Apresentação

Este curso tem como objetivo, oferecer uma noção geral sobre a construção de sistemas de banco de dados. Para isto, é necessário estudar modelos para a construção de projetos lógicos de bancos de dados, modelos para a construção de projetos físicos de banco de dados, técnicas de controle de dependência de dados e métodos de consultas.

Para construção dos modelos lógicos, será estudado o modelo Entidade Relacionamento, utilizando a abordagem proposta em [ELMAS89] que oferece uma notação rica em recursos, permitindo a modelagem de entidades normais, fracas, atributos simples, compostos, multivalorados, derivados e a modelagens de objetos mais complexos como classes e subclasses (modelo Entidade Relacionamento Extendido).

Para construção dos modelos físicos, será estudado o modelo Relacional como originalmente proposto por Codd.

Para eliminar dependência de dados, utilizaremos a normalização, abordando a 1ª, a 2ª, a 3ª Formas Normais, propostas originalmente por Codd.

Para a elaboração de consultas, será estudado a Álgebra Relacional, que nada mais é do que uma forma canônica para as linguagens de consulta e a linguagem de consultas SQL.

1. Introdução e 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" é mais restrito que simplesmente a definição dada acima. Um banco de dados possui as seguintes propriedades:

- um banco de dados é uma coleção lógica coerente de dados com um significado inerente; uma disposição desordenada dos dados não pode ser referenciada como um banco de dados;

- 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" ; 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".

1.1. Abordagem Banco de Dados X Abordagem Processamento Tradicional de Arquivos

1.1.1. 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.

1.1.2. 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.

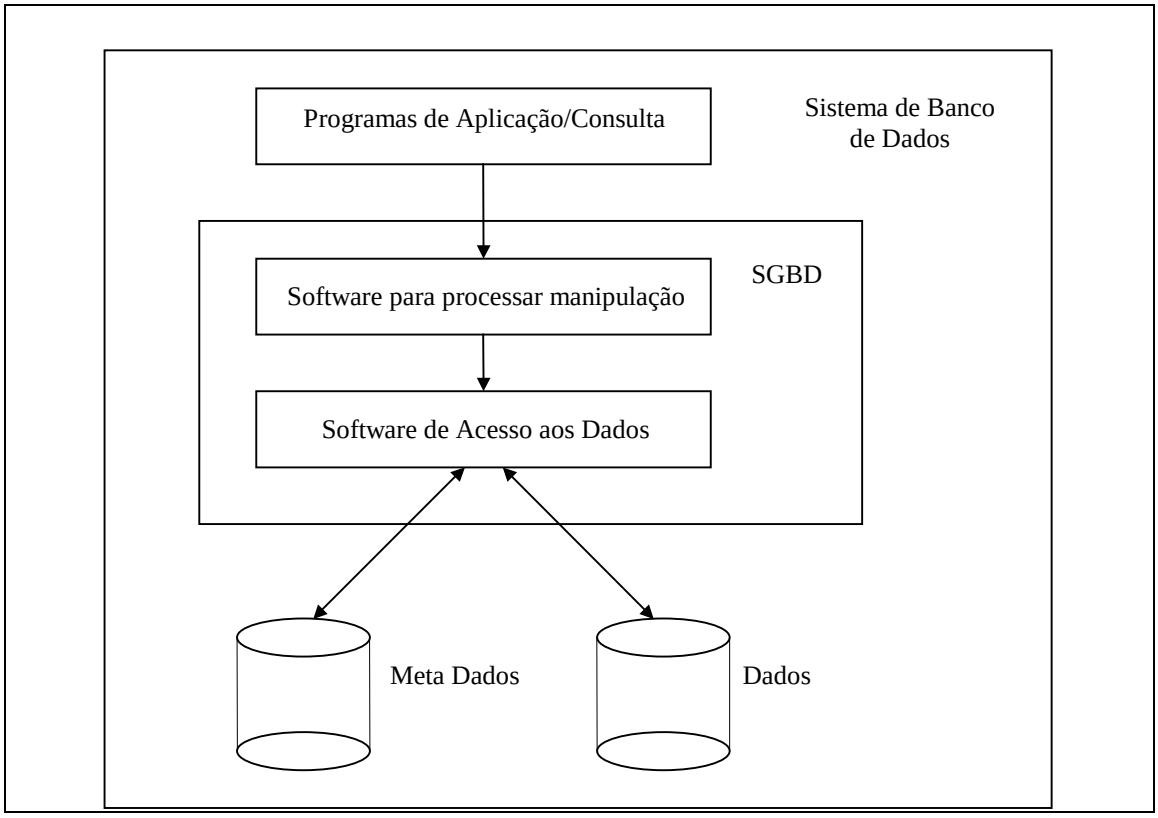

Figura 1. Um ambiente de Sistema de Banco de Dados

1.1.3. 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 utilizada 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.

1.1.4. 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.

1.2. Usuários

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

1.2.1. 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.

1.2.2. 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.

1.2.3. 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.

1.2.4. 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.

1.3. Vantagens e desvantagens do uso de um SGBD

1.3.1. 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.

1.3.2. 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.

1.3.3. 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.

1.3.4. 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.

1.3.5. Tolerância a Falhas

Um SGBD deve fornecer recursos para recuperação de falhas tanto de software quanto de hardware.

1.3.6. Quando não Utilizar um SGBD

Em algumas 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;
- generalidade que um SGBD fornece na definição e processamento de dados;
- 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:

- o banco de dados e as aplicações são simples, bem definidas e não se espera mudanças no projeto;
- a necessidade de processamento em tempo real de certas aplicações, que são terrivelmente prejudicadas pela sobrecarga causada pelo uso de um SGBD;

- não haverá múltiplo acesso ao banco de dados.

2. Conceitos e Arquiteturas de um SGBD

2.1. 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 de dados podem ser basicamente de dois tipos:

- alto nível: ou modelo de dados conceitual, que fornece uma visão mais próxima do modo como os usuários visualizam os dados realmente;

- baixo nível: ou modelo de dados físico, que fornece uma visão mais detalhada do modo como os dados estão realmente armazenados no computador.

2.2. Esquemas e Instâncias

Em qualquer modelo de dados utilizado, é importante distinguir a "descrição" do banco de dados do "banco de dados" por si próprio. A descrição de um banco de dados é chamada de "esquema de um banco de dados" e é especificada durante o projeto do banco de dados. Geralmente, poucas mudanças ocorrem no esquema do banco de dados.

Os dados armazenados em um banco de dados em um determinado instante do tempo formam um conjunto chamado de "instância do banco de dados". A instância altera toda vez que uma alteração no banco de dados é feita.

O SGBD é responsável por garantir que toda instância do banco de dados satisfaça ao esquema do banco de dados, respeitando sua estrutura e suas restrições. O esquema de um banco de dados também pode ser chamado de "intensão" de um banco de dados e a instância de "extensão" de um banco de dados.

2.3. A Arquitetura Três Esquemas

A principal meta da arquitetura "três esquemas" (figura 2) é separar as aplicações do usuário do banco de dados físico. Os esquemas podem ser definidos como:

- nível interno: ou esquema interno, o qual descreve a estrutura de armazenamento físico do banco de dados; utiliza um modelo de dados e descreve detalhadamente os dados armazenados e os caminhos de acesso ao banco de dados;

- nível conceitual: ou esquema conceitual, o qual descreve a estrutura do banco de dados como um todo; é uma descrição global do banco de dados, que não fornece detalhes do modo como os dados estão fisicamente armazenados;

- nível externo: ou esquema de visão, o qual descreve as visões do banco de dados para um grupo de usuários; cada visão descreve quais porções do banco de dados um grupo de usuários terá acesso.

