_cod, 200

from EMPRESTIMO, AGENCIA

where EMPRESTIMO.agencia_cod= AGENCIA.agencia_cod **and** agencia_nome = 'Princesa Isabel'

c) Atualizações: Em certas situações, podemos desejar mudar um valor em uma tupla sem mudar todos os valores na tupla. Para isso, o comando **update** pode ser usado.



Suponha que esteja sendo feito o pagamento de juros e que, em todos os saldos, sejam acrescentados em 5%. Escrevemos:

update CONTA

set saldo = saldo * 1,05



Suponha que todas as contas com saldo superiores a 10000 recebam aumento de 6% e as demais de 5%.

Update CONTA

set saldo = saldo * 1,06

where saldo >10000

Update CONTA

set saldo = saldo * 1,05

where saldo<=10000

A cláusula **where** pode conter uma série de comandos select aninhados.



Considere, por exemplo, que todas as contas de pessoas que possuem empréstimos no banco terão acréscimo de 1%.

Update CONTA

set saldo = saldo * 1,01

where cliente_cod in

(**select** cliente_cod

from EMPRESTIMO)



No esquema EMPRESA, atualize o salário de todos os empregados que trabalham no departamento 2 para R\$ 3.000,00.

update empregado

set salario = 3000

where depto = 2;



No esquema BANCO, atualize o valor dos ativos. Os ativos são os valores dos saldos das contas da agência.

update agencia

set ativos =

(**select sum**(saldo)

from conta

where conta.agencia_cod = agencia.agencia_cod)

d) Valores Nulos: É possível dar valores a apenas alguns atributos do esquema para tuplas inseridas em uma dada relação. Os atributos restantes são designados como nulos.



Considere a requisição:

insert into CLIENTE (cliente_cod, cliente_nome, rua, cidade)

values (123,'Andrea',null,null)

A palavra chave **null** pode ser usada em um predicado para testar se um valor é nulo. Assim, para achar todos os clientes que possuem valores nulos para rua, escrevemos:

select distinct cliente_nome

from CLIENTE

where rua **is null**

O predicado **is not null** testa a ausência de um valor nulo.

4.3.1.9. Definição de Dados

O conjunto de relações de um Banco de Dados precisa ser especificado ao sistema por meio de uma linguagem de definição de dados - DDL. A SQL DDL permite a especificação não apenas de um conjunto de relações, mas também de informações sobre cada relação, incluindo:

- esquema para cada relação;
- domínio de valores associados a cada atributo;
- conjunto de índices a ser mantido para cada relação;
- restrições de integridade;
- a estrutura física de armazenamento de cada relação no disco.

Uma relação SQL é definida usando o comando **create table**. A forma geral do comando **create table** então é:

```
create table <nome_tabela> (  
<nome_coluna1> <tipo_coluna1> <NOT NULL>,  
<nome_coluna2> <tipo_coluna2> <NOT NULL>,  
...  
<nome_colunan> <tipo_colunan> <NOT NULL>);
```

A restrição **not null** indica que o atributo deve ser obrigatoriamente preenchido; se não for especificado, então o *default* é que o atributo possa assumir o valor nulo.



Para criar a tabela EMPREGADO do esquema EMPRESA, teríamos o seguinte comando:

```
create table EMPREGADO  
(nome char (30) NOT NULL,  
rg integer NOT NULL,  
cic integer,  
depto integer NOT NULL,  
rg_supervisor integer,  
salario, decimal (7,2) NOT NULL)
```

O comando **create table** inclui opções para especificar certas restrições de integridade, conforme veremos. A relação criada acima está inicialmente vazia. O comando insert poderá ser usado para carregar os dados para uma relação.

Para remover uma tabela de banco de dados SQL, usamos o comando **drop table**. O comando **drop table** remove todas as informações sobre a relação retirada do banco de dados. A forma geral para o comando **drop table** é: **drop table** <nome_tabela>;



Para eliminar a tabela EMPREGADO do esquema EMPRESA, teríamos o seguinte comando:

drop table EMPREGADOS;

Observe que neste caso, a chave da tabela EMPREGADOS, (rg), é utilizada como chave estrangeira ou como chave primária composta em diversas tabelas que devem ser devidamente corrigidas. Esse processo não é assim tão simples, pois, como vemos nesse caso, a exclusão da tabela EMPREGADO implica na alteração do projeto físico de diversas tabelas. Isso acaba implicando na construção de uma nova base de dados.

O comando **alter table** é usado para alterar a estrutura de uma relação existente. Permite que o usuário faça a inclusão de novos atributos em uma tabela. A forma geral para o comando **alter table** é a seguinte: **alter table** <tabela> <add,drop,modify> <coluna> <tipo_coluna>;

em que *add* adiciona uma coluna; *drop* remove uma coluna; *modify* modifica algo em uma tabela. No caso do comando **alter table**, a restrição NOT NULL não é permitida, pois assim que se insere um novo atributo na tabela, o valor para ele em todas as tuplas da tabela receberão o valor NULL.

Não é permitido eliminar algum atributo de uma relação já definida. Assim caso você desejar eliminar uma chave primária devidamente referenciada em outra tabela como chave estrangeira, ao invés de obter a eliminação do campo, obterá apenas um erro.

4.3.2.0. Visões (VIEWS)

Uma view em SQL é uma tabela que é derivada de outras tabelas ou de outras views. Uma view não necessariamente existe em forma física; é considerada uma tabela virtual (em contraste com as tabelas cujas tuplas são efetivamente armazenadas no banco de dados).

Uma view é definida usando o comando **create view**. Para definir uma visão, precisamos dar a ela um nome e definir a consulta que a processa.

A forma geral é:

create view <nomevisao> **as** <expressão de consulta>

em que <expressão de consulta> é qualquer consulta SQL válida.



Considere a visão que consiste em nomes de agências e de clientes.

Create view TOTOS_CLIENTES **as**

(**select** agencia_nome,cliente_nome

from CLIENTE, CONTA, AGENCIA

where CLIENTE.cliente_cod = CONTA.cliente_cod and

```
CONTA.agencia_cod = AGENCIA.agencia_cod)
```

union

```
(select agencia_nome,cliente_nome  
from CLIENTE, EMPRESTIMO, AGENCIA  
where CLIENTE.cliente_cod = EMPRESTIMO.cliente_cod and  
EMPRESTIMO.agencia_cod = AGENCIA.agencia_cod)
```

Nomes de visões podem aparecer em qualquer lugar onde nome de relação (tabela) possa aparecer. Usando a visão TOTOS_CLIENTES, podemos achar todos os clientes da agência 'Princesa Isabel', escrevendo:

```
select cliente_nome  
from TOTOS_CLIENTES  
where agencia_nome = 'Princesa Isabel'
```

Uma modificação é permitida por meio de uma visão apenas se a visão em questão está definida em termos de uma relação do atual banco de dados relacional.



Selecionando tuplas de empréstimos.

