B A N C O D E D A D O S

Apostila 1

2005

Prof. Paulo Roberto Rodrigues de Souza

❖ INTRODUÇÃO

Esta apostila tem o objetivo de nos orientar durante este período em nossas aulas de Banco de Dados, não tem a pretensão de ser a única fonte de informação para nosso êxito no curso. A intenção de elaborar este material é de possuirmos um referencial e facilitar o estudo do aluno nesta disciplina. Ela engloba materiais de livros diversos, apostilas, pesquisas na Internet e de conhecimento próprio em minha experiência no uso e construção de banco de dados.

Iremos ver durante o curso, entre outros tópicos, uma noção geral sobre a construção de sistemas de banco de dados, construção de projetos lógicos de bancos de dados, modelos para a construção de projetos físicos de banco de dados, controle de dependência de dados, consultas, redundância entre vários outros.

Veremos em nossas aulas práticas o processo de criação de tabelas, relacionamentos, views, stored procedures entre outros recursos importantes.

1 - Conceitos Gerais

A tecnologia aplicada aos métodos de armazenamento de informações vem crescendo e gerando um impacto cada vez maior no uso de computadores, em qualquer área em que os mesmos podem ser aplicados.

Um "banco de dados" pode ser definido como um conjunto de "dados" devidamente relacionados. Por "dados" podemos compreender como "fatos conhecidos" que podem ser armazenados e que possuem um significado implícito. Porém, o significado do termo "banco de dados" pode significar algo mais que a definição acima. Um banco de dados possui as seguintes propriedades:

- um banco de dados é uma coleção lógica coerente de dados com um significado inerente;
- um banco de dados é projetado, construído e populado com dados para um propósito específico; um banco de dados possui um conjunto pré definido de usuários e aplicações;
- um banco de dados representa algum aspecto do mundo real, o qual é chamado de "mini-mundo" ou "mundo real"; qualquer alteração efetuada no mini-mundo é automaticamente refletida no banco de dados.

Um banco de dados pode ser criado e mantido por um conjunto de aplicações desenvolvidas especialmente para esta tarefa ou por um "Sistema Gerenciador de Banco de Dados" (SGBD). Um SGBD permite aos usuários criarem e manipularem bancos de dados de propósito geral. O conjunto formado por um banco de dados mais as aplicações que manipulam o mesmo é chamado de "Sistema de Banco de Dados".

Outra Definição

Um BD também pode ser definido como sendo:

*"Uma **coleção** de dados **operacionais inter-relacionados**. Estes dados são armazenados de forma **independente dos programas** que os utilizam, **servindo assim a múltiplas aplicações** de uma organização."*

Algumas partes desta definição são fundamentais para a compreensão do conceito de BD:

- **Coleção**: agrupamento com repetição;
- **Operacionais**: vitais; estratégicos para a tomada de decisões; permanentes;
- **Inter-relacionados**: um BD mantém um agrupamento de entidades (fatos da realidade em questão) e de relacionamentos entre estas entidades;
- **Independentes dos programas**: dados são mantidos em um meio de armazenamento destinado aos dados da organização e não necessariamente no espaço local do programa de aplicação;
- **Serve à múltiplas aplicações**: dados em um BD podem ser compartilhados por várias aplicações da organização. Cada uma delas define exatamente os dados que deseja manipular.

❖ Abordagem Banco de Dados X Abordagem Processamento Tradicional de Arquivos

- Auto Informação

Uma característica importante da abordagem Banco de Dados é que o SGBD mantém não somente os dados mas também a forma como os mesmos são armazenados, contendo uma descrição completa do banco de dados. Estas informações são armazenadas no catálogo do SGBD, o qual contém informações como a estrutura de cada arquivo, o tipo e o formato de armazenamento de cada tipo de dado, restrições, etc. A informação armazenada no catálogo é chamada de "Meta Dados". No processamento tradicional de arquivos, o programa que irá manipular os dados deve conter este tipo de informação, ficando limitado a manipular as informações que o mesmo conhece. Utilizando a abordagem banco de dados, a aplicação pode manipular diversas bases de dados diferentes.

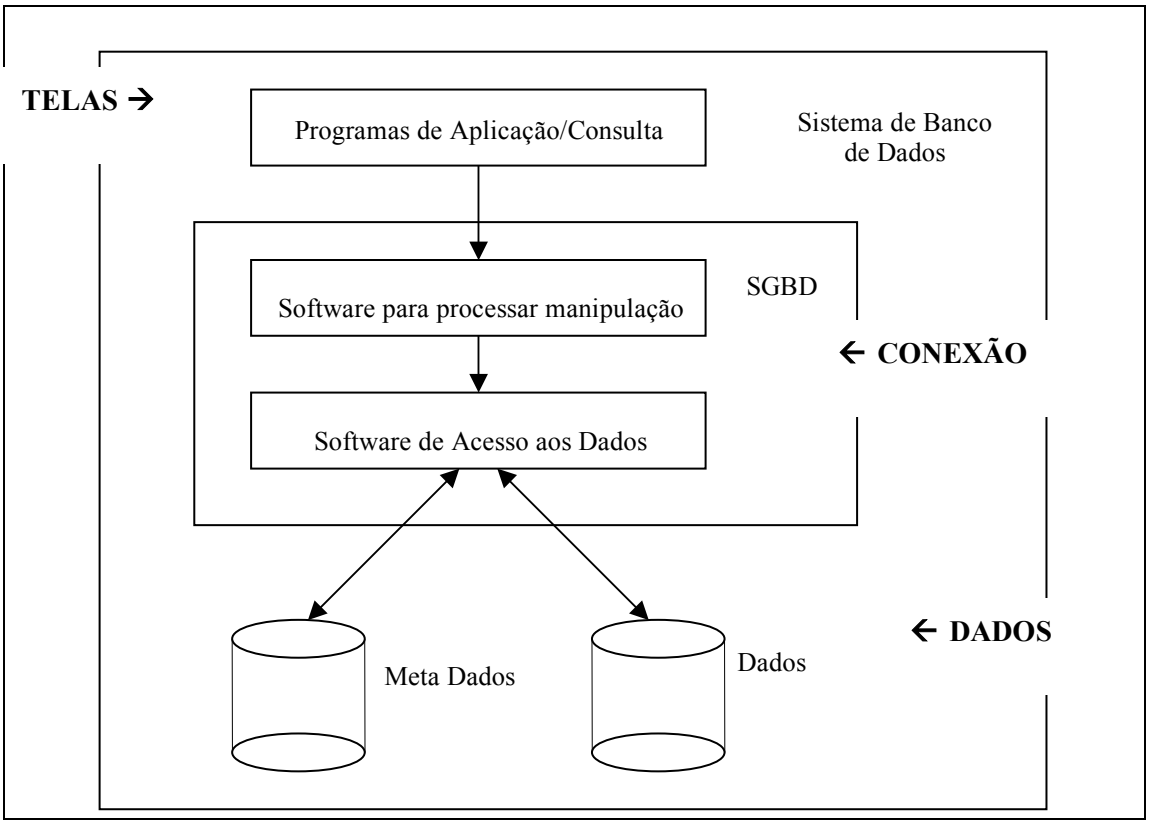
- Separação entre Programas e Dados

No processamento tradicional de arquivos, a estrutura dos dados está incorporada ao programa de acesso. Desta forma, qualquer alteração na estrutura de arquivos implica na alteração no código fonte de todos os programas. Já na abordagem banco de dados, a estrutura é alterada apenas no catálogo, não alterando os programas.

A diferença principal de um banco de dados evoluído de uma estrutura de armazenamento de dados tradicional ou menos evoluída é o fato destas estruturas estarem bem identificadas em um banco de dados. Existem no mercado algumas "bases de dados" que, embora possuam estrutura de armazenamento de dados, não possuem estruturas e ferramentas exigíveis de um banco de dados evoluído e completo. Poderíamos citar: Stored Procedures,

Dados separados de aplicações (consultas, formulários), Triggers, Views, controle de acesso evoluído, dentre outros recursos.

Podemos verificar a característica de "separação de dados e aplicações" bem detalhada abaixo:



Um ambiente de Sistema de Banco de Dados

- Abstração de Dados

O SGBD deve fornecer ao usuário uma "representação conceitual" dos dados, sem fornecer muitos detalhes de como as informações são armazenadas. Um "modelo de dados" é uma abstração de dados que é utileizada para fornecer esta representação conceitual utilizando conceitos lógicos como objetos, suas propriedades e seus relacionamentos. A estrutura detalhada e a organização de cada arquivo são descritas no catálogo.

Descrevendo de forma bem prática, abstração de dados nada mais é do que conseguir se representar e armazenar dados de uma situação real em um banco de dados. Por exemplo, é como conseguir se "montar" um banco de dados de um controle de estoques de uma loja. A abstração é a nossa capacidade de "enxergar" as necessidades deste banco.

- Múltiplas Visões de Dados

Como um conjunto de informações pode ser utilizada por um conjunto diferenciado de usuários, é importante que estes usuários possam ter "visões" diferentes da base de dados. Uma "visão" é definida como um subconjunto de uma base de dados, formando deste modo, um conjunto "virtual" de informações. Também de forma prática, uma visão ou "views" como são

chamadas em bancos evoluídos são consultas especializadas de parte dos dados de um banco de dados.

❖ Usuários

Para um grande banco de dados, existe um grande número de pessoas envolvidas, desde o projeto, uso até manutenção.

- Administrador de Banco de Dados (DBA)

Em um ambiente de banco de dados, o recurso primário é o banco de dados por si só e o recurso secundário o SGBD e os softwares relacionados. A administração destes recursos cabe ao Administrador de Banco de Dados, o qual é responsável pela autorização de acesso ao banco de dados e pela coordenação e monitoração de seu uso.

- Projetista de Banco de Dados

O Projetista de Banco de Dados é responsável pela identificação dos dados que devem ser armazenados no banco de dados, escolhendo a estrutura correta para representar e armazenar dados. Muitas vezes, os projetistas de banco de dados atuam como "staff" do DBA, assumindo outras responsabilidades após a construção do banco de dados. É função do projetista também avaliar as necessidades de cada grupo de usuários para definir as visões que serão necessárias, integrando-as, fazendo com que o banco de dados seja capaz de atender a todas as necessidades dos usuários.

- Usuários Finais

Existem basicamente três categorias de usuários finais que são os usuários finais do banco de dados, fazendo consultas, atualizações e gerando documentos:

- usuários casuais: acessam o banco de dados casualmente, mas que podem necessitar de diferentes informações a cada acesso; utilizam sofisticadas linguagens de consulta para especificar suas necessidades;

- usuários novatos ou paramétricos: utilizam porções pré-definidas do banco de dados, utilizando consultas preestabelecidas que já foram exaustivamente testadas;

- usuários sofisticados: são usuários que estão familiarizados com o SGBD e realizam consultas complexas.

- Analistas de Sistemas e Programadores de Aplicações

Os analistas determinam os requisitos dos usuários finais e desenvolvem especificações para transações que atendam estes requisitos, e os programadores implementam estas especificações como programas, testando, depurando, documentando e dando manutenção no mesmo. É importante que, tanto analistas quanto programadores, estejam a par dos recursos oferecidos pelo SGBD.

❖ Vantagens e desvantagens do uso de um SGBD

- Controle de Redundância

No processamento tradicional de arquivos, cada grupo de usuários deve manter seu próprio conjunto de arquivos e dados. Desta forma, acaba ocorrendo redundâncias que prejudicam o sistema com problemas como:

- toda vez que for necessário atualizar um arquivo de um grupo, então todos os grupos devem ser atualizados para manter a integridade dos dados no ambiente como um todo;

- a redundância desnecessária de dados levam ao armazenamento excessivo de informações, ocupando espaço que poderia estar sendo utilizado com outras informações.

O Controle de redundância é o ato de se evitar repetições de dados de forma desnecessária em um banco de dados. É não repetirmos informações desnecessárias, porém, sem deixarmos de armazenar o que é importante e crucial para o funcionamento do banco e a resolução do problema a ser solucionado.

- Compartilhamento de Dados

Um SGBD multi-usuário deve permitir que múltiplos usuários acessem o banco de dados ao mesmo tempo. Este fator é essencial para que múltiplas aplicações integradas possam acessar o banco.

O SGBD multi-usuário deve manter o controle de concorrência para assegurar que o resultado de atualizações sejam corretos. Um banco de dados multi-usuários deve fornecer recursos para a construção de múltiplas visões.

- Restrição a Acesso não Autorizado

Um SGBD deve fornece um subsistema de autorização e segurança, o qual é utilizado pelo DBA para criar "contas" e especificar as restrições destas contas; o controle de restrições se aplica tanto ao acesso aos dados quanto ao uso de softwares inerentes ao SGBD. A restrição de acesso não autorizado é o ato de se limitar o acesso dos usuários ao banco ou a parte dele. Bancos de dados evoluídos devem conter recursos para se efetuar estes controles de forma fácil, prática e segura.

- Representação de Relacionamentos Complexos entre Dados

Um banco de dados pode incluir uma variedade de dados que estão interrelacionados de várias formas. Um SGBD deve fornecer recursos para se representar uma grande variedade de relacionamentos entre os dados, bem como, recuperar e atualizar os dados de maneira prática e eficiente. Os relacionamentos entre dados atualmente estão mais simples de serem efetuados, além dos relacionamentos entre tabelas, devemos nos atentar e preocupar com relacionamentos entre bancos diferentes.

- Tolerância a Falhas

Um SGBD deve fornecer recursos para recuperação de falhas tanto de software quanto de hardware. Tolerância a falhas é o recurso que o SGBD deve ter no caso de uma queda de energia inesperada por exemplo.

- Quando não Utilizar um SGBD

Em raras situações, o uso de um SGBD pode representar uma carga desnecessária aos custos quando comparado à abordagem processamento tradicional de arquivos como por exemplo:

- alto investimento inicial na compra de software e hardware adicionais;
- sobrecarga na provisão de controle de segurança, controle de concorrência, recuperação e integração de funções.

Problemas adicionais podem surgir caso os projetistas de banco de dados ou os administradores de banco de dados não elaborem os projetos corretamente ou se as aplicações não são implementadas de forma apropriada. Se o DBA não administrar o banco de dados de forma apropriada, tanto a segurança quanto a integridade dos sistemas podem ser comprometidas. A sobrecarga causada pelo uso de um SGBD e a má administração justificam a utilização da abordagem processamento tradicional de arquivos em casos como:

- Quando não haverá múltiplo acesso ao banco de dados ou a estrutura empresarial é muito pequena (SGDB).

2. Conceitos e Arquiteturas de um SGBD

❖ Modelos de Dados

Uma das principais características da abordagem banco de dados, é que a mesma fornece alguns níveis de abstração de dados omitindo ao usuário final, detalhes de como estes dados são armazenados. 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 possibilitam aos administradores e projetistas de bancos de dados a entenderem melhor o banco de dados.

Os modelos de dados podem ser basicamente de dois tipos:

- **Modelo de dados conceitual** (alto nível) → Fornece uma visão mais próxima do modo como os usuários visualizam os dados realmente;

- **Modelo Lógico** (baixo nível) → Fornece uma visão mais detalhada do modo como os dados estão realmente armazenados no computador.

❖ Esquemas

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.

❖ 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. O esquema de nível externo é como se fossem consultas especializadas de acordo com o usuário ou seu grupo.

❖ Independência de Dados

A "independência de dados" pode ser definida como a capacidade de se alterar um programa qualquer sem ter que se alterar o banco. Um exemplo bastante claro desta independência é que um mesmo banco pode servir à múltiplas aplicações.

Um sistema de notas da secretaria de uma Universidade, por exemplo, pode ter o mesmo banco para acesso via web e para alterações na própria secretaria da Universidade. A independência dos dados seria alterarmos o programa web, sem alterarmos o banco.

❖ 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 onde 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 são bem claras, 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).

Quando se fala em linguagem SQL (Structure Query Language), estamos falando em comandos que compõem a DDL e a DML preferencialmente. Resumindo, DDL é a linguagem para Definição dos dados, ou seja, através destes comandos é possível se criar a estrutura do banco (Create database, create table, etc). DML é a linguagem para manipulação dos dados ou seja: Insert, update, delete, select, etc...

❖ 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.