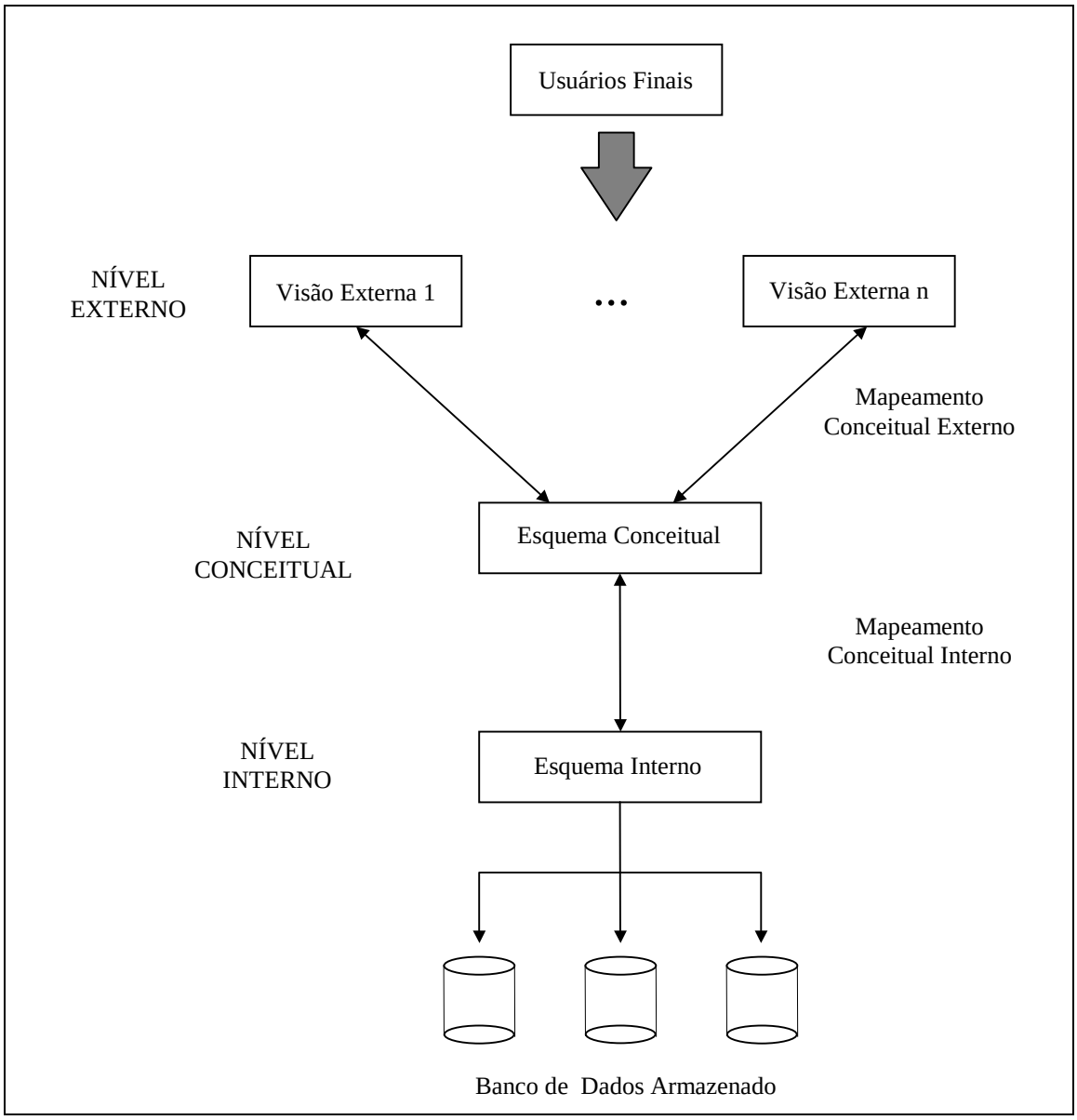

Figura 2: Arquitetura Três Esquemas

2.4. Independência de Dados

A "independência de dados" pode ser definida como a capacidade de se alterar um esquema em um nível em um banco de dados sem ter que alterar um nível superior (figura 2). Existem dois tipos de independência de dados:

- independência de dados lógica: é a capacidade de alterar o esquema conceitual sem ter que alterar o esquema externo ou as aplicações do usuário;

- independência de dados física: é a capacidade de alterar o esquema interno sem ter que alterar o esquema conceitual, o esquema externo ou as aplicações do usuário.

2.5. As Linguagens para Manipulação de Dados

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

2.6. Os Módulos Componentes de um SGBD

Um SGBD é um sistema complexo, formado por um conjunto muito grande de módulos. A figura 3 mostra um esquema da estrutura de funcionamento de um SGBD.

2.7. Classificação dos SGBDs

O principal critério para se classificar um SGBD é o modelo de dados no qual é baseado. A grande maioria dos SGBDs conteporâneos 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.

contato@pedrofcarvalho.com.br
S. J. Rio Preto – SP 2009

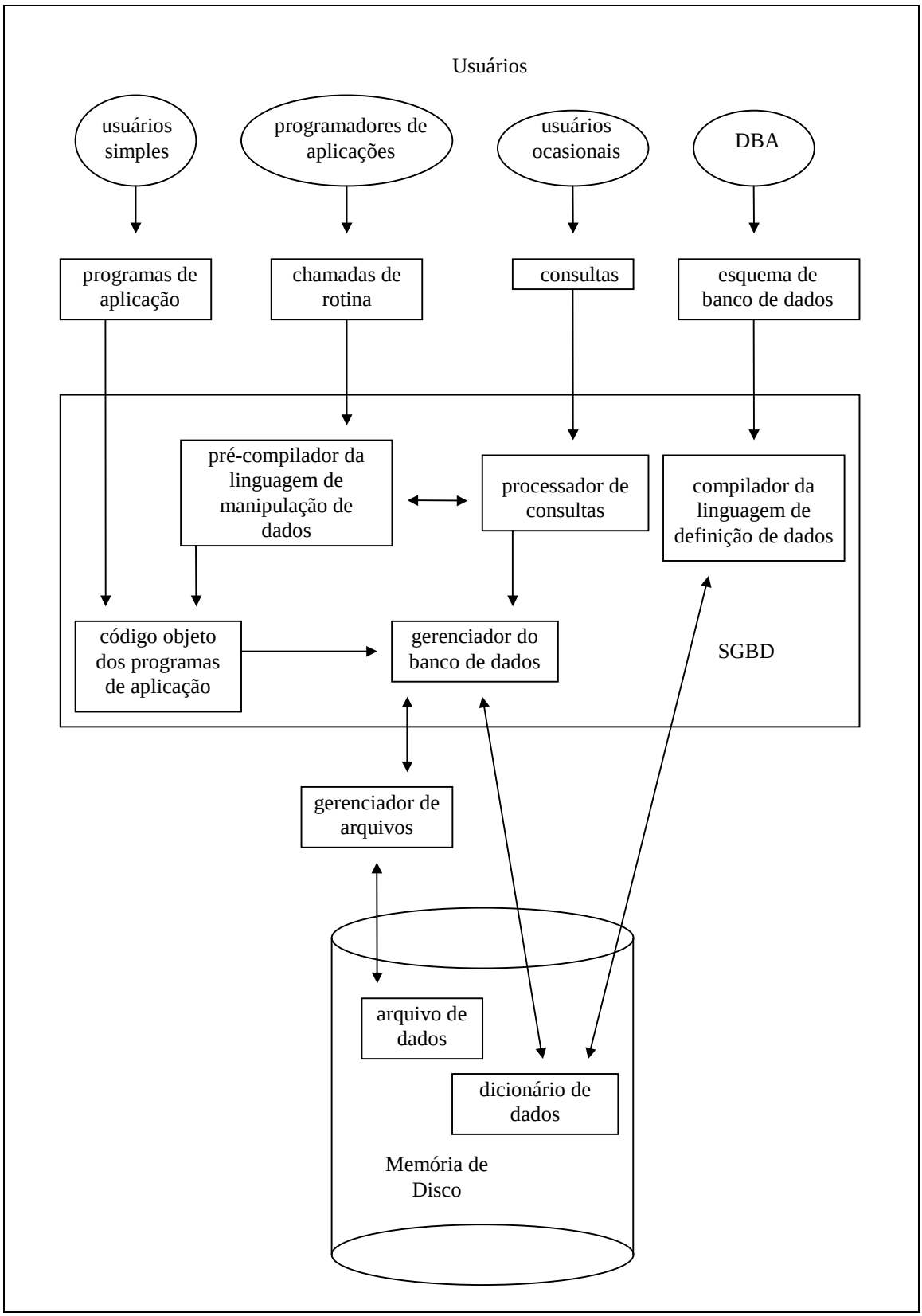

Figura 3: Estrutura de um Sistema Gerenciador de Banco de Dados

contato@pedrofcarvalho.com.br
S. J. Rio Preto – SP 2009

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.

3.1. Modelo de Dados Conceitual de Alto Nível

A figura 4 faz uma descrição simplificada do processo de projeto de um banco de dados.

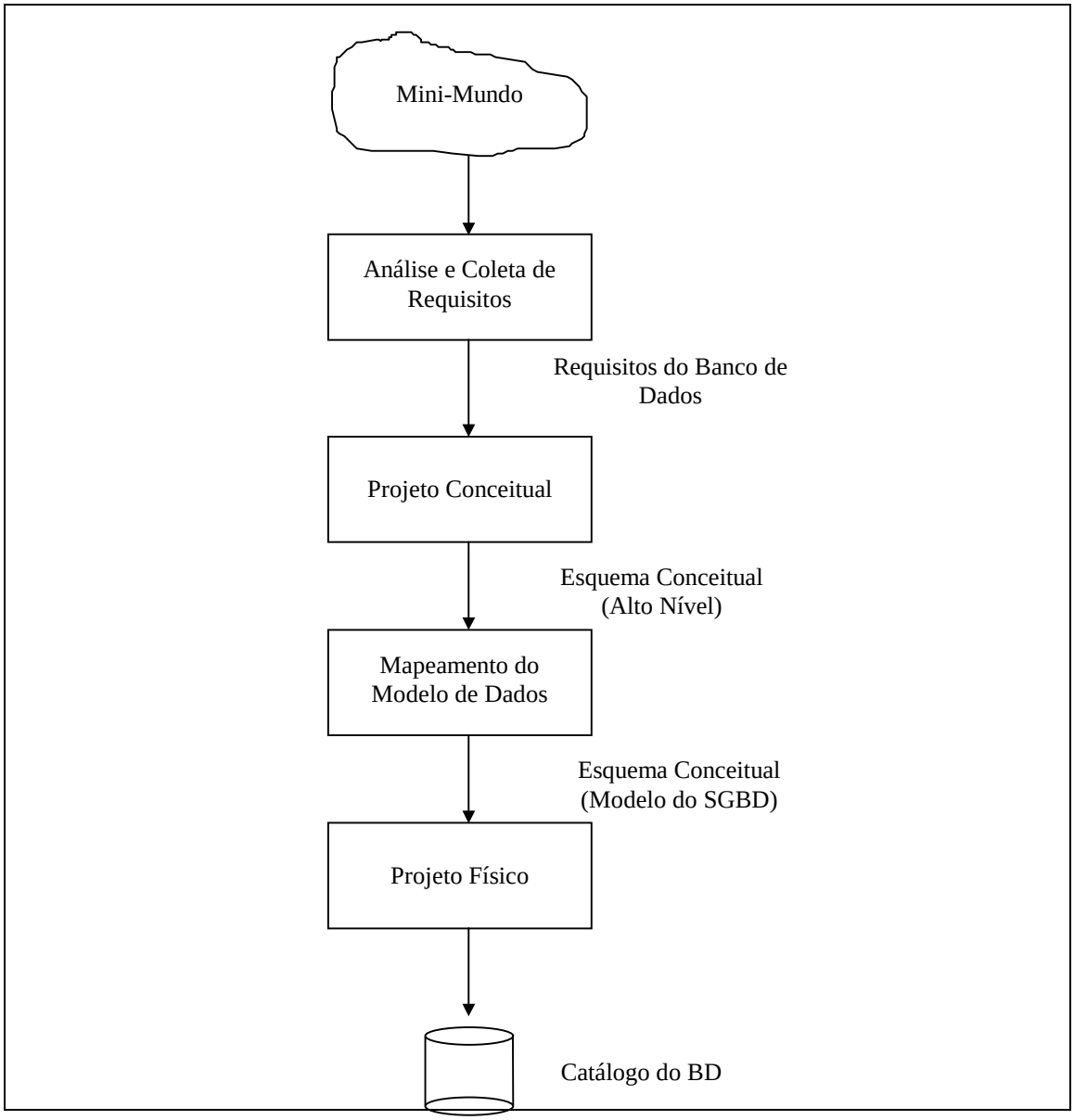

Figura 4: Fases do Projeto de um Banco de Dados

3.2. 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. Cada entidade possui um conjunto particular de propriedades que a descreve chamado "atributos". Um atributo pode ser dividido em diversas sub-partes com significado independente entre si, recebendo o nome de "atributo composto". Um atributo que não pode ser subdividido é chamado de "atributo simples" ou "atômico".