Create view emprestimo_info as

```
(select agencia_cod, emprestimo_numero, cliente_cod  
from EMPRESTIMO)
```

Uma vez que a SQL permite a um nome de visão aparecer em qualquer lugar em que um nome de relação aparece, podemos escrever:

```
insert into emprestimo_info  
values (1,40,7)
```

Essa inserção é representada por uma inserção na relação EMPRESTIMO, uma vez que é a relação a partir do qual a visão emprestimo_info foi construída. Devemos, entretanto, ter algum valor para *quantia*. Esse valor é um valor nulo. Assim, o **insert** acima resulta na inserção da tupla: (1,40,7,null) na relação EMPRESTIMO.

Da mesma forma, poderíamos usar os comandos *update* e *delete*.

Para apagar uma visão, usamos o comando

drop view <nomevisão>



Apagar a visão emprestimo_info.

drop view emprestimo_info

5. ARQUITETURA DE SISTEMAS DE BANCO DE DADOS

5.1. Modelos de Dados

De acordo com Korth e Silberschatz (1999), um “modelo de dados” é um conjunto de conceitos que podem ser utilizados para descrever a estrutura “lógica” e “física” de um banco de dados. Por “estrutura”, podemos compreender o tipo dos dados, os relacionamentos e as restrições que podem recair sobre os dados.

Os modelos de dados podem ser basicamente de dois tipos:

- **Alto Nível:** ou modelo de dados conceitual, que fornece uma visão mais próxima do modo como os usuários visualizam os dados realmente.
- **Baixo Nível:** ou modelo de dados físico, que fornece uma visão mais detalhada do modo como os dados estão realmente armazenados no computador.

5.2. Esquemas e Instâncias

Em qualquer modelo de dados utilizado, é importante distinguir a “descrição” do banco de dados do “banco de dados” por si próprio. A **descrição de um banco de dados** é chamada de “**esquema de um banco de dados**” e é especificada durante o projeto do banco de dados.

Geralmente, poucas mudanças ocorrem no esquema do banco de dados. Os **dados armazenados em um banco de dados em um determinado instante do tempo** formam um **conjunto chamado de “instância do banco de dados”**. A instância altera toda vez que uma alteração no banco de dados é feita.

O SGBD é responsável por garantir que toda instância do banco de dados satisfaça ao esquema do banco de dados, respeitando sua estrutura e suas restrições. O

esquema de um banco de dados também pode ser chamado de “**intensão**” de um banco de dados e a **instância** de “**extensão**” de um banco de dados.

5.3. A Arquitetura Três Esquemas

A principal meta da arquitetura “**três esquemas**” é **separar as aplicações do usuário do banco de dados físico**. Os esquemas podem ser definidos como:

- **Nível Interno:** ou esquema interno, o qual descreve a estrutura de armazenamento físico do banco de dados, utiliza um modelo de dados e descreve detalhadamente os dados armazenados e os caminhos de acesso ao banco de dados.
- **Nível Conceitual:** ou esquema conceitual, o qual descreve a estrutura do banco de dados como um todo, é uma descrição global do banco de dados, que não fornece detalhes do modo como os dados estão fisicamente armazenados.
- **Nível Externo:** ou esquema de visão, o qual descreve as visões do banco de dados para um grupo de usuários, cada visão descreve quais porções do banco de dados um grupo de usuários terá acesso.

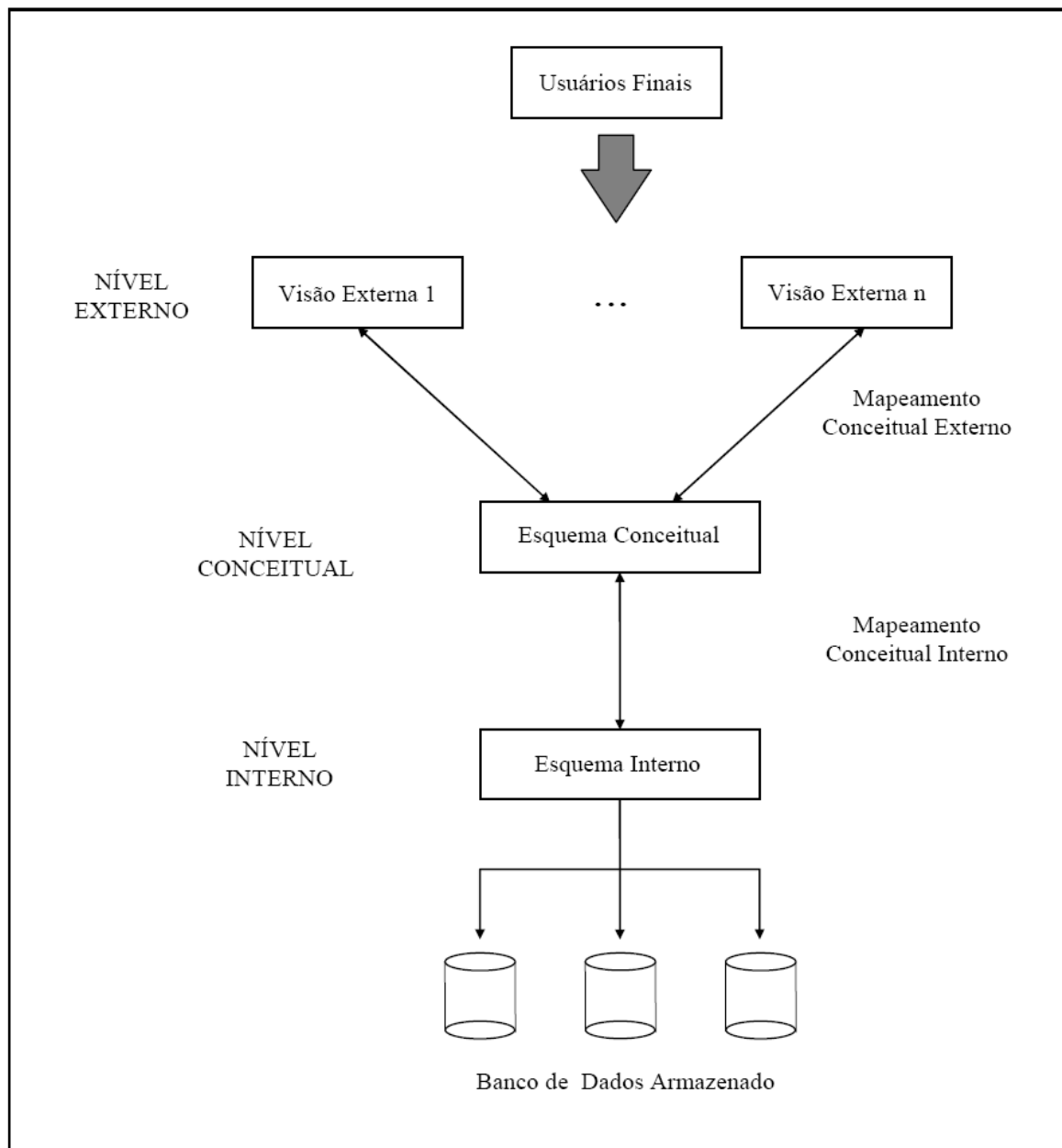


Figura 23 – Arquitetura Três Esquemas

5.4. Independência de Dados

A “**independência de dados**” pode ser definida como a **capacidade de se alterar um esquema em um nível em um banco de dados sem ter que alterar um nível superior**.

