

CAPÍTULO I – Bancos de Dados

Dados e Informações

“**Dados** são conjuntos de fatos distintos e objetivos, relativos a eventos” ¹.

O funcionamento das organizações gera grandes quantidades de dados. Mas, para o processo de tomada de decisões, os dados não apresentam uma importância direta, pois as pessoas podem não compreender o significado desses dados, da maneira como eles são apresentados.

Quando compreendemos o significado dos dados, esses se transformam em **Informações**, a matéria prima para a tomada de decisão.

Daí podemos deduzir algumas coisas importantes. Para a tomada de decisão dentro das organizações, necessitamos obter informações, mas essas dependem fundamentalmente dos dados existentes interna e externamente a essas organizações. Além disso, os dados são a matéria prima e o resultado das transações que as organizações realizam no seu dia-a-dia, para realizar os seus negócios.

Portanto, é vital para que as organizações possam funcionar normalmente que dados, que apresentem importância para essas organizações, sejam armazenados de forma organizada e segura.

Banco de Dados

Banco de Dados é um “aplicativo que armazena dados de uma forma organizada e que permita a recuperação desses dados”.

Os Bancos de Dados podem ser **Integrados** e **Compartilhados**. Eles são Integrados, quando representam a unificação de diferentes arquivos de dados e são Compartilhados quando permitem que mais do que uma aplicação acesse os dados armazenados nele.

Além disso, eles devem procurar evitar a **Redundância** e a **Inconsistência**. A Redundância, ou seja, o armazenamento do mesmo dado em diferentes locais, é algo útil para efeito de segurança (funciona como um backup dos dados), mas ela pode apresentar um problema inaceitável para o processo de armazenamento de dados: a Inconsistência.

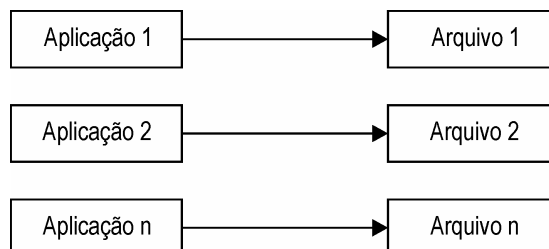
Um tipo de inconsistência que os Bancos de Dados apresentam é quando os mesmos tipos de dados são armazenados em locais diferentes (por exemplo, o endereço de um cliente), mas apresenta valores diferentes em cada um dos locais (por exemplo, o cliente atualizou o seu endereço, mas o processo só ocorreu em um dos locais de armazenamento dos dados).

Dessa forma, podemos deduzir que, nos Bancos de Dados, a redundância pode ser aceita (apesar de apresentar outro problema, que é o aumento do espaço necessário para armazenar esses dados), mas a inconsistência deve ser evitada de qualquer maneira.

¹ DAVENPORT, Thomas H.; PRUSAK, Laurence. *Conhecimento Empresarial: Como as organizações gerenciam o seu capital intelectual*. Rio de Janeiro: Campus, 1998. p. 2.

Histórico dos Bancos de Dados

Inicialmente, os dados necessários ao funcionamento das organizações eram armazenados em **Arquivos**, que são agrupamentos de dados armazenados em algum dispositivo de armazenamento físico, e para acessar esses dados e utilizá-los no dia-a-dia, eram criadas **Aplicações** (programas de computador).

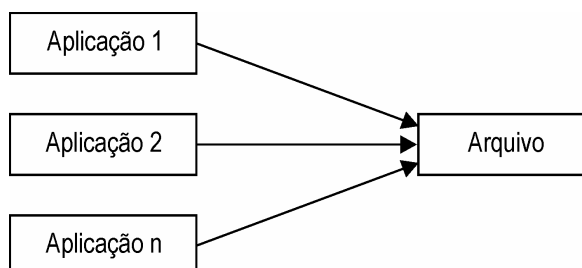


O problema dessa abordagem é que cada tipo de aplicação possuía o seu arquivo de dados. Vejamos, imagine uma empresa de manufatura, onde existem arquivos armazenando dados referentes ao projeto do produto (Departamento de Engenharia), aos detalhes dos componentes do produto (Departamento de Compras) e aos detalhes de montagem do produto (Departamento de Produção).

Cada um desses departamentos possui a sua aplicação específica e, portanto, seu arquivo de dados específico. Mas, quando um novo produto é criado pelo Departamento de Engenharia é necessário que os dados dos componentes desse produto sejam enviados para os Departamentos de Compra e de Produção. Como esse modelo de armazenamento não permite que uma aplicação acesse o arquivo de dados da outra, os dados eram transferidos na forma de papel impresso para o outro departamento e digitados novamente, na outra aplicação.

Essa redundância de dados não existia devido a preocupação com a segurança, mas sim por uma questão técnica, o que levava a grandes quantidades de dados idênticos sendo armazenados em diversos arquivos diferentes. Além desse problema, pois nesta época os dispositivos de armazenamento eram muito caros, a possibilidade de ocorrerem inconsistência nos dados era enorme.

A solução apresentada para esse problema foi unificar arquivos que armazenavam os mesmos dados e permitir um acesso compartilhado a esse arquivo, por parte das aplicações.



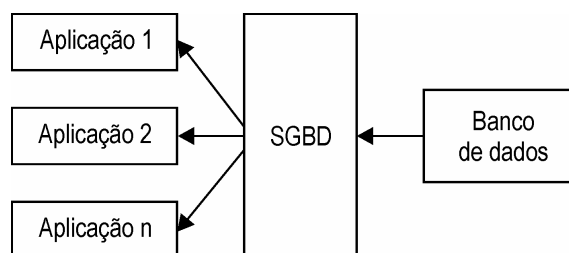
Isso resolveu o problema da redundância e da inconsistência, mas apresentou outros. O primeiro deles é que permitindo o acesso de todos ao mesmo arquivo, todas as aplicações eram obrigadas a tratar dados que podiam não apresentar importância para ela e, até mesmo, nem deveriam ser acessados. Observe o exemplo a seguir.

Imagine um arquivo armazenando os dados de um empregado. Nesse arquivo estão armazenados todos os dados necessários sobre as pessoas que trabalham na organização. Agora imagine que existem dois aplicativos acessando esse arquivo, o aplicativo do Departamento de Recursos Humanos e o aplicativo do Departamento de Operações, que cria uma escala de trabalho para os empregados.

Como todos os aplicativos têm acesso a todos os dados armazenados no arquivo, o Departamento de Operações terá acesso a dados que não tem nenhuma importância para ele, como por exemplo, o número de um determinado documento do empregado, como também terá acesso a dados que são sigilosos e que nem deveriam ser acessados, como por exemplo, o salário do empregado.

O segundo problema é que, qualquer alteração na estrutura do arquivo (estrutura do armazenamento dos dados), que seja feita para adequar o arquivo a uma determinada aplicação, implica na modificação de todas as outras aplicações para se adequarem a essa mudança também.

Para resolver isso, surgiu o conceito do **Sistema Gerenciador de Bancos de Dados (SGBD)**², que consiste em uma aplicação que administra o acesso aos dados dos arquivos de dados. O SGBD funciona como um “intermediário” entre as aplicações e os arquivos de dados (o Banco de Dados), mantendo a unificação dos arquivos e o compartilhamento do acesso, além de evitar a inconsistência dos dados, mas controlando o acesso a esses dados.



Esse controle permite que a aplicação acesse somente os dados que sejam importantes para o seu funcionamento (mesmo que existam mais dados armazenados) e inibe o acesso àqueles dados que deveriam ser sigilosos para aquela aplicação.

Além disso, ele resolve outro problema existente nos dois modelos anteriores. Antes do surgimento dos SGBD, as aplicações eram responsáveis por determinar o armazenamento físico dos dados, ou seja, se preocupar em como armazenar os dados nos dispositivos de armazenamento. Essa tarefa foi passada para os SGBDs, o que facilita em muito a vida dos programadores de aplicações.

Modelos de Bancos de Dados

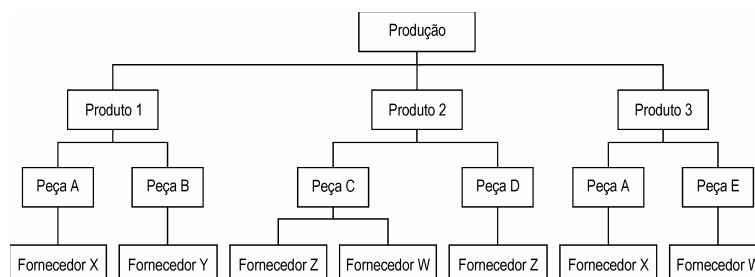
Como vimos, com o surgimento dos SGBDs, para a criação de uma aplicação que acesse Banco de Dados não é mais necessário se preocupar com o armazenamento físico desses dados, dessa forma era necessário que esses dados fossem modelados (mostrados) de uma forma compreensível para o desenvolvedor.

² Em inglês **DBMS (Database Management System)**

Existem diversas formas de mostrar esses dados, esses são os **Modelos de Bancos de Dados**. Lembrando que um modelo é apenas a forma como os dados são mostrados e não como eles são fisicamente armazenados (preocupação do SGBD e não do desenvolvedor).

Modelo Hierárquico

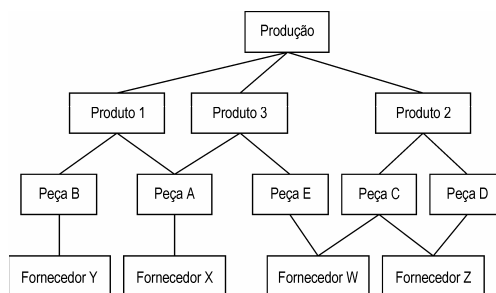
Um dos primeiros modelos a surgir foi o **Modelo Hierárquico**. Esse modelo mostrava os dados na forma de uma “árvore” invertida, conforme figura abaixo.



Nele os dados eram ligados na forma de dados pai (mais acima na estrutura) e dados filhos (mais abaixo na estrutura). Isso gerava uma estrutura muito grande e com muitos dados sendo apresentados de forma repetida, o que poderia gerar dificuldade de compreensão. Observe na figura acima o Fornecedor W, ele aparece duas vezes, porque ele é fornecedor das peças C e E.

Modelo em Rede

Para resolver o problema de representação apresentado pelo Modelo Hierárquico, surgiu o **Modelo de Rede**. Esse modelo é muito semelhante ao anterior, mas com a diferença que cada dado é apresentado somente uma vez.



Por exemplo, observe novamente o Fornecedor W, ele agora aparece somente uma vez, mas com duas ligações (uma para cada peça que ele fornece). O problema desse modelo é que ele pode se tornar bastante confuso, quando as linhas que ligam os diferentes níveis começam a se cruzar.

Esses dois modelos apresentavam os dados de uma forma mais próxima ao armazenamento físico dos arquivos. Com o surgimento dos SGBDs, era necessário mostrar os dados de outra forma.

Modelo Relacional

O **Modelo Relacional** foi criado para permitir que os dados fossem apresentados de uma forma mais próxima da realidade e mais adequada para o uso dos SGBDs. Os dados são apresentados na forma de **Tabelas**, sendo que cada linha da tabela é um relacionamento entre um conjunto de valores.

Alguns conceitos relacionados a esse modelo são apresentados a seguir.

Domínios

Domínios são conjuntos de valores possíveis para uma entidade³ e seus atributos. Abaixo exemplos de domínios da tabela **Roupa** (para os atributos **Código**, **Nome**, **Local** e **Cor**).

Código	Nome	Local	Cor
001	Camisa	São Paulo	Branco
002	Calça	Campinas	Preto
003	Saia	Rio de Janeiro	Vermelho
004	Bermuda	Vitória	Verde
005	Camiseta	Manaus	Azul
		Peruíbe	Amarelo
		Bauru	Rosa

Uma representação dos dados usando o Modelo Relacional ficaria da seguinte forma (tabela **Roupa**).