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".

Um atributo que é gerado a partir de outro atributo é chamado de "atributo derivado".

3.3. Tipos Entidade, Conjunto de Valores, Atributo Chave

Um banco de dados costuma conter grupos de entidades que são similares, possuindo os mesmos atributos, porém, cada entidade com seus próprios valores para cada atributo. Este conjunto de entidades similares definem um "tipo entidade". Cada tipo entidade é identificada por seu nome e pelo conjunto de atributos que definem suas propriedades. A descrição do tipo entidade é chamada de "esquema do tipo entidade", especificando o nome do tipo entidade, o nome de cada um de seus atributos e qualquer restrição que incida sobre as entidades.

Uma restrição muito importante em uma entidade de um determinado tipo entidade é a "chave". Um tipo entidade possui um atributo cujos valores são distintos para cada entidade individual. Este atributo é chamado "atributo chave" e seus valores podem ser utilizados para identificar cada entidade de forma única. Muitas vezes, uma chave pode ser formada pela composição de dois ou mais atributos. Uma entidade pode também ter mais de um atributo chave.

Cada atributo simples de um tipo entidade 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.

3.4. Tipos e Instâncias de Relacionamento

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

Um "tipo relacionamento" **R** entre *n* entidades **E1, E2, ..., En**, é um conjunto de associações entre entidades deste tipo. Informalmente falando, cada instância de relacionamento **r1** em **R** é uma associação de entidades, onde a associação inclui exatamente uma entidade de cada tipo entidade participante no tipo relacionamento. Isto significa que estas entidades estão relacionadas de alguma forma no mini-mundo. A figura 5 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.

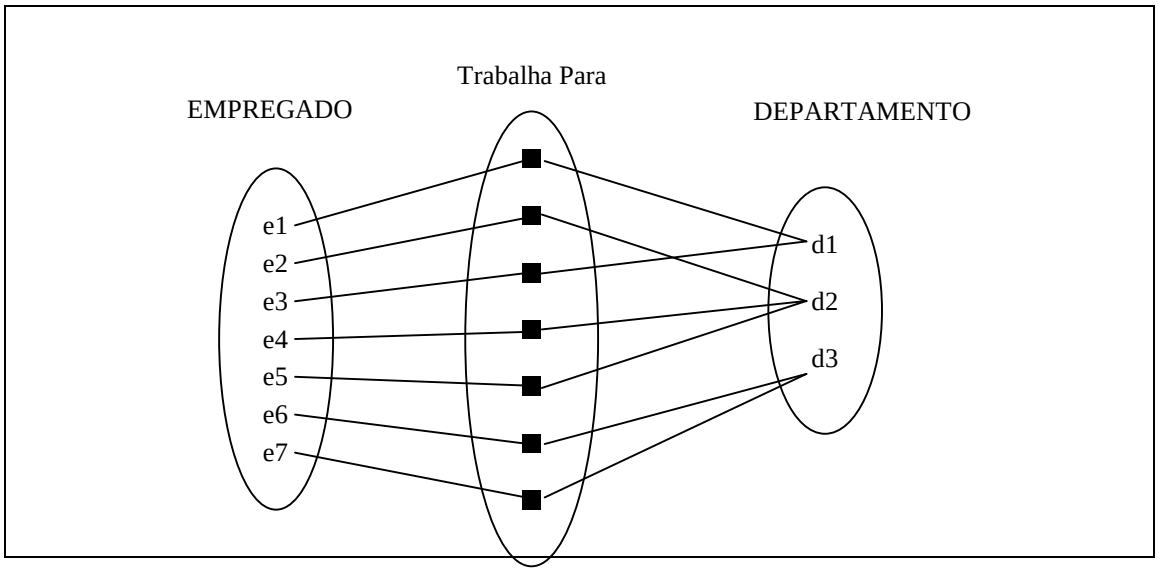

Figura 5: Exemplo de um Relaciomento

3.5. 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 5, 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.

3.6. Outras Características de um Relacionamento

3.6.1. Relacionamentos como Atributos

Algumas vezes é conveniente pensar em um relacionamento como um atributo. Considere o exemplo da figura 5. Podemos pensar departamento como sendo um atributo da entidade empregado, ou empregado, como um atributo multivalorado da entidade departamento. Se uma entidade não possuir existência muito bem definida, talvez seja mais interessante para a coesividade do modelo lógico que ela seja representada como um atributo.

3.6.2. Nomes de Papéis e Relacionamentos Recursivos

Cada tipo entidade que participa de um tipo relacionamento desempenha um **papel** particular no relacionamento. O nome do papel representa o papel que uma entidade de um tipo entidade participante desempenha no relacionamento. No exemplo da figura 5, nós temos o papel empregado ou trabalhador para o tipo entidade EMPREGADO e o papel departamento ou empregador para a entidade DEPARTAMENTO. Nomes de papéis não são necessariamente importantes quando todas as entidades participantes desempenham papéis diferentes. Algumas vezes, o papel torna-se essencial para distinguir o significado de cada participação. Isto é muito comum em "elacionamentos recursivos".

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

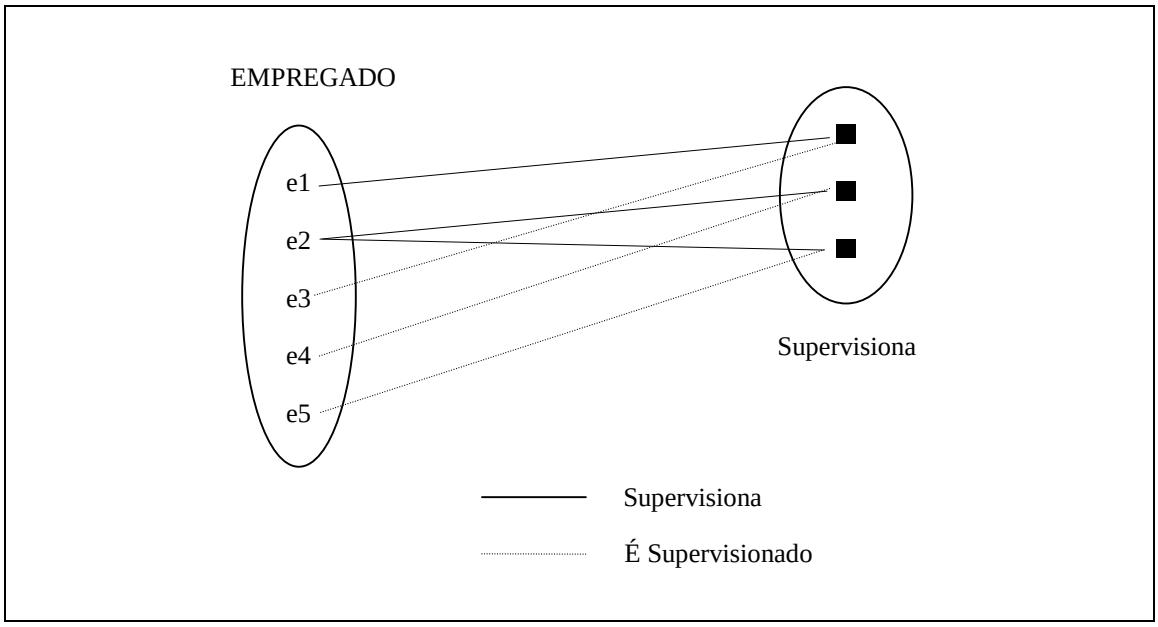

Figura 6 - Um Relacionamento Recursivo

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.

3.6.3. 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. Veja o exemplo da figura 7.