Existem dois tipos de independência de dados:

- **Independência de Dados Lógica:** é a capacidade de alterar o esquema conceitual sem ter que alterar o esquema externo ou as aplicações do usuário.
- **Independência de Dados Física:** é a capacidade de alterar o esquema interno sem ter que alterar o esquema conceitual, o esquema externo ou as aplicações do usuário.

5.5. As Linguagens para Manipulação de Dados

Para a definição dos esquemas conceitual e interno, pode-se utilizar uma linguagem chamada DDL (Data Definition Language - Linguagem de Definição de Dados). O SGBD possui um compilador DDL que permite a execução das declarações para identificar as descrições dos esquemas e para armazená-las no catálogo do SGBD. A DDL é utilizada em SGBDs nos quais a separação entre os níveis interno e conceitual não é muito clara.

Em um SGBD em que a separação entre os níveis conceitual e interno é bem clara, é utilizado uma outra linguagem, a SDL (Storage Definition Language - Linguagem de Definição de Armazenamento) para a especificação do esquema interno. A especificação do esquema conceitual fica por conta da DDL.

Em um SGBD que utiliza a arquitetura três esquemas, é necessária a utilização de mais uma linguagem para a definição de visões, a VDL (Vision Definition Language - Linguagem de Definição de Visões).

Uma vez que o esquema esteja compilado e o banco de dados esteja populado, usa-se uma linguagem para fazer a manipulação dos dados, a DML (Data Manipulation Language - Linguagem de Manipulação de Dados).

5.6. Os Módulos Componentes de um SGBD

Um SGBD é um sistema complexo, formado por um conjunto muito grande de módulos.

A Figura abaixo mostra um esquema da estrutura de funcionamento de um SGBD.

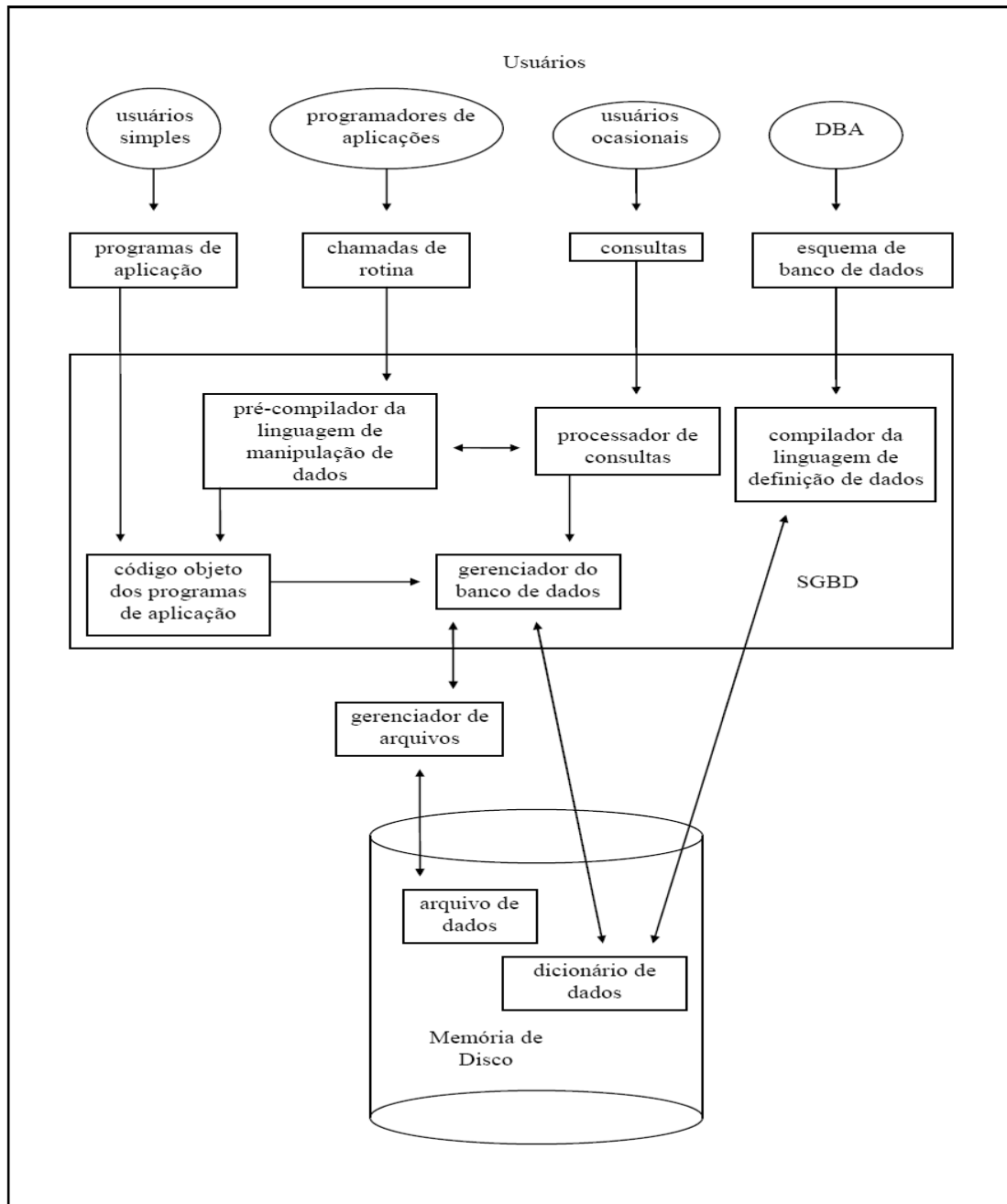


Figura 24 - Estrutura de um Sistema Gerenciador de Banco de Dados

5.7. Classificação dos SGBDs

O principal critério para se classificar um SGBD é o modelo de dados no qual é baseado. A grande maioria dos SGBDs contemporâneos são baseados no modelo relacional, alguns em modelos conceituais e alguns em modelos orientados a objetos. Outras classificações são:

- **Usuários:** um SGBD pode ser mono-usuário, comumente utilizado em computadores pessoais ou multi-usuários, utilizado em estações de trabalho, mini-computadores e máquinas de grande porte;
- **Localização:** um SGBD pode ser localizado ou distribuído; se ele for localizado, então todos os dados estarão em uma máquina (ou em um único disco) ou distribuído, onde os dados estarão distribuídos por diversas máquinas (ou diversos discos);
- **Ambiente:** ambiente homogêneo é o ambiente composto por um único SGBD, e um ambiente heterogêneo é o ambiente compostos por diferentes SGBDs.

5.8 Profissionais e Atividades Envolvidas em um SGBD

Conforme Takai, Italiano e Ferreira (2005):

- **Administrador da Base de Dados:** em qualquer organização onde muitas pessoas compartilham muitos recursos, existe a necessidade de um administrador chefe para supervisionar e gerenciar esses recursos. Num ambiente de base de dados, o recurso primário é a própria base de dados, e os recursos secundários são o próprio SGBD e softwares relacionados. A administração desses recursos é de responsabilidade do DBA (*"Database Administrator"*). O DBA é responsável por autorizar acesso à base de dados e coordenar e monitorar seu uso. O DBA é

responsável por problemas, tais como, quebra de segurança ou baixo desempenho. Em grandes organizações, o DBA é auxiliado por técnicos.