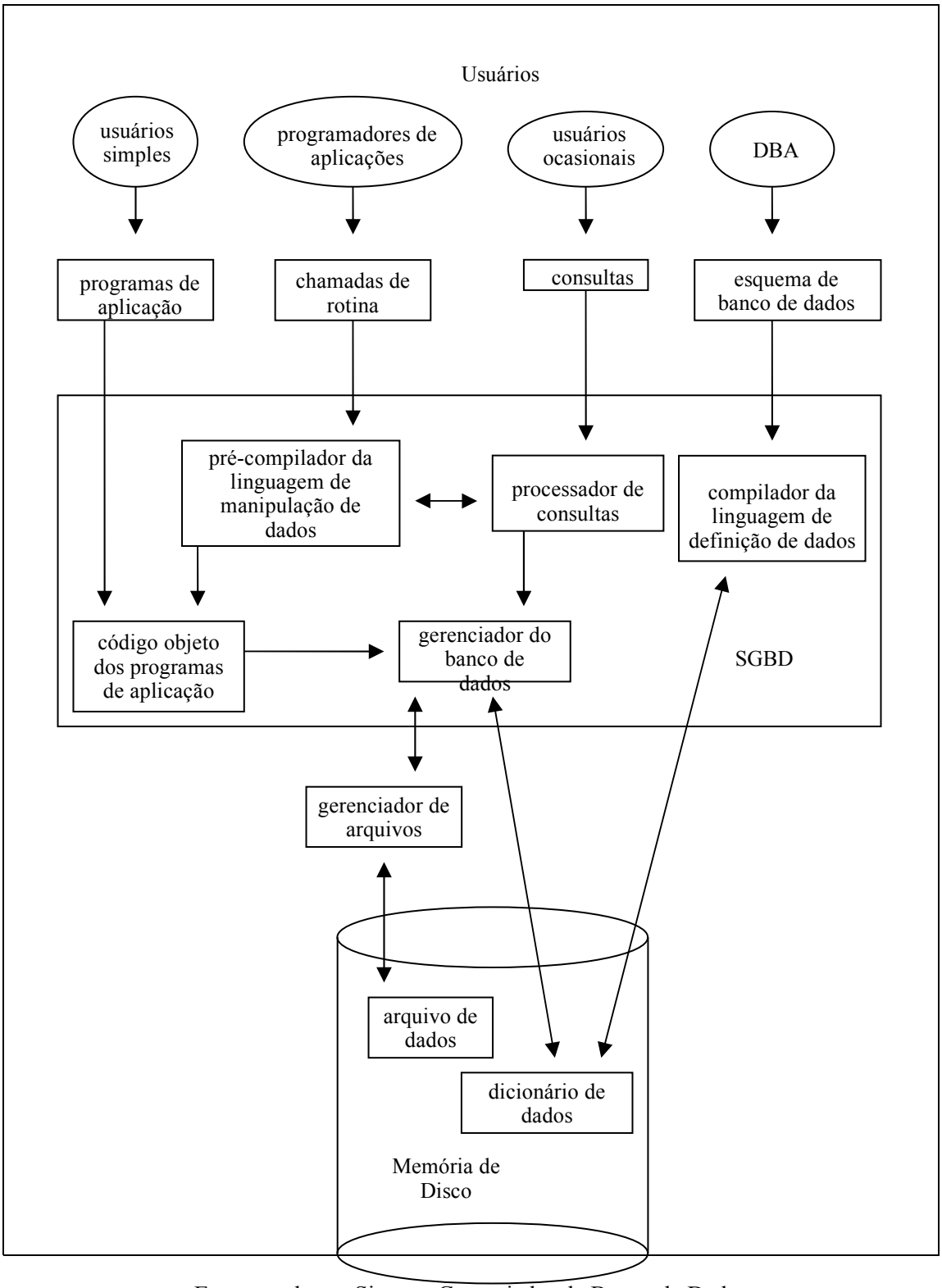
❖ Classificação dos SGBDs

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. Podemos ter em um único sistema, um banco Oracle se comunicando com outro banco MySQL, por exemplo.



Estrutura de um Sistema Gerenciador de Banco de Dados

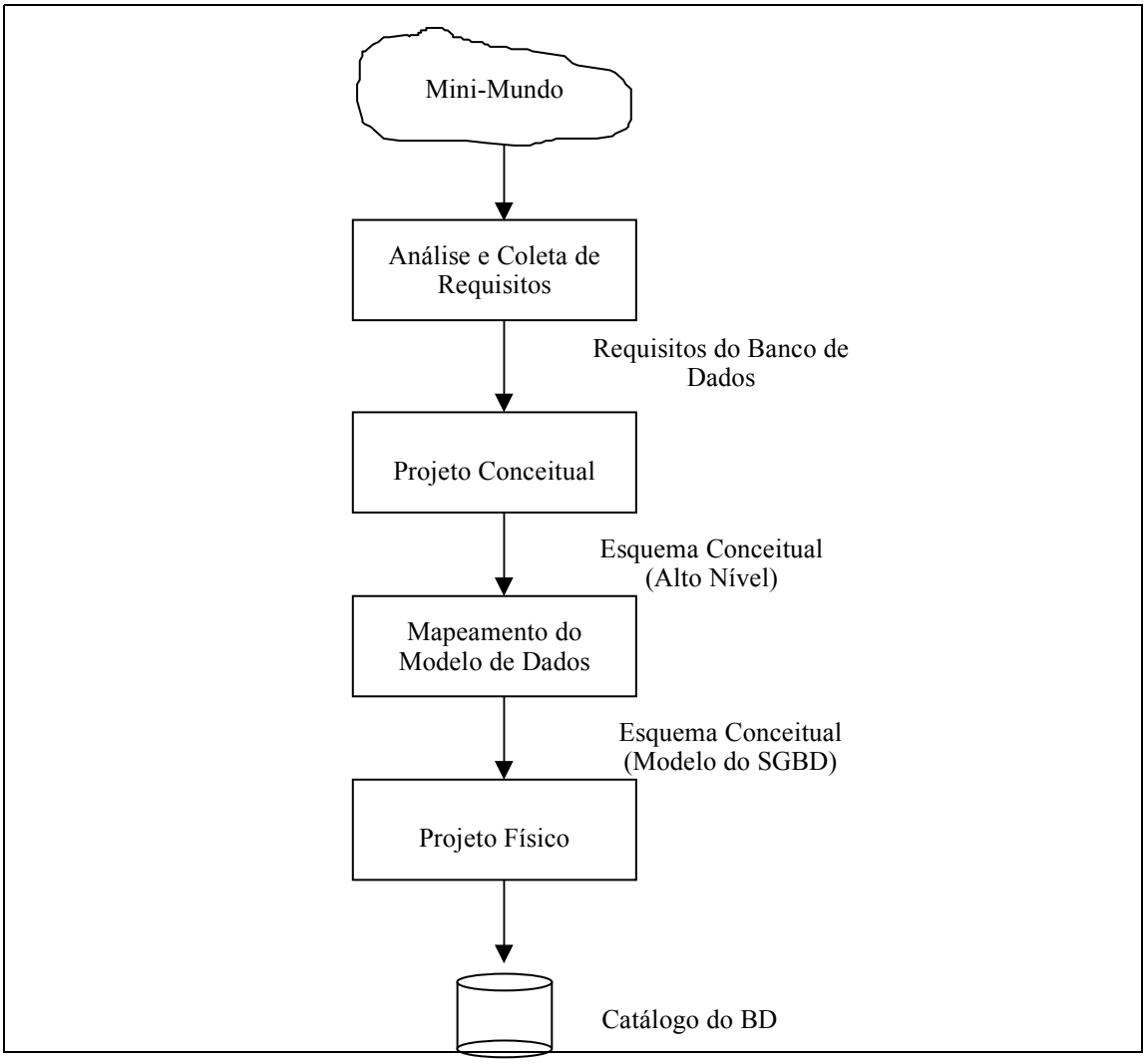
3. Modelagem de Dados Utilizando o Modelo Entidade Relacionamento (ER)

O modelo Entidade-Relacionamento é um modelo de dados conceitual de alto nível, cujos conceitos foram projetados para estar o mais próximo possível da visão que o usuário tem dos dados, não se preocupando em representar como estes dados estarão realmente armazenados. O modelo ER é utilizado principalmente durante o processo de projeto de banco de dados.

Montar um modelo correto e consistente talvez seja a tarefa mais difícil e importante em no processo de construção de um banco de dados.

❖ Modelo de Dados Conceitual de Alto Nível

A figura abaixo faz uma descrição simplificada do processo de projeto de um banco de dados.



Fases básicas do Projeto de um Banco de Dados

O modelo de dados conceitual de alto nível é composto de várias etapas como vimos na figura anterior. Dentre elas, uma de vital importância para elaborarmos um projeto e, conseqüentemente, um banco de dados eficaz é a extração de requisitos.

É nesta etapa que conseguiremos descrever os requisitos, ou seja, as necessidades que o cliente ou usuário necessita daquilo que se deseja armazenar no banco. Se o levantamento de requisitos for falho, todo o modelo lógico, físico e a implementação do banco poderá ser comprometida.

Dentre as etapas importantes na construção do banco temos o **modelo lógico** que é composto de algumas estruturas, dentre elas:

❖ Entidades e Atributos

O objeto básico tratado pelo modelo ER é a "entidade", que pode ser definida como um objeto do mundo real, concreto ou abstrato e que possui existência independente.

IMPORTANTE

Uma tabela que se cadastre "clientes" é uma entidade. Um cadastro de "produtos" é outra entidade, uma entidade de "alunos" é outra entidade. Também podemos chamar entidade de "tabela".

Cada entidade possui um conjunto particular de propriedades que a descreve chamado "atributos" ou "campos". Por exemplo, código é um campo, nome é outro campo, endereço também é outro campo.

Os atributos que podem assumir apenas um determinado valor em uma determinada instância é denominado "atributo simplesmente valorado", enquanto que um atributo que pode assumir diversos valores em uma mesma instância é denominado "multi valorado".

Antigamente, em alguns bancos de dados, há a possibilidade de se obter uma propriedade específica. É a profundidade. Por exemplo: Poderíamos ter, em um mesmo campo, três valores diferentes. Exemplo: Tel[1] Tel[2] Tel[3]...

Atualmente, para esta implementação "multi-valorada" utilizamos uma entidade à parte. Por exemplo: Se um cliente possui diversos telefones, porém, alguns podem ter somente um. Para não termos campos vazios, utilizamos uma tabela de clientes e outra de telefones relacionada à clientes para digitarmos os telefones. Na tabela telefones, utilizaríamos o campo "CodCliente", "TipoTel" e "NroTel". Com isso, implementaríamos, de forma racional, o conceito de multi-valoração.

❖ Estrutura básica de Banco de Dados, Conjunto de Valores, Atributo Chave

Na verdade a estrutura básica de um banco de dados é a seguinte:

Um banco é formado por um conjunto de entidades ou tabelas,

Uma tabela ou entidade é formada por um conjunto de registros (Um cliente, Um aluno, etc).

Um registro é formado por um conjunto de atributos ou campos(Ex: código, nome, endereço, etc...)

Cada informação dessa é um campo ou atributo.

Uma restrição muito importante em um registro é o campo ou atributo chamado "**atributo chave**" e seus valores podem ser utilizados para identificar cada registro de forma única. Muitas vezes, uma chave pode ser formada pela composição de dois ou mais atributos. Quando a chave é formada por mais de um atributo ou campo chamamos de **Chave Primária composta**. Quando a chave é formada somente por um campo (Ex: Código) chamamos de Chave Primária Simples.

Cada atributo simples de um registro está associado com um conjunto de valores denominado "**domínio**", o qual especifica o conjunto de valores que podem ser designados para este determinado atributo para cada entidade. Ou seja, quais valores podem assumir cada campo. No campo sexo por exemplo, só podemos aceitar F ou M, ou seja F,M é o domínio do atributo sexo.

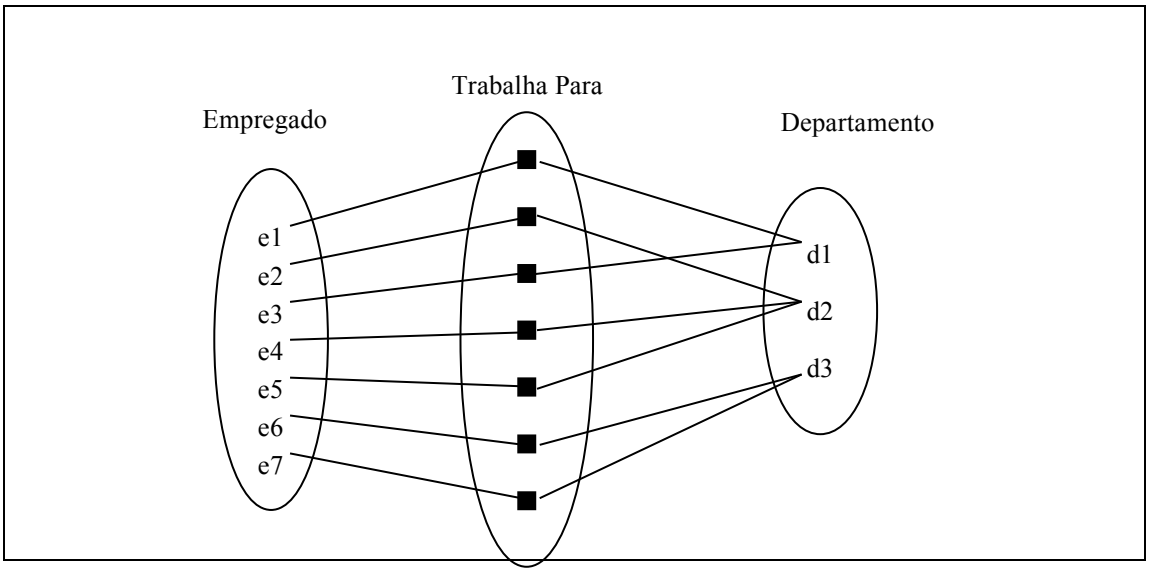
❖ Relacionamentos

Além de conhecer detalhadamente os tipos entidade, é muito importante conhecer também os relacionamentos entre as entidades.

Um relacionamento, como o próprio nome diz, é a relação que ocorre entre registros de duas ou mais entidades(tabelas). Para que o relacionamento ocorra, deve haver "interesses" envolvidos, ou seja, uma entidade deve se "interessar" pelas informações da outra, seus campos ou atributos devem ter o **mesmo tipo e tamanho e devem ser afins**, porém, não necessariamente o mesmo nome.

A figura à seguir mostra um exemplo entre dois tipos entidade (empregado e departamento) e o relacionamento entre eles (trabalha para). Repare que para cada relacionamento, participam apenas uma entidade de cada tipo entidade, porém, uma entidade pode participar de mais do que um relacionamento.

Os relacionamentos existem para que as entidades possam compartilhar as informações, evitando-se que haja repetição de informações sem necessidade. O nome de um cliente por exemplo, só deve ocorrer uma única vez. Se outras entidades necessitam das informações dos clientes devem se relacionar com a tabela "Clientes". Os relacionamentos ocorrem normalmente em atributos numéricos de pequena largura, ou seja, devem ser campos pequenos. Campos dos tipos texto não devem se relacionar, excetos em raros casos onde isso for necessário.



Exemplo de um Relacionamento

❖ Grau de um Relacionamento

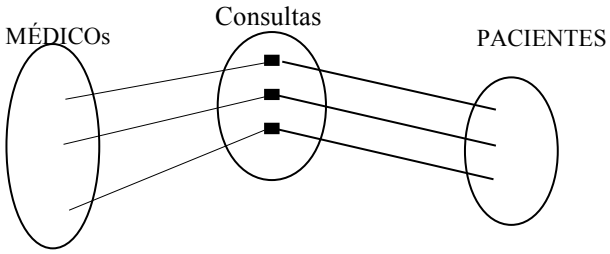
O "grau" de um tipo relacionamento é o número de tipos entidade que participam do tipo relacionamento. No exemplo da figura acima, temos um relacionamento binário. O grau de um relacionamento é ilimitado, porém, a partir do grau 3 (ternário), a compreensão e a dificuldade de se desenvolver a relação corretamente se tornam extremamente complexas.