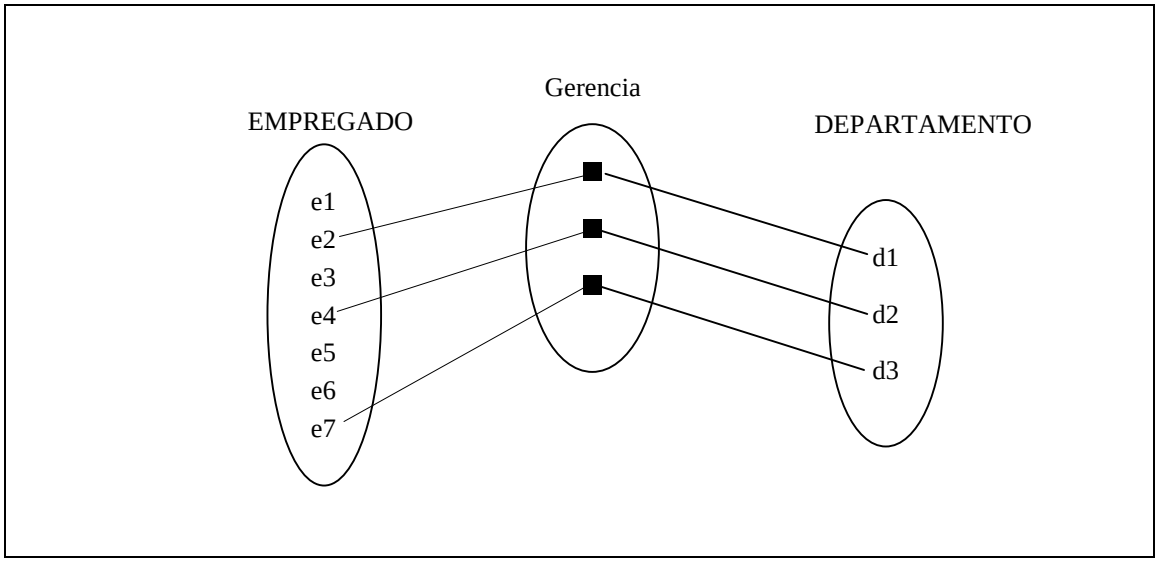

Figura 7 - Relacionamento EMPREGADO gerencia DEPARTAMENTO

No exemplo da figura 7, 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 o número de relacionamentos dos quais uma entidade pode participar. A cardinalidade pode ser: 1:1, 1:N, M:N. No exemplo da figura 7, 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 da figura 5, no 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 8 temos um exemplo de um relacionamento com cardinalidade N:M.

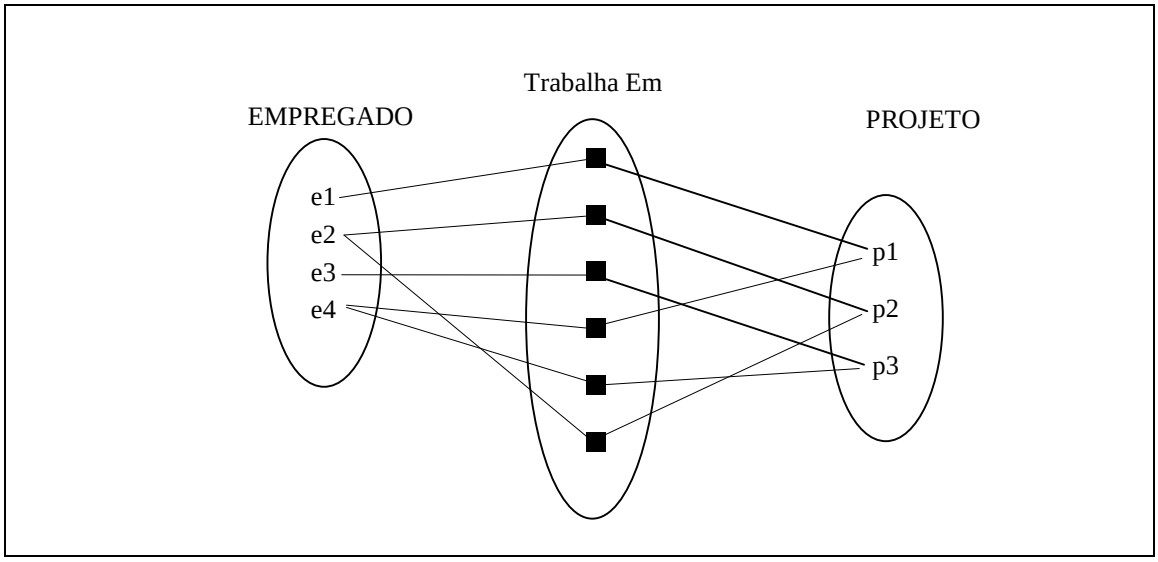

Figura 8 - Relacionamento N:M

No exemplo da figura 8, nós temos que um empregado pode trabalhar em vários projetos enquanto que um projeto pode ter vários empregados trabalhando.

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 7. 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** as entidade do tipo entidade EMPREGADO precisam participar do relacionamento. Já no exemplo da figura 5, 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 7. 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 8. Caso queiramos armazenar quantas horas cada empregado trabalhou em cada projeto, então este deverá ser um atributo do relacionamento.

3.6.4. 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 da figura 9.

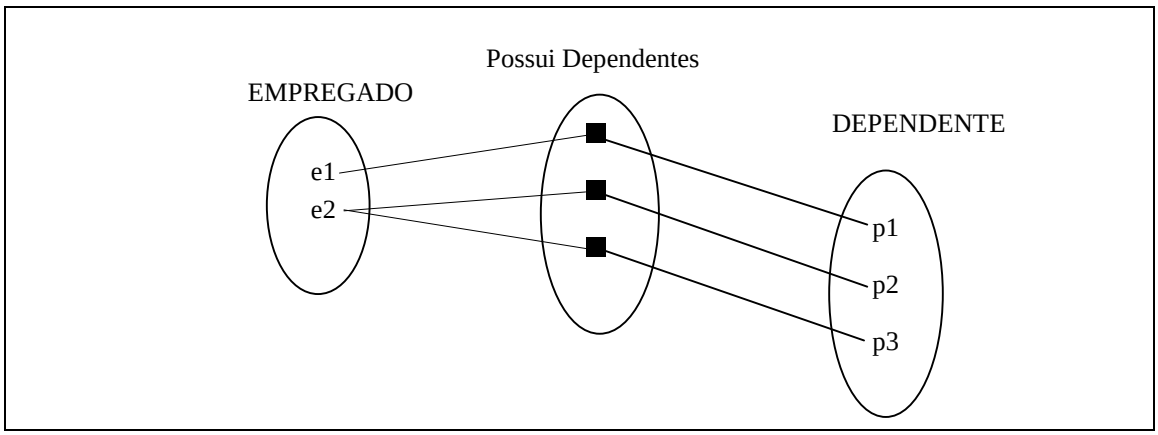

Figura 9 - Relacionamento com uma Entidade Fraca (Dependente)

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.

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 criassemos um relacionamento ternário. Veja o resultado como fica no exemplo da figura 10.

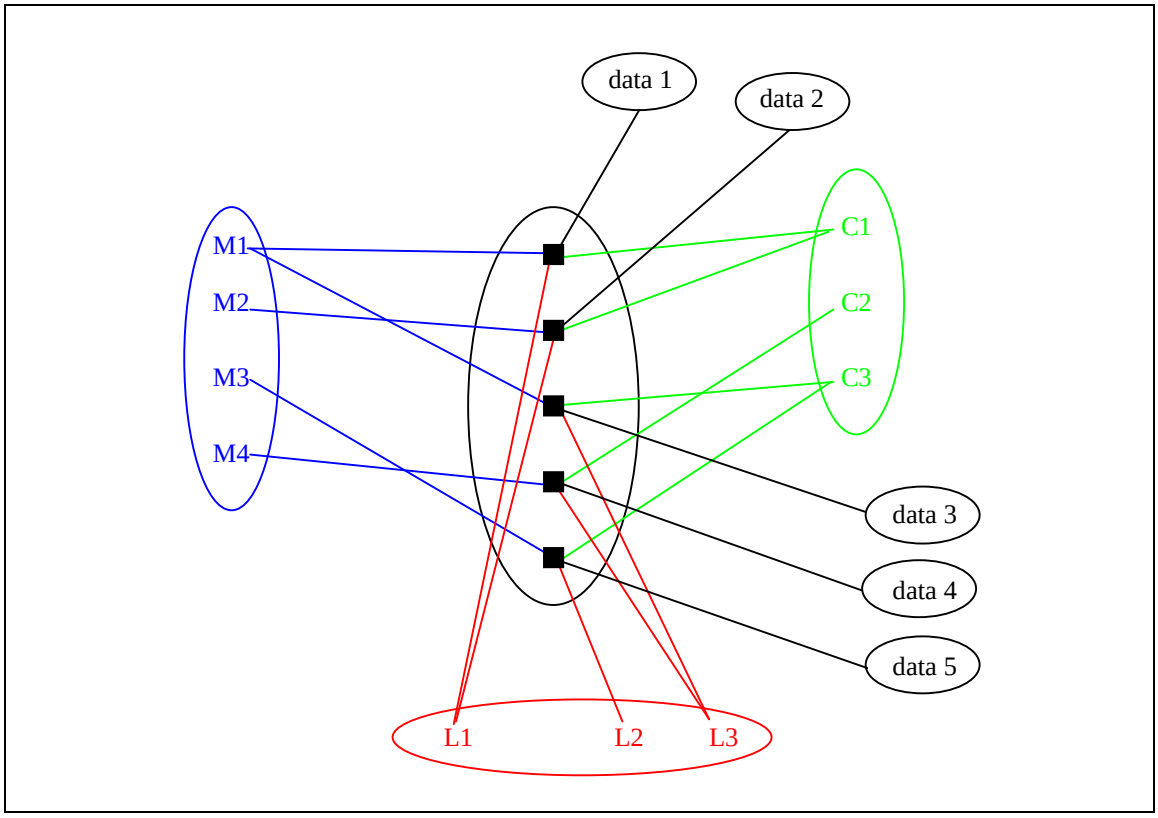

Figura 10 - Relacionamento Ternário

3.7. Diagrama Entidade Relacionamento

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

O diagrama ER fornece uma visão lógica do banco de dados, 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.