- **Projetistas da Base de Dados:** os projetistas de base de dados têm a responsabilidade de identificar os dados a serem armazenados na Base de Dados e escolher estruturas apropriadas para representar e armazenar tais dados. Essas tarefas são geralmente executadas antes que a base de dados seja utilizada. É responsabilidade desses projetistas obter os requisitos necessários dos futuros usuários da base. Tipicamente, os projetistas interagem com cada grupo de usuários em potencial e definem visões da base de dados para adequar os requisitos e processamentos de cada grupo. Essas visões são então analisadas e, posteriormente, integradas para que, ao final, o projeto da base de dados possa ser capaz de dar subsídio aos requisitos de todos os grupos de usuários.

- **Analistas de Sistemas e Programadores de Aplicação:**

- ⇒ Analistas de sistemas: determinam os requisitos (especificações) de usuários finais, especialmente dos usuários comuns, e desenvolvem especificações das transações para atender a estes requisitos;

- ⇒ Programadores de aplicações: implementam essas especificações produzindo programas e, então, testam, depuram, documentam e mantêm esses programas.

- ⇒ Analistas e programadores: devem estar familiarizados com todas as capacidades fornecidas pelo SGBD para desempenhar essas tarefas.

- **Usuários Finais:** existem profissionais que precisam ter acesso à base de dados para consultar, modificar e gerar relatórios. A base de dados existe para esses usuários. Existem algumas categorias de usuários finais:

⇒ Usuários ocasionais: ocasionalmente fazem acesso à base de dados, mas podem necessitar de diferentes informações a cada vez que fazem acesso. Eles podem usar uma linguagem de consulta sofisticada para especificar suas requisições e são, tipicamente, gerentes de médio ou alto-nível.

⇒ Usuários comuns ou paramétricos: estes usuários realizam operações padrões de consultas e atualizações, chamadas “Transações Permitidas”, que foram cuidadosamente programadas e testadas. Eles constantemente realizam recuperações e modificações na base de dados.

⇒ Usuários sofisticados: incluem engenheiros, analistas de negócios e outros que procuraram familiarizar-se com as facilidades de um SGBD para atender aos seus complexos requisitos.

- **Profissionais de Apoio:**

- ⇒ Projetistas e Implementadores de SGBD
- ⇒ Desenvolvedores de Ferramentas
- ⇒ Operadores de Manutenção

6. SEGURANÇA E MECANISMO DE PROTEÇÃO

Uma das maiores preocupações de implementação de um SGBD diz respeito à segurança e autorização do acesso aos dados. Os dados armazenados no banco de dados precisam ser protegidos de acessos não autorizados, destruição ou alteração insidiosa e introdução acidental de inconsistência



6.1. Segurança e Violação de Integridade

Segundo Elmasri (2000), o mau uso do banco de dados pode ser classificado como intencional (insidioso) ou acidental. A perda acidental de consistência de dados pode ser consequência de:

- Quedas durante o processamento de transações.
- Anomalias causadas por acesso ao banco de dados.
- Anomalias causadas pela distribuição de dados pelos diversos computadores.
- Erros lógicos que violam as regras impostas para que as transações preservem as restrições de consistência do banco de dados.

É mais fácil a proteção contra perda acidental da consistência dos dados do que as proteger contra o acesso insidioso ao banco de dados. Dentre as formas de acesso insidioso estão as seguintes:

- Leitura não autorizada de dados (roubo de informação)
- Modificação não autorizada de dados
- Destruição não autorizada de dados

Para proteger o banco de dados, devemos tomar medidas de segurança sob vários aspectos. Os mecanismos de proteção utilizados são:

- **Discrecionários:** usados para conceder privilégios aos usuários, incluindo direito de acesso a itens específicos (arquivos, registros ou campos) de acordo com um modo especificado (leitura, escrita ou modificações).
- **Obrigatórios:** usados para garantir múltiplos níveis de segurança por meio da classificação dos dados e usuários em várias classes (ou níveis), e posterior implementação de uma política de segurança.

Outros aspectos de segurança envolvidos na tecnologia de Banco de Dados:

- **Controle de acesso ao Banco de Dados:** o mecanismo de segurança de um SGBD deve incluir opções para restringir o acesso para o Banco de Dados como um todo.
- **Segurança nos Bancos de Dados Estatísticos:** alguns arquivos internos do SGBD, usados para gerar informações ou resumos estatísticos sobre os Bancos de Dados não podem ser acessados livremente.
- **Criptografia:** é uma técnica de segurança geralmente utilizada em algum tipo de comunicação de dados, na qual estes são codificados no envio e decodificados no recebimento.



O **DBA** é a autoridade central que administra um SGBD. É o responsável por conceder privilégios de acesso e classificar os dados e usuários de acordo com políticas específicas.

O controle de acesso discrecionário baseado em privilégios é o método típico de controle de acesso implementado em um SGBD. Geralmente implementado por meio de comandos específicos existentes na linguagem adotada pelo SGBD.

Em um SGBD relacional, em geral existem dois níveis de atribuições de privilégios:

1) Nível de conta: aplica-se a cada conta independente do BD considerado.

2) Nível de relação: aplica-se individualmente às relações (ou visões) de um BD, definidas por meio de uma linguagem, por exemplo, SQL. Para controlar a concessão e revogação de privilégios, a cada relação é atribuída uma **conta proprietária**, que geralmente é a conta por meio da qual a relação foi criada. Ao proprietário de uma relação são dados todos os privilégios sobre ela.

Em SQL, o DBA pode atribuir um proprietário a todo esquema utilizando o comando CREATE SCHEMA. A propagação de privilégios pode ser feita com ou sem a GRANT OPTION.



Os tipos de privilégios que podem ser concedidos a uma relação são:

- Seleção (consulta)
- Modificação (alteração, remoção e inserção)

6.2. Autorizações



Um usuário pode ter diferentes formas de autorizações sobre o Banco de Dados:

- **Autorização de leitura – (read):** permite leitura, mas não a modificação dos dados;
- **Autorização de inserção – (insert):** permite a inserção de novos dados, mas não a modificação de dados existentes;
- **Autorização de atualização – (update):** permite modificação, mas não a remoção de dados;
- **Autorização de eliminação – (delete):** permite a eliminação de dados.

6.3. Especificação de segurança em SQL

A linguagem SQL inclui comandos para conceder e revogar privilégios. O conjunto de privilégios depende da versão SQL considerada.