❖ Outras Características de um Relacionamento

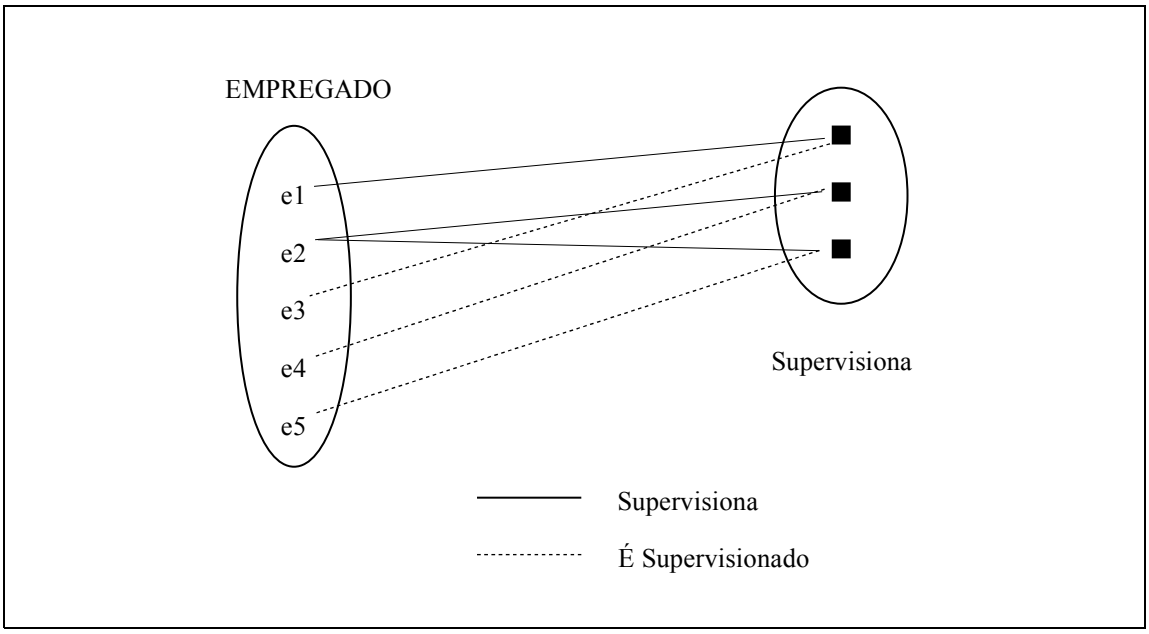
- Relacionamentos com Atributos

Algumas vezes é conveniente pensar em um relacionamento como um atributo.

Considere uma situação onde diversos médicos possuem diversos pacientes e cada paciente pode ter N médicos. Portanto, não é possível armazenarmos o código do médico na entidade paciente (já que ele possui n médicos) e nem o código do paciente em médico (ele possui n clientes). Caracteriza-se portanto um relacionamento de muitos para muitos (n para n). Neste caso, é necessário criarmos o que chamamos "Entidade Associativa" ou "Relacionamento com atributos". Neste caso, teríamos a entidade "Consulta". Nesta entidade poderíamos ter o código do médico, o código do paciente, data da consulta e observações.



Um relacionamento recursivo é um relacionamento entre entidades do mesmo tipo entidade. Veja o exemplo da figura 6.

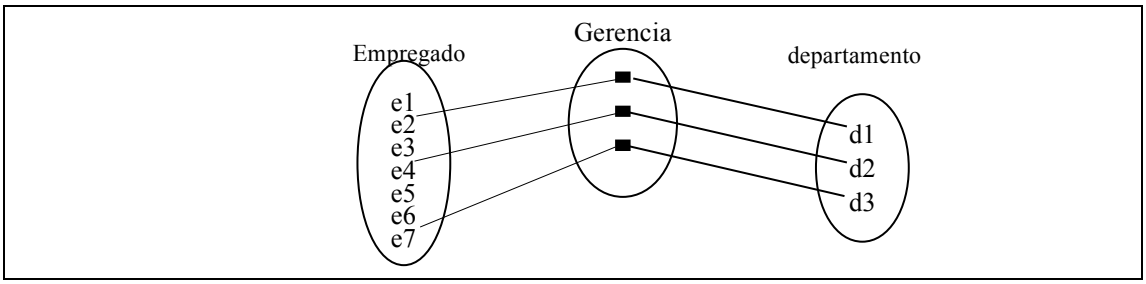


Um Relacionamento Recursivo

No exemplo, temos um relacionamento entre o tipo entidade EMPREGADO, onde um empregado pode supervisionar outro empregado e um empregado pode ser supervisionado por outro empregado.

- Restrições em Tipos Relacionamentos

Geralmente, os tipos relacionamentos sofrem certas restrições que limitam as possíveis combinações das entidades participantes. Estas restrições são derivadas de restrições impostas pelo estado destas entidades no mini-mundo ou mundo real. Veja o exemplo da figura a seguir.

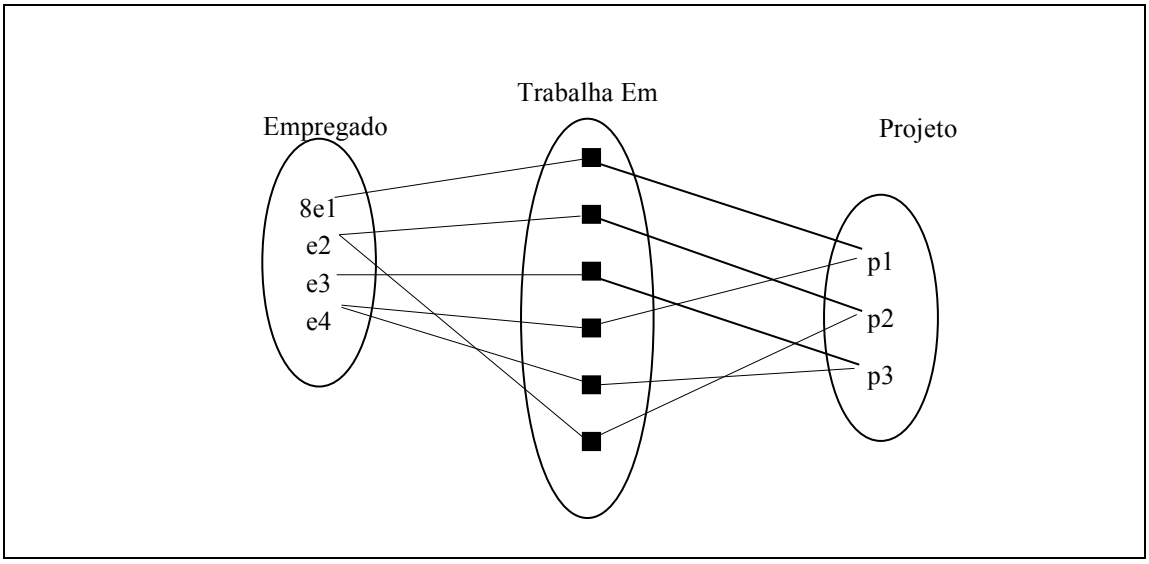


Relacionamento EMPREGADO gerencia DEPARTAMENTO

No exemplo acima, temos a seguinte situação: um empregado pode gerenciar apenas um departamento, enquanto que um departamento, pode ser gerenciado por apenas um empregado. A este tipo de restrição, nós chamamos **cardinalidade**. A cardinalidade indica a forma que uma entidade participa do relacionamento.

A cardinalidade pode ser: 1:1(um para um), 1:N (um para muitos), M:N (muitos para muitos).

No exemplo anterior, a cardinalidade é 1:1, pois cada entidade empregado pode gerenciar apenas um departamento e um departamento pode ser gerenciado por apenas um empregado. No exemplo do relacionamento EMPREGADO Trabalha Para DEPARTAMENTO, o relacionamento é 1:N, pois um empregado pode trabalhar em apenas um departamento, enquanto que um departamento pode possuir vários empregados. Na figura abaixo temos um exemplo de um relacionamento com cardinalidade N:M.



Relacionamento N:M

No exemplo acima, nós temos que um empregado pode trabalhar em vários projetos enquanto que um projeto pode ter vários empregados trabalhando. É o mesmo caso que citamos acima de Médicos e Pacientes.

Outra restrição muito importante é a **participação**. A participação define a existência de uma entidade através do relacionamento, podendo ser **parcial** ou **total**. Veja o exemplo da figura 6. A participação do empregado é **parcial** pois nem todo empregado gerencia um departamento, porém a participação do departamento neste relacionamento é **total** pois todo departamento precisa ser gerenciado por um empregado. Desta forma, **todas** as entidades do tipo entidade DEPARTAMENTO precisam participar do relacionamento, mas **nem todas** os registros da entidade EMPREGADO precisam participar do relacionamento. Já no exemplo da figura 4, ambas as participações são totais pois todo empregado precisa trabalhar em um departamento e todo departamento tem que ter empregados trabalhando nele.

Estas restrições são chamadas de **restrições estruturais**.

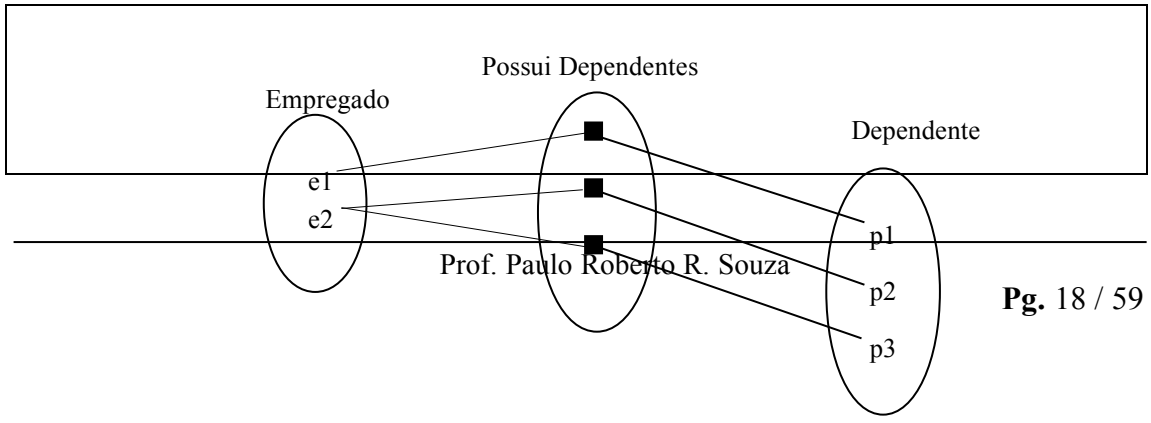
Algumas vezes, torna-se necessário armazenar um atributo no tipo relacionamento. Veja o exemplo da figura "Empregados" e "Departamentos". Eu posso querer saber em que dia o empregado passou a gerenciar o departamento. É difícil estabelecer a qual tipo entidade pertence atributo, pois o mesmo é definido apenas pela existência do relacionamento. Quando temos relacionamentos com cardinalidade 1:1, podemos colocar o atributo em uma das entidades, de preferência, em uma cujo tipo entidade tenha participação total. No caso, o atributo poderia ir para o tipo entidade departamento. Isto porque nem todo empregado participará do relacionamento. Caso a cardinalidade seja 1:N, então podemos colocar o atributo no tipo entidade com participação N. Porém, se a cardinalidade for N:M, então o atributo deverá mesmo ficar no tipo relação. Veja o exemplo da figura que aparecem "Empregados" e "Projetos". Caso queiramos armazenar quantas horas cada empregado trabalhou em cada projeto, então este deverá ser um atributo do relacionamento.

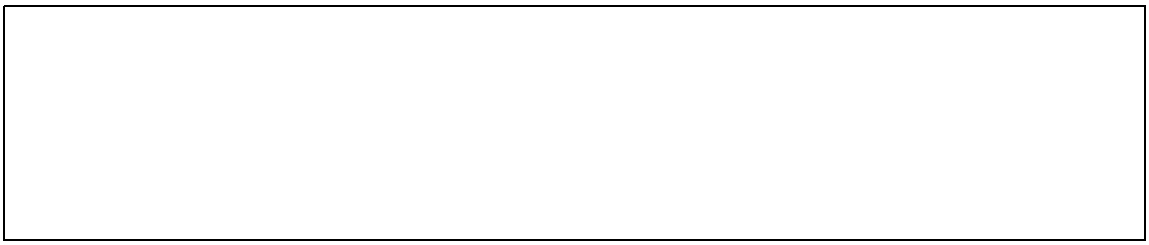
Ou seja, quando temos o relacionamento muitos para muitos, devemos utilizar a entidade-relacionamento ou relacionamento com atributos. Esta entidade serve de "ponte" entre as duas relacionadas, ou seja, quando não conseguimos armazenar algumas informações em um ou em outra entidade, utilizamos uma terceira (entidade-relacionamento) para armazena-las.

Um bom exemplo disso é o relacionamento entre MÉDICOS e PACIENTES. Um médico possui muitos pacientes e um paciente possui muitos médicos. Algumas informações não são possíveis de serem armazenadas nem em médicos nem tampouco em pacientes. A data da consulta por exemplo, o diagnóstico, etc. Se colocamos em médicos, ele tem vários pacientes e estas informações ficarão falhas, se colocamos em pacientes, se ele fez várias consultas e só possuímos um espaço ou seja, um campo para armazenar cada informação, também ficará falha. Portanto, será necessário a utilização da entidade-relacionamento CONSULTAS para armazenarmos estas informações específicas de CONSULTAS.

- Tipos Entidades Fracas

Alguns tipos entidade podem não ter um atributo chave por si só. Isto implica que não poderemos distinguir algumas entidades por que as combinações dos valores de seus atributos podem ser idênticas. Estes tipos entidade são chamados **entidades fracas**. As entidades deste tipo precisam estar relacionadas com uma entidade pertencente ao tipo entidade **proprietária**. Este relacionamento é chamado de **relacionamento identificador**. Veja o exemplo abaixo.





Relacionamento com uma Entidade Fraca (Dependente)

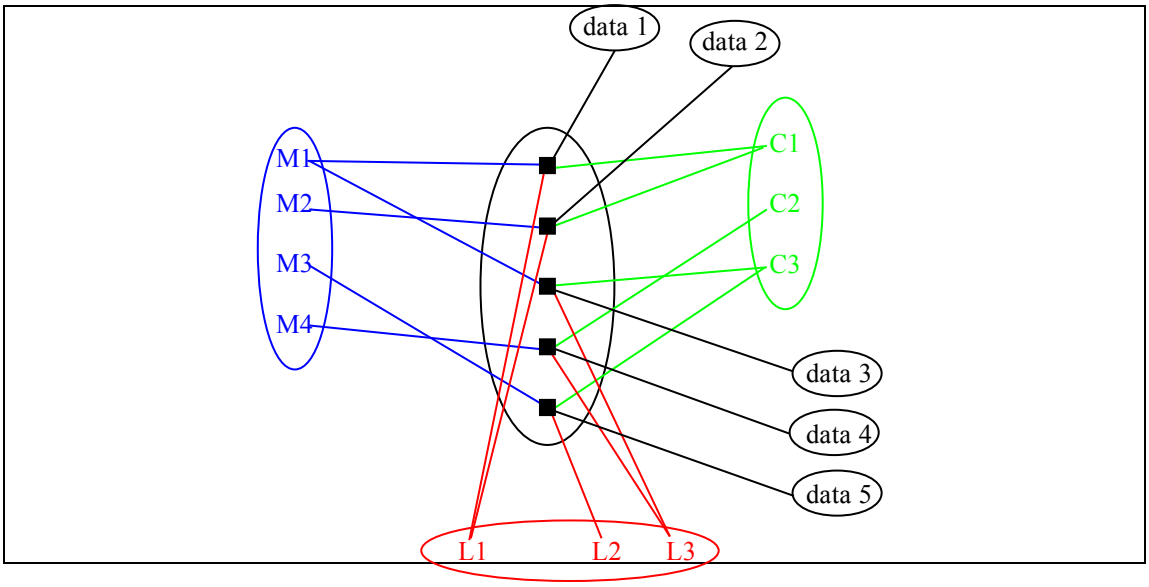
O tipo entidade DEPENDENTE é uma entidade fraca pois não possui um método de identificar uma entidade única. O EMPREGADO não é uma entidade fraca pois possui um atributo para identificação (atributo chave). O número do RG de um empregado identifica um único empregado. Porém, um dependente de 5 anos de idade não possui necessariamente um documento. Desta forma, esta entidade é um tipo entidade fraca. Um tipo entidade fraca possui uma **chave parcial**, que juntamente com a chave primária da entidade proprietária forma uma chave primária composta. Neste exemplo: a chave primária do EMPREGADO é o RG. A chave parcial do DEPENDENTE é o seu nome, pois dois irmãos não podem ter o mesmo nome. Desta forma, a chave primária desta entidade fica sendo o RG do pai ou mãe mais o nome do dependente.

Algumas características marcantes nas entidades fracas são o fato de sua chave primária ser composta (Formada por mais de um campo) e seus registros só existem se existirem registros correspondentes na entidade forte. Ex. Alunos e Notas. Só existem notas se existirem alunos cadastrados.

Todos os exemplos vistos acima foram para relacionamentos binários, ou seja, entre dois tipos entidades diferentes ou recursivos. Porém, o modelo entidade relacionamento não se restringe apenas à relacionamentos binários. O número de entidades que participam de um tipo relacionamento é irrestrito e armazenam muito mais informações do que diversos relacionamentos binários. Considere o seguinte exemplo:

Um motorista pode efetuar uma viagem para uma localidade dirigindo um determinado caminhão em uma determinada data.

Se efetuarmos três relacionamentos binários, não teremos estas informações de forma completa como se criássemos um relacionamento ternário. Veja o resultado como fica no exemplo da figura abaixo.



Relacionamento Ternário

❖ 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, fornecendo um conceito mais generalizado de como estão estruturados os dados de um sistema.

Os objetos que compõem o diagrama ER estão listados a seguir, na figura 11.