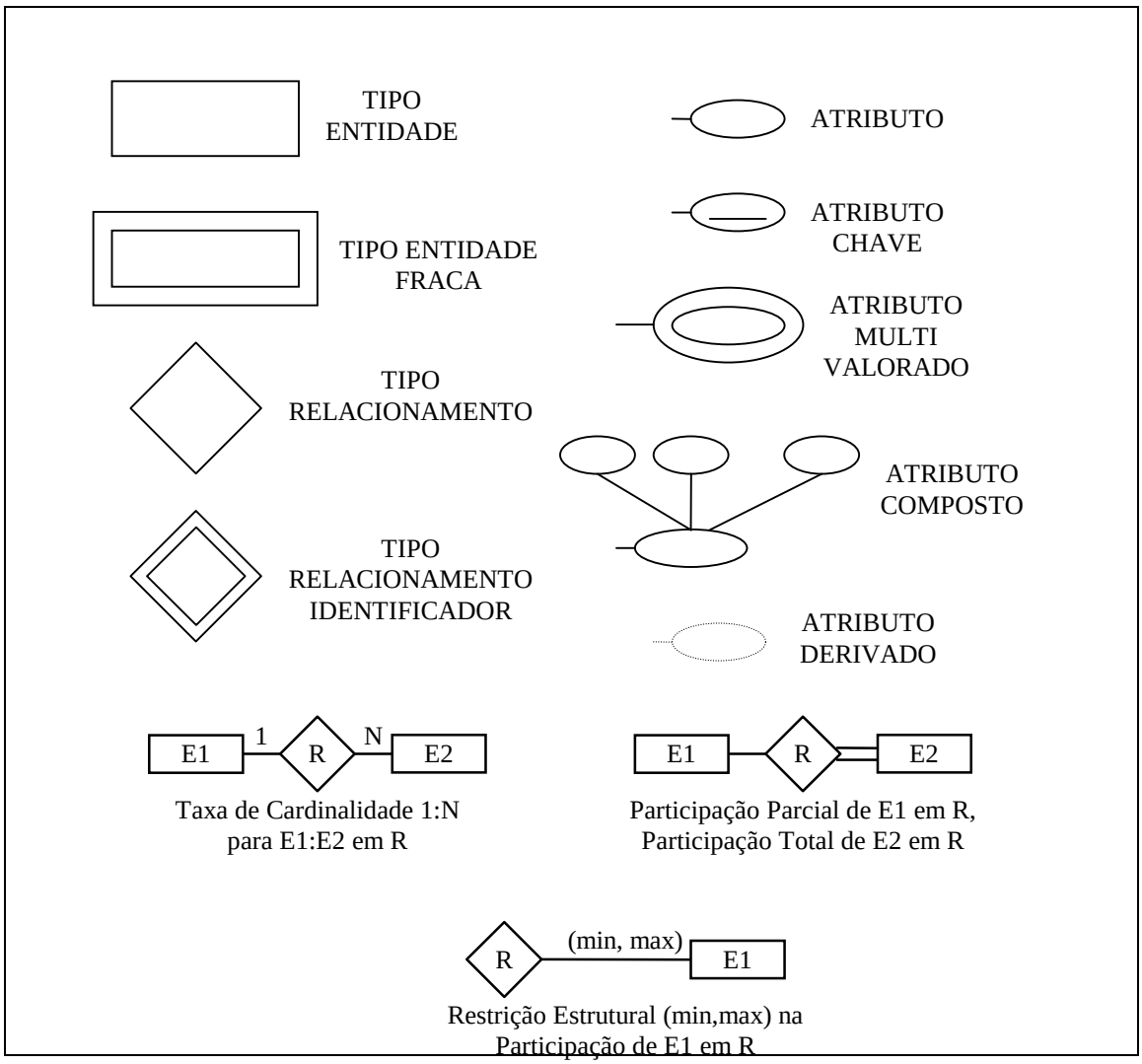

Figura 11- Objetos que Compõem o Diagrama ER

3.8. Modelo Entidade Relacionamento Extendido

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**.

3.8.1. 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, podemo 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 caracteríscticas de uma mesma entidade da superclasse. Uma subclasse pode herdar atributos de superclasses diferentes.

A figura 12 mostra a representação diagramática do exemplo acima.

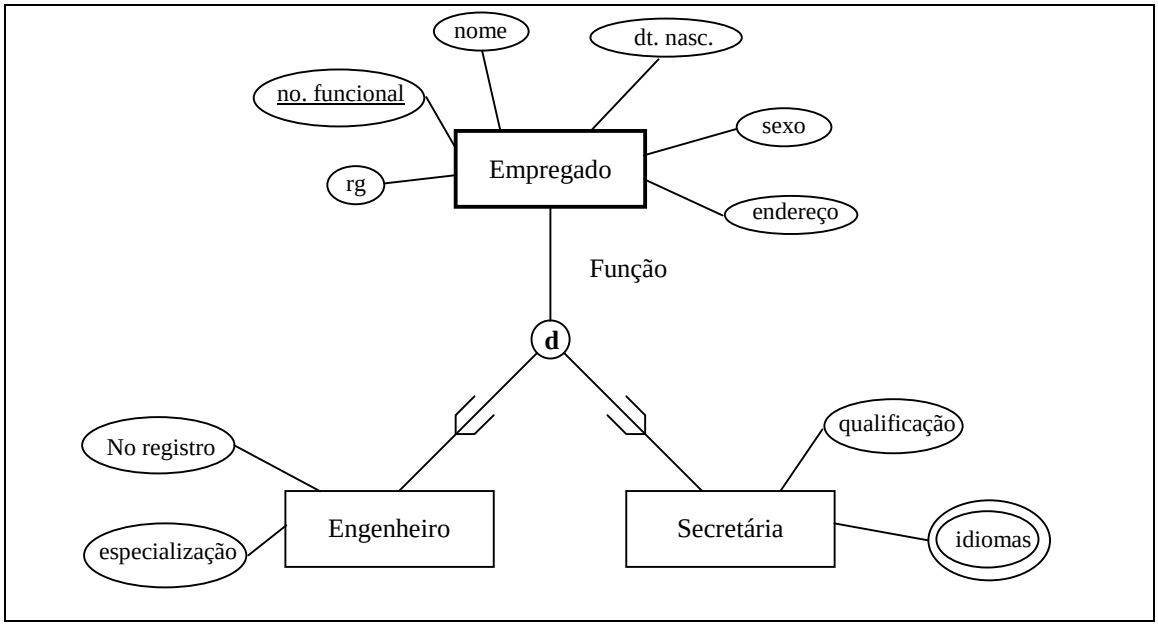

Figura 12 - Representação de Superclasse e Subclasses

3.8.3. 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 distingua 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.

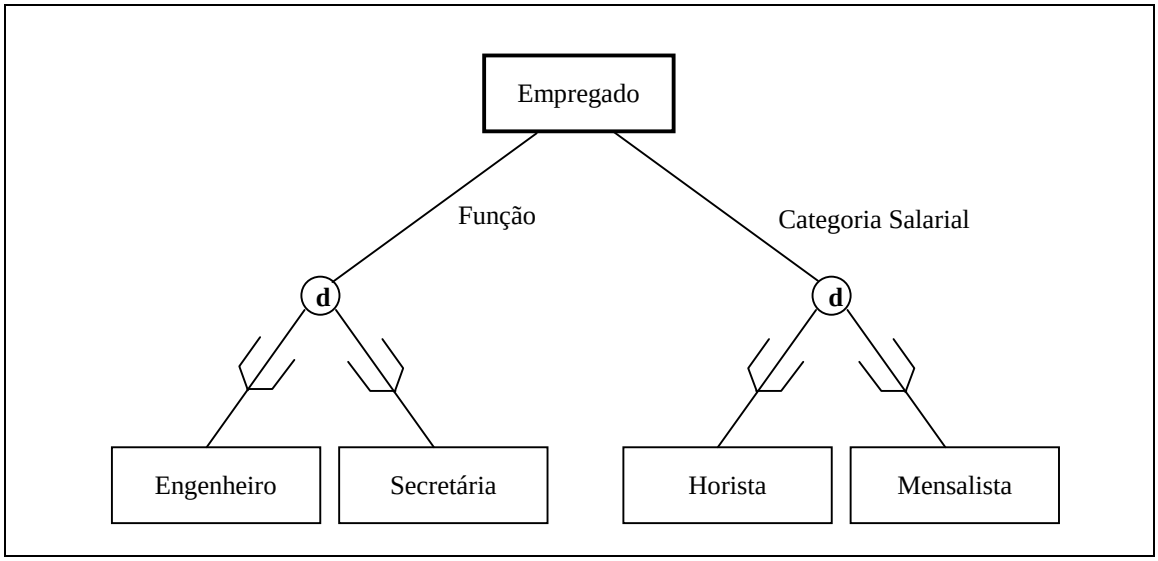

Figura 13 - Duas Especializações para Empregado: *Função* e *Categoria Salarial*

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.

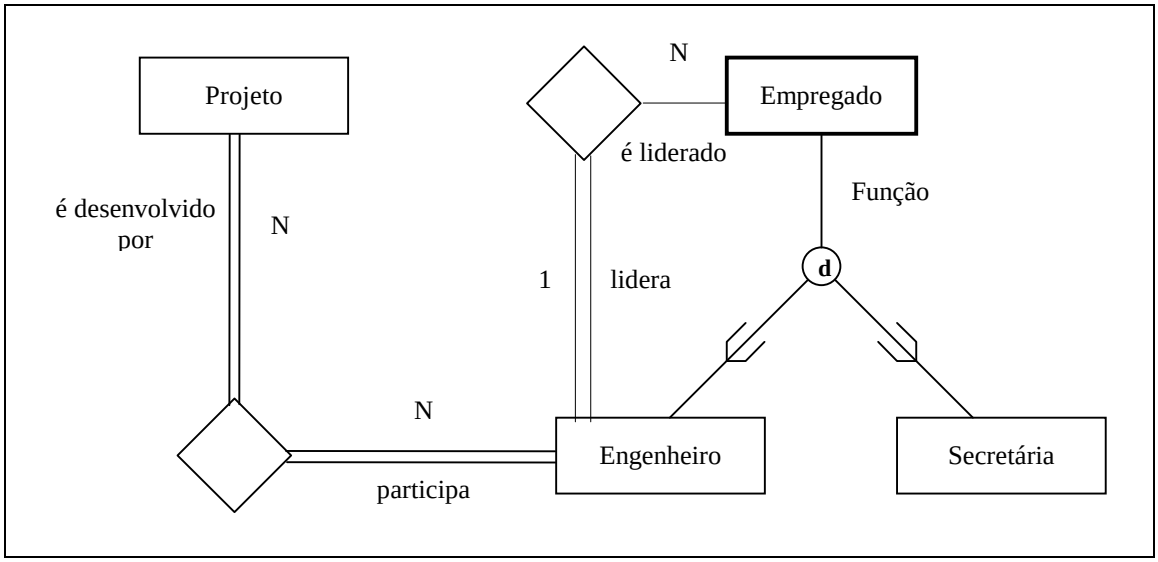

Figura 14 - Relacionamentos Entre Subclasses e Entidades

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.