O comando **grant** é usado para conferir autorização. Sua forma básica é:

Grant <lista de autorizações> **on** <nome da relação> **to** <lista de usuários>;

E

Concessão de Select sobre uma relação funcionários, para os usuários U1, U2:

Grant select on funcionarios to U1, U2;

E

A autorização update pode ser conferida a todos os campos da relação ou somente a alguns deles:

Grant update(nome) on funcionarios to U1, U2;

E

O privilégio insert também funciona como o update, sendo que os atributos restantes receberão o seu valor default. A instrução **all privileges** pode ser utilizada como forma abreviada para todos os privilégios permitidos:

Grant all privileges on emprestimo to U1;

E

Por default, não é permitido ao usuário conceder seus privilégios a outros usuários, para que isto aconteça o DBA deve dar esta autorização:

Grant select on emprestimo to U1 with grant option;

E

Revogando privilégios

Revoke all privileges on emprestimo from U1;

Revoke select on emprestimo from U1, U2, U3 cascade;

7. RECUPERAÇÃO

As informações podem ser perdidas de um sistema computacional devido à ocorrência de falhas. Essas falhas variam de falta de energia a erros de software. Um sistema de banco de dados deve garantir propriedades como atomicidade e durabilidade de suas transações mesmo que falhas ocorram.

Para Korth e Silberschatz (1999), um mecanismo de recuperação que restaure o banco de dados para um estado consistente existente antes da falha é, portanto, de extrema importância. Os algoritmos de recuperação em banco de dados são responsáveis por assegurar a consistência do banco de dados e a atomicidade das transações.

Para isso eles devem efetuar ações durante o processamento normal de uma transação garantindo que haja informação suficiente para permitir a recuperação, e ações após a falha, que efetivamente recuperam o conteúdo do banco de dados para um estado consistente, garantindo a atomicidade da transação e a durabilidade.



7.1. Falhas em Bancos de Dados

Podem-se classificar as falhas como:

- **Falha de Transação:** esta falha pode ser causada por dois tipos de erro, erro lógico ou erro de sistema. Erro lógico ocorre quando uma condição interna impede a execução normal da transação, como uma entrada inadequada ou um dado não encontrado. Quando o sistema entra num estado inadequado, impedindo a execução normal de uma transação, diz-se que ocorreu um erro de sistema.

- **Queda do Sistema:** esta falha causa a perda do conteúdo do armazenamento volátil. Causas comuns desta falha são mau funcionamento de hardware ou bug no software de banco de dados.
- **Falha de Disco:** Esta falha leva à perda do conteúdo do disco. Pode ser causada pela quebra do cabeçote ou da falha durante uma operação de transferência de dados. São usadas, para recuperação do sistema após a falha, cópias dos dados em outros discos ou backups de arquivos em meios terciários, como fitas.

7.2. Recuperação e Atomicidade

Ao ocorrer uma falha durante uma transação, soluções do tipo re-executar ou ignorar poderão levar o banco a um estado inconsistente. O problema está em modificar o banco de dados sem que a transação seja de fato efetivada.

Para evitar inconsistências, é necessário que todas, ou nenhuma, alterações relacionadas a uma transação sejam realizadas no banco de dados. Para manter a atomicidade, devem-se mandar todas as alterações para um armazenamento estável, sem modificar o banco de dados.

Isso permite, apesar de falhas, manter um estado consistente dos dados. Abaixo estão descritas duas técnicas para realizar essa gravação, de acordo com SETZER (1998):

7.2.1. Recuperação Utilizando Log

Recuperação baseada em log é a técnica mais utilizada para gravar modificações em um banco de dados. O log documenta as atividades em um banco de dados.

Em especial, o log de atualizações documenta todas as alterações de registros. Os registros do log são gravados antes da modificação no banco, de forma que, caso ocorra uma falha que interrompa uma transação, o banco de dados possa retornar a um estado consistente.

8. NOÇÕES DE BANCOS DE DADOS DISTRIBUÍDOS

8.1 Definições

Um **Sistema de Banco de Dados Distribuído** consiste em um conjunto de sites (máquinas), ligados entre si por meio de algum tipo de rede de comunicações, em que:

- Cada *site* é um *site* completo de sistema de banco de dados.
- Porém, os *sites* atuam juntos, de modo que um usuário em qualquer *site* pode ter acesso a dados em qualquer lugar da rede, como se estes estivessem armazenados no *site* do próprio usuário.

Assim, Oara Date (1990), um banco de dados distribuído é na verdade uma espécie de banco de dados virtual, no qual seus componentes estão fisicamente armazenados em vários bancos de dados reais distintos.

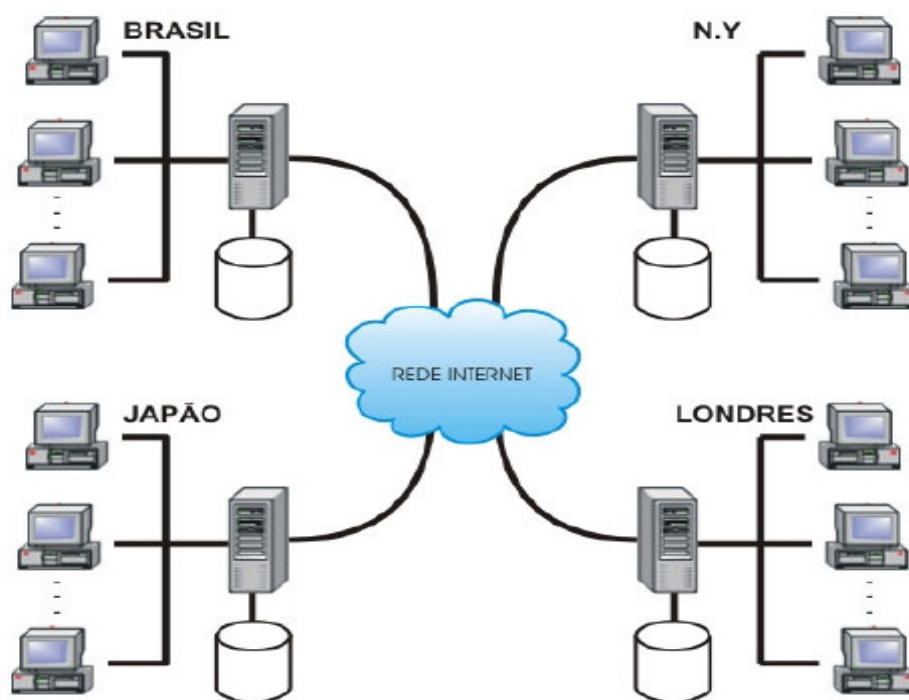


Figura 25 - Comunicação entre banco de dados distribuídos geograficamente

Observe que cada *site* é, ele próprio, do sistema de banco de dados por si mesmo, ou seja, o sistema de banco de dados distribuído pode, portanto, ser considerado como um tipo de parceria entre SGBDs individuais locais nos *sites* locais individuais; um novo componente de software em cada *site* - logicamente uma extensão do SGBD local - fornece as necessárias funções da parceria e é a combinação desse novo componente com o SGBD existente.

A propósito, é comum deduzir que os *sites* estão fisicamente separados - talvez até geograficamente. Na verdade, o foco em sistemas distribuídos se deslocou nos últimos anos; enquanto a maior parte da pesquisa tendia a assumir a distribuição geográfica, a maior parte das primeiras instalações comerciais envolvia a distribuição local.



Conforme Date (1990), vários *sites*, no mesmo edifício, interligados por meio de uma rede local (LAN). Porém, mais recentemente, a enorme proliferação de redes remotas (WAN) reativou o interesse na possibilidade de distribuição geográfica.