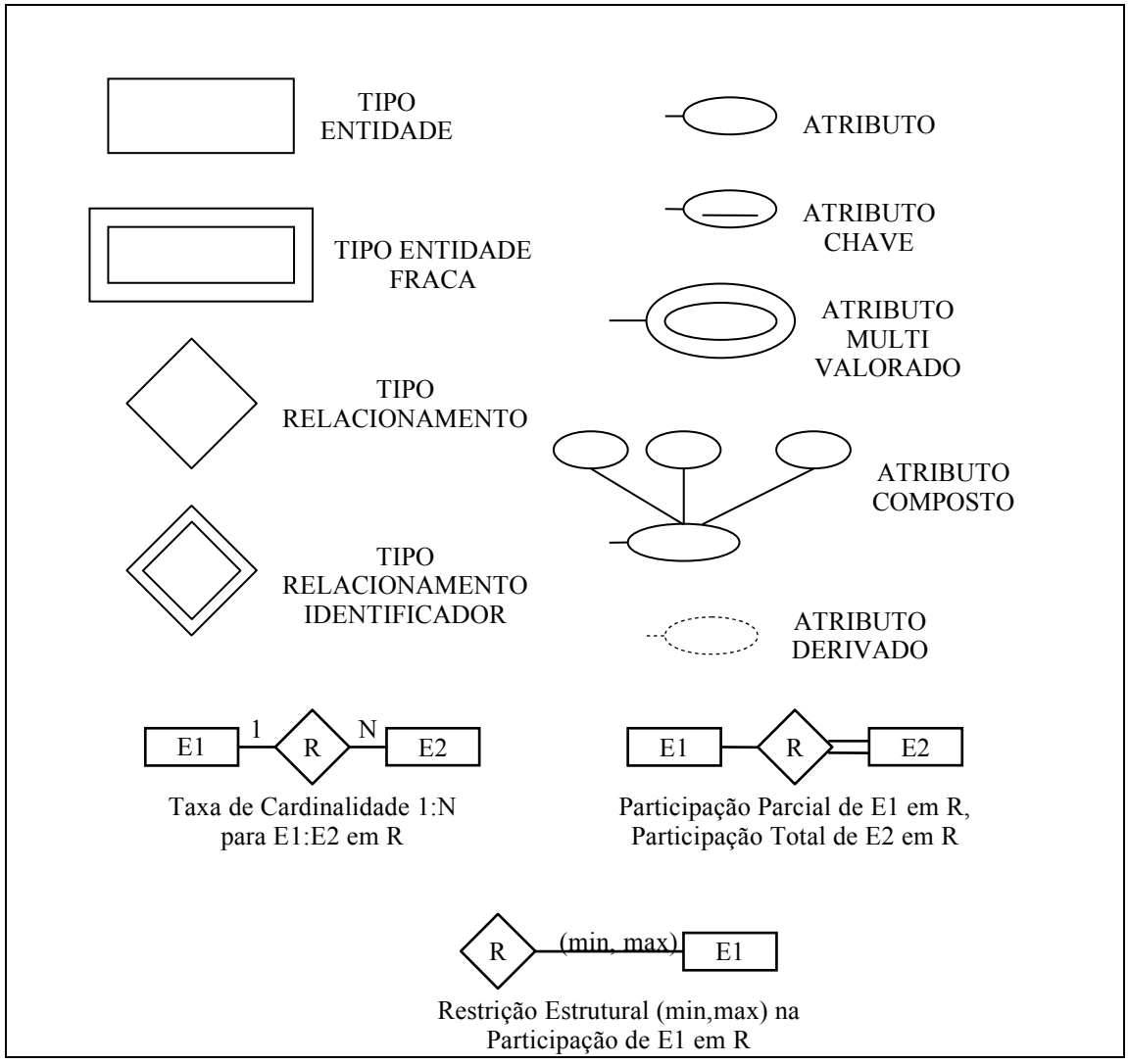


Figura 11- Objetos que Compõem o Diagrama ER

❖ Modelo Entidade Relacionamento Estendido

Os conceitos do modelo Entidade Relacionamento discutidos anteriormente são suficientes para representar logicamente a maioria das aplicações de banco de dados. Porém, com o surgimento de novas aplicações, surgiu também a necessidade de novas semânticas para a modelagem de informações mais complexas. O modelo **Entidade Relacionamento Extendido (ERE)** visa fornecer esta semântica para permitir a representação de informações complexas. É importante frisar que embora o modelo **ERE** trate classes e subclasses, ele não possui a mesma semântica de um modelo orientado a objetos.

O modelo **ERE** engloba todos os conceitos do modelo **ER** mais os conceitos de **subclasse**, **superclasse**, **generalização** e **especialização** e o conceito de **herança de atributos**.

- Subclasses, Superclasses e Especializações

O primeiro conceito do modelo **ERE** que será abordado é o de **subclasse** de um tipo entidade. Como visto anteriormente, um tipo entidade é utilizado para representar um conjunto de entidades do mesmo tipo. Em muitos casos, um tipo entidade possui diversos subgrupos adicionais de entidades que são significativas e precisam ser representadas explicitamente devido ao seu significado à aplicação de banco de dados. Leve em consideração o seguinte exemplo:

> *Para um banco de dados de uma empresa temos o tipo entidade empregado, o qual possui as seguintes características: nome, rg, cic, número funcional, endereço completo (rua, número, complemento, cep, bairro, cidade), sexo, data de nascimento e telefone (ddd e número); caso o(a) funcionário(a) seja um(a) engenheiro(a), então deseja-se armazenar as seguintes informações: número do CREA e especialidade (Civil, Mecânico, Elétro/Eletrônico); caso o(a) funcionário(a) seja um(a) secretário(a), então deseja-se armazenar as seguinte informações: qualificação (bi ou tri língue) e os idiomas no qual possui fluência verbal e escrita.*

Se as informações *número do CREA*, *especialidade*, *tipo* e *idiomas* forem representadas diretamente no tipo entidade *empregado* estaremos representando informações de um conjunto limitados de entidades *empregado* para os todos os funcionários da empresa. Neste caso, podemos criar duas subclasses do tipo entidade *empregado*: ***engenheiro*** e ***secretária***, as quais irão conter as informações acima citadas. Além disto, ***engenheiro*** e ***secretária*** podem ter relacionamentos específicos.

Uma entidade não pode existir meramente como componente de uma subclasse. Antes de ser componente de uma subclasse, uma entidade deve ser componente de uma superclasse. Isto leva ao conceito de **herança de atributos**; ou seja, a subclasse herda todos os atributos da superclasse. Isto porque a entidade de subclasse representa as mesmas características de uma mesma entidade da superclasse. Uma subclasse pode herdar atributos de superclasses diferentes.

A figura 12 mostra a representação diagramática do exemplo acima.

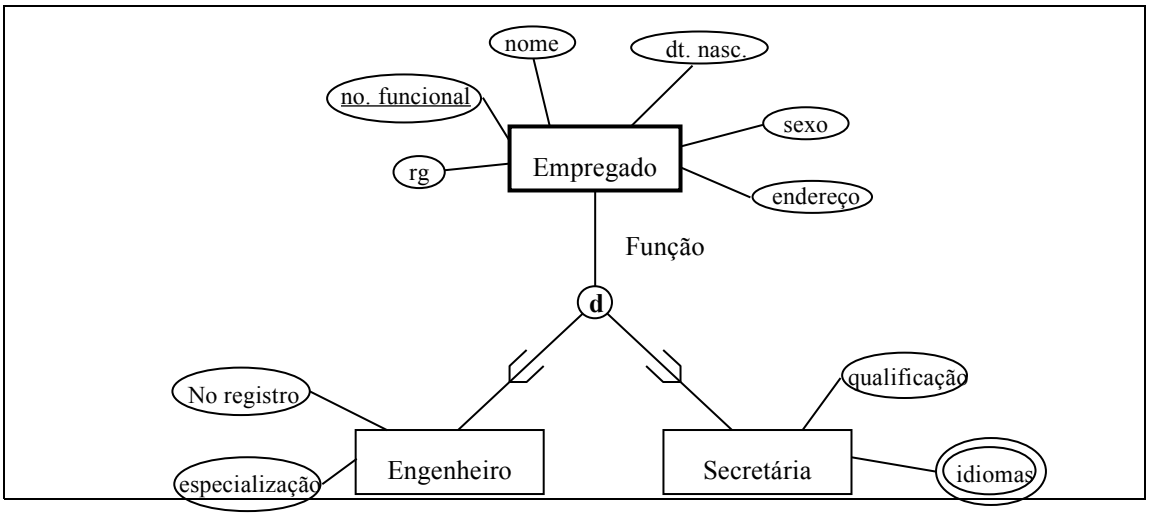


Figura 12 - Representação de Superclasse e Subclasses

- Especialização

Especialização é o processo de definição de um conjunto de classes de um tipo entidade; este tipo entidade é chamado de superclasse da especialização. O conjunto de subclasses é formado baseado em alguma característica que distinga as entidades entre si.

No exemplo da figura 12, temos uma especialização, a qual podemos chamar de *função*. Veja agora no exemplo da figura 13, temos a entidade empregado e duas especializações.

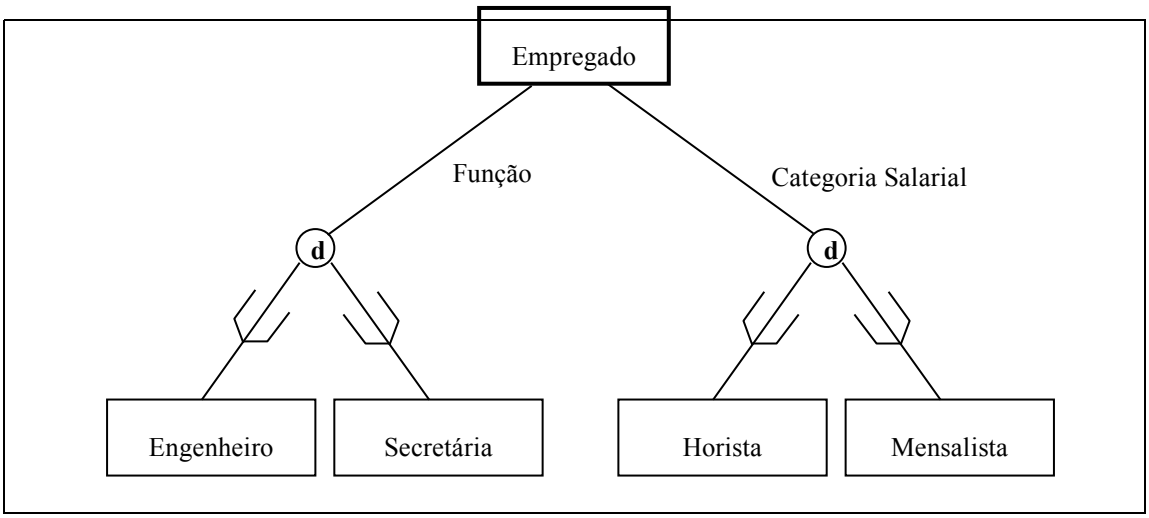


Figura 13 - Duas Especializações para Empregado: *Função* e *Categoria Salarial*

Como visto anteriormente, uma subclasse pode ter relacionamentos específicos com outras entidades ou com a própria entidade que é a sua superclasse. Veja o exemplo da figura 14.

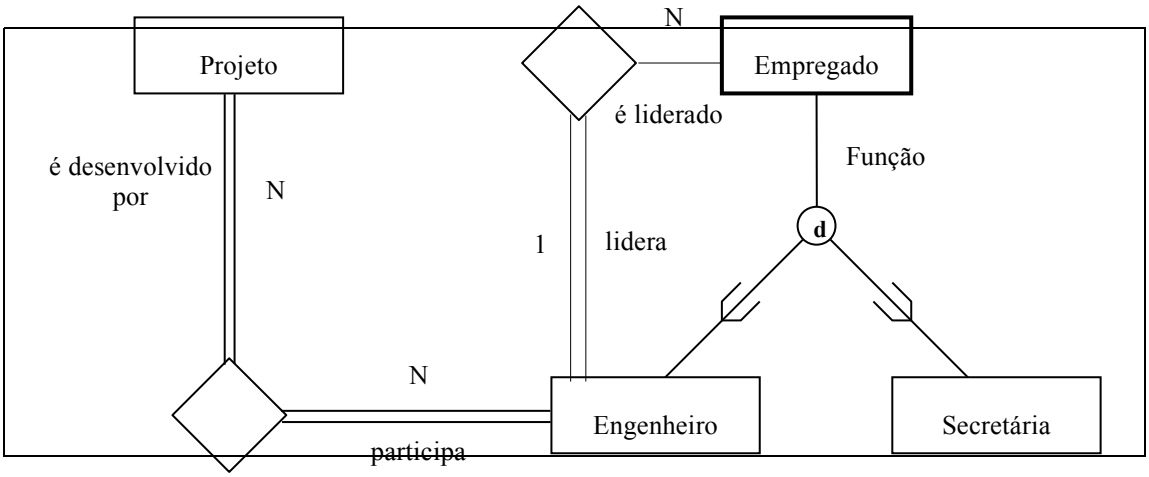


Figura 14 - Relacionamentos Entre Subclasses e Entidades

O processo de *especialização* nos permite:

- definir um conjunto de subclasses de um tipo entidade;
- associar atributos específicos adicionais para cada subclasse;
- estabelecer tipos relacionamentos específicos entre subclasses e outros tipos entidades.

- Generalização

A *generalização* pode ser pensada como um processo de abstração reverso ao da *especialização*, no qual são suprimidas as diferenças entre diversos tipos entidades, identificando suas características comuns e generalizando estas entidades em uma superclasse

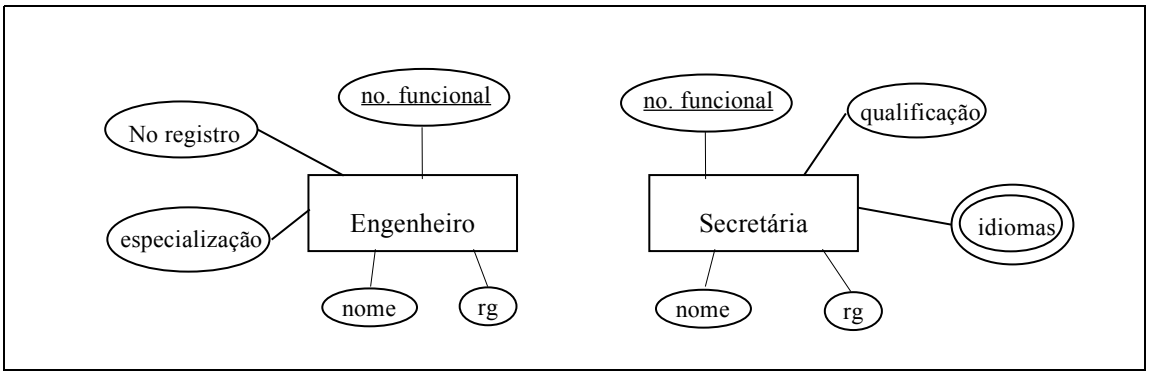


Figura 15 - Tipos Entidades *Engenheiro* e *Secretária*

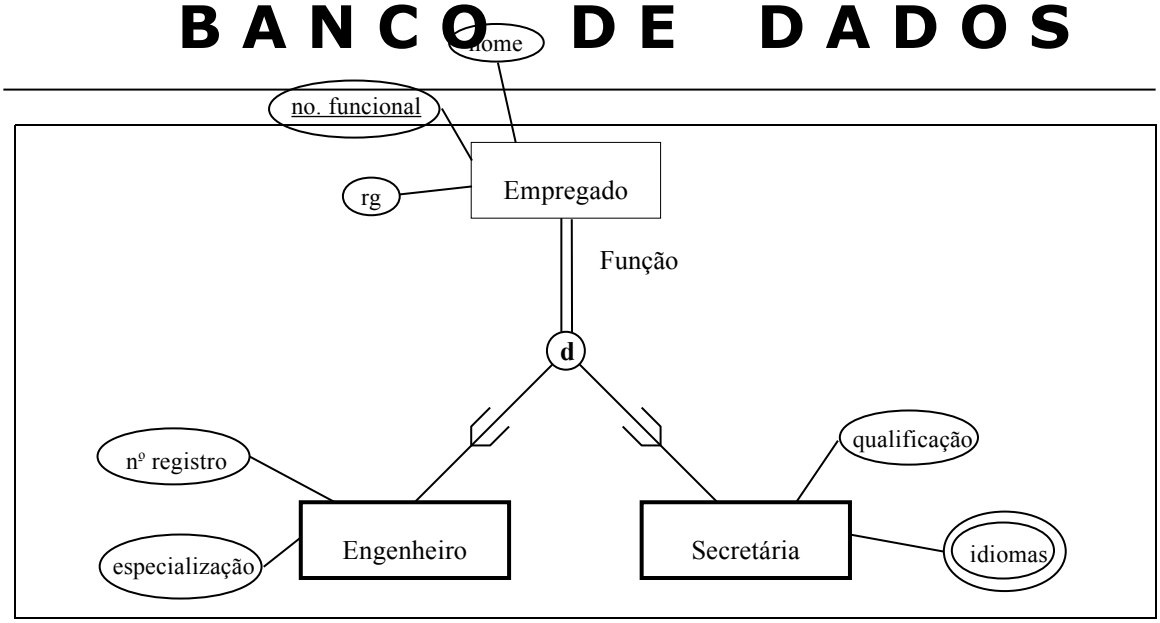


Figura 16 - Generalização *Empregado* para os Tipos Entidades Engenheiro e Secretária

É importante destacar que existe diferença semântica entre a *especialização* e a *generalização*. Na *especialização*, podemos notar que a ligação entre a superclasse e as subclasses é feita através de um traço simples, indicando **participação parcial** por parte da superclasse. Analisando o exemplo da figura 12, é observado que um empregado **não** é obrigado a ser um *engenheiro* ou uma *secretária*. Na generalização, podemos notar que a ligação entre a superclasse e as subclasses é feita através de um traço duplo, indicando **participação total** por parte da superclasse. Analisando o exemplo da figura 16, é observado que um empregado **é** obrigado a ser um *engenheiro* ou uma *secretária*.