3.8.4. 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

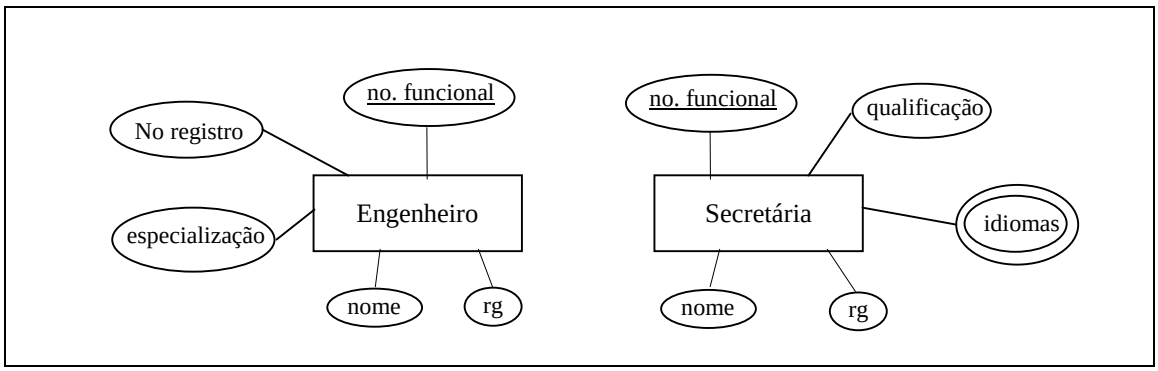

Figura 15 - Tipos Entidades *Engenheiro* e *Secretária*

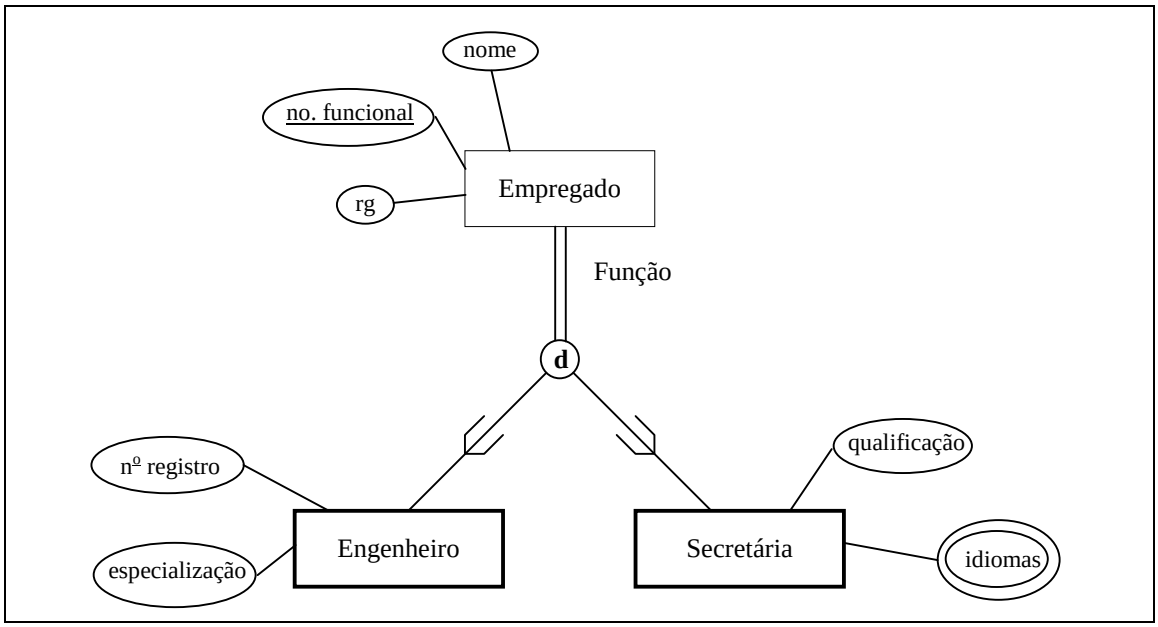

Figura 16 - Generalização *Empregado* para os Tipos Entidades Engenheiro e Secretária

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*.

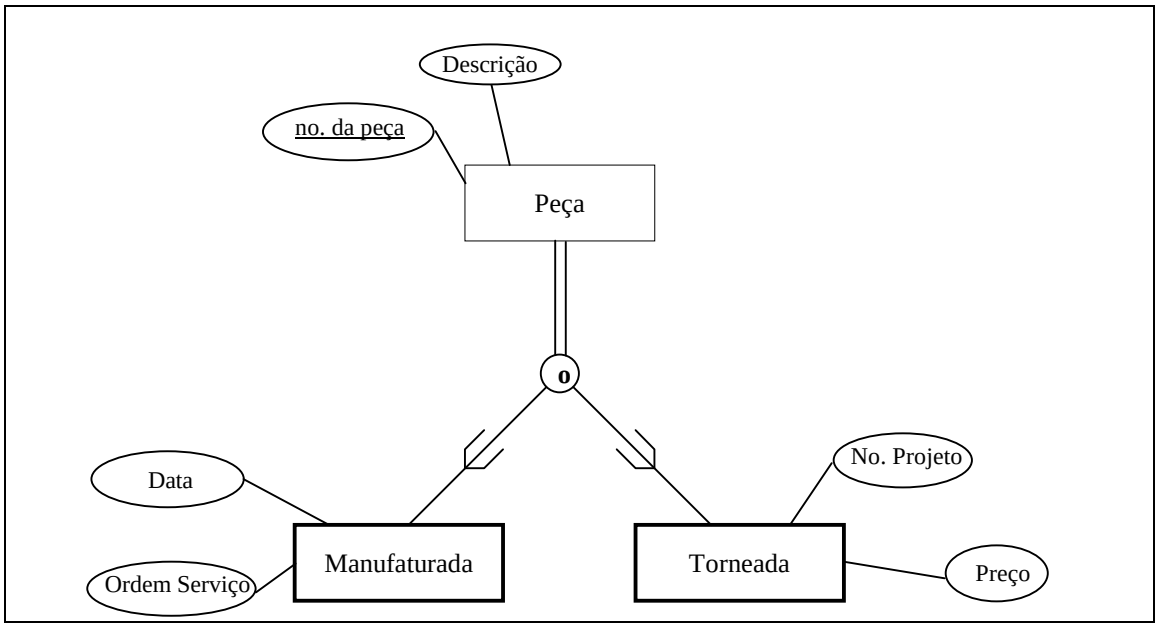

Figura 17 - Uma Generalização com "Overlap"

3.8.5. "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:

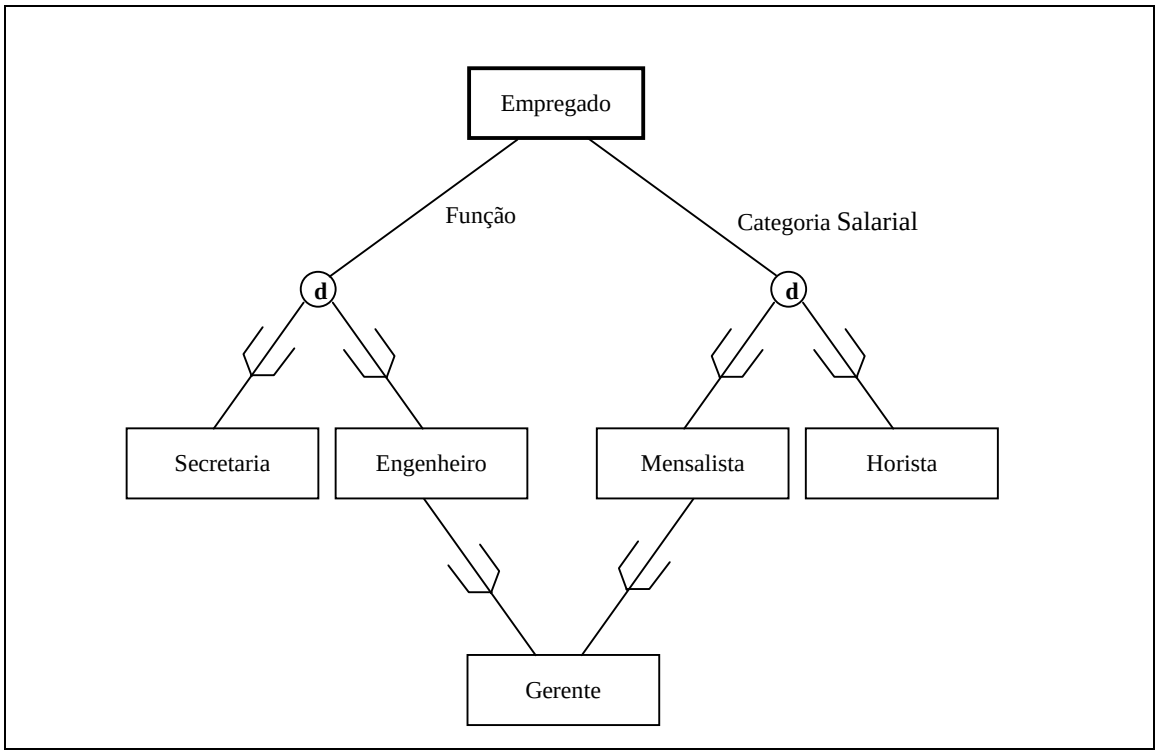

Figura 18 - Um "Lattice" com a Subclasse Gerente Compartilhada

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*.