8.2 Visão geral

Um *site* pode participar da execução de uma transação global; transações essas que mantêm acesso em diversos *sites*. A execução de transações globais exige comunicação entre os *sites*.

Há muitas razões para construir um banco de dados distribuído, incluindo o compartilhamento de dados, confiabilidade, disponibilidade e rapidez no processamento de consultas. Entretanto, essas vantagens são acompanhadas de muitas desvantagens, como o alto custo de desenvolvimento de software e o alto potencial de bugs. A principal desvantagem dos sistemas de bancos de dados

distribuídos é a complexidade adicional para garantir a coordenação adequada entre os *sites*.

Um sistema de banco de dados múltiplo proporciona um ambiente em que novas aplicações de banco de dados podem manter acesso a dados de diversos bancos de dados preexistentes, localizados em vários ambientes com hardware e software heterogêneos. Segundo Korth e Silbershatz (1999), os sistemas de bancos de dados locais podem empregar modelos lógicos diferentes e linguagens de definição e manipulação de dados distintos; podem ainda diferir em relação aos mecanismos para controle de concorrência e gerenciamento de transações. Um sistema de banco de dados múltiplo cria a ilusão da integração lógica do banco de dados, sem impor uma integração física.

8.3. Funcionamento

Podemos iniciar esse tópico com a seguinte pergunta: por que bancos de dados distribuídos são desejáveis? A resposta para essa pergunta é que normalmente as empresas já são distribuídas, pelo menos logicamente (em divisões, departamentos, grupos de trabalho, etc.) e com grande probabilidade também fisicamente (em fábricas, laboratórios, etc.) - e isso decorre que os dados já estão normalmente distribuídos, porque cada unidade organizacional dentro da empresa, necessariamente, manterá dados que são relevantes para sua própria operação.

O patrimônio total de informações da empresa é desse modo disseminado naquilo que às vezes se costuma chamar de ilhas de informações. Um sistema distribuído fornece as pontes necessárias para interligar essas linhas. Para Korth e Silbershatz (1999), em outras palavras, ele permite que a estrutura do banco de dados reflita a estrutura da empresa - dados locais podem ser mantidos em instalações locais, às quais eles pertencem logicamente – enquanto, ao mesmo tempo, dados remotos estão disponíveis para acesso quando necessário.



Considere este exemplo apresentado por Connaleem (1999): suponha que há apenas dois *sites*, São Paulo e Santa Catarina, e suponha que o sistema é bancário, com dados de contas para as contas de São Paulo armazenados em São Paulo, e dados de contas para contas de Santa Catarina armazenados em Santa Catarina. Então, as vantagens são óbvias: o arranjo distribuído combina eficiência de processamento (os dados são mantidos próximos ao local em que são usados mais freqüentemente) com maior facilidade de acesso (é possível ter acesso a uma conta em São Paulo, a partir de Santa Catarina e vice-versa, por meio da rede de comunicações).

Fazer com que a estrutura do banco de dados reflita a estrutura da empresa é provavelmente (como acabamos de explicar) a principal vantagem de sistemas distribuídos. Contudo, devemos mencionar que também há algumas desvantagens, das quais a maior é o fato de sistemas distribuídos serem complexos, pelo menos do ponto de vista técnico.

8.4. Armazenamento distribuído

Há diversos enfoques para o armazenamento de informações em um banco de dados distribuído de acordo com Date (1990):

- **Replicação:** o sistema mantém réplicas idênticas da relação. Cada réplica é armazenada em diferentes *sites*, resultando na replicação dos dados. A alternativa para a replicação é armazenar somente uma cópia de uma determinada relação *r*.
- **Fragmentação:** a relação é particionada em diversos fragmentos. Cada fragmento é armazenado em um *site* diferente. Podendo se dividir em horizontal, separam-se os registros (linhas) da tabela; e vertical, separa-se as colunas da tabela.
- **Replicação e fragmentação:** a relação é particionada em vários segmentos. Os sistemas mantêm diversas réplicas de cada fragmento.

8.5. Arquitetura

De acordo com Connale (1999), um sistema é um sistema distribuído em que: (a) alguns *sites* podem consultar diversos dados em lugares distintos, (b) o cliente não sabe em hipótese alguma em que sistema de dados ele está, (c) todas as aplicações são executadas no mesmo *site*.

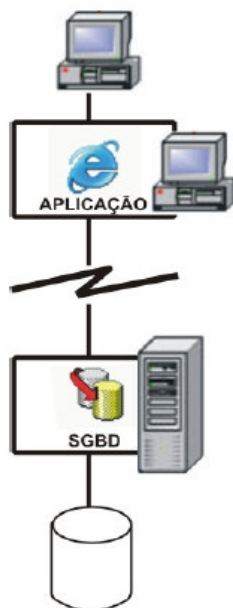


Figura 26 - Um Sistema Web Acessando um Banco de Dados



Lembramos ainda que são possíveis diversas variações sobre o tema básico:

- Vários clientes poderiam compartilhar o mesmo SGBD-D.
- Um único cliente poderia ser capaz de deter acesso a vastos SGBD-D. Esse caso por sua vez se subdivide em dois outros:
 - O cliente está limitado a obter acesso a um único SGBD-D de cada vez, isto é, cada solicitação individual de banco de dados deve ser dirigida a um só SGBD-D; não é possível, dentro de uma única solicitação, combinar dados de dois ou mais SGBD-D.
 - Para o cliente pode aparentar ser um único SGBD-D e o usuário não precisa saber qual SGBD-D armazena quais fragmentos de dados.

8.6. Autonomia local

Os *sites* em um sistema distribuído devem ser autônomos, significando que todas as operações em um determinado *site* são controladas por esse *site*; nenhum *site* X deve depender de algum outro *site* Y para que sua operação seja bem-sucedida.

A autonomia local também implica que dados locais são de propriedades e gerenciamentos locais, com contabilidade local, todos os dados “realmente” pertencem a algum banco de dados local, mesmo que sejam acessíveis a partir de outros *sites* remotos. Assim, questões como segurança, integridade e representação de armazenamento de dados locais permanecem sob controle.

8.7. Independência de localização

A idéia básica da independência de localização (também chamada transparência de localização) é simples: os usuários não devem ser obrigados a saber onde estão fisicamente armazenados os dados, embora devam ser capazes de se comportar - pelo menos de um ponto de vista lógico - como se os dados estivessem todos armazenados em seu próprio *site* local.

A independência de localização é desejável porque simplifica programas e atividades em terminal dos usuários. Em particular, permite que dados migrem de um *site* para outro sem invalidar qualquer desses programas ou atividades. Para Date (1990), Essa capacidade de migração desejável permite que dados sejam deslocados pela rede em resposta a alterações de exigências de desempenho.

8.8. Independência de fragmentação

Um sistema admite fragmentação de dados se uma dada variável de relação armazenada pode ser dividida em pedaços ou fragmentos para fins de armazenamento físico. A fragmentação é desejável por razões de desempenho: os

dados podem ser armazenados no local em que são mais freqüentemente utilizados, de modo que a maior parte das operações seja apenas local, e o tráfego de rede seja reduzido.