A letra **d** dentro do círculo que especifica uma especialização ou uma generalização significa **disjunção**. Uma disjunção em uma especialização ou generalização indica que uma entidade do tipo entidade que representa a superclasse pode assumir apenas um papel dentro da mesma. Analisando o exemplo da figura 13. Temos duas especializações para a superclasse *Empregado*, as quais são restringidas através de uma disjunção. Neste caso, um empregado pode ser um *engenheiro* ou uma *secretária* e o mesmo pode ser *horista* ou *mensalista*.

Além da *disjunção* podemos ter um "**overlap**", representado pela letra **o**. No caso do "overlap", uma entidade de uma superclasse pode ser membro de mais que uma subclasse em uma especialização ou generalização. Analise a generalização no exemplo da figura 17. Suponha que uma peça fabricada em uma tornearia pode ser *manufaturada* ou *torneada* ou ainda, pode ter sido *manufaturada* e *torneada*.

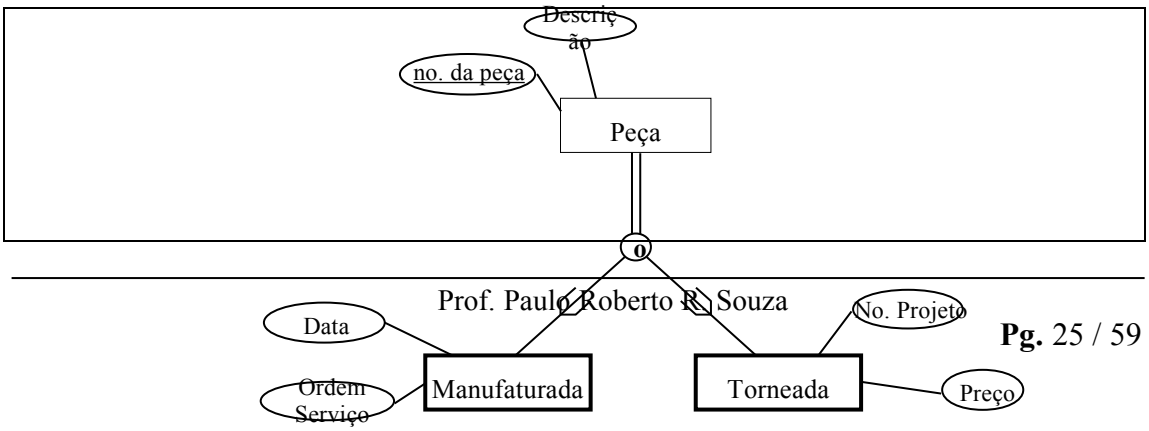


Figura 17 - Uma Generalização com “Overlap”

- “Lattice” ou Múltipla Herança

Uma subclasse pode ser definida através de um “lattice”, ou múltipla herança, ou seja, ela pode ter diversas superclasses, herdando características de todas. Leve em consideração o seguinte exemplo:

> *Uma construtora possui diversos funcionários, os quais podem ser engenheiros ou secretárias. Um funcionário pode também ser assalariado ou horista. Todo gerente de departamento da construtora deve ser um engenheiro e assalariado.*

O modelo lógico da expressão acima tem o seguinte formato:

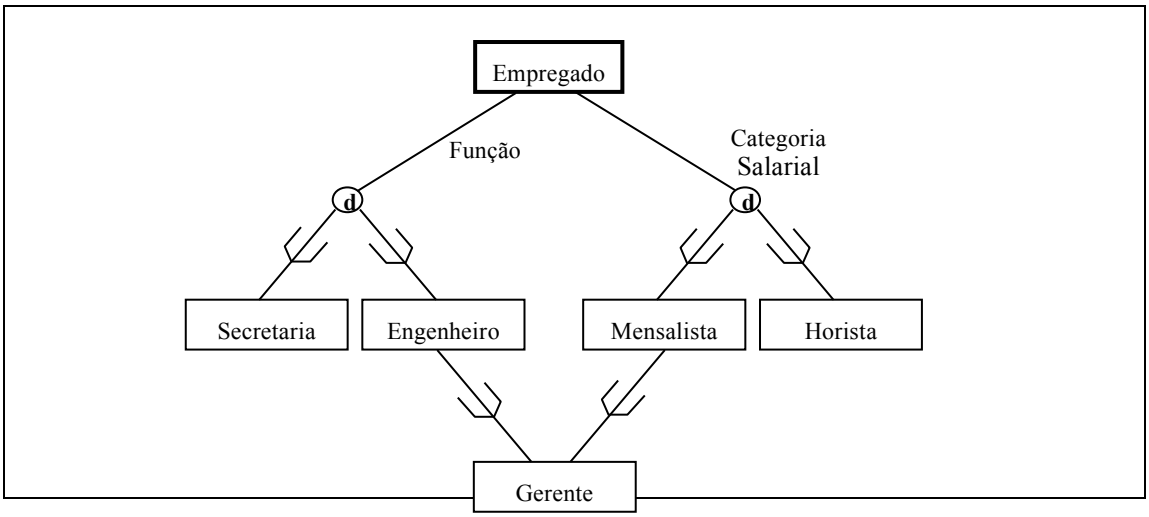


Figura 18 - Um “Lattice” com a Subclasse Gerente Compartilhada

Neste caso então, um gerente será um funcionário que além de possuir as características próprias de *Gerente*, herdará as características de *Engenheiro* e de *Mensalista*.

Tabela Resumo

O **modelo relacional** foi criado por Codd em 1970 e tem por finalidade representar os dados como uma coleção de relações, onde cada relação é representada por uma **tabela**, ou falando de uma forma mais direta, um arquivo. Porém, um arquivo é mais restrito que uma tabela. Toda tabela pode ser considerada um arquivo, porém, nem todo arquivo pode ser considerado uma tabela.

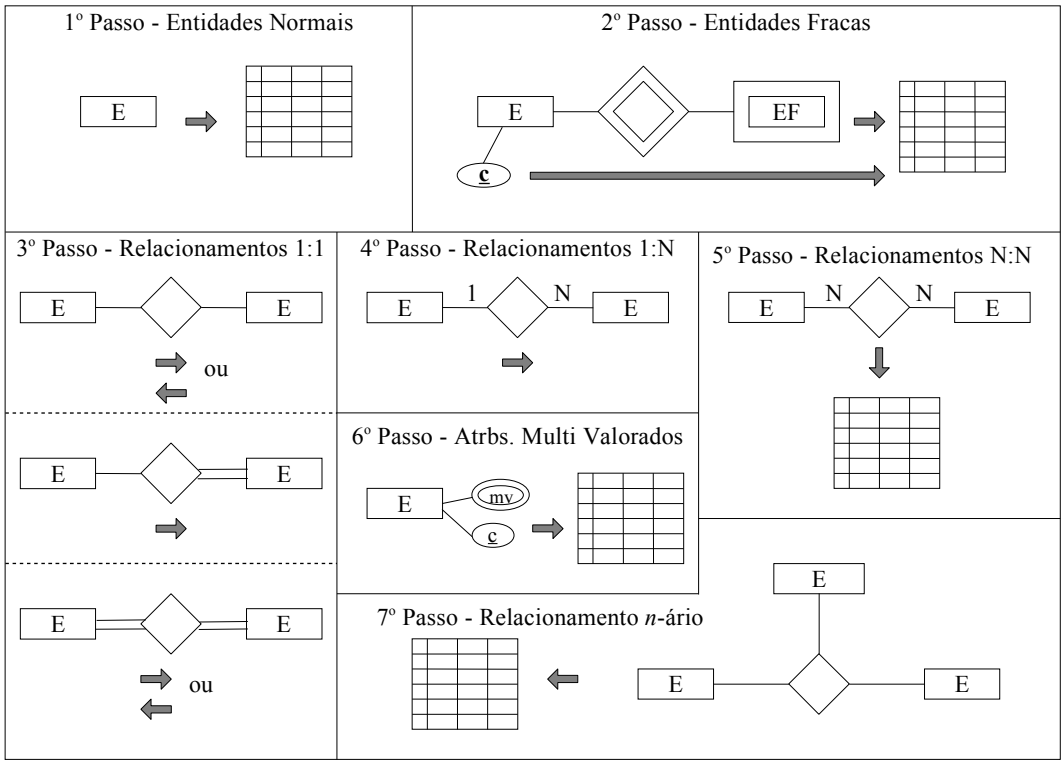


Figura 19 - Mapeamento para o Modelo Relacional

4 - A Álgebra Relacional

A **álgebra relacional** é uma coleção de operações canônicas que são utilizadas para manipular as relações. Estas operações são utilizadas para selecionar tuplas de relações individuais e para combinar tuplas relacionadas de relações diferentes para especificar uma consulta em um determinado banco de dados. O resultado de cada operação é uma nova operação, a qual também pode ser manipulada pela álgebra relacional.

NOTA: Tuplas são conjuntos de registros. Por exemplo: Se em uma tabela de cadastro de clientes selecionamos somente os clientes que são do estado = a MG. Portanto, este "set" de registros ou este conjunto de registros selecionados damos o nome de **Tupla.**

Todos os exemplos envolvendo álgebra relacional implicam na utilização do banco de dados descrito no apêndice A.

- A Operação *Select*

A operação **select** é utilizada para selecionar um subconjunto de tuplas de uma relação, sendo que estas tuplas devem satisfazer uma **condição de seleção**. A forma geral de uma operação **select** é:

$$\sigma_{\text{<condição de seleção>}}(\text{<nome da relação>})$$

A letra grega σ é utilizada para representar a operação de seleção; **<condição de seleção>** é uma expressão *booleana* aplicada sobre os atributos da relação e **<nome da relação>** é o nome da relação sobre a qual será aplicada a operação **select**.

Exemplos:

$$\textbf{consulta1} = \sigma_{\text{salário} < 2.500,00}(\text{EMPREGADO})$$

gera a seguinte tabela como resultado:

Tabela *consulta1*					
Nome	RG	CIC	Depto.	RG Supervisor	Salário
Ricardo	*30303030*	*33333333*	*2*	*10101010*	*2.300,00*
Renato	*50505050*	*55555555*	*3*	*20202020*	*1.300,00*

$$\textbf{consulta2} = \sigma_{(\text{relação} = \text{“Filho”})\ .\text{and.}\ (\text{sexo} = \text{“Feminino”})}(\text{DEPENDENTES})$$

gera a seguinte tabela como resultado:

Tabela consulta2				
RG Responsável	Nome Dependente	Dt. Nascimento	Relação	Sexo
30303030	*Adreia*	*01/05/90*	*Filho*	*Feminino*

As operações relacionais que podem ser aplicadas na operação **select** são:

<, >, ≤ , ≥ , =, ≠

além dos operadores booleanos:

and, **or**, **not**.

A operação **select** é unária, ou seja, só pode ser aplicada a uma única relação. Não é possível aplicar a operação sobre tuplas de relações distintas.

- A Operação Project

A operação project seleciona um conjunto determinado de colunas de uma relação. A forma geral de uma operação **project** é:

$$\pi_{\text{<lista de atributos>}}(\text{<nome da relação>})$$

A letra grega π representa a operação **project**, **<lista de atributos>** representa a lista de atributos que o usuário deseja selecionar e **<nome da relação>** representa a relação sobre a qual a operação **project** será aplicada.

Exemplos:

$$\textbf{consulta3} = \pi_{\text{Nome, Dt. Nascimento}}(\text{DEPENDENTES})$$

gera a seguinte tabela como resultado:

Tabela consulta3	
Nome Dependente	Dt. Nascimento
Jorge	*27/12/86*
Luiz	*18/11/79*
Fernanda	*14/02/69*
Angelo	*10/02/95*
Adreia	*01/05/90*

- Seqüencialidade de Operações

As operações **project** e **select** podem ser utilizadas de forma combinada, permitindo que apenas determinadas colunas de determinadas tuplas possam ser selecionadas.

A forma geral de uma operação sequencializada é:

$$\pi_{\text{<lista de atributos>}}\left(\sigma_{\text{<condição de seleção>}}(\text{<nome da relação>})\right)$$

Veja o seguinte exemplo:

$$\textbf{consulta4} = \pi_{\text{nome, depto., salario}}\left(\sigma_{\text{salario} < 2.500,00}(\text{EMPREGADO})\right)$$

produz a tabela a seguir como resultado:

Tabela consulta4		
Nome	Depto.	Salário
Ricardo	*2*	*2.300,00*
Renato	*3*	*1.300,00*

A **consulta4** pode ser reescrita da seguinte forma:

consulta5 = $\sigma_{\text{salario} < 2.500,00}$(EMPREGADO)

Tabela consulta5					
Nome	RG	CIC	Depto.	RG Supervisor	Salário
Ricardo	*30303030*	*33333333*	*2*	*10101010*	*2.300,00*
Renato	*50505050*	*55555555*	*3*	*20202020*	*1.300,00*

consulta6 = $\pi_{\text{nome, depto., salario}}$ (CONSULTA5)

Tabela consulta6		
Nome	Depto.	Salário
Ricardo	*2*	*2.300,00*
Renato	*3*	*1.300,00*

porém é mais elegante utilizar a forma descrita na **consulta4**.

- Operações Matemáticas

Levando em consideração que as relações podem ser tratadas como conjuntos, podemos então aplicar um conjunto de operações matemáticas sobre as mesmas. Estas operações são: **união** ($\cup$) , **intersecção** ($\cap$) e **diferença** ($-$). Este conjunto de operações não é unário, ou seja, podem ser aplicadas sobre mais de uma tabela, porém, existe a necessidade das tabelas possuirem tuplas exatamente do mesmo tipo.

Estas operações podem ser definidas da seguinte forma:

- **união** - o resultado desta operação representada por $\mathbf{R \cup S}$ é uma relação **T** que inclui todas as tuplas que se encontram em **R** e todas as tuplas que se encontram em **S**;

- **intersecção** - o resultado desta operação representada por $\mathbf{R \cap S}$ é uma relação **T** que inclui as tuplas que se encontram em **R** e em **S** ao mesmo tempo;

- **diferença** - o resultado desta operação representada por $\mathbf{R - S}$ é uma relação **T** que inclui todas as tuplas que estão em **R** mas não estão em **S**.

Leve em consideração a seguinte consulta:

Selecione todos os empregados que trabalham no departamento número 2 ou que supervisionam empregados que trabalham no departamento número 2.