4. O Modelo Relacional

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.

Quando uma relação é pensada como uma tabela de valores, cada linha nesta tabela representa uma coleção de dados relacionados. Estes valores podem ser interpretados como fatos descrevendo uma instância de uma entidade ou de um relacionamento. O nome da tabela e das colunas desta tabela são utilizados para facilitar a interpretação dos valores armazenados em cada linha da tabela. Todos os valores em uma coluna são necessariamente do mesmo tipo.

Na terminologia do modelo relacional, cada tabela é chamada de **relação**; uma linha de uma tabela é chamada de **tupla**; o nome de cada coluna é chamado de **atributo**; o tipo de dado que descreve cada coluna é chamado de **domínio**.

4.1. Domínios, Tuplas, Atributos e Relações

Um **domínio D** é um conjunto de valores atômicos, sendo que por atômico, podemos compreender que cada valor do domínio é indivisível. Durante a especificação do domínio é importante destacar o tipo, o tamanho e a faixa do atributo que está sendo especificado. Por exemplo:

Coluna	Tipo	Tamanho	Faixa
RG	Numérico	10,0	03000000-25999999
Nome	Caracter	30	a-z, A-Z
Salário	Numérico	5,2	00100,00-12999,99

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

A **instância r** de um esquema relação denotado por $r(R)$ é um conjunto de n-tuplas
$r = [t_1, t_2, ..., t_n]$ onde os valores de $[t_1, t_2, ..., t_n]$ devem estar contidos no domínio **D**. O valor **nulo** também pode fazer parte do domínio de um atributo e representa um valor não conhecido para uma determinada tupla.

4.2. Atributo Chave de uma Relação

Uma relação pode ser definida como um conjunto de tuplas distintas. Isto implica que a combinação dos valores dos atributos em uma tupla não pode se repetir na mesma tabela. Existirá sempre um subconjunto de atributos em uma tabela que garantem que não haverá valores repetidos para as diversas tuplas da mesma, garantindo que $t1[SC] \neq t2[SC]$.

SC é chamada de **superchave** de um esquema de relação. Toda relação possui ao menos uma superchave - o conjunto de todos os seus atributos. Uma **chave C** de um esquema de relação **R** é uma superchave de **R** com a propriedade adicional que removendo qualquer atributo **A** de **K**, resta ainda um conjunto de

atributos **K**' que não é uma superchave de **R**. Uma chave é uma superchave da qual não se pode extrair atributos. Por exemplo, o conjunto: *(RA, Nome, Endereço)* é uma superchave para estudante, porém, não é uma chave pois se tirarmos o campo *Endereço* continuaremos a ter uma superchave. Já o conjunto *(Nome da Revista, Volume, Nº da Revista)* é uma superchave e uma chave, pois qualquer um dos atributos que retirarmos, deixaremos de ter uma superchave, ou seja, *(Nome da Revista, Volume)* não identifica uma única tupla.

Em outras palavras, uma superchave é uma **chave composta**, ou seja, uma chave formada por mais que um atributo. Veja o exemplo abaixo:

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	Fernanda	01/05/90	Filho	Feminino

Quando uma relação possui mais que uma chave (não confundir com chave composta) - como por exemplo RG e CIC para empregados - cada uma destas chaves são chamadas de **chaves candidatas**. Uma destas chaves candidatas deve ser escolhida como **chave primária**.

Uma **chave estrangeira CE** de uma tabela $\mathbf{R_1}$ em $\mathbf{R_2}$ ou vice-versa, especifica um relacionamento entre as tabelas $\mathbf{R_1}$ e $\mathbf{R_2}$.

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

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*

4.3. Mapeamento do Modelo Entidade Relacionamento para o Modelo Relacional

O mapeamento do modelo entidade relacionamento para o Modelo Relacional segue oito passos básicos a saber:

1. Para cada entidade **E** no modelo ER é criada uma tabela $\mathbf{T_1}$ no Modelo Relacional que inclua todos os atributos simples de **E**; para cada atributo composto, são inseridos apenas os componentes simples de cada um; um dos atributos chaves de **E** deve ser escolhida como a chave primária de $\mathbf{T_1}$;

2. Para cada entidade fraca **EF** com entidade proprietária **E** no modelo ER, é criada uma tabela $\mathbf{T_1}$ no Modelo Relacional incluindo todos os atributos simples de **EF**; para cada atributo composto , são inseridos apenas os componentes simples de cada um; a chave primária desta relação $\mathbf{T_1}$ será composta pela chave parcial da entidade fraca **EF** mais a chave primária da entidade proprietária **E**;

3. Para cada relacionamento regular com cardinalidade 1:1 entre entidades $\mathbf{E_1}$ e $\mathbf{E_2}$ que geraram as tabelas $\mathbf{T_1}$ e $\mathbf{T_2}$ respectivamente, devemos escolher a chave primária de uma das relações ($\mathbf{T_1}$, $\mathbf{T_2}$)e inseri-la como chave estrangeira na outra relação; se um dos lados do relacionamento tiver participação total e outro parcial, então é interessante que a chave do lado com participação **parcial** seja inserido como chave estrangeira no lado que tem participação **total**;

4. Para cada relacionamento regular com cardinalidade 1:N entre entidades $\mathbf{E_1}$ e $\mathbf{E_2}$ respectivamente e que geraram as tabelas $\mathbf{T_1}$ e $\mathbf{T_2}$ respectivamente, deve-se inserir a chave primária de $\mathbf{T_1}$ como chave estrangeira em $\mathbf{T_2}$;

5. Para cada relacionamento regular com cardinalidade N:N entre entidades $\mathbf{E_1}$ e $\mathbf{E_2}$, cria-se uma nova tabela $\mathbf{T_1}$, contendo todos os atributos do relacionamento mais o atributo chave de $\mathbf{E_1}$ e o atributo chave de $\mathbf{E_2}$; a chave primária de $\mathbf{T_1}$ será composta pelos atributos chave de $\mathbf{E_1}$ e $\mathbf{E_2}$;

6. Para cada atributo multivalorado $\mathbf{A_1}$, cria-se uma tabela $\mathbf{T_1}$, contendo o atributo multivalorado $\mathbf{A_1}$, mais o atributo chave **C** da tabela que representa a entidade ou relacionamento que contém $\mathbf{A_1}$; a chave primária de $\mathbf{T_1}$ será composta por $\mathbf{A_1}$ mais **C**; se $\mathbf{A_1}$ for composto, então a tabela $\mathbf{T_1}$ deverá conter todos os atributos de $\mathbf{A_1}$;

7. Para cada relacionamento n-ário, n > 2, cria-se uma tabela $\mathbf{T_1}$, contendo todos os atributos do relacionamento; a chave primária de $\mathbf{T_1}$ será composta pelos atributos chaves das entidades participantes do relacionamento;

8. Converta cada especialização com *m* subclasses $\mathbf{\{S_1, S_2, ..., S_m\}}$ e superclasse **SC**, onde os atributos de **SC** são $\mathbf{\{c, a_1, a_2, ..., a_n\}}$ onde **c** é a chave primária de **SC**, em tabelas utilizando uma das seguintes opções:

   8.1. Crie uma tabela **T** para **SC** com os atributos $\mathbf{A(T) = \{c, a_1, a_2, ..., a_n\}}$ e chave $\mathbf{C(T) = c}$; crie uma tabela $\mathbf{T_i}$ para cada subclasse $\mathbf{S_i}$, $1 \leq i \leq$

m, com os atributos
A(T$_i$) = **{c}** ∪ **A(S$_i$)**, onde **C(T)** = **c**;

8.2. Crie uma tabela **T$_i$** para cada subclasse **S$_i$**, 1 ≤ i ≤ m, com os atributos **A(T$_i$) = A(S$_i$)** ∪ **{c, a$_1$, a$_2$, ..., a$_n$}** e **C(T$_i$)** = **c**;

8.3. Crie uma tabela **T** com os atributos **A(T) = {c, a$_1$, a$_2$, ..., a$_n$}** ∪ **A(S$_1$)** ∪ **...** ∪ **A(S$_m$)** ∪ **{t}** e **C(T)** = **c**, onde **t** é um atributo **tipo** que indica a subclasse à qual cada tupla pertence, caso isto venha a ocorrer;

8.4. Crie uma tabela **T** com atributos **A(T) = {c, a$_1$, a$_2$, ..., a$_n$}** ∪ **A(S$_1$)** ∪ **...** ∪ **A(S$_m$)** ∪ **{t$_1$, t$_2$, ..., t$_m$}** e **C(T)** = **c**; esta opção é para generalizações com "overlapping", e cada **t$_i$**, 1 ≤ i ≤ m, é um atributo "booleano" indicando se a tupla pertence ou não à subclasse **S$_i$**; embora funcional, esta opção pode gerar uma quantidade muito grande de valores nulos;