E

Considere a variável de relação de empregados EMP, com a amostra de valores apresentada na parte superior da Figura abaixo:

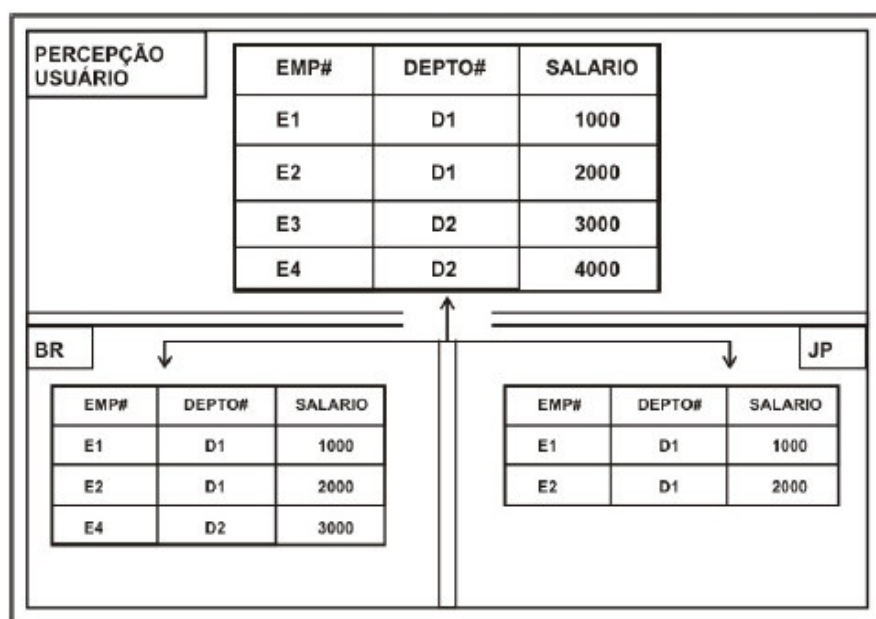


Figura 27 – Um Exemplo de Fragmentação entre Dados de BR e JP e a Percepção do Usuário

Em um sistema que admitisse a fragmentação, poderíamos definir dois fragmentos.

```

FRAGMENT EMP AS N_EMP AT SITE 'BR'
WHERE DEPTO# = 'D1' OR DEPTO# = 'D2',
L_ EMP AT SITE 'JP'
WHERE DEPTO# = 'D2';
    
```

Estamos supondo que: (a) tuplas de empregados são mapeadas no armazenamento físico de modo bastante direto; (b) números de empregados e números de departamentos são apenas string de caracteres, e salários são apenas números; (c) D1 e D3 são departamentos da empresa no Brasil, e D2 é um departamento da empresa no Japão. Em outras palavras, tuplas para empregados no Brasil serão

armazenadas no *site* aqui no Brasil, e tuplas para empregados no Japão serão armazenadas no *site* lá no Japão. Observe os nomes dos fragmentos internos do sistema, N_EMP e L_EMP.

8.9. Independência de replicação

Um sistema admite replicação de dados se uma dada variável de relação armazenada, ou geralmente, um dado fragmento de uma determinada variável de relação armazenada - pode ser representada por muitas cópias ou réplicas distintas, armazenadas em muitos *sites* distintos.

E

```
REPLICATE N EMP AS  
LN_EMP AT SITE 'Brasil'
```

```
REPLICATE L EMP AS  
NL_EMP AT SITE "Japão"
```

BRASIL			JAPÃO		
BR_EMP			JP_EMP		
EMP#	DEPTO#	SALARIO	EMP#	DEPTO#	SALARIO
E1	D1	1000	E2	D2	3000
E2	D2	2000	E3	D2	4000
E5	D3	5000	E4	D3	6000
L_BR_EMP (RÉPLICA)			L_JP_EMP (RÉPLICA)		
EMP#	DEPTO#	SALARIO	EMP#	DEPTO#	SALARIO
E1	D1	1000	E1	D1	4200
E2	D2	2000	E2	D2	5600
E5	D3	5000	E5	D3	7000

Figura 28 - Um Exemplo de Replicação

Observe os nomes internos de réplicas do sistema, NL EMP e LN EMP. A replicação é desejável por pelo menos duas razões. Primeiro, pode significar melhor desempenho (aplicações podem operar sobre cópias locais, em vez de terem de se comunicar com *sites* remotos); segundo, também pode significar melhor disponibilidade (um dado objeto replicado permanece disponível para processamento - pelo menos para acesso - enquanto houver no mínimo uma cópia disponível).

Naturalmente, a maior desvantagem da replicação é que, quando um determinado objeto replicado é atualizado, todas as cópias desse objeto precisam ser atualizadas; o problema da propagação de atualizações. Observamos de passagem que a replicação em um sistema distribuído representa uma aplicação específica da idéia de redundância controlada.

Agora, a replicação, como a fragmentação, deve no caso ideal ser “transparente para o usuário”. Em outras palavras, um sistema que admita replicação de dados também deve admitir independência de replicação (também chamada transparência de replicação), isto é, os usuários devem ser capazes de se comportar, pelo menos de um ponto de vista lógico, como se os dados de fato não fossem replicados de modo algum.

8.10. Gerenciamento

O acesso a diversos itens de dados em um sistema distribuído é normalmente acompanhado de transações que têm de preservar as propriedades. Há dois tipos de transações que devemos considerar. As transações locais, que são aquelas que mantêm acesso e atualizam somente a base de dados local; e as transações globais, que são aquelas que mantêm acesso e atualizam diversas bases de dados locais.

Entretanto, no caso das transações globais, essa tarefa é bem mais complicada, já que diversos *sites* podem participar de sua execução. Conforme Date (1990), uma

falha em um desses *sites* ou uma falha de comunicação entre *sites* pode resultar em erros de processamento.

8.10.1. Tipos de Falhas no Sistema

Um sistema distribuído pode sofrer os mesmos tipos de falhas de um sistema centralizado (por exemplo, erros de software, erros de hardware ou erros em disco). Há, entretanto, tipos adicionais de falhas que precisam ser tratadas em um ambiente distribuído. Os tipos de falhas característicos são:

- Falha em um *site*.
- Perda de mensagens.
- Falha de comunicação.
- Problemas de particionamento da rede.

A perda ou comprometimento das mensagens é sempre uma possibilidade nos sistemas distribuídos. O sistema usa protocolos de controle de transmissão, como TCP/IP, para o tratamento desses erros. Os *sites* de um sistema podem ser conectados fisicamente de diversas formas.

Cada configuração apresenta vantagens e desvantagens. As configurações podem ser comparadas umas com as outras, com base nos seguintes critérios:

- **Custo de instalação.** O custo da ligação física entre os *sites* do sistema.

Custo de comunicação. O custo de tempo e dinheiro para envio de uma mensagem do *site* A para o B.

- **Disponibilidade, relativo ao grau de acesso aos dados, a despeito de falhas de ligação entre ou nos *sites*.** Um nó A para o B corresponde a uma comunicação direta entre dois *sites*. Em uma rede totalmente conectada, cada *site* está diretamente conectado a todos os outros *sites*.