Código	Nome	Local	Cor
001	Camisa	São Paulo	Branco
002	Calça	Peruíbe	Rosa
003	Camisa	São Paulo	Amarelo
004	Bermuda	Vitória	Verde
005	Camiseta	Manaus	Verde

Observe que nem todos os valores dos domínios foram utilizados, apenas aqueles que tem relação com os objetos lógicos representados (lembre que essa tabela é uma representação lógica dos dados referentes aos objetos físicos).

Nesse modelo as tabelas apresentam as seguintes propriedades:

- 1) As linhas são distintas.
- 2) Os nomes das colunas são únicos.
- 3) A ordem das linhas é irrelevante.
- 4) A ordem das colunas é irrelevante.

Cada coluna de uma tabela do Modelo Relacional, que correspondem aos atributos, é chamada de **Campo** e cada linha de uma tabela é chamada de **Registro** (também conhecida como **Tupla**).⁴

³ Conjunto de objetos da realidade modelada sobre os quais se deseja manter informações no banco de dados.

⁴ Na literatura, quando fazemos o projeto lógico do banco de dados utilizamos as denominações **Atributo** e **Tupla**, e quando fazemos o projeto físico utilizamos **Campo** e **Registro**. Na verdade são a mesma coisa.

Chaves

Chave é o campo (atributo) que identifica de maneira unívoca o registro (tupla), ou seja, para que não existam duas linhas com os dados iguais em todas as colunas. Na tabela exemplo, o campo **Código** é usado para essa identificação. Esse campo é conhecido como **Chave Primária (PK – Primary Key)** da tabela **Roupa**.

Existem casos onde é preciso utilizar mais do que um campo como Chave. Esses campos unidos que representam a chave primária são conhecidos como **Chave Primária Composta** (observe o exemplo abaixo da tabela **Produto**).

Nota Fiscal	Código	Quantidade
N1	P1	100
N1	P2	200
N1	P3	300
N2	P4	400
N2	P1	100
N3	P2	200
N3	P3	300

Obs: como o campo Chave Primária identifica o registro, ele não pode apresentar valor nulo (nulo não significa zero, mas sim sem nenhum valor armazenado).

Chave Estrangeira

Observe as tabelas **Roupa** e **Fornecedor** abaixo:

Roupa					Fornecedor		
PK			FK		PK		
Código	Nome	Cor	Cód_Forn	Local	Código	Nome	Cidade
R1	Camisa	Preto	F2	São Paulo	F1	ABC	São Paulo
R2	Saia	Preto	F1	Bauru	F2	XYZ	Salvador
R3	Calça	Branco	F1	São Paulo	F3	WWW	São Paulo
R4	Camisa	Verde	F3	São Paulo			
R5	Bermuda	Branco	F2	Campinas			

No campo **Cód_Forn** da tabela **Roupa** são encontrados valores correspondentes aos valores encontrados no campo **Código** da tabela **Fornecedor**, ou seja, existe um relacionamento entre esses dois campos. Só é possível acrescentar valores em **Cód_Forn** que tenham correspondência em **Código** (tabela **Fornecedor**). O campo **Cód_Forn** é conhecido como **Chave Estrangeira (FK - Foreign Key)**, ou seja, é um campo que não é chave em uma tabela (tabela **Roupa**) que se relaciona com um campo chave em outra tabela (tabela **Fornecedor**).

As chaves primárias e as chaves estrangeiras fornecem os meios para representar os **Relacionamentos** entre tabelas no Modelo Relacional.

Visão

Visão é uma tabela que é derivada de outras tabelas (uma ou mais) e não existe por si só. As visões são “instantâneos” dos dados armazenados nas tabelas, portanto não existem fisicamente.

As visões são obtidas por meio de **Consultas** e estas são realizadas por meio de **Linguagens de Consulta**, sendo a mais famosa o **SQL (Structured Query Language)**.

No exemplo abaixo é apresentada a tabela **Pessoa_Brasil** que é uma visão da tabela **Pessoa**, na qual são selecionados os registros onde o campo **País** é igual ao valor “*Brasil*”.

Pessoa				Pessoa_Brasil			
Cod	Nome	Cidade	País	Cod	Nome	Cidade	País
01	João	São Paulo	Brasil	01	João	São Paulo	Brasil
02	Peter	Nova Iorque	Estados Unidos	04	Maria	Porto Alegre	Brasil
03	Brigitte	Paris	França				
04	Maria	Porto Alegre	Brasil				
05	Hans	Berlin	Alemanha				
06	Joanna	Los Angeles	Estados Unidos				

Bancos de Dados Multidimensionais

Como já vimos anteriormente, os Bancos de Dados são muito importantes para o processo de tomada de decisões nas organizações, pois eles armazenam os dados das transações que ocorrem no dia-a-dia delas. Os sistemas tradicionais armazenam os dados do que aconteceu na organização, mas determinados dados vão sendo alterados ao longo do processo (como por exemplo, os dados sobre a quantidade de determinado produto em estoque de um supermercado ou os dados sobre a cotação de uma determinada moeda estrangeira). Esses são os **Sistemas Transacionais** e utilizam um processamento conhecido como **OLTP (On-Line Transaction Processing)**.

Com o tempo, surgiu a necessidade de dados históricos e consolidados (de diferentes fontes) sobre as transações realizadas nas organizações. Isso ocorreu devido à necessidade de dados consolidados para a tomada de decisão em um nível mais estratégico da organização. Surgiu um outro tipo de processamento o **OLAP (On-Line Analytical Processing)**. A tabela abaixo apresenta uma comparação de algumas características dos dois tipos de processamento.

Características	Sistemas Transacionais(OLTP)	Sistemas Analíticos(OLAP)
Exemplos	CRM, ERP, Supply Chain	SIG, SAD, SIE
Atualizações	Mais freqüentes	Menos freqüentes
Tipo de Informação	Detalhes (maior granularidade)	Agrupamento (menor granularidade)
Quantidade de Dados	Poucos	Muitos
Precisão	Dados atuais	Dados históricos
Complexidade do Resultado da Pesquisa (para o negócio)	Baixa	Alta
Terminologia	Linhas e Colunas	Dimensões, Medidas e Fatos

Para apoiar esse tipo de processamento surgiram os **Bancos de Dados Multidimensionais**, também conhecidos como **Data Warehouses**. Um Data Warehouse é um conjunto de bancos de dados, geralmente do modelo relacional, que consolida as informações empresariais, provenientes das mais distintas fontes de dados.

A sua principal característica é ser a consolidação de dados de diferentes bancos de dados transacionais e sempre ter que incorporar um componente temporal. Simplificando para facilitar a compreensão, podemos imaginar diversas tabelas do modelo relacional, agrupadas e sobrepostas, sendo que cada uma dessas tabelas está relacionada com um período de tempo (uma tabela por dia, semana, mês ou ano). É possível imaginar como se as tabelas fossem páginas que vão sendo colocadas umas sobre as outras, representando o instantâneo dos dados consolidados naquele determinado período de tempo.

O processo de construção desse tipo de banco de dados é moroso e complexo, principalmente devido à grande quantidade de dados, provenientes de diversas fontes heterogêneas de dados, algumas vezes inconsistentes e envolvendo várias áreas diferentes da organização.

A dificuldade de consolidação dos dados transacionais é decorrente de:

- ♦ Várias fontes heterogêneas de dados;
- ♦ Dados de entrada precisam se “limpos”, ou seja, adequados a um formato padrão;
- ♦ A periodicidade de obtenção dos dados (granularidade) deve ser ajustada;
- ♦ Pode ser preciso resumir os dados; e
- ♦ Necessidade de um componente temporal, nem sempre presente nas fontes de dados.

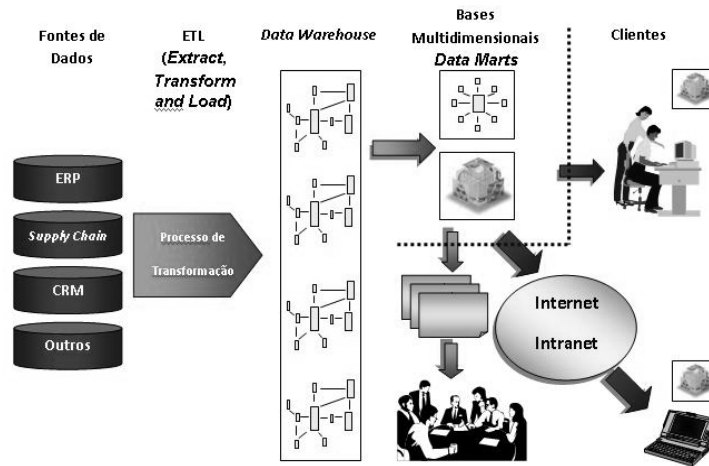
Como exemplo, podemos mostrar algumas situações comuns encontradas nas fontes de dados para a montagem de bancos de dados multidimensionais. São elas:

- ♦ Mesmos dados com nomes de campos diferentes;
- ♦ Dados diferentes com o mesmo nome de campo;
- ♦ Dados exclusivos de uma determinada aplicação; e
- ♦ Chaves diferentes para um mesmo tipo de tabela.

Tão importante quanto saber quais dados armazenar, é saber quando e quais dados remover do Data Warehouse, porque com o tempo esse tipo de banco de dados começa a ficar muito grande. Algumas estratégias existentes, e que podem ser utilizadas em conjunto, são:

- ♦ Resumir dados mais antigos; e
- ♦ Armazenar dos dados antigos em mídias de armazenamento mais baratas (como fitas, por exemplo);

Um esquema de banco de dados multidimensional pode ser visto na figura abaixo:



Obs: Data Marts podem ser compreendidos como sendo Data Warehouses departamentais ou intermediários.

CAPÍTULO II – Modelagem Conceitual – Entidades e Atributos

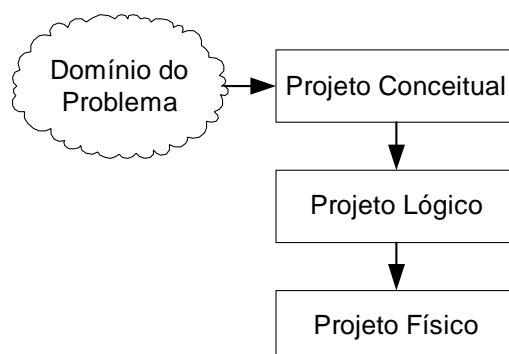
Os sistemas computadorizados, para que possam ser utilizados, requerem um certo grau de sistematização das atividades e a identificação de uma série de elementos-chave.

Algumas definições são elaboradas em nível corporativo, servindo como orientação para todas as atividades da organização. São definições estratégicas, relacionadas às políticas adotadas na condução dos negócios e às próprias características do ramo de atividades em si. Pode-se definir personagens, eventos e procedimentos envolvidos no dia a dia da organização; características particulares a cada um desses elementos; relacionamentos existentes entre eles; objetivos da organização; prioridades, etc. São as chamadas **regras de negócio**.

As regras de negócio são a base para o detalhamento das informações necessárias ao desempenho das atividades normais de cada setor.

A tarefa de levantamento e estruturação dos dados segundo as regras de negócio da organização é chamada **Modelagem de Dados**.

O **Modelo de Dados** descreve de maneira formal os tipos de dados que estão armazenados em um banco de dados. Para a construção de um modelo de dados é necessária a utilização de uma linguagem de Modelagem de Dados.