Vamos primeiro selecionar todos os funcionários que trabalham no departamento número 2.

consulta7 = $\sigma_{depto = 2}$ (EMPREGADOS)

Tabela consulta7					
Nome	RG	CIC	Depto.	RG Supervisor	Salário
Fernando	*20202020*	*22222222*	*2*	*10101010*	*2.500,00*
Ricardo	*30303030*	*33333333*	*2*	*10101010*	*2.300,00*
Jorge	*40404040*	*44444444*	*2*	*20202020*	*4.200,00*

Vamos agora selecionar os supervisores dos empregados que trabalham no departamento número 2.

consulta8 = $\pi_{rg\_supervisor}$ (CONSULTA7)

Tabela consulta8
RG Supervisor
10101010
20202020

Vamos projetar apenas o rg dos empregados selecionados:

consulta9 = π_{rg}(CONSULTA7)

Tabela consulta9
RG
20202020
30303030
40404040

E por fim, vamos unir as duas tabelas, obtendo o resultado final.

consulta10 = CONSULTA8 $\cup$ CONSULTA9

Tabela consulta10
RG
20202020
30303030
40404040
10101010

Leve em consideração a próxima consulta:

selecione todos os empregados que desenvolvem algum projeto e que trabalham no departamento número 2;

Vamos primeiro selecionar todos os empregados que trabalham em um projeto.

consulta11 = $\pi_{rg\_empregado}$ (EMPREGADO/PROJETO)

Tabela consulta11
RG_Empregado
20202020
30303030
40404040
50505050

Vamos agora selecionar todos os empregados que trabalham no departamento 2.

consulta12 = π_{rg} ($\sigma_{depto = 2}$ (EMPREGADOS))

Tabela consulta12
RG
20202020
30303030
40404040

Obtemos então todos os empregados que trabalham no departamento 2 e que desenvolvem algum projeto.

consulta13 = CONSULTA11 ∩ CONSULTA12

Tabela consulta13
RG
20202020
30303030
40404040

Leve em consideração a seguinte consulta:

selecione todos os usuários que não desenvolvem projetos;

consulta14 = $\pi_{rg\_empregado}$ (EMPREGADO/PROJETO)

Tabela consulta14
RG Empregado

20202020
30303030
40404040
50505050

consulta15 = π_{rg} (EMPREGADOS)

Tabela consulta15
RG
10101010
20202020
30303030
40404040
50505050

consulta16 = CONSULTA15 – CONSULTA14

Tabela consulta16
RG
10101010

- Produto Cartesiano

O **produto cartesiano** é uma operação binária que combina todas as tuplas de duas tabelas. Diferente da operação **união**, o **produto cartesiano** não exige que as tuplas das tabelas possuam exatamente o mesmo tipo. O **produto cartesiano** permite então a consulta entre tabelas relacionadas utilizando uma condição de seleção apropriada. O resultado de um **produto cartesiano** é uma nova tabela formada pela combinação das tuplas das tabelas sobre as quais aplicou-se a operação.

O formato geral do **produto cartesiano** entre duas tabelas **R** e **S** é:

R X S

Leve em consideração a seguinte consulta:

encontre todos os funcionários que desenvolvem projetos em Campinas;

consulta16 = EMPREGADO/PROJETO X PROJETO

Tabela consulta16				
RG Empregado	Número Projeto	Nome	Número	Localização

20202020	*5*	*Financeiro 1*	*5*	*São Paulo*
20202020	*5*	*Motor 3*	*10*	*Rio Claro*
20202020	*5*	*Prédio Central*	*20*	*Campinas*
20202020	*10*	*Financeiro 1*	*5*	*São Paulo*
20202020	*10*	*Motor 3*	*10*	*Rio Claro*
20202020	*10*	*Prédio Central*	*20*	*Campinas*
30303030	*5*	*Financeiro 1*	*5*	*São Paulo*
30303030	*5*	*Motor 3*	*10*	*Rio Claro*
30303030	*5*	*Prédio Central*	*20*	*Campinas*
40404040	*20*	*Financeiro 1*	*5*	*São Paulo*
40404040	*20*	*Motor 3*	*10*	*Rio Claro*
40404040	*20*	*Prédio Central*	*20*	*Campinas*
50505050	*20*	*Financeiro 1*	*5*	*São Paulo*
50505050	*20*	*Motor 3*	*10*	*Rio Claro*
50505050	*20*	*Prédio Central*	*20*	*Campinas*

Vamos agora selecionar as tuplas resultantes que estão devidamente relacionadas que são as que possuem o mesmo valor em número do projeto e número e cuja localização seja 'Campinas'.

$$\textbf{consulta17} = \pi_{\text{rg\_empregado, número}} (\sigma_{\text{(número\_projeto = número) .and. (localização = 'Campinas')}}^{(\text{CONSULTA16})})$$

Tabela consulta17	
RG	Número
40404040	*20*
50505050	*20*

A operação **produto cartesiano** não é muito utilizada por não oferecer um resultado otimizado. Veja o ítem seguinte.

- Operação Junção

A operação **junção** atua de forma similar á operação **produto cartesiano**, porém, a tabela resultante conterá apenas as combinações das tuplas que se relacionam de acordo com uma determinada condição de junção. A forma geral da operação **junção** entre duas tabelas **R** e **S** é a seguinte:

$$\mathbf{R} \bowtie_{\text{<condição de junção>}} \mathbf{S}$$

Leve em consideração a consulta a seguir:

encontre todos os funcionários que desenvolvem projetos em Campinas;

consulta18 = EMPREGADOS/PROJETOS$\bowtie$ número_projeto = número PROJETOS

Tabela **consulta18**				
RG_Empregado	Número_Projeto	Nome	Número	Localização
20202020	*5*	*Financeiro 1*	*5*	*São Paulo*
20202020	*10*	*Motor 3*	*10*	*Rio Claro*
30303030	*5*	*Financeiro 1*	*5*	*São Paulo*
40404040	*20*	*Prédio Central*	*20*	*Campinas*
50505050	*20*	*Prédio Central*	*20*	*Campinas*

consulta19 = $\sigma_{\text{localização = 'Campinas'}}$ (CONSULTA18)

Tabela **consulta18**				
RG_Empregado	Número_Projeto	Nome	Número	Localização
40404040	*20*	*Prédio Central*	*20*	*Campinas*
50505050	*20*	*Prédio Central*	*20*	*Campinas*

5 - SQL - Structured Query Language

SQL é um conjunto de declarações que é utilizado para acessar os dados utilizando gerenciadores de banco de dados. Nem todos os gerenciadores utilizam SQL. SQL não é uma linguagem procedural pois processa conjuntos de registros, ao invés de um por vez, provendo navegação automática através dos dados, permitindo ao usuário manipular tipos complexos de dados. SQL pode ser utilizada para todas as atividades relativas a um banco de dados podendo ser utilizada pelo administrador de sistemas, pelo DBA, por programadores, sistemas de suporte à tomada de decisões e outros usuários finais.

❖ Comandos para Definição de Dados Utilizando SQL (Data Definition Language)

- Comando CREATE TABLE – Criar tabela.

O comando **create table** permite ao usuário criar uma nova tabela (ou relação). Para cada atributo da relação é definido um nome, um tipo, máscara e algumas restrições. Os tipos de uma coluna são:

- **char(*n*)**: caracteres e strings onde ***n*** é o número de caracteres;
- **integer**: inteiros
- **float**: ponto flutuante;
- **decimal(*m*,*n*)**: onde ***m*** é o número de casas inteiras e ***n*** o número de casas decimais.

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.

A forma geral do comando **create table** então é:

create table <nome_tabela> (<nome_coluna$_1$> <tipo_coluna$_1$> <NOT NULL>,
<nome_coluna$_2$> <tipo_coluna$_2$> <NOT NULL>,
:
<nome_coluna$_n$> <tipo_coluna$_n$> <NOT NULL>
);

Por exemplo, para criar a tabela EMPREGADOS do apêndice A, teríamos o seguinte comando:

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

- Comando DROP TABLE – Apagar a estrutura da tabela com os dados.

O comando **drop table** permite a exclusão de uma tabela (relação) em um banco de dados.

A forma geral para o comando **drop table** é:

drop table <nome_tabela>;

Por exemplo, para eliminar a tabela EMPREGADOS do apêndice A 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 diversos tabelas que devem ser devidamente corrigidas.

Este processo não é assim tão simples pois, como vemos neste caso, a exclusão da tabela EMPREGADOS implica na alteração do projeto físico de diversas tabelas. Isto acaba implicando na construção de uma nova base de dados.

- Comando ALTER TABLE – Alterar a estrutura da tabela.

O comando **alter table** 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 <nome_tabela> add <nome_coluna> <tipo_coluna>;

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 o mesmo em todas as tuplas da tabela receberão o valor NULL.

Comandos para Manipulação de Dados (DML - Data Manipulation Language)

❖ Consultas em SQL

- O comando SELECT – Selecionar registros.

O comando **select** permite a seleção de tuplas e atributos em uma ou mais tabelas. A forma básica para o uso do comando **select** é:

select <*lista de atributos*>
from <*lista de tabelas*>
where <*condições*>*;*

Por exemplo, para selecionar o nome e o rg dos funcionários que trabalham no departamento número 2 na tabela EMPREGADOS utilizamos o seguinte comando:

```
select nome, rg
from EMPREGADOS
where depto = 2;
```

obteremos então o seguinte resultado:

Nome	***RG***
Fernando	*20202020*
Ricardo	*30303030*
Jorge	*40404040*

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

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

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:

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

A operação *select-from-where* em SQL pode envolver quantas tabelas forem necessárias. Leve em consideração a seguinte consulta:

selecione o número do departamento que controla projetos localizados em Rio Claro;

```
select t1.numero_depto
from departamento_projeto t1, projeto 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 isto pode ficar cansativo em consultas muito complexas além do que, impossibilitaria a utilização da mesma tabela mais que uma vez em uma expressão SQL. Considere a seguinte consulta:

selecione o nome e o rg de todos os funcionários que são supervisores;

```
select e1.nome, e1.rg
from empregado e1, empregado e2
where e1.rg = e2.rg_supervisor;
```

que gera o seguinte resultado:

Nome	***RG***
João Luiz	*10101010*
Fernando	*20202020*

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

$$\pi_{\text{nome, rg}} (\bowtie \text{EMPREGADOS}_{\text{tg\_t1 = rg\_supervisor\_t2}} \text{EMPREGADOS});$$

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. Desta forma, a expressão:

```
select *
from empregados;
```

gera o seguinte resultado:

Nome	***RG***	***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*
Jorge	*40404040*	*44444444*	*2*	*20202020*	*4.200,00*
Renato	*50505050*	*55555555*	*3*	*20202020*	*1.300,00*

Diferente de álgebra relacional, a operação *select* em SQL permite a geração de tuplas duplicadas como resultado de uma expressão. Para evitar isto, devemos utilizar o especificador **distinct**. Veja a seguir os exemplos com e sem o especificador **distinct**.

```
select depto
from empregado;
```

```
select distinct depto
from empregado;
```

que gera os seguintes resultados:

Depto.
1

Depto.
1

2
2
2
3

2
3

Podemos gerar consultas aninhadas em SQL utilizando o especificador **in**, que faz uma comparação do especificador **where** da consulta mais externa com o resultado da consulta mais interna. Considere a consulta a seguir:

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');
```

Para selecionar um conjunto de tuplas de forma ordenada devemos utilizar o comando **order by**. Leve em consideração a seguinte consulta:

selecione todos os empregados por ordem alfabética:

```
select nome, rg, depto
from empregado
order by nome;
```

- Inserções e Atualizações – Inserir registros.

Para elaborar inserções em SQL, utiliza-se o comando **insert into**. A forma geral para o comando **insert into** é:

```
insert into    <nome da tabela> <(lista de colunas)>
values     <(lista de valores)>;
```

Considere a seguinte declaração:

insira na tabela empregados, os seguintes dados:

nome: *Jorge Goncalves*

rg: *60606060*

cic: *66666666*

departamento: *3*

rg_supervisor: *20202020*

salário: *R$ 4.000,00*

```
insert into    empregados
values     ('Jorge Goncalves', '60606060', '66666666', 3, '20202020', 4000,00);
```

ou ainda:

insira na tabela empregados os seguintes dados:

nome: *Joao de Campos*

rg: *70707070*

cic: *77777777*

departamento: *3*

salário: *R$2.500,00*

insert into *empregados (nome, rg, cic, depto, salario)*
values ('*Joao de Campos*', '*70707070*', '*77777777*', 3, 2500,00);

Como na primeira inserção todos os campos foram inseridos, então não foi necessário especificar o nome das colunas. Porém, na segunda inserção, o campo *rg_supervisor* não foi inserido, então especificou-se as colunas. Outra forma de se elaborar esta inserção seria:

insert into empregados values ('Joao de Campos*, '*70707070*', '*77777777*', 3, '', 2500,00);*

Neste caso, utilizou-se os caracteres **''** para se declarar que um valor nulo seria inserido nesta coluna.

Para se efetuar uma alteração em uma tabela, é utilizado o comando **update**. A forma geral do comando **update** é:

update <*tabela*>
set <*coluna*> = <*expressão*>
where <*condição*>

Considere a seguinte declaração:

atualize o salário de todos os empregados que trabalham no departamento 2 para R$ 3.000,00;

update *empregado*
set *salario = 3.000,00*
where *depto = 2;*

Para se eliminar uma tupla de uma tabela, utiliza-se o comando **delete**. A forma geral do comando **update** é:

delete from <*tabela*>
where <*condição*>*;*

Leve em consideração a seguinte expressão:

elimine os registros nos quais o empregado trabalhe no departamento 2 e possua salário maior que R$ 3.500,00;

delete from *empregado*
where *salario > 3.500,00 and depto = 2;*

Nos casos de atualização que foram vistos, todas as <*condições*> podem ser uma consulta utilizando o comando **select**, onde o comando será aplicado sobre todos os registros que satisfizerem as condições determinadas pelo comando de seleção.

6 - Normalização

A normalização é uma metodologia para projeto de BDs relacionais, uma vez que sugere uma organização de dados em tabelas. Assim sendo, a normalização é uma ferramenta para projeto lógico de BDs relacionais.

O objetivo desta técnica é evitar problemas de redundância e anomalias de atualização, que podem estar presentes em uma relação. A solução para resolver estes problemas é a decomposição de uma relação em uma ou mais relações, com base na aplicação de certas regras de normalização (formas normais). Esta proliferação de relações nem sempre é ideal do ponto de vista de performance, devendo ser balanceado vantagens e desvantagens antes da efetivação dos resultados de uma forma normal (FN).

Temos ao todo, cinco formas normais, porém, se diz que ao passar pela terceira forma normal, a maioria dos bancos já estará normalizado. Vejamos quais são:

1) Uma representação de tabela não normalizada:

CódProj	Tipo	D
LSC001	Novo Desenv	Si

→ Documento exemplo na forma não normalizada

Esquema de tabelas relacionais:

Proj (codproj, tipo, descr, (codemp, nome, cat, sal, dataini, tempal))

O parêntese apresenta aninhamento de tabelas. No caso de tabelas aninhadas, as colunas sublinhadas indicam a coluna ou grupo de colunas que servem para distinguir diferentes linhas da tabela aninhada referente a uma linha da tabela de seu nível externo. Assim, a coluna CodProj distingue as linhas (cada uma referente a um projeto) da tabela PROJ. Já a coluna CodEmp, serve para distinguir diferentes linhas de empregado dentro do grupo referente a um projeto.

2) Passagem à primeira forma normal (1FN)

Diz-se que uma tabela está na primeira forma normal, quando ela não contém tabelas aninhadas.

O próximo passo da normalização consta da transformação do esquema de tabela não normalizada em um esquema relacional na primeira forma normal. Uma tabela encontra-se na primeira forma normal quando ela não contém tabelas aninhadas. Portanto, a passagem à primeira forma normal consta da eliminação das tabelas aninhadas eventualmente existentes.

Para transformar um esquema de tabela não-normalizada em um esquema da 1FN há duas alternativas:

- ✓ Construir uma única tabela com redundância de dados.

Cria-se uma tabela na qual os dados das linhas externas à tabela aninhada são repetidos para cada linha da tabela aninhada. No caso que estamos analisando seria o seguinte:

PROJEMP (CodProj, Tipo, Descr, CodEmp, Nome, Cat, Sal, DataIni, TempAl).

Neste caso, os dados aparecem repetidos para cada linha de empregados.

- ✓ Construir uma tabela para cada tabela aninhada.

Cria-se uma tabela referente a própria tabela que está sendo normalizada e uma tabela para cada tabela aninhada. O resultado seria o seguinte:

PROJ(CodProj, Tipo, Descr)

PROJEMP (CodProj, CodEmp, Nome, Cat, Sal, DataIni, TempAl).

Ao se decompor desta forma, em duas tabelas, as relações entre as informações podem ser perdidas.

Na decomposição de tabelas, a passagem à primeira forma normal por decomposição de tabelas é feita nos seguintes passos:

1. É criada uma tabela na 1FN referente a tabela não normalizada e que contém as apenas colunas com valores atômicos, isto é, sem as tabelas aninhadas. A chave primária da tabela na 1FN é idêntica a chave da tabela não normalizada.

2. Para cada tabela aninhada, é criada uma tabela na 1FN composta pelas seguintes colunas:
   a. A chave primária de cada uma das tabelas na qual a tabela em questão está aninhada.
   b. As colunas da própria tabela aninhada.
3. São definidas as chaves primárias das tabelas da 1FN que correspondem as tabelas aninhadas.

Portanto, neste exemplo, a decomposição gera duas tabelas com o seguinte esquema:

1FN

Proj(CodProj, Tipo, Descr)

ProjEmp(CodProj, CodEmp, Nome, Cat, Sal, DataIni, TempAl).