Em uma rede parcialmente conectada, há as ligações diretas entre alguns - mas não entre todos - pares de *sites*. Então, o custo de instalação dessas configurações é

menor que o das redes totalmente conectadas. Entretanto, se dois *sites*, A e B, não estão diretamente conectados, as mensagens entre eles precisam ser roteadas por uma seqüência de ligações. Essa necessidade resulta no crescimento dos custos de comunicação.

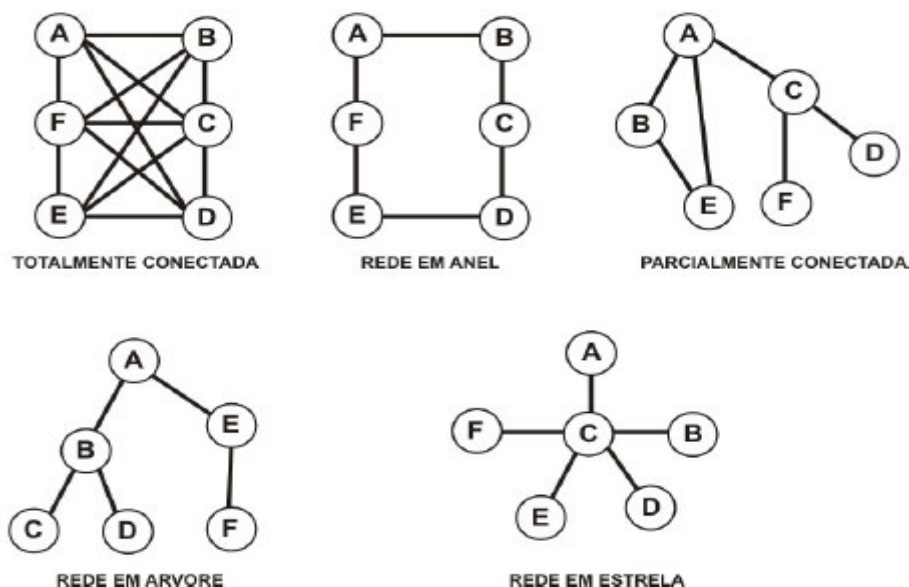


Figura 29 - Topologia de Redes de Computadores

Se uma ligação falha, as mensagens que poderiam ser transmitidas por meio dela deverão ser roteadas novamente. Em alguns casos, é possível encontrar outra rota por meio da rede, assim as mensagens poderão alcançar seus destinos. Em outros casos, uma falha pode ocorrer onde não haja qualquer ligação entre o par de *sites*. Note que, sob essa definição, um subsistema pode ser considerado um único nó.



Os diferentes tipos de redes parcialmente conectados, que foram apresentados, têm características diferentes em relação aos tipos de falhas, instalações e custos de comunicação.

8.11. Robustez

Para um sistema distribuído ser robusto, ele precisa detectar falhas, reconfigurar o sistema para que ele possa continuar funcionando e recuperar a situação original durante o retomo de uma ligação ou de um processador.

As falhas diferentes são tratadas de modos diferentes. Perda de mensagens são retransmitidas. Retransmissão repetida de uma mensagem, sem mensagem de reconhecimento é normalmente sintoma de perda de comunicação. Em geral, a rede tenta encontrar rotas alternativas para uma mensagem. Falhas nessas tentativas sugerem particionamento da rede.

Entretanto, geralmente não é possível ter certeza se houve falha na comunicação entre *sites* ou se houve particionamento da rede. O sistema detecta a falha, mas não consegue identificar o tipo.



Considere este exemplo apresentado por Korth e Silbershatz (1999): suponha que o *site* S1 não consiga se comunicar com o *site* S2. Pode ser que S2 esteja com problemas. Entretanto, pode ser que a ligação entre S1 e S2 não esteja funcionando. Suponha que o *site* S1 tenha identificado uma falha. Ele iniciará um procedimento para reconfiguração do sistema e, então, prosseguirá no modo de operação normal.



- Se há dados replicados no *site* que não estão funcionando, o catálogo deverá ser atualizado para que as consultas não solicitem acesso à cópia desse *site*.
- Se no *site* com problemas houver transações ativas no momento da falha, essas transações deverão ser abortadas. É preferível abortar essas transações rapidamente, já que elas poderão manter bloqueios em dados em *sites* ainda ativos.

- Se o *site* que não está funcionando é um servidor de algum subsistema deverá ser feita uma eleição para determinar o novo servidor.

Já que, em geral, não é possível distinguir entre falhas de rede e falhas nos *sites*, qualquer esquema de reconfiguração precisa ser projetado de modo a trabalhar corretamente caso haja o particionamento da rede.



Em particular, a seguinte situação precisa ser evitada:

- Dois ou mais servidores centrais são eleitos em partições distintas.
- Mais de uma partição atualiza e replica itens de dados.

A reintegração de um *site* ou da comunicação entre pares dentro de um sistema também exige cuidados. Quando um *site* volta a integrar a rede, é preciso que um procedimento seja executado para atualização das tabelas do sistema, de modo a refletir as alterações sofridas enquanto ele esteve fora da rede. Se o *site* possui réplicas de itens de dados, é preciso que ele obtenha os valores atualizados desses itens e que se possa garantir que ele passe a receber todas as atualizações futuras. A reintegração de um *site* é mais complicada do que parece à primeira vista, já que os dados podem sofrer atualizações durante a reintegração do *site* em questão.



Para Date (1990), uma solução simples para esse problema seria manter o sistema temporariamente sem alterações enquanto o *site* é reintegrado. Na maioria das aplicações, entretanto, tal interrupção seria inaceitável. Quando a comunicação entre *sites* é restaurada, pode-se estar juntando partições da rede. Dado que o particionamento da rede limita os tipos de operações possíveis em um ou todos os *sites*.

REFERÊNCIAS

CONNALEM, Jim. **Desenvolvendo aplicações web com UML**. Tradução da 2ª Edição. Editora Campus, 1999.

DATE, C. J., **Introdução a Sistemas de Banco de Dados**. 8ª Edição. Campus, 1990.

ELMASRI, R. NAVATHE, S. B. **Fundamentals of Database Systems**. 3ª Edição. Menlo Park: Addison-Wesley, 2000.

GARCIA-MOLINA, H. , ULLMAN, J. D. e WIDOM, J. **Implementação de Sistemas de Banco de Dados**. Rio de Janeiro: Ed. Campus, 2002.

HEUSER C. A. **Projeto de Banco de Dados**. 3ª Edição. Porto. Alegre: Sagra-Luzzato, 2001.

KORTH, .F. & SILBERSHATZ, A. **Sistemas de banco de dados**. 9ª Edição. São Paulo: Makron Books, 1999.

O'BRIEN J.A. **Sistemas de Informação e as Decisões Gerenciais na Era da Internet**. São Paulo: Saraiva, 2001.

O.K. Takai; I.C.Italiano; J.E. Ferreira. **Apostila de Banco de Introdução a Banco de Dados**. Departamento IME-USP, 2005.

SETZER, V. M.. **Banco de dados**. 3ª Edição. São Paulo: Ed. Edgard Blucher, 1998.

SQL Language - Oracle Reference Manual; Version 7.2.