A modelagem possui os seguintes objetivos principais:

- Identificar os fatos (estruturas de dados) a serem armazenados no banco de dados, espelhando o mundo real (problema a ser resolvido)
- Construir modelos **conceituais**

Modelo Conceitual – Apresenta a estrutura dos dados que podem aparecer no banco de dados, independente do aplicativo que será utilizado (SGBD)

Modelo Lógico – Apresenta o nível de abstração visto pelo usuário do SGBD, portanto depende do aplicativo que será utilizado.

Modelo Físico – Apresenta os detalhes de armazenamento interno de informações que, apesar de não influenciarem a programação de aplicações, influenciam a performance delas.

Este capítulo começa a tratar da **Modelagem Conceitual**. O Modelo Conceitual deve apresentar as seguintes características:

- Deve registrar a estrutura dos dados que podem aparecer no banco de dados.
- O modelo deve ser independente do Sistema Gerenciador de Banco de Dados, portanto não registrando como os dados estão armazenados nele.

O Modelo Conceitual pode apresentar duas interpretações:

1. Modelo da organização – define as entidades da organização que tem informações armazenadas no banco de dados
2. Modelo do banco de dados – define que arquivos (tabelas) farão parte do banco de dados.

Modelo Entidade-Relacionamento (MER)

O **Modelo Entidade-Relacionamento (MER)** foi criado em 1976 por Peter Chen. É o modelo mais utilizado atualmente, devido, principalmente, à sua simplicidade e eficiência. Baseia-se na percepção do mundo real, que consiste em uma coleção de objetos básicos, chamados **Entidades**, e em **Relacionamentos** entre esses objetos. Veremos aqui como modelar um sistema, identificando as entidades e os relacionamentos, e utilizando o **Diagrama Entidade-Relacionamento (DER)**. Para isso, precisamos conhecer suas características, os elementos que o compõem, e como representá-lo.

Não é a única abordagem para a modelagem conceitual, existem também:

- NIAM/ORM (técnica europeia da década de 70)
- UML (técnica para modelos orientados a objeto) – apesar de usar uma abordagem orientada a objetos, a UML se baseia nos conceitos da MER.

O Modelo Entidade-Relacionamento possui diversos componentes. Vamos apresentá-los.

Entidades

Conjunto de objetos da realidade modelada sobre os quais deseja-se manter informações no banco de dados. É algo no ambiente do domínio do problema.

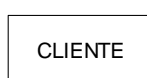
Exemplo: Em um sistema de um Banco, as seguintes entidades são possíveis:

- Clientes
- Contas correntes
- Transações
- Agências

A entidade pode ser:

- Objeto Concreto – pessoa, automóvel, etc.
- Objeto Abstrato – departamento, qualidade, etc.

Simbologia – retângulo com o nome da entidade.



Instância é um objeto em particular, por exemplo, uma determinada pessoa, um determinado departamento.

A entidade isoladamente não informa nada, por isso possuem propriedades, tais como:

- Atributos
- Relacionamentos
- Generalizações / Especializações

Atributos

Dado, ou informação, que é associada a cada ocorrência de uma entidade ou de um relacionamento. Ele descreve a entidade.

Ao observarmos objetos (entidades) em um ambiente, estaremos, na verdade, reconhecendo tais objetos através da identificação de suas características próprias. Essas características serão as mesmas para todos os elementos de um determinado conjunto.

Classificação dos Atributos quanto a sua finalidade

Quanto à finalidade, os atributos de uma entidade podem ser classificados como:

- **Atributos Descritivos** – representam características intrínsecas das entidades. Todo atributo pode ser classificado como sendo do tipo Descritivo.
- **Atributos Nominativos** – definidores de nomes ou rótulos de identificação das entidades (Exemplo: Código de..., Nome de..., Sigla do...). Provavelmente serão atributos identificadores (como veremos a seguir) e serão Chaves candidatas no Modelo Lógico.
- **Atributos Referenciais** – não pertencem propriamente a entidade onde estão alocados, mas fazem algum tipo de citação, ou ligação, desse objeto com outro (Exemplo, Local de trabalho, Nome do fabricante).

Simbologia – atributos são representados apenas pelo seu nome ligado à entidade por uma linha reta.



Obs: o nome de um atributo deve ser único dentro do modelo de dados.

Podemos ter vários tipos de atributos: simples, composto, multivalorado e determinante. Vejamos as características de cada um deles.

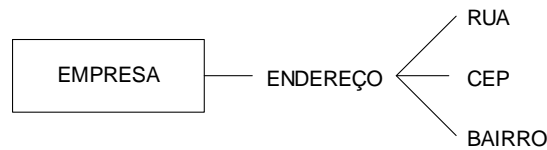
Atributo Simples

Não possui qualquer característica especial. Por exemplo, o nome da empresa é um atributo sem qualquer característica especial.



Atributo Composto

O seu conteúdo é formado por itens menores. Por exemplo, Endereço é formado por Rua, CEP e Bairro.



Atributo Multivalorado

O seu conteúdo pode ser formado por mais de uma informação. É indicado colocando-se um asterisco precedendo o nome do atributo, como no caso da empresa que pode possuir mais do que um telefone.



Atributo Armazenado e Derivado

São atributos fortemente relacionados, ou seja, um atributo é derivado, a partir de um atributo armazenado. Por exemplo, o atributo idade de uma pessoa pode ser derivado a partir do atributo data de nascimento.

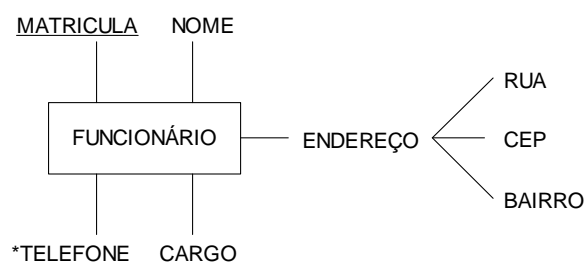
Atributo Chave, Determinante ou Identificador

O atributo chave (determinante ou identificador) é aquele que define univocamente as instâncias de uma entidade, ou seja, é *único* para as instâncias de uma entidade. É indicado sublinhando-se o nome do atributo. No exemplo de uma empresa, o CNPJ é um atributo chave, pois não podem existir duas empresas com o mesmo valor para esse atributo. No exemplo da escola, a Matrícula é um atributo chave, pois não pode existir mais de um funcionário com a mesma matrícula.



Obs: o atributo identificador pode ser composto por um único atributo ou por uma coleção de atributos.

Vejamos um exemplo no qual se aplicam os conceitos vistos até aqui:



Observe os seguintes pontos:

- Apesar de um atributo *Nome* parecer um atributo chave, não podemos esquecer que existem pessoas com nomes iguais, ou mesmo, quando não é o caso, que se tornam iguais na abreviatura. Exemplo: José Pinheiro Costa (José P. Costa) e José Pereira Costa (José P. Costa). Já a *Matrícula* tem que ser única, pois identifica o funcionário na empresa.

- O *Endereço* é um atributo composto de *Rua*, *CEP* e *Bairro*, logo sua representação foi explicitada. Não é errado colocar os atributos *Rua*, *CEP* e *Bairro* diretamente ligados à entidade *Funcionário*, porém a utilização do atributo composto se mostra menos “poluente” para o diagrama e este, portanto, fica mais fácil de ser entendido.
- O *Telefone* foi explicitado como multivalorado; logo, foi precedido de um asterisco. Isso não significa, no entanto, que todos os funcionários tenham, obrigatoriamente, mais de um telefone, nem mesmo que tenham telefone.

Construindo Modelos Entidade-Relacionamento (MER)

O modelo deve ser preciso e não ambíguo. Diferentes leitores de um mesmo modelo Entidade-Relacionamento devem sempre entender exatamente o mesmo. Além disso, dois modelos entidade-relacionamento diferentes podem ser equivalentes.

Ele apresenta apenas algumas propriedades de um banco de dados (concebido para bancos de dados relacionais), sendo pouco poderoso para expressar restrições de integridade (regras de negócio).

Para construir um modelo de um objeto da realidade:

- Não pode ser feita uma observação isolada do objeto. Necessidade de conhecer o contexto onde o objeto está inserido
- A decisão sobre uma construção (layout dos componentes do modelo) está sujeita a alterações durante a modelagem. O desenvolvimento do modelo e o aprendizado sobre a realidade modelada vão refinar e aperfeiçoar o modelo.

Identificação das Entidades

A análise, para identificação das entidades que devem fazer parte do modelo, inicia com o enunciado de um problema gerado pelos clientes e, possivelmente, pelos desenvolvedores. Esse enunciado pode ser incompleto e informal.

Deve-se tornar o enunciado mais preciso, com isso, expondo as suas inconsistências e ambigüidades.

Normalmente, o enunciado surge a partir da fase de Análise, dentro do desenvolvimento de um sistema. É necessário trabalhar junto com o cliente para refinar os requisitos que constam do enunciado, pois esses requisitos representam a verdadeira intenção do cliente.

Essa etapa envolve a confirmação de requisitos que estejam explícitos no enunciado do problema e a busca de informações que estejam faltando sobre os requisitos que estejam implícitos.

É necessário utilizar uma certa técnica para a escolha das entidades que deverão fazer parte do modelo (e, por conseguinte, parte do banco de dados resultante). A seguir, é apresentada uma seqüência de atividades que podem auxiliar na identificação das entidades para o Modelo Conceitual.

1. Identificação de Entidade candidatas

O primeiro detalhe importante é que, muitas vezes, as entidades correspondem aos substantivos do enunciado do problema.

Segundo Shlaer e Mellor, em seu livro *Object-Oriented System Analysis, Modelling the World of Data*, é importante observar cinco grandes grupos de elementos:

- **Coisas tangíveis** – tudo aquilo que pode ser tocado.
- **Funções exercidas por objetos, ou elementos** – especifica a atuação do elemento no ambiente onde está inserido.
Exemplo: Médico (Cirurgião, Pediatra), Engenheiro (Civil, Naval)
- **Eventos ou ocorrências** – que podem ser materializáveis.
Exemplo: Voo comercial, Acidente de trânsito, Jogo de Futebol
- **Interações** – materialização da interação entre objetos, onde cada objeto preserva suas características.
Exemplo: Compra de Automóvel
- **Especificações** – especificações que quando aplicadas ou seguidas darão origem a objetos (entidades).

Podem ser tratados como:

- a. Podem ser omitidos do modelo conceitual, desde que os atributos sejam alocados às entidades aos quais se referenciam.

Exemplo:

Refrigerador
Cor
Capacidade principal
Voltagem
Modelo
Tempo de produção
Altura
Largura
Profundidade
Data de produção
Número de série

- b. Podem ser mantidos no modelo conceitual, mas retratarão uma realidade mais próxima do Modelo Lógico (o que pode ser prejudicial).

Exemplo:

Refrigerador	Modelo de Refrigerador
Cor	Capacidade principal
Modelo	Voltagem
Tempo de produção	Modelo
Data de produção	Altura
Número de série	Largura
	Profundidade

Obs: O nível de abstração poderá ser maior ou menor, mas definido no processo de especificação de requisitos.

Como foi dito anteriormente, a entidade pode ser um objeto concreto (pessoa, automóvel, etc.) ou um objeto abstrato (departamento, qualidade, etc.), mas devem fazer sentido no domínio do problema (domínio da aplicação a ser desenvolvida). Isso significa, escolher entidades sobre as quais é necessário armazenar dados, para que a aplicação funcione de acordo com as necessidades do cliente.

É necessário evitar construções de implementação em computador como entidades (por exemplo, listas encadeadas, sub-rotinas, etc.).

Nem todas as entidades surgem explicitamente, a partir do enunciado do problema, algumas estão implícitas no domínio da aplicação (entidades necessárias para que a aplicação funcione) ou no conhecimento geral (entidades que não são enunciadas, por serem evidentes).