PROJ

CodProj	Tipo	Descr
LSC001	Novo Desenv.	Sistema de Estoque
PAG002	Manutenção	Sistema de RH

PROJEMP

CodProj	CodEmp	Nome	Cat	Sal	DataIni	TempAl
LSC001	2146	João	A1	4	1/11/91	24
LSC001	3145	Silvio	A2	4	2/10/91	24
PAG002	4112	João	A2	4	4/01/91	24
PAG002	6126	José	B1	9	1/11/92	12

Dependência Funcional

Dizemos que determinada coluna é dependente funcionalmente de outra coluna desde que a primeira coluna determine a existência da segunda. Por exemplo, salário é dependente funcionalmente de código do funcionário. Isso ocorre já que, sem funcionário não há salário.

3) Passagem à segunda forma normal (2FN)

A passagem à segunda forma normal objetiva eliminar um certo tipo de redundância de dados. Uma tabela encontra-se na segunda forma normal quando, além de encontrar-se na primeira forma normal, cada coluna não chave depende da chave primária completa. Uma tabela que não se encontra na segunda forma contém dependências funcionais parciais, ou seja, contém colunas não chave que dependem apenas de uma parte da chave primária.

→ Uma tabela encontra-se na 2FN quando, além de estar na 1FN, não contém dependências parciais.

→ Dependência parcial: Uma dependência funcional parcial ocorre quando uma coluna depende apenas de parte da chave primária composta.

Portanto, toda tabela que está na 1FN e que possui apenas uma coluna como chave primária já está na 2FN. O mesmo aplica-se para uma tabela que contenha apenas colunas chave primária.

No nosso exemplo, a tabela PROJEMP deve ser examinada para procurar depend6encias parciais, pois possui uma chave primária composta. As colunas Nome, Cat e Sal dependem, cada uma, apenas da coluna CodEmp, já que, nome, categoria funcional e salário são determinados somente pelo código do empregado. Porém, DataIni e TempAl dependem da chave primária completa.

Para passarmos o esquema PROJ e PROJEMP para a segunda forma normal, vamos dividir a tabela PROJEMP em duas tabelas:

PROJ: (CodProj, Tipo, Descr)
PROJEMP: (CodProj, CodEmp, DataIni, TempAl)
EMP: (CodEmp, Nome, Cat, Sal)

PROJ

CodProj	Tipo	Descr
LSC001	Novo Desenv.	Sistema de Estoque
PAG002	Manutenção	Sistema de RH

PROJEMP

CodProj	CodEmp	DataIni	TempAl
LSC001	2146	1/11/91	24
LSC001	3145	2/10/91	24
PAG002	4112	4/01/91	24
PAG002	6126	1/11/92	12

EMP

CodEmp	Nome	Cat	Sal
2146	João	A1	4
3145	Silvio	A2	4
4112	João	A2	4
6126	José	B1	9

Portanto, a passagem para a segunda forma normal obedece a alguns passos importantes:

1) Copiar para a 2FN cada tabela que tenha chave primária simples ou que não tenha colunas além da chave.
2) Para cada tabela com chave primária composta e com pelo menos uma coluna não chave:
    a. Criar na 2FN uma tabela com as chaves primárias da tabela na 1FN.
    b. Para cada coluna não chave fazer a seguinte pergunta:
        i. A coluna depende de toda chave ou de apenas parte dela?

Caso a coluna dependa de toda a chave completa na 2FN (a coluna DataIni e TempAl) → Criar a coluna correspondente na tabela com a chave completa na 2FN.
Caso a coluna dependa apenas de parte da chave (as colunas, nome, sal e cat da tabela ProjEmp) → Criar, caso ainda não existir, uma tabela na 2FN que tenha como chave primária a parte da chave que é determinante da coluna em questão (tabela EMP).
Criar a coluna dependente dentro da tabela na 2FN (Colunas nome, sal e Cat).

4) Passagem à terceira forma normal (3FN)

A passagem à terceira forma normal objetiva eliminar um outro tipo de redundância. Vamos supor que o salário do funcionário seja determinado pela sua categoria funcional. Neste caso, a informação de que salário é pago à uma categoria funcional está representado redundantemente na tabela. Tantas vezes quantos empregados possui a categoria funcional. A passagem à 3FN objetiva eliminar este tipo de redundância de dados.

Uma tabela encontra-se na 3FN quando, além de estar na 2FN, toda coluna não chave depende diretamente de chave primária, isto é, quando não há dependências funcionais transitivas ou indiretas. Uma dependência funcional transitiva ou indireta acontece quando uma coluna não chave primária depende funcionalmente de outra coluna ou combinação de colunas não chave primária.

Dependência transitiva: Uma dependência funcional transitiva ocorre quando uma coluna, além de depender da chave primária da tabela, depende de outra coluna ou conjunto de colunas da tabela.

3FN
PROJ(CodProj, Tipo, Descr)
PROJEMP: (CodProj, CodEmp, DataIni, TempAl)
EMP: (CodEmp, Nome, Cat)
CAT(Cat, Sal)

PROJ

CodProj	Tipo	Descr
LSC001	Novo Desenv.	Sistema de Estoque
PAG002	Manutenção	Sistema de RH

PROJEMP

CodProj	CodEmp	DataIni	TempAl
LSC001	2146	1/11/91	24
LSC001	3145	2/10/91	24
PAG002	4112	4/01/91	24
PAG002	6126	1/11/92	12

EMP

CodEmp	Nome	Cat
2146	João	A1
3145	Silvio	A2
4112	João	A2
6126	José	B1

CAT

Cat	Sal
A1	4
A2	4
A2	4
B1	9

Portanto, a passagem da 2FN para a 3FN é o seguinte:

1) Para tabelas com duas ou mais colunas não chave.

   a) Criar uma tabela no esquema 3FN com a chave primária da tabela em questão.
   b) Para cada coluna não chave fazer a seguinte pergunta:

   "a coluna depende de alguma outra coluna não chave"

   Caso a coluna dependa apenas da chave: Copiar a coluna para a tabela 3FN.

   Caso a coluna depender da outra coluna:
   - Criar, caso ainda não exista, uma tabela no esquema na 3FN que tenha como chave primária a coluna da qual há a dependência indireta.
   - Copiar a coluna dependente para a tabela criada.
   - A coluna determinante deve permanecer também na tabela original.

5) Passagem à quarta forma normal (4FN)

Para a maioria dos documentos e arquivos, a decomposição até a 3FN é suficiente para obter um esquema de banco de dados correspondente ao documento.

Existem outras formas normais, como a forma normal de Boyce/Codd, a 4FN e a 5FN. Destas a única que teria uma importância relevante seria a quarta forma normal. Porém, como a grande maioria dos bancos de dados estarão normalizados até a terceira forma normal, sugiro que o acadêmico procure complementar seus conhecimentos, se necessário, pesquisando na bibliografia indicada pela disciplina.

6) Problemas da Normalização

A normalização pode apresentar alguns problemas, porém, acredito que o maior deles está realmente no próprio profissional, que exagera na hora de normalizar as tabelas, criando um problema maior para o desenvolvedor de aplicações ou analista, e até mesmo para o usuário, pois, o excesso de normalização leva à lentidão nas consultas e relatórios.

☺ ☺ ☺ ▪ ▪ ▪ ▪ ▪ 🖫 🖫 ☺ ☺ ☺

Apêndice A - Exemplo de um Banco de Dados

Tabela EMPREGADO					
Nome	RG	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*
Jorge	*40404040*	*44444444*	*2*	*20202020*	*4.200,00*
Renato	*50505050*	*55555555*	*3*	*20202020*	*1.300,00*

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

Tabela PROJETO		
Nome	Número	Localização
Financeiro 1	*5*	*São Paulo*
Motor 3	*10*	*Rio Claro*
Prédio Central	*20*	*Campinas*

Tabela DEPENDENTES				
RG Responsável	Nome Dependente	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	Adreia	01/05/90	Filho	Feminino

Tabela DEPARTAMENTO_PROJETO	
Número Depto.	Número Projeto
2	*5*
3	*10*
2	*20*

Tabela EMPREGADO_PROJETO		
RG Empregado	Número Projeto	Horas
20202020	*5*	*10*
20202020	*10*	*25*
30303030	*5*	*35*
40404040	*20*	*50*
50505050	*20*	*35*

❖

7 - ARTIGOS ANEXOS e Materiais Complementares

1) Bancos de Dados Livre x Pago

Software livre é uma realidade cada vez mais constante no mundo da informática. Com o inegável sucesso do Linux, a comunidade de informática passou a notar outros nichos que poderiam ser explorados pelo software livre.

Com banco de dados não poderia ser diferente. Com um mercado de sistemas pagos bastante consolidado, incluindo gigantes como IBM, Microsoft e Oracle, os bancos de dados livres começam a mostrar sua força. O fato é que, como aconteceu com o Linux, o banco de dados relacional está se tornando uma commodity, seja pelo aumento de bons produtos no mercado, seja pela concorrência dos softwares livres. Vamos analisar neste artigo quais são os principais bancos de dados livres e em que eles podem concorrer de fato com os gigantes do mercado.

Inicialmente é necessário diferenciar o uso que se fará do banco de dados. Já há algum tempo os bancos de dados deixaram de ser simples armazenadores de informações. Hoje em dia, pelo menos para os principais produtos do mercado, há uma vasta gama de serviços agregados a eles. Estruturas de dados cada vez mais complexas estão sendo geridas e armazenadas pelos bancos de dados para atender necessidades específicas, em especial para internet.

O padrão para manipulação da informação nos bancos de dados relacionais é o SQL (Structured Query Language). O primeiro padrão surgiu em 1986 e foi definido pela ANSI (American National Standards Institute). Ficou conhecido como SQL-86. Em 1989, surge a nova versão com modificações significativas (SQL-89). Em 1992 chegou-se a um padrão de regras básicas que definem os bancos de dados relacionais. A maior parte dos bancos de dados em uso atualmente é, em alguma escala, compatível com esta versão. Em 1999 houve a última alteração deste padrão. É conhecida como SQL-3 ou SQL-99. Houve importantes avanços nesta última versão. Algumas das mudanças mais significativas estão relacionadas com a definição dos padrões para bancos de dados Objeto-Relacionais.

Tanto os bancos de dados pagos como os livres adotam este mesmo padrão. Claro que isso não quer dizer que tanto faz utilizar um ou outro banco de dados. O que diferencia de fato o banco de dados é a forma como a informação armazenada é tratada . Cada software possui um gerenciamento diferenciado dos dados, utilizando metodologias de replicação, garantia de integridade das informações e controle de transações. Outros são mais simples, baseados em arquivos de dados. Alguns possuem linguagens de programação completas e complexas, enquanto outros não as possuem.

Portanto, dizer que os banco de dados seguem o padrão SQL não quer dizer que todos eles possuem as mesmas características. Geralmente, o termo SQL é apenas um indicativo que, sabendo-se o padrão, será fácil para o usuário implementar soluções personalizadas em diversos bancos de dados.

Alguns fatores influenciam na escolha de um banco de dados:

a.) **Controle de redundância:** o banco de dados deve ser capaz de garantir que os dados não tenham duplicidade. Isso normalmente é conhecido como integridade referencial. Desta forma, não seria possível incluir dois registros com o

mesmo código (chave primária). Também não seria possível excluir um registro que tivesse relacionamento com outras tabelas (chave estrangeira). Esta integridade é a base do modelo relacional, portanto é necessário que o banco de dados tenha a capacidade de gerenciar o controle de redundância.

b.) **Compartilhamento de dados:** a informação deve estar disponível para qualquer número de usuários de maneira rápida, concomitante e segura. É impensável, nos dias atuais, imaginar um banco de dados exclusivo para um usuário. A informação, cada vez mais, deve ser compartilhada por diversas pessoas da empresa. Disponibilizar a informação com rapidez e segurança é requisito fundamental para determinar a escolha do banco de dados.

c.) **Controle de acesso:** é essencial saber quem fez e o que cada usuário pode fazer dentro do banco de dados. Disponibilizar a informação é pouco. Deve haver controle sobre o que é disponibilizado. Deve-se analisar as possibilidades de controle de acesso às tabelas e colunas do banco de dados e às operações que cada usuário pode realizar (inclusão, alteração, consulta ou exclusão).

d.) **Cópias de Segurança:** deve haver rotinas específicas para realizar cópias de segurança dos dados armazenados.

e.) **Suporte às Transações:** as transações são originadas em qualquer operação que seja feita nos dados armazenados. Realizar o controle sobre essas transações, garantindo a integridade das informações armazenadas mesmo quando há diversos usuários realizando operações ao mesmo tempo, é uma necessidade cada vez mais importante para os bancos de dados. Há diversos níveis para o controle de transações. O mínimo necessário para os dias atuais é o bloqueio por linha, ou seja, cada alteração bloqueará apenas uma linha no banco de dados. Com isso, há uma maior disponibilidade da informação armazenada visto que poucas linhas estarão efetivamente bloqueadas por transações pendentes.

f.) **Suporte à programação:** mesmo para quem utiliza arquitetura de desenvolvimento em três ou mais camadas, algumas operações continuam sendo mais rápidas se forem realizadas diretamente no banco de dados. Com isso, o banco de dados deve possuir uma linguagem de programação que permita realizar rotinas específicas diretamente sobre ele. Além disso, muitas regras de negócio são implementadas diretamente no banco de dados. Por exemplo, realizar o pedido de compra toda vez que o estoque do produto chegar ao nível mínimo ou bloquear a venda, caso o cliente esteja comprando além do seu limite de crédito.

g.) **Recuperação:** falhas acontecem. O que fazer quando houver uma quebra total do banco de dados? Qual o caminho para recuperação deste desastre? Claro que um bom backup pode resolver boa parte dos problemas, mas o que, além disso, o banco de dados poderá fazer? Mecanismos de backup online e em diversos servidores e clusters, entre outros, são ferramentas importantes quando um problema acontece.

h.) **Desempenho:** de nada adianta ter um banco de dados completo se este for lento para as necessidades da empresa. O desempenho do banco de dados muitas vezes pode ser melhorado com técnicas de tunning (ajuste) realizadas diretamente no banco. Convém ter certeza de que o banco de dados permite realizar estes ajustes, se eles podem ser realizados e em qual escala.

i.) **Escalabilidade:** é necessário saber os limites do banco de dados. Convém, principalmente para os bancos de dados livres, considerar o tamanho máximo do banco de dados e o número máximo de linhas em cada tabela. Conhecer casos de sucesso é fundamental para determinar se o banco de dados está dentro da necessidade da empresa.

Algumas opções do mercado de banco de dados livre

O mais popular banco de dados livre do mercado é o MySQL (www.mysql.com). Milhares de sites na internet são mantidos neste banco de dados. Simples e poderoso, o MySQL é um sucesso inegável para empresas de diversos portes.

A produtora do software é a MySQL AB. O MySQL pode ser obtido gratuitamente no site ou ser adquirido por preços bem mais baixos que os concorrentes. Este banco de dados segue o padrão ANSI SQL-99 e, como os demais, possui extensões à linguagem padrão. A nova versão do produto possui suporte às transações e integridade referencial. Também é possível realizar replicação de dados e ainda possui um mecanismo de cache de buscas para agilizar as consultas às tabelas. Contudo, um dos recursos mais importantes para banco de dados ainda está ausente no MySQL: triggers (gatilhos), procedimentos e funções armazenadas e views (visões). Estes recursos estão prometidos para a próxima versão do produto.

Uma outra grande vantagem do MySQL é a velocidade. Testes mostram que ele compete com os líderes do mercado. Contudo, é necessário esperar para ver o que acontecerá quando tiver todos os recursos prometidos. Afinal, um banco de dados com menos recursos, poderá ser mais rápido. A independência de plataforma é outro fator relevante. Há versões do MySQL para Linux, Windows, Novell Netware, FreeBSD, Mac OS, AIX, HP-UX e Sun, dentre outros. É um produto que, devido ao crescente uso no mundo todo, mesmo tendo um longo caminho a percorrer até chegar a oferecer os mesmos recursos dos outros gigantes do mercado de banco de dados, tem um futuro garantido.

Outro grande concorrente dos bancos de dados livres é o PostgreSQL (www.postgresql.com). Da mesma forma que o MySQL, o PostgreSQL é compatível com o padrão SQL-99 e possui suas próprias extensões à linguagem. Também pode ser obtido da internet e é totalmente grátis. É um projeto bastante maduro (consta no site que o produto existe há 16 anos) e possui recursos que somente os banco de dados pagos possuem. Além do suporte à integridade referencial, replicação de dados e controle de transações, é possível criar triggers (gatilhos), procedimentos e funções armazenados, views (visões) e até herança de objetos. Existem diversas linguagens de programação que podem ser utilizadas no PostgreSQL. Uma delas é muito semelhante ao Oracle PL/SQL. Isso diminui o tempo de aprendizado do produto. Da mesma forma que o Oracle, o PostgreSQL é um banco de dados objeto-relacional. Por esta semelhança com o banco de dados líder do mercado, o PostgreSQL tem se tornado uma alternativa bastante atraente para profissionais que dominam aquela plataforma.