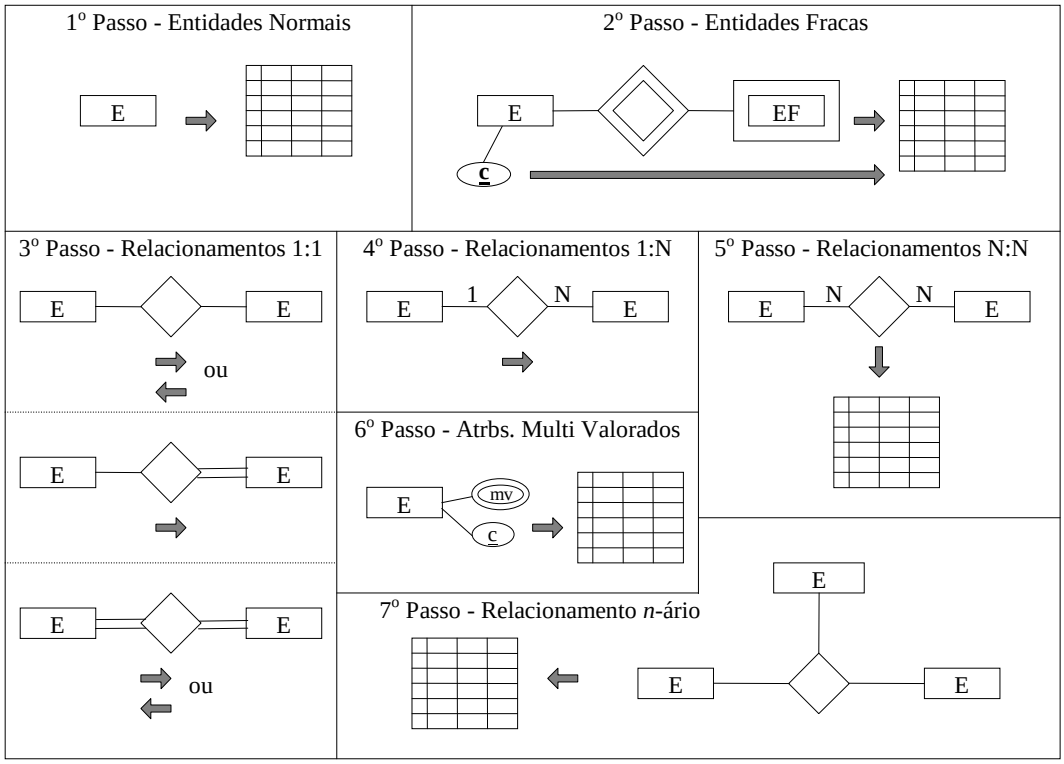

Figura 19 - Mapeamento para o Modelo Relacional

4.4. Dependência Funcional e Normalização

4.4.1. Dependência Funcional

Uma **dependência funcional** é uma restrição entre dois conjuntos de atributos de uma base de dados. Suponha que o esquema de uma base de dados **R** possua *n* atributos
A_1, A_2, ..., A_n; pense em **R** = **{ A$_1$, A$_2$, ... , A$_n$ }** como a representação universal da base de dados. Uma dependência funcional, representada por **X → Y** entre dois conjuntos de atributos **X** e **Y** que são subconjuntos de **R** especificam uma restrição nas tuplas que podem compor uma instância relação *r* de **R**. A restrição estabelece que para qualquer par de tuplas t_1 e t_2 em *r* de forma que t_1.**[X]** = t_2.**[X]**, é obrigado a existir t_1.**[Y]** = t_2.**[Y]**. Isto significa que os valores do componente **Y** em uma tupla em *r* **depende de**, ou **é determinada pelos** valores do componente **X**. Para **X → Y** lê-se: ***Y** é funcionalmente dependente de **X***, ou ***X** infere sobre **Y***.

Para a base de dados do apêndice A temos como exemplo as seguintes dependências funcionais:

1. RG → { Nome, CIC, Depto., RG_Supervisor, Salário }

2. Número_Projeto → { Nome_Projeto, Localização }

3. { RG_Empregado, Número_Projeto } → Horas

além de outras.

A dependência 1 implica que o número de um **RG** define de forma única o nome do empregado e o CIC do empregado. A dependência 2 implica que o **número do projeto** define de forma única o nome do projeto e sua localização e a dependência 3 implica que o
RG do empregado mais o **número do projeto** define de forma única o número de horas que o empregado trabalhou no projeto. A especificação das inferências deve ser elaborada pelo projetista de banco de dados em conjunto com o analista de sistemas, pois os mesmos deverão ter conhecimento da semântica da base de dados.

4.4.2. Normalização

O processo de **normalização** pode ser visto como o processo no qual são eliminados esquemas de relações (tabelas) não satisfatórios, decompondo-os, através da separação de seus atributos em esquemas de relações menos complexas mas que satisfaçam as propriedades desejadas.

O processo de normalização como foi proposto inicialmente por Codd conduz um esquema de relação através de um bateria de testes para certificar se o mesmo está na **1ª**, **2ª** e **3ª Formas Normais**. Estas três Formas Normais são baseadas em dependências funcionais dos atributos do esquema de relação.

4.4.2.1. 1ª Forma Normal

A **1ª Forma Normal** prega que todos os atributos de uma tabela devem ser **atômicos** (indivisíveis), ou seja, não são permitidos atributos multivalorados, atributos compostos ou atributos multivalorados compostos. Leve em consideração o esquema a seguir:

- **CLIENTE**

1. Código
2. { Telefone }
3. Endereço: (Rua, Número, Cidade)

gerando a tabela resultante:

		Telefone 1	Endereço		
Cliente	Código	Telefone *n*	Rua	No	Cidade

sendo que a mesma não está na 1ª Forma Normal pois seus atributos não são atômicos. Para que a tabela acima fique na 1ª Forma Normal temos que eliminar os atributos não atômicos, gerando as seguintes tabelas como resultado:

Cliente	Código	Rua	Número	Cidade

Cliente_Telefone	Código_Cliente	Telefone_Cliente

4.4.2.2. 2ª Forma Normal

A **2ª Forma Normal** prega o conceito da **dependência funcional total**. Uma dependência funcional **X → Y** é **total** se removemos um atributo **A** qualquer do componente **X** e desta forma, a dependência funcional deixa de existir. A dependência funcional **X → Y** é uma **dependência funcional parcial** se existir um atributo **A** qualquer do componente **X** que pode ser removido e a dependência funcional **X → Y** não deixa de existir.

Veja a dependência funcional 3 do ítem **4.4.1. Dependência Funcional**:

{ RG_Empregado, Número_Projeto } → Horas

é uma dependência funcional total, pois se removermos o atributo **RG_Empregado** ou o atributo **Número_Projeto**, a dependência funcional deixa de existir.

Uma tabela **T** está na 2ª Forma Normal se estiver na 1ª Forma Normal e todo atributo que não compõem a chave primária **C** for totalmente funcionalmente dependente da chave primária **C**. Se uma tabela não está na 2ª Forma Normal a mesma pode ser normalizada gerando outras tabelas cujos atributos que não façam parte da chave primária sejam totalmente funcionalmente dependente da mesma, ficando a tabela na 2ª Forma Normal.

4.4.2.3. 3ª Forma Normal

A **3ª Forma Normal** prega o conceito de **dependência transitiva**. Uma dependência funcional **X → Y** em uma tabela **T** é uma dependência transitiva se existir um conjunto de atributos **Z** que não é um subconjunto de chaves de **T** e as dependências **X → Z**, **Z → Y**, são válidas. Considere a seguinte tabela como exemplo:

Empregado	RG	Nome	Nº_Departamento	Nome_Depto	RG_Ger_Depto

onde temos a seguinte dependência transitiva:

RG → { Nome_Depto, RG_Ger_Depto }

RG → Nº_Departamento

Nº_Departamento → { Nome_Depto, RG_Ger_Depto }

Porém, verifique o caso da tabela abaixo:

Empregado	RG	CIC	Nome	Nº_Funcional

Neste caso, a dependência transitiva:

RG → { Nome, Nº_Funcional }

RG → CIC

CIC → { Nome, Nº_Funcional }

não é valida pois o atributo CIC é uma chave candidata.

Uma tabela está na 3ª Forma Normal se estiver na 2ª Forma Normal e não houver dependência transitiva entre atributos não chave.

4.5. 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.

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

4.5.1. 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	<u>RG</u>	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				
<u>RG Responsável</u>	<u>Nome Dependente</u>	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.

4.5.2. 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*

4.5.3. Sequencialidade 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:

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

Veja o seguinte exemplo:

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

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

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

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**.

4.5.4. 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 (∪) , **intersecção** (∩) 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 **R** ∪ **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 **R** ∩ **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 **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 = $\pi_{rg}^{(CONSULTA7)}$

Tabela consulta9
RG
20202020
30303030
40404040

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

consulta10 = CONSULTA8 ∪ 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

4.5.5. 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'.

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

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.

4.5.6. 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 $\bowtie_{\text{número\_projeto = número}}$ 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*

4.6. 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.

4.6.1. Definição de Dados Utilizando SQL

4.6.1.1. Comando CREATE TABLE

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 );
```

4.6.1.2. Comando DROP TABLE

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.

4.6.1.3. Comando ALTER TABLE

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.

4.6.2. Consultas em SQL

4.6.2.1. O comando SELECT

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:

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

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

```
select nome, rg, salario
from EMPREGADOS
where depto = 2 AND salario > 2500.00;
```

que fornece o seguinte resultado:

Nome	***RG***	***Salário***
Jorge	*40404040*	*4.200,00*

e que é originária da seguinte função em álgebra relacional:

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

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:

$\pi_{nome,\ rg}$ (EMPREGADOS $\bowtie_{tg\_t1\ =\ rg\_supervisor\_t2}$ 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.

Depto.

1
2
2
2
3

1
2
3

Podemos gerar consultas aninhadas em SQL utilizando o espeficicador **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;
```

4.6.3. Inserções e Atualizações

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.

5. Bibliografia

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

Sistema de Banco de Dados; Henry F. Korth, Abraham Silberschatz; Makro Books; 1995;

SQL Language - Oracle Reference Manual; Version 7.2;

contato@pedrofcarvalho.com.br

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*

contato@pedrofcarvalho.com.br

contato@pedrofcarvalho.com.br
S. J. Rio Preto – SP 2009

40404040	*20*	*50*
50505050	*20*	*35*

contato@pedrofcarvalho.com.br
S. J. Rio Preto – SP 2009