Comece listando as entidades candidatas (nesse momento, não seja muito seletivo). Um detalhe importante é que, muitas vezes, as entidades correspondem aos substantivos do enunciado do problema.

Algumas perguntas úteis:

- Que coisas são trabalhadas?
- O que pode ser identificado por número, código?
- Tem atributos? Esses atributos são relevantes, pertinentes?
- Essa coisa pode assumir a forma de uma tabela?
- É um documento externo (recibo, fatura, nota fiscal)? Se sim, é forte candidato a entidade.
- Tem significado próprio?
- Qual a entidade principal do contexto?

Dicas:

- Substantivos que não possuem atributos podem ser atributos de outras entidades.
- Adjetivos colocados pelos usuários indicam normalmente atributos de uma entidade.
- Verbos indicam prováveis relacionamentos (que serão vistos no próximo capítulo)
- Advérbios temporais indicam prováveis atributos de um relacionamento
- Procure sempre visualizar qual é a entidade principal do contexto sob análise
- Entidades cujo nome termine por “ento” ou por “ão” geralmente são procedimentos

2. Regras para a conservação das Entidades corretas

Com a lista de entidades candidatas em mãos, descarte aquelas que são desnecessárias e incorretas, de acordo com os critérios a seguir:

a. Entidades Redundantes

Se duas entidades expressarem a mesma informação, o nome de descreva melhor essa entidade deve ser escolhido.

Exemplo: em um sistema bancário, as entidades *Cliente* e *Usuário* podem ser redundantes.

b. Entidades Irrelevantes

Se uma entidade tiver pouco, ou nada, relacionado com o problema, deve ser eliminada da lista. É necessário ser criterioso nessa escolha, deve-se observar o contexto do problema.

Exemplo: em um sistema bancário, a entidade *Preço do Equipamento* pode ser irrelevante.

c. Entidades Vagas

A entidade deve ser específica. Algumas entidades podem ter limites mal definidos ou abrangência demasiadamente ampla.

Exemplo: em um sistema bancário, a entidade *Rede Bancária* pode ser vaga.

d. Atributos

Nomes que descrevem objetos isolados devem ser considerados como atributos (que veremos adiante, nesse capítulo). Somente o transforme em uma entidade se a sua existência for importante.

Exemplo: em um sistema bancário, a entidade *Dados de Transações* ou *Extrato* são atributos.

e. Operações

Se um nome descreve uma operação que é aplicada a uma entidade e não é manipulada em si mesma, então não é uma entidade. Operações com características próprias podem ser modeladas como entidades.

Exemplo: *Chamada Telefônica*. Em um sistema de controle de funcionários é uma entidade, mas em um sistema telefônico é apenas uma operação.

f. Papéis

O nome da entidade deve refletir a sua natureza intrínseca e não o papel que ela desempenha em um relacionamento. Além disso, uma entidade física pode corresponder a diversas entidades.

Exemplo: *Proprietário* não seria um bom nome para uma entidade, em um sistema de uma montadora de automóveis. *Pessoa* e *Empregado* podem corresponder à mesma entidade, dependendo do sistema.

g. Construções de Implementação

Construções inadequadas ao mundo real devem ser eliminadas no modelo analisado.

Exemplo: *Software*, *Registro de Transações* são construções de implementação e não entidades.

3. Dois critérios práticos para distinguir as entidades dos atributos

Deve-se ter em mente esses dois critérios durante todo o processo de escolha das entidades.

Critério 1:

- Objeto está vinculado a outros objetos – modelado como entidade
- Objeto não está vinculado a outros objetos – modelado como atributo

Critério 2:

- Conjunto de valores de um determinado objeto é fixo (domínio fixo) – modelado como atributo
- Existem transações no sistema que alteram o conjunto de valores do objeto (domínio variável) – modelado como entidade

CAPÍTULO III – Modelagem Conceitual – Relacionamentos

Como vimos no capítulo anterior, uma entidade possui, principalmente, atributos e relacionamentos. Relacionamentos esses entre ela e uma, ou mais, entidades.

Identificação dos relacionamentos

Muitas vezes a observação de um relacionamento será o ponto de partida para a identificação das entidades relacionadas a ele.

Existem dois grupos básicos de relacionamentos: Relacionamentos entre instâncias de objetos de diferentes tipos e relacionamentos entre instâncias de um mesmo tipo de objeto.

Relacionamentos entre instâncias de objetos de diferentes tipos

Relacionamento onde, no papel de entidades formadoras, ou participantes, do relacionamento encontramos entidades diferentes.

As etapas a serem seguidas para trabalhar com esse tipo de relacionamento são:

1. Identificar os objetos envolvidos – processo já apresentado anteriormente
2. Caracterizar os objetos – processo já apresentado anteriormente (identificação de atributos)
3. Representar os objetos – processo já apresentado anteriormente
4. Identificar os novos relacionamentos entre os objetos

Denominação dos Relacionamentos – Basicamente, um relacionamento é expresso através de uma construção verbal.

Ex: Relacionamento entre os objetos CARRO e PESSOA.

PESSOA **utiliza** CARRO ou CARRO **é utilizado** por PESSOA.

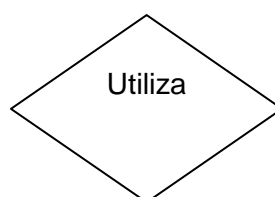
5. Caracterizar os relacionamentos entre os objetos

Ex: Utilizando o exemplo do relacionamento entre os objetos CARRO e PESSOA.

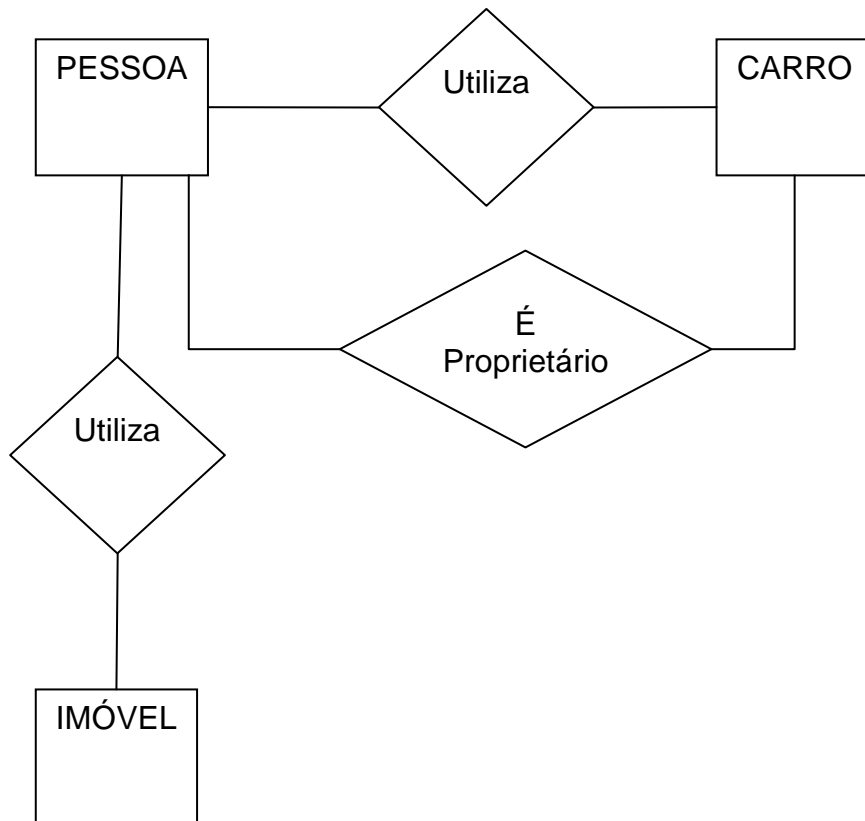
- Nem toda PESSOA utiliza um CARRO.
- Um CARRO pode ser utilizado por uma, ou mais, PESSOAS.
- Algumas PESSOAS utilizam mais de um CARRO.
- Um CARRO sempre será utilizado por, pelo menos, uma PESSOA.

6. Representar os novos relacionamentos no modelo

Simbologia:



Exemplo:



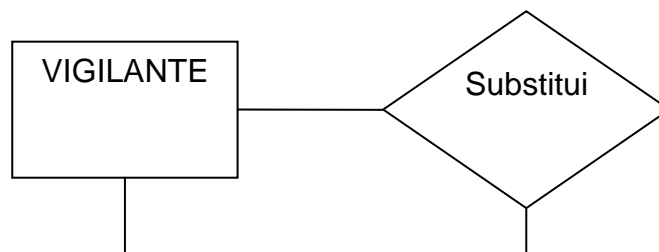
Como convenção, adota-se que o relacionamento deve ser entendido da esquerda para a direita ou de cima para baixo.

Ex: PESSOA utiliza o CARRO, e não vice-versa.

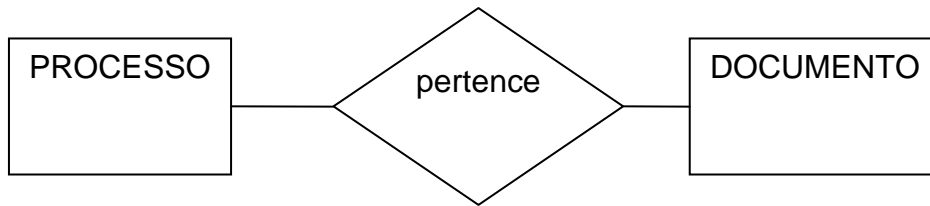
Relacionamentos entre instâncias de um mesmo tipo de objeto

Relacionamento onde, no papel de entidades formadoras, ou participantes, do relacionamento encontramos o mesmo tipo de entidade.

As etapas a serem seguidas são as mesmas apresentadas no item anterior, apenas ressaltando o modo como será representado esse relacionamento:



Nem sempre o DER é suficiente para definir o relacionamento entre entidades. Observe o exemplo abaixo:



Nesse caso poderia existir uma incerteza quanto ao relacionamento, pois tanto o documento pode pertencer ao documento, como o documento pode pertencer ao processo. Para que essa anomalia seja sanada é necessário criar o Dicionário de Relacionamentos.

Caracterização dos Relacionamentos

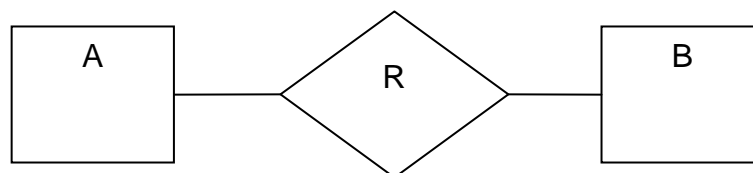
Esse é um dos pontos mais importantes da criação de um modelo conceitual íntegro e deve se basear no atendimento de alguns requisitos:

- Grau, ou Cardinalidade, do relacionamento
- Número de elementos que participam do relacionamento
- Condição de participação dos elementos no relacionamento
- Condição de estabelecimento do relacionamento

Cardinalidade ou Grau do Relacionamento

Esse requisito procura apresentar regras para o estabelecimento das associações entre os elementos. Essas regras não devem ser impostas, mas sim observadas no ambiente em questão.

Genericamente, temos:



Devemos responder às seguintes perguntas:

1. Com quantos elementos do tipo B pode se relacionar *cada um* dos elementos do tipo A?
2. *Dado um* elemento do tipo B, com quantos elementos do tipo A ele se relaciona?

Exemplo: Vamos utilizar o relacionamento entre PESSOA e CARRO, apresentado anteriormente.

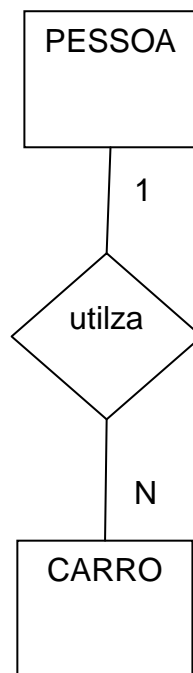
Quantos CARROS utiliza cada uma das PESSOAS?

R: Uma PESSOA utiliza vários CARROS, um de cada vez.

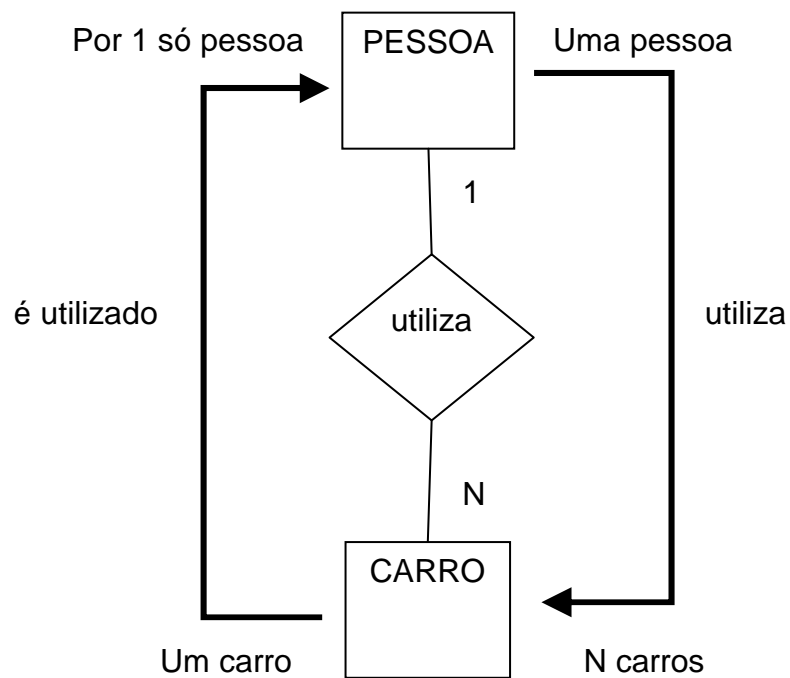
Dado um CARRO, por quantas PESSOAS ele pode ser utilizado?

R: Um CARRO é utilizado por uma única PESSOA, em nosso caso.

A representação desse relacionamento é:



O raciocínio seguido é:



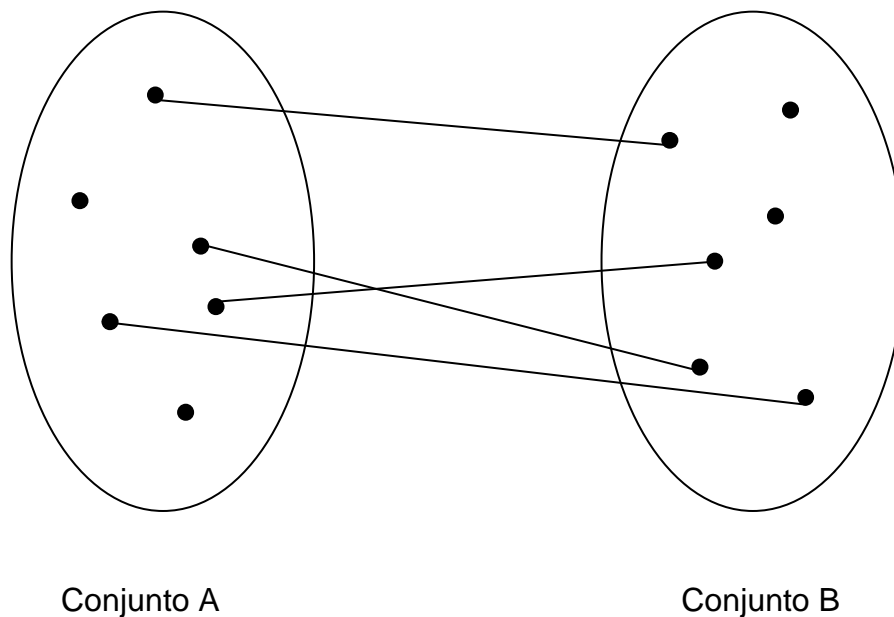
Obs: Sempre a análise e descrição do relacionamento deve ter como ponto de partida **um elemento individualizado**. A indicação do grau do relacionamento é colocado na outra entidade.

Os graus de relacionamento, ou cardinalidade, podem ser dos seguintes tipos:

- 1:1 (Um para Um)
- 1:N (Um para Muitos)
- M:N (Muitos para Muitos)

Relacionamentos 1:1

Os relacionamentos 1:1 são casos especiais do relacionamento 1:N, onde N é igual a 1. Nesse tipo de relacionamento um elemento do tipo A se relaciona com um, e somente um elemento do tipo B e um elemento do tipo B se relaciona com um, e somente um elemento do tipo A.



Obs: Nesse momento da modelagem de dados não estamos ainda preocupados com detalhes do tipo: algum elemento do conjunto A não se relacionar com nenhum elemento do conjunto B ou nenhum elemento de B se relaciona com qualquer elemento de A.

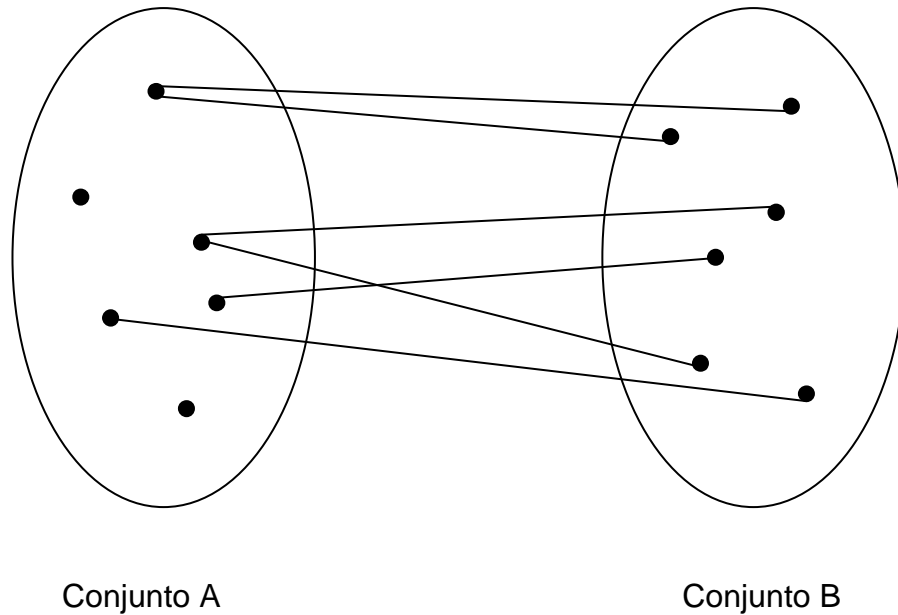
Exemplo:



Esse tipo de relacionamento é bastante difícil de ser caracterizado, pois qualquer modificação na interpretação do ambiente levará a uma mudança no grau do relacionamento. Por exemplo, se estivermos observando um ambiente onde o que está sendo observado é o histórico de casamento das pessoas, MARIDO pode estar relacionado com mais de uma ESPOSA.

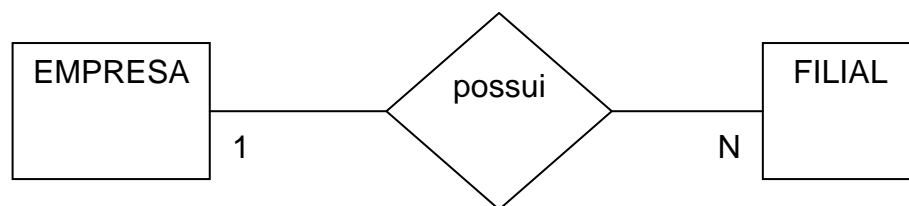
Relacionamentos 1:N

Nesse tipo de relacionamento um elemento do tipo A se relaciona com vários elementos do tipo B, mas um elemento do tipo B se relaciona com um, e somente um elemento do tipo A.

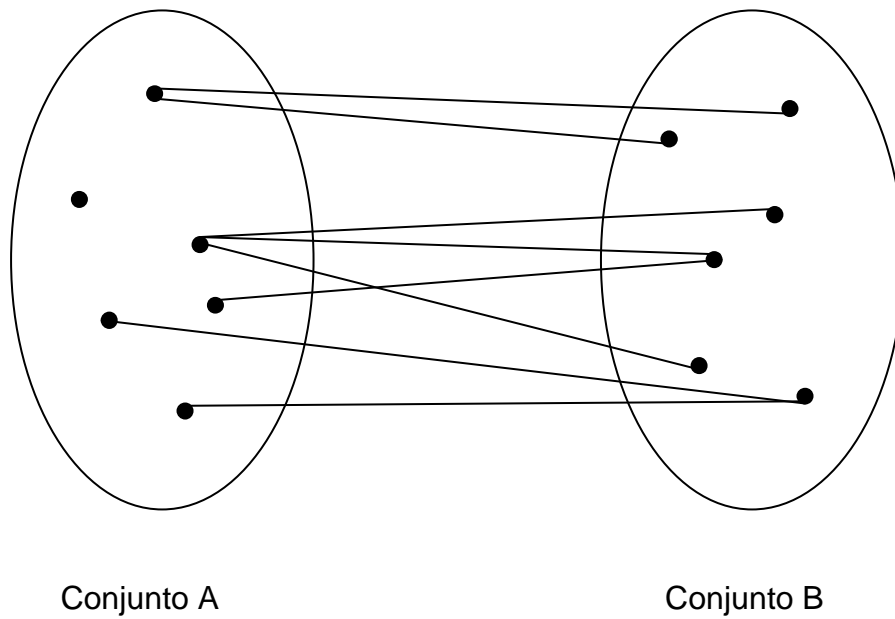


Como analogia, podemos imaginar esse relacionamento como uma estrutura em árvore, onde uma raiz pode dar origem à vários ramos, mas um ramo só será originado de uma raiz.

Exemplo:

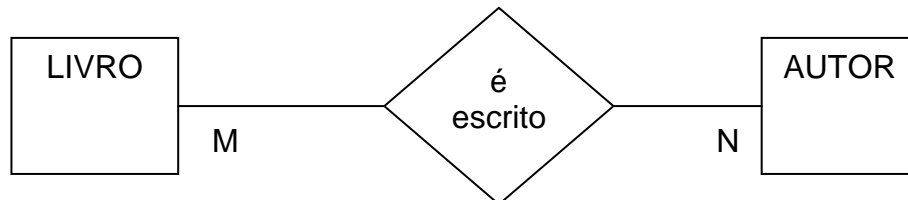
**Relacionamentos M:N**

Nesse tipo de relacionamento, teoricamente, não existem restrições quanto a relacionamentos entre elementos do tipo A e elementos do tipo B.



Obs: Usamos a notação M:N para denotar que as duas grandezas são independentes, ou seja, o número de elementos do tipo A que se relacionam com elementos do tipo B, pode ser diferente do número de elementos do tipo B que se relacionam com elementos do tipo A.

Exemplos



Número de Elementos que Participam do Relacionamento

Um relacionamento pode ser estabelecido entre *dois ou mais* elementos, e não somente entre dois elementos (no máximo) como pode denotar os exemplos utilizados até agora. Portanto, são permitidos relacionamentos binários (duas entidades), ternário (três entidades), etc.

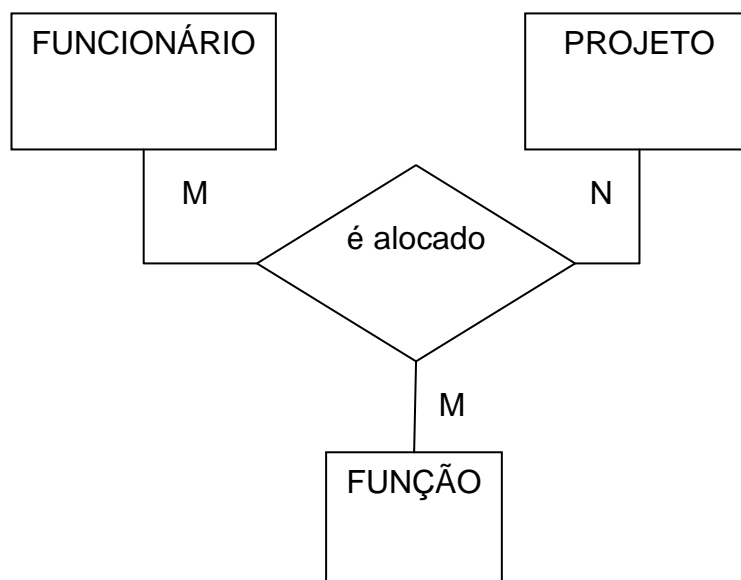
Observemos o exemplo abaixo:

Funcionário	Projeto	Função
Antonio Carlos	Recursos Humanos	Analista de Sistemas
Antonio Carlos	Adm. Financeira	Consultor
Antonio Carlos	Adm. Materiais	Analista de Sistemas
Carla Martins	Recursos Humanos	Programador
Carla Martins	Adm. Tráfego	Usuário
Marcos Silveira	Adm. Materiais	Consultor

Observe que se quisermos definir adequadamente o modelo de dados desse ambiente observado, deveríamos criar um relacionamento entre as três entidades (FUNCIONÁRIO, PROJETO e FUNÇÃO).

Para entender melhor, observe a seguinte questão: Quem é o Analista de Sistemas de um determinado Projeto? Para responder essa questão são necessários dados (relacionados) às três entidades.

A representação de um relacionamento ternário é:



Presença dos Elementos no Relacionamento

Outro aspecto importante, sobre relacionamentos, que deve ser levado em consideração quando da modelagem conceitual, é a obrigatoriedade ou não da participação dos elementos nas associações.

A resposta para essa questão deve ser obtida a partir da observação do ambiente a ser analisado.

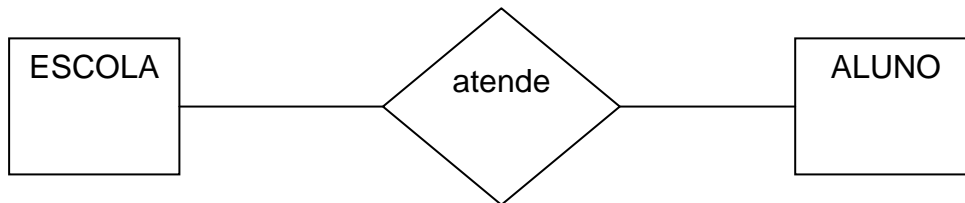
Existem duas formas de descrever essa presença dos elementos no relacionamento. Uma delas é quando usamos a notação de Relacionamentos Incondicionais e os Relacionamentos Condicionais.

Até agora, apresentamos a representação dos relacionamentos como se eles fossem Relacionamentos Incondicionais, ou seja, ele se enquadra somente como 1:1, 1:N ou M:N. Mas, esse tipo de representação não apresenta informações suficientes em boa parte dos ambientes estudados. Para melhorar a representação do relacionamento entre entidades é necessário acrescentar o conceito da cardinalidade (grau) mínima e máxima.

O Cardinalidade (Grau) Máximo é representado pelo Cardinalidade (Grau) que representamos até agora, podendo ser 1:1, 1:N ou M:N.

O Cardinalidade (Grau) Mínimo indica o menor valor possível de participação dos elementos do conjunto A e B no relacionamento.

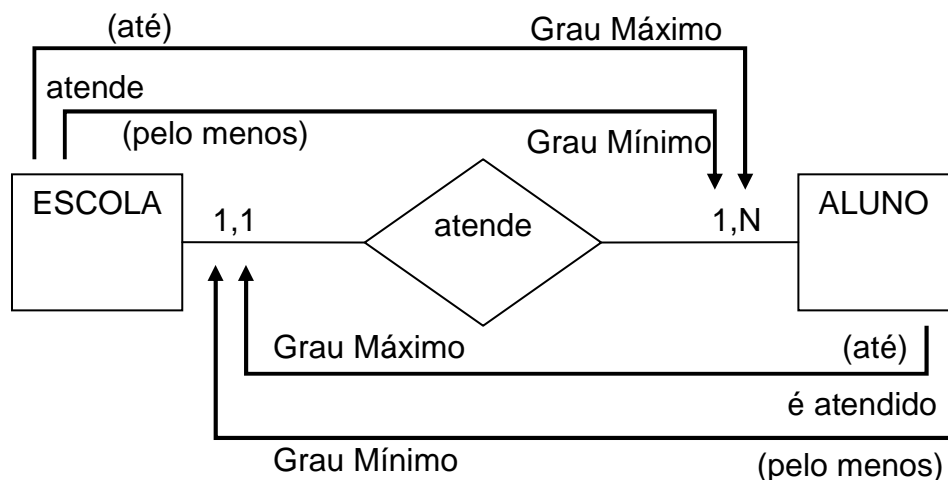
Observemos o exemplo das entidades ESCOLA e ALUNO:



Analisemos o caso:

- Uma ESCOLA *atende* **no mínimo (pelo menos) 1 ALUNO** e **no máximo (até) N ALUNOS**.
- Um ALUNO *é atendido* **no mínimo (pelo menos) por 1 ESCOLA** e **no máximo (até) por 1 ESCOLA**.

A representação dos graus mínimo e máximo do relacionamento é:



O Cardinalidade (Grau) Mínimo pode assumir dois valores:

- 0 (zero) – indica a possibilidade de não participação ou opcionalidade
- 1 (um) – indica a obrigatoriedade de participação

Quanto a Existência Simultânea de Relacionamentos

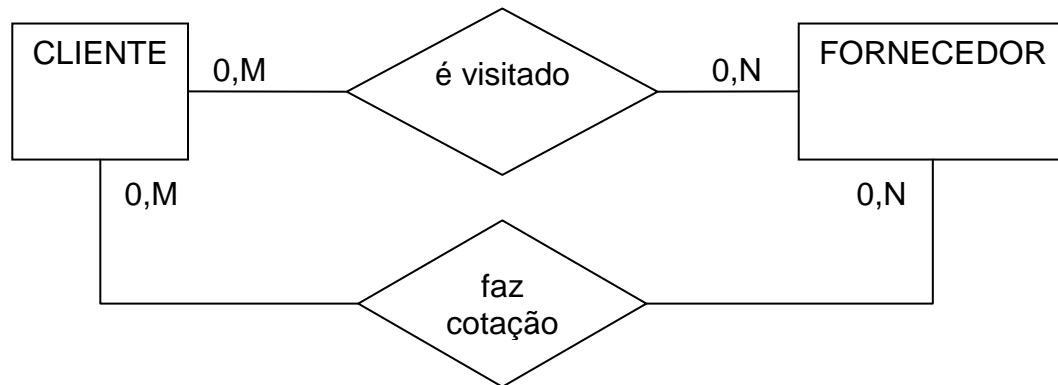
Uma entidade pode estabelecer relacionamentos simultâneos com diversas outras entidades. Podemos ter três tipos de associações simultâneas:

- Relacionamentos Independentes
- Relacionamentos Contingentes
- Relacionamentos Mutuamente Exclusivos

Relacionamentos Independentes

Relacionamentos que possam ser estabelecidos sem que haja necessidade de avaliação simultânea de outro relacionamento.

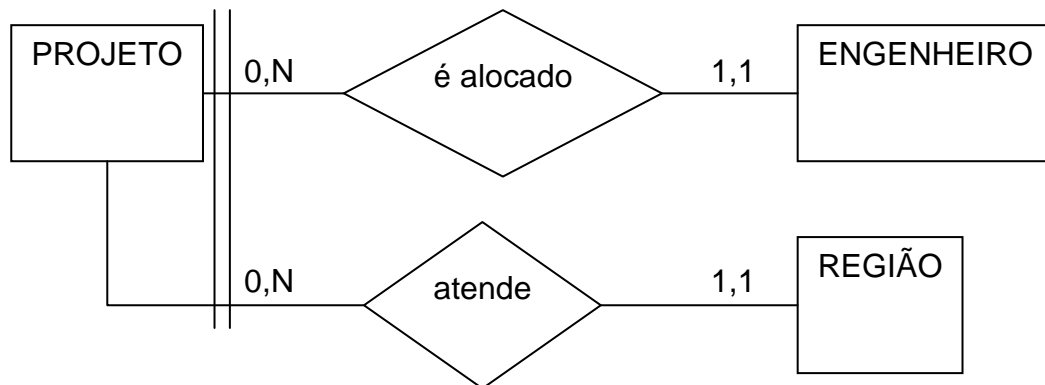
Exemplo:



Relacionamentos Contingentes

São os relacionamentos que, tendo dependência uns com os outros, impõem o estabelecimento simultâneo de associações entre os vários elementos envolvidos. São típicos em associações que se processam com uma mesma finalidade, mas que envolvem elementos diferentes.

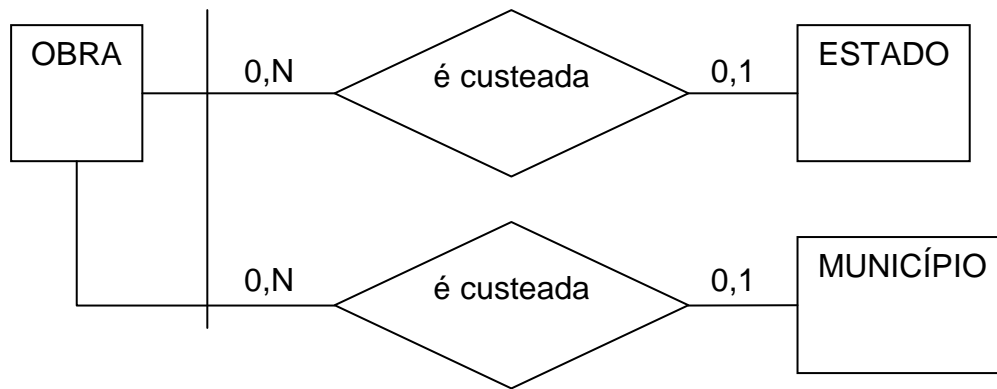
A representação desse tipo de relacionamento no modelo é:



Relacionamentos Mutuamente Exclusivos

Acontece se a associação de um elemento for estabelecida através de um dos relacionamentos, não poderá ser estabelecida pelos demais.

A representação desse tipo de relacionamento no modelo é:

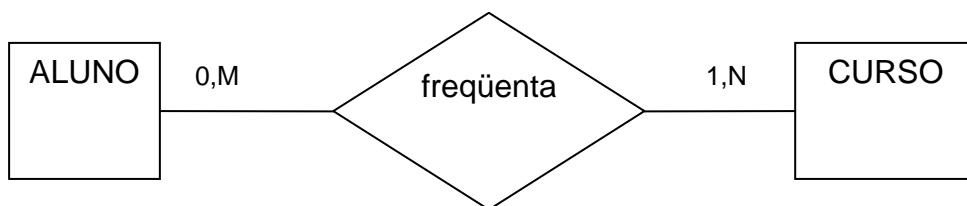


Relacionamentos com Atributos

Em alguns casos é necessário, para que o modelo conceitual reflita a realidade do ambiente observado, representar determinados atributos como sendo alocados ao relacionamento. Isso acontece quando o atributo não está ligado unicamente a uma ou outra entidade, mas sim às duas relacionadas, portanto ao relacionamento.

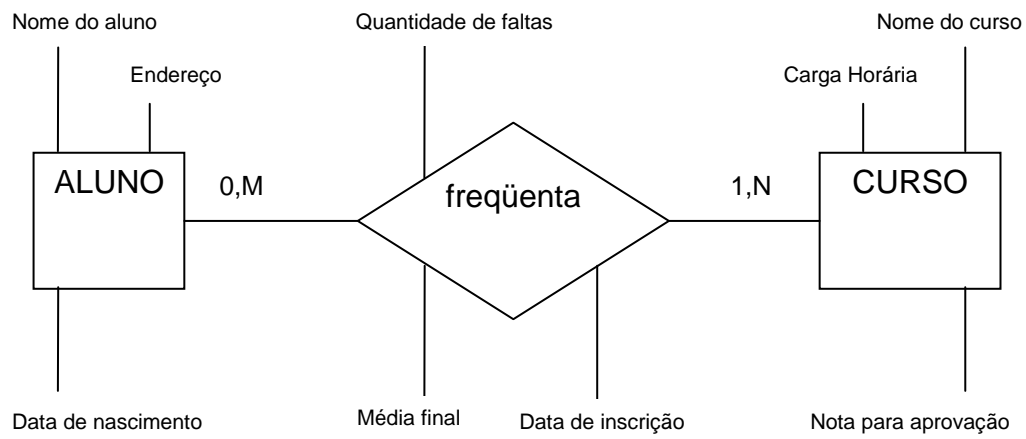
Para identificar isso, basta observar se o atributo tem sentido, sem a existência do relacionamento, nesse caso ele não deve ser alocado ao relacionamento, mas sim a uma entidade. Caso contrário, o atributo deve ser alocado ao relacionamento.

Observemos o exemplo abaixo:



Atributo	Local de Alocação	Proprietário
Nome do aluno	ENTIDADE	ALUNO
Endereço	ENTIDADE	ALUNO
Data de nascimento	ENTIDADE	ALUNO
Nome do curso	ENTIDADE	CURSO
Carga horária	ENTIDADE	CURSO
Nota para aprovação	ENTIDADE	CURSO
Quantidade de faltas	RELACIONAMENTO	freqüenta
Média final	RELACIONAMENTO	freqüenta
Data de inscrição	RELACIONAMENTO	freqüenta

A representação no modelo será:



CAPÍTULO IV – Modelagem Conceitual – Dicionário de Dados

Como vimos nos capítulos anteriores, na maioria dos casos, o Diagrama Entidade-Relacionamento não é suficiente para traduzir todos os conceitos necessários sobre o ambiente analisado, podendo gerar ambigüidade na compreensão do mesmo.

Nesses casos é necessário lançar mão da dicionarização, ou seja, definir aspectos importantes que esclareçam os aspectos que não tenham ficado claro.

Um aspecto importante a levar em consideração é que o Dicionário de Dados não deve se restringir a definir aquilo que já é observado através do diagrama, como, por exemplo, não devemos nos limitar a definir a entidade PESSOAS como “entidade que possui os dados sobre pessoas”, pois isso já é mostrado pelo diagrama.

Regras de dicionarização:

- Nenhum modelo é suficientemente claro se não for acompanhado de uma definição formal dos elementos (Dicionário de Dados)
- Conceitos triviais para uma pessoa podem não ser triviais para outras pessoas.
- Estabelecer definições completas e inequívocas. Ser claro, preciso e específico.

Dicionarização de Entidades

Respostas que a dicionarização das entidades deve responder, conforme visto acima, de forma completa e inequívoca (além de com clareza, precisão e especificidade):

1. O que é o elemento que compõe a entidade?
2. O que faz o elemento que compõe a entidade?
3. Para que serve o elemento que compõe a entidade?
4. O que engloba essa categoria de elementos (entidade)? Que elementos podem fazer parte dela?
5. O que está excluído dessa categoria de elementos (entidade)?
6. Quando algo (ou alguém) passa a ser, ou deixa de ser, um elemento desse tipo?
7. Sua permanência nessa categoria é imutável?

Além dessas respostas, toda informação útil deve ser agregada, mas lembre-se que nem toda informação será útil.

Dados formais que devem fazer parte da dicionarização (além da definição da entidade, através das respostas às questões acima):

- **Nome da entidade**
- **Atributos dessa entidade (apenas os nomes)**
- **Relacionamentos (apenas os nomes daqueles relacionamentos com os quais tem relação direta)**

Dicionarização de Atributos

Na dicionarização de atributos não devemos cometer a falha da simplificação.

Respostas que a dicionarização dos atributos deve responder, conforme visto acima, de forma completa e inequívoca (além de com clareza, precisão e especificidade):

1. Qual a finalidade do atributo?
2. Como os elementos são enquadrados nesse atributo?
3. O que significa esse atributo?
4. Quais as diferenças entre as diversas opções desse atributo?
5. Quem define o atributo?
6. Quando é atribuído?
7. Pode mudar depois de definido?
8. Qual o papel do atributo no ambiente? E no sistema?

Como no caso anterior, se necessário, informações úteis adicionais devem ser incluídas no dicionário de atributos.

Dados formais que devem fazer parte da dicionarização (além da definição do atributo, através das respostas às questões acima):

- **Nome do atributo**
- **Nome da entidade (ou relacionamento) da qual faz parte**

Dicionarização de Relacionamentos

Respostas que a dicionarização dos relacionamentos deve responder, conforme visto acima, de forma completa e inequívoca (além de com clareza, precisão e especificidade):

1. Qual a função do relacionamento?
2. O que ele representa?
3. Quais são as regras de seu estabelecimento?
4. Quais são as exceções a seu estabelecimento?
5. Quando ele ocorre?
6. Quando ele pode deixar de existir?

Dados formais que devem fazer parte da dicionarização (além da definição do relacionamento, através das respostas às questões acima):

- **Nome do relacionamento**
- **Atributos desse relacionamento (apenas os nomes, se existirem)**
- **Cardinalidade do relacionamento (máxima e mínima)**
- **Entidades (apenas os nomes daquelas entidades com os quais tem relação direta)**

CAPÍTULO V – Estrutura de Generalização e Especialização

A **Estrutura de Generalização e Especialização** é utilizada quando, na montagem do Modelo Conceitual, identificamos sub-conjuntos distintos, dentro de conjuntos específicos. Ou seja, identificamos entidades que podem possuir “sub-categorias” com atributos diferentes para cada uma delas.

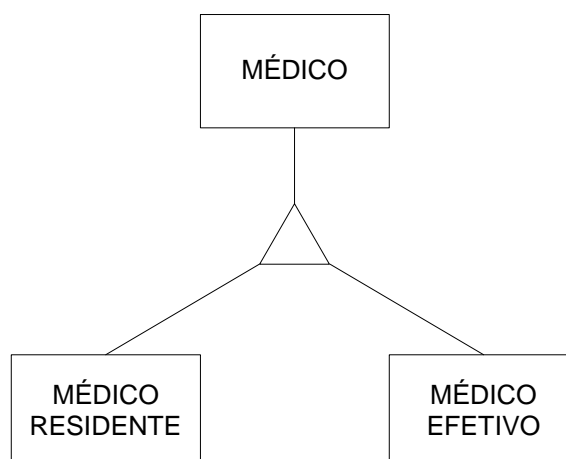
Como exemplo podemos observar um ambiente hospitalar, onde são reconhecidas as seguintes entidades:

- MÉDICOS
- PACIENTES
- QUARTOS
- SALAS DE CIRURGIA

Após uma análise mais detalhada, descobrimos que existem dois tipos de médico no ambiente hospitalar: MÉDICO RESIDENTE e MÉDICO EFETIVO, sendo que existem atributos que se aplicam somente a uma dessas categorias. Observe a tabela abaixo:

Atributo	MÉDICO RESIDENTE	MÉDICO EFETIVO
Especialidade	X	X
Nome do Médico	X	X
Local de Atuação		X
Tempo de Permanência		X
Data de Efetivação		X
Data de Início da Residência	X	
Nome do Orientador	X	
Data da Avaliação Prevista	X	
Tempo de Experiência na Função	X	X

Para representar isso no Modelo Conceitual, é possível utilizar um novo elemento gráfico que representa o grupo e os sub-grupos. Observe o diagrama abaixo:



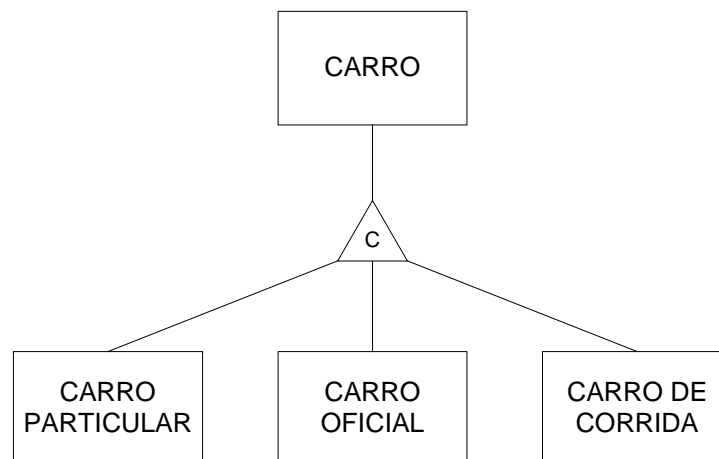
No diagrama acima, dizemos que a entidade MÉDICO é a entidade **Generalizada** (apresenta as características “gerais” da categoria) e as entidades MÉDICO RESIDENTE e MÉDICO EFETIVO são entidades **Especializadas** (apresentam as características “especiais” da categoria”).

Existem dois tipos distintos de Especialização:

- Especialização Mutuamente Exclusiva
- Especialização não Mutuamente Exclusiva

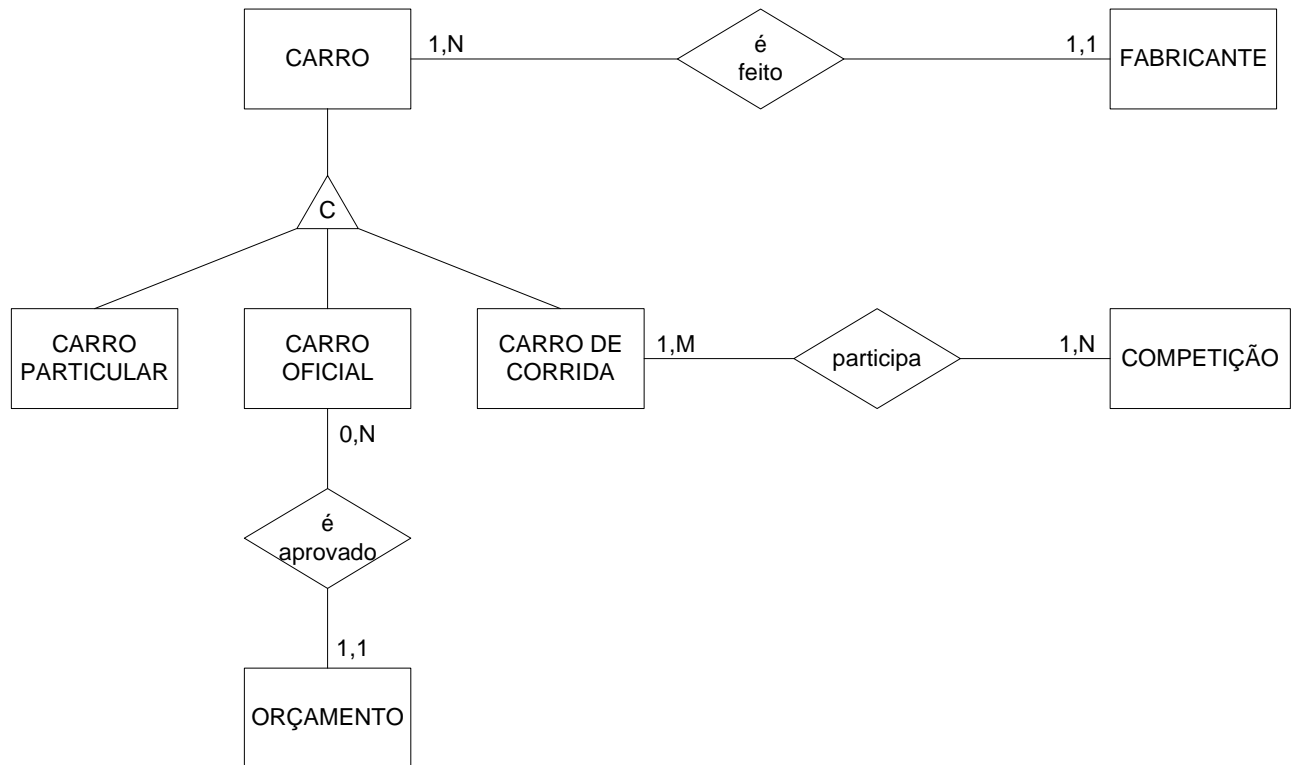
5.1 Especialização Mutuamente Exclusiva

A **Especialização Mutuamente Exclusiva**, também conhecida como **Especialização – Categoria**, acontece quando, ao identificarmos as entidades especializadas, percebemos que uma instância de uma dessas entidades nunca poderá ser instância de outra das entidades especializadas, simultaneamente. Ou seja, estamos classificando essas entidades especializadas como categorias distintas da entidade generalizada. Observe o exemplo abaixo:



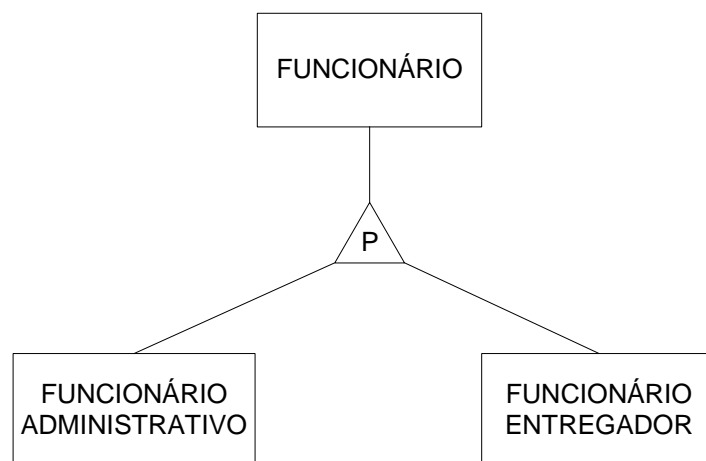
Esse diagrama indica que, no ambiente analisado, um carro pode ser Particular, Oficial ou de Corrida (nunca poderá ser classificado como sendo de duas categorias ao mesmo tempo). Isso é indicado pela letra “C” dentro do triângulo.

Quando essa representação é utilizada no Modelo Conceitual, tanto a entidade geral (CARRO), como as entidades especializadas (CARRO PARTICULAR, CARRO OFICIAL ou CARRO DE CORRIDA) podem ser usadas para criar relacionamentos com outras entidades. Observe o exemplo abaixo:



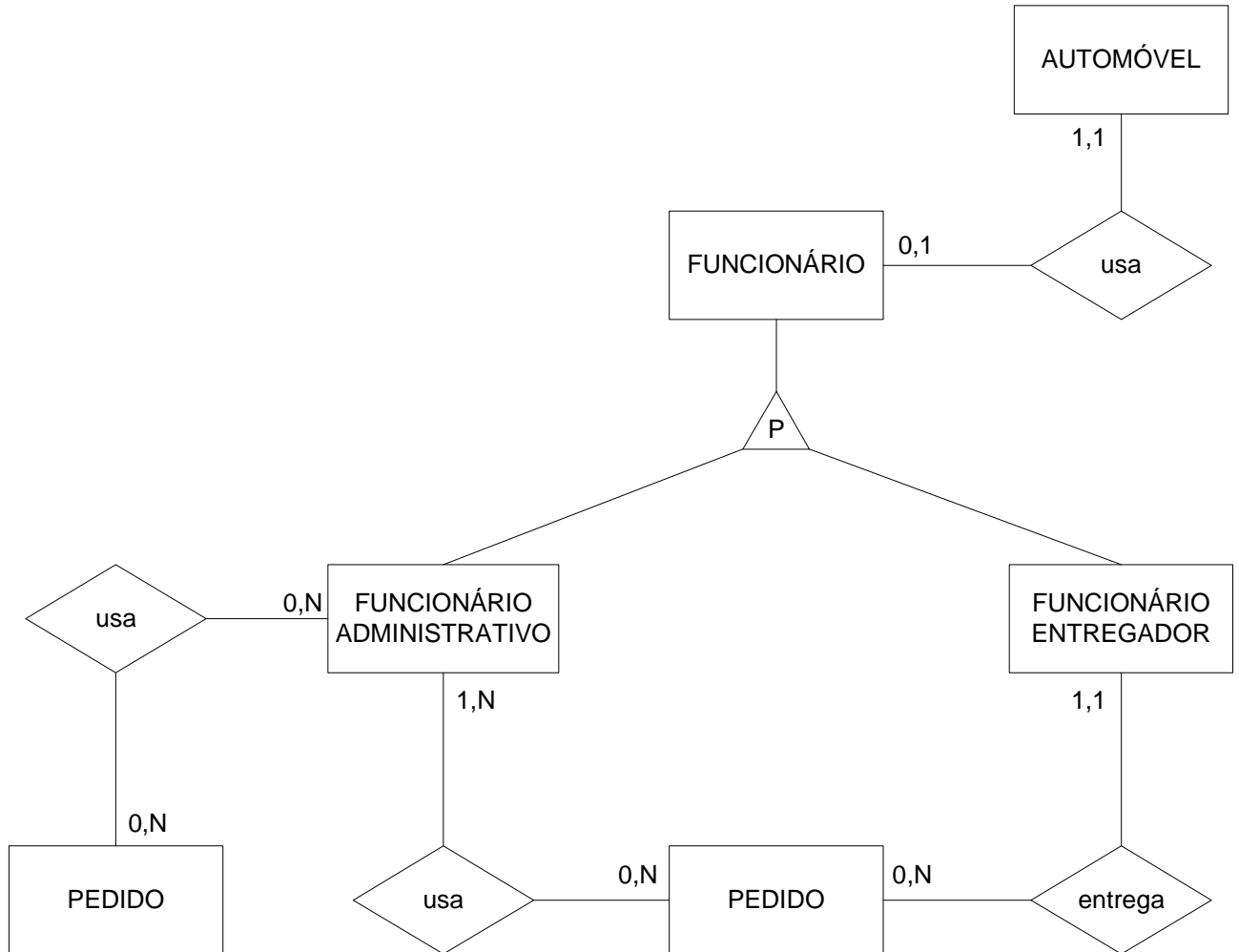
5.2 Especialização não Mutuamente Exclusiva

A **Especialização Mutuamente Exclusiva**, também conhecida como **Especialização – Papéis**, acontece quando, ao identificarmos as entidades especializadas, percebemos que uma instância de uma dessas entidades pode ser instância de outra das entidades especializadas, simultaneamente. Ou seja, as entidades especializadas representam “papéis” que o elemento pode representar em determinado momento. Observe o diagrama abaixo:



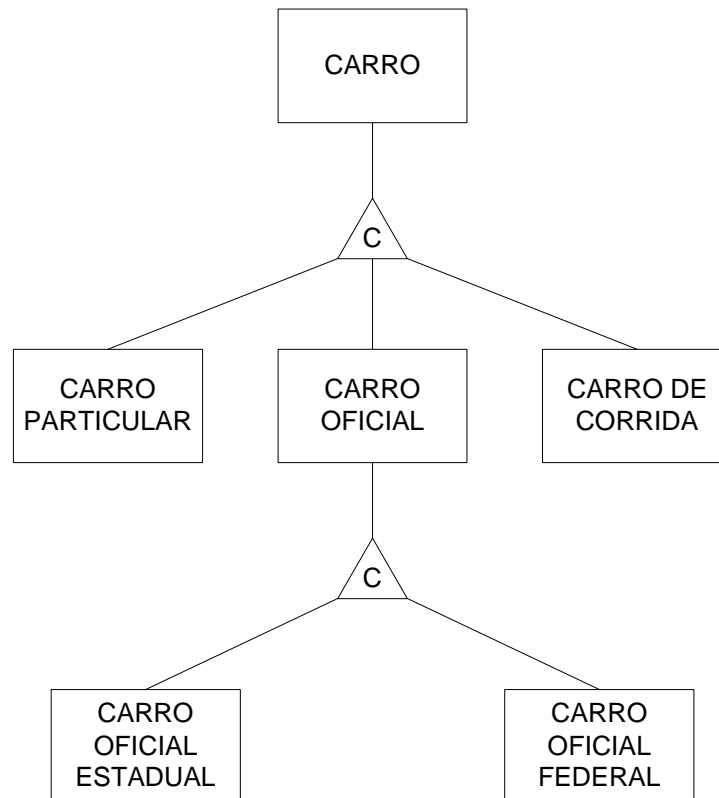
Esse diagrama indica que, no ambiente analisado, um funcionário pode ser tanto administrativo, quanto entregador (ao mesmo tempo, dependendo do “papel” que desempenha no momento). Isso é indicado pela

letra “**P**” dentro do triângulo. As regras quanto aos relacionamentos continuam as mesmas que a situação anterior, ou seja, qualquer uma das entidades pode criar relacionamentos com outras entidades. A regra para a criação de relacionamentos é a mesma do tipo de especialização anterior. Observe o exemplo abaixo:



5.3 Níveis de Especialização

É possível utilizar vários níveis de especialização para melhorar a modelagem do problema real, que está sendo analisado. Observe o diagrama abaixo:



CAPÍTULO VI – Modelo Lógico

O **Modelo Lógico** é obtido a partir de um Modelo Conceitual previamente criado. O processo de obtenção de um modelo relacional é baseado na realização de três atividades distintas:

- Normalização das estruturas de dados;
- Derivação de estruturas de Agregação e Generalização-Especialização; e
- Derivação de relacionamentos.

4.1. Normalização

Normalização é um formalismo que tem por objetivo organizar o Modelo Conceitual, facilitando a sua derivação para um Modelo Lógico. O processo analisa as entidades e seus atributos visando evitar anomalias de armazenamento que possam ocorrer.

Ele possui **Formas Normais** (nas quais as entidades se enquadram), sendo que os principais aplicativos utilizam até a Terceira Forma Normal. Isso significa que o modelo se enquadra na Primeira, na Segunda e na Terceira Formas Normais.

4.1.1. Primeira Forma Normal (1FN)

Uma entidade se encontra na **Primeira Forma Normal (1FN)** quando o relacionamento entre o seu atributo chave e seus outros atributos for unívoco, ou seja, para cada chave há a ocorrência de um, e somente um, dado de cada atributo.

Para exemplificar a normalização utilizaremos a entidade **PEDIDO**, composta pelos atributos apresentados abaixo:

PEDIDO

Número_Pedido (PK)	Código_Cliente	Nome_Cliente	Código_Produto	Quantidade_Produto	Descrição_Produto
-----------------------	----------------	--------------	----------------	--------------------	-------------------

Etapas para a normalização (1FN)

1 – Identifique os atributos (ou grupo de atributos) que possuam múltiplos valores para uma ocorrência da entidade.

PEDIDO