Mesmo com todos estes recursos, o PostgreSQL é bastante veloz. Contudo, está disponível apenas para Unix e Linux. Pode ser instalado no Windows utilizando uma biblioteca específica ou utilizando o cygwin, produto que simula o ambiente Unix no Windows. Está prometida para a próxima versão compatibilidade nativa com o Windows. Quando isso acontecer – principalmente para os usuários Windows - por ser o banco de dados livre com maior número de recursos comparáveis aos líderes do mercado de banco de dados, o PostgreSQL estará se firmando como uma opção real de uso e substituição de outros produtos.

O terceiro dos bancos de dados livres a considerar para implantação é o Firebird (www.firebirdsql.com). Ele utiliza o padrão SQL-92 e possui recursos semelhantes aos bancos de dados pagos. Possui controle de transações, inclusive com o two fase commit (gravação em duas fases). Também está disponível em diversas

plataformas, como Linux, Windows, Solaris, HP-UX, Mac, dentre outros. É possível adquirir uma versão gratuita no site e é considerado bastante estável (versões desde 1981 com nomes diferentes). Como o PostgreSQL, o Firebird permite a criação de triggers (gatilhos) mas os procedimentos e funções são externos ao banco de dados. Isso dá a flexibilidade de utilizar diversas linguagens de programação, mas por outro lado compromete o desempenho. Sem dúvida alguma é um produto que, apesar de estar há anos no mercado, ainda tem um bom caminho a evoluir. Contudo, como vem de uma base de usuários fiéis ao InterBase, pode ser uma opção atrativa aos conhecedores deste banco de dados.

Conclusão

É inegável que o software livre veio para ficar também em banco de dados. Os produtos que existem no mercado estão provando isso. Com a natural maturidade que um projeto deste consegue com o tempo, aliado a um grupo de pessoas verdadeiramente obcecadas pela qualidade do produto, fica cada vez mais claro que as empresas que vivem de venda de banco de dados têm que começar a se preocupar.

Muitas empresas que possuem softwares proprietários alegam que os bancos de dados livres estão há anos-luz de distância dos seus produtos. A realidade não parece ser bem esta. Dependendo da necessidade pode-se utilizar um banco de dados livre sem perda de qualidade para a empresa. Se a solução for departamental, a opção do software livre é ainda mais viável. Muitas empresas estão optando por instalar versões destes bancos de dados em soluções periféricas aos sistemas principais para poder testá-los e, em caso de sucesso, migrar completamente para a nova base. Basta fazer uma visita ao site dos produtos citados acima e verificar a quantidade e o porte das empresas que estão realizando projetos-piloto com os produtos.

Este caminho é conhecido do mercado. Assim foi e está sendo com Linux. Houve um crescimento significativo de servidores Linux no mercado a ponto de a Microsoft assumir que ele é um grande competidor atualmente. É difícil vermos uma empresa de qualquer porte que não tenha um único servidor Linux instalado.

Com os bancos de dados não custa repetir o caminho. Afinal, o custo de um banco de dados é significativo para qualquer empresa e a qualidade do software livre não deixa a desejar. Se o banco de dados relacional virará uma commodity ainda não é possível precisar. Hoje em dia, como poucas empresas possuem produtos de peso para competir, os preços ainda são muito semelhantes. Quem necessita de bancos de dados para toda a organização dificilmente utilizará um dos bancos de dados livres. Contudo, empresas menores poderão fazê-lo sem grandes problemas. Aí será o momento em que as grandes empresas irão notar que o midlle e o low market existem e estão à espera de produtos com qualidade e preços competitivos.

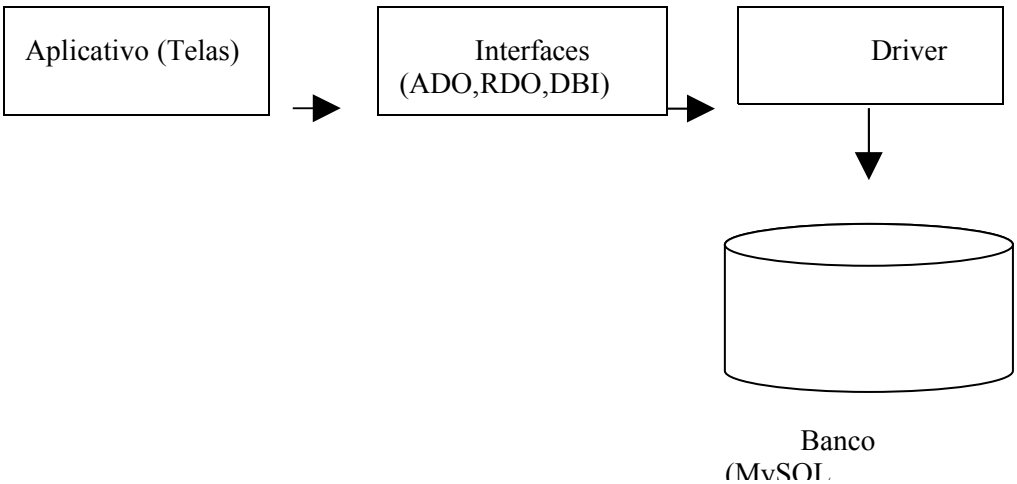
Celso Henrique Poderoso de Oliveira

henrique@nq.com.br

Economista pós-graduado em Sistemas de Informação, trabalha desde 1986 com informática e desde 1996 com banco de dados Oracle. É Gerente de TI na Net Quality Informática (www.nq.com.br), empresa desenvolvedora ERP para Construção Civil, web-commerce e professor de Banco de Dados Oracle na FIAP – Faculdade de Informática e Administração Paulista iap (www.fiap.com.br). Autor dos livros Oracle PL/SQL 8i, Oracle PL/SQL 9i, Oracle Builtins 9i e SQL Curso Prático, todos publicados pela Editora Novatec (www.novateceditora.com.br).

2) Conexão ASP MYSQL

Como já vimos, utilizar um banco de dados exige habilidades diversas. Na aula de hoje, iremos estar trabalhando um pouco com interfaces para o mysql. Vejamos como é esta estrutura:



Na criação de páginas web (html), quando desejamos que o conteúdo da página seja alterado, temos que alterar seu código. Utilizando uma linguagem para web (Asp, PHP, etc), a página é alterada automaticamente, no momento em que mudamos os dados constantes nas tabelas no banco.

Como vimos no desenho acima, isso é realizado através de uma interface e de um driver que se conectam ao banco possibilitando que as informações que montam a página sejam dinâmicas, ou seja, se modifiquem automaticamente.

Na verdade, isso funciona da seguinte forma: O aplicativo diz para a interface conectar-se ao banco de dados. O aplicativo fornece algumas informações, como o endereço e o nome do banco com que o aplicativo quer falar. A interface pega essas informações e faz uma chamada para o driver. O driver recebe essas informações e faz a conexão. Ele sabe com que porta falar e como conversar com o banco. A porta é muito semelhante ao número de telefone da pessoa com que você está tentando chamar. É uma linha de comunicação. Agora, os dados podem fluir livremente entre aplicativo (telas) e o banco de dados.

Resumidamente, necessitamos de alguns recursos para que isso funcione. Para conectarmos o ASP com MySQL necessitamos dos seguintes recursos:

1) Servidor web (PWS-Personal Web Server ou IIS – Internet Information Server) – Deve ser instalado. O pws consta no cd do Windows 98.
2) MySql – www.mysql.com
3) Interface (ADO). – nativo.
4) Driver (ODBC). Deve ser instalado.

5) Nome endereço IP do servidor web (máquina onde você instalou o pws).
6) Nome e local do banco de dados.
7) Usuário e senha.

O MySql também pode ser utilizado com a linguagem PHP, porém, esta necessita de um servidor mais apurado (apache) e complicado de configurar. Por esta razão, estaremos trabalhando com o pws e asp.

Vejamos um código ASP para conexão à um banco e consulta da tabela clientes:

```
<%

set con = Server.CreateObject("ADODB.Connection")

con.Open "DSN=localhost"

set rs = con.Execute("Select * from clientes")

while not rs.EOF

 response.write rs("Codigo") & " " & rs("Nome") & "<br>"

 rs.movenext

wend

%>
```

3 – Alguns comandos SQL

Comando	Função
Create Database nomebanco	Cria o banco de dados com o nome (nomebanco).
Drop Database nomebanco	Exclui o banco de dados com o nome (nomebanco).
Use nomebanco	Abre o banco de dados de nome nomebanco
Show Databases;	Mostra os bancos de dados do servidor.
Show tables;	Mostra as tabelas do banco selecionado.
Desc nometabela;	Mostra a estrutura da tabela informada (nometabela)
Show columns from nometabela;	Mostra a estrutura da tabela informada (nometabela)
Alter table nometabela change campo_1 campo1 varchar(20);	Altera o nome do campo campo_1 para campo1.
Alter table nometabela change nome nome varchar(50);	Altera o tipo do campo nome de varchar(20) para varchar(50).
Create table nometabela (campo1 tipo, campo2 tipo, campo3 tipo);	Cria uma tabela no banco de dados selecionado.
Ex: create table clientes (codigo integer auto_increment primary key, nome varchar(40) not null, ender varchar(40), bairro varchar(20), cidade varchar(20), estado char(2));	Cria a tabela clientes com os campos especificados. A cláusula auto_increment significa auto numeração, primary key indica chave primária e not null diz que o preenchimento do campo é obrigatório ou seja, não aceita valores nulos.
Insert into tabela (campos) values (valores)	Insere valores na tabela especificada, deve-se informar campos e valores.
Ex: insert into clientes (codigo, nome, ender, cidade, bairro, estado) values(1, “Paulo Roberto”, “Av. Renato Azeredo”, “Jardim América”, “Três Corações”, “MG”);	Insere um registro na tabela clientes
Select * from nometabela;	Seleciona (Mostra) todos campos e todos registros da tabela (nometabela)
Ex: Select * from clients order by nome;	Mostra todos campos de todos registros da tabela clients ordenados pelo nome.

Ex2: Select codigo, nome from clientes where estado=”MG”;	Mostra os campos codigo e nome da tabela clientes enquanto o estado for igual a MG.
Ex3: Select codigo, nome, Max(salário) from funcionários;	Mostra o codigo, nome e maior salário da tabela funcionários.
Update nometabela set nomecampo = valor where condição	Atualiza o campo (nomecampo) da tabela (nometabela) para valor (valor) enquanto condição
Ex: update tabprecos set precovenda=(precovenda*1,2) where tipoproduto=2;	Reajusta em 20% os preços dos produtos cujo tipo for igual a 2
Delete from nometabela where condição;	Exclui todos registros da tabela (nometabela) enquanto condição.
Ex: delete from clientes where estado=”SP”;	Exclui todos registros dos clientes cujo estado for igual a SP.
Select campo.tab1, campo.tab1, campo.tab2, campo.tab3 from tab1 inner join tab2 on campo.tab1=campo.tab2	Mostra todos registros relacionados das tabelas tab1 e tab2 desde a condição após a cláusula “on” seja satisfeita.
Select campo.tab1, campo.tab1, campo.tab2, campo.tab3 from tab1 left join tab2 on campo.tab1=campo.tab2	Mostra todos registros da tabela tab1 (left – esquerda) e os relacionados da tabelas tab2 desde a condição após a cláusula “on” seja satisfeita.
Select campo.tab1, campo.tab1, campo.tab2, campo.tab3 from tab1 right join tab2 on campo.tab1=campo.tab2	Mostra todos registros da tabela tab2 (right – direita) e os relacionados da tabelas tab1 desde a condição após a cláusula “on” seja satisfeita.

4. Tecnologias Emergentes que utilizam Bancos de Dados

Business Inteligence

O que é Business Intelligence (BI)?

Conjunto de ferramentas e aplicações que visam potenciar a utilização de informação de gestão por todos os elementos da organização de forma a melhorar os processos internos e os seus resultados.

Data Warehouse/Data Marts

Pela ótica do usuário final, Data Warehouse é um banco de dados de alta disponibilidade, com todas as informações gerenciais/estratégicas de que precisam para tomar decisões, com dados confiáveis e em formato e gregação adequados. Os dados são sempre atualizados com a freqüência correta, preservando o histórico para permitir análises comparativas e a realização de projeções.

Data Mart é uma abordagem específica do DW, focada em uma área ou departamento

Características das ferramentas de BI para usuários finais

- Acesso aos dados e análise das informações;
- Busca dos dados de várias fontes, sendo possível o correlacionamento dos mesmos;

- Apresentação dos dados já agregados, trabalhados, contextualizados permitindo, assim, um melhor entendimento do negócio e um melhor planejamento das ações.

Ferramentas de BI

- Planilhas eletrônicas;
- EIS (Executive Information System);
- Softwares de Query & Report;
- Ferramentas OLAP (On Line Analytical Processing);
- Data Mining

CRM - Customer Relationship Management

CRM é a denominação adotada para o conjunto de ferramentas e processos que envolvem as atividades de atendimento ao cliente, o registro de suas manifestações e encaminhamento destas às áreas verticais que cuidam de sua resolução, dentro ou fora da empresa, abastecendo uma base de conhecimentos deste cliente através da formação do histórico de seu relacionamento com a organização, nas suas mais variadas formas, por qualquer meio de contato.

ERP

Enterprise Resource Planning designa pacotes integrados de gestão, sistemas de informação com módulos integrados que dão suporte a diversas áreas operacionais, tais como vendas, produção, gestão de materiais, contabilidade e pessoal.

Os sistemas ERP têm como objetivo aprimorar a gestão das empresas, aumentando a produtividade dos processos operacionais, ajudando a controlar custos e reduzir despesas, e criando uma base confiável de informações gerenciais, permitindo um planejamento estratégico mais seguro.

A implantação de um sistema ERP possui diversas fases, que se iniciam com a escolha do produto mais adequado às necessidades e características de sua empresa, passam pela instalação do sistema e continuam com as atividades pós-implantação. Modificações, atualizações, novas funcionalidades, treinamento, suporte aos usuários, são algumas das atividades necessárias após a implantação do ERP. Para todas estas atividades você precisa contar com uma empresa com conhecimento e experiência.

EIS - Executive Information System

Os EIS (Executive Information Systems) - Sistemas de Informações Executivas - são sistemas

desenvolvidos para atender as necessidades dos executivos de uma empresa, de obterem informações gerenciais de uma maneira simples e rápida.

Características:

• Atender às necessidades de informações dos executivos de alto nível

• Possibilidade de customização

• Possuir recursos gráficos de alta qualidade

• Facilidade de uso

Data Warehouse

Data Warehouse (DW), que consiste em organizar os dados corporativos da melhor maneira, para dar subsídio de informações aos gerentes e diretores das empresas para tomada de decisão. Tudo isso num banco de dados paralelo aos sistemas operacionais da empresa.

Mineração de dados - Data Mining

A mineração de dados consiste no uso do computador para vasculhar imensos bancos de dados em busca de tendências que passariam despercebidas aos olhos humanos.

ASP (Application Service Provider)

Uma empresa funciona como ASP, ou Application Service Provider, quando oferece um serviço para o fornecimento, armazenamento e gerenciamento de uma solução ou software a partir de uma base centralizada, de forma que essas aplicações são acessadas remotamente pela Internet.

O que se exigir de seus fornecedores ASP?

• Desempenho

• Segurança

• Flexibilidade para implantar novas tecnologias

Data Center

Os Data centers são instalações compostas de centenas de servidores que oferecem serviços de gerenciamento do ambiente, alta disponibilidade 24x7(24 horas sete dias por semana 365 dias por ano), planejamento de capacidade, avaliação de performance de sistema operacional e aplicativos, monitoramento, backup, balanceamento de carga - entre outros -, para a hospedagem de dados/aplicações, e-commerce, sites, etc.

5. Bibliografia

RAMEZ Elmasri, Shamkant Navathe; **Fundamentals of Database Systems**; The Benjamin CummingsPublishing Company; 1989;

HENRY F. Korth, Abraham Silberschatz; **Sistema de Banco de Dados**; Makron Books; 1995;

SQL Language - Oracle Reference Manual; Version 7.2;

http://www.dbqconsult.com.br/dbqc_ads1.htm acessado em 29/01/2005

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

DATE, C. J., **Introdução a Sistemas de Banco de Dados**, 8ª ed, Campus, 1990. Livros de referência:

KORTH, .F. & SILBERSHATZ, A. **Sistemas de banco de dados**. São Paulo; Makron Books; 2a ed. 1999.

DATE, C. J., **An Introduction to Database System**, sixth edition, 1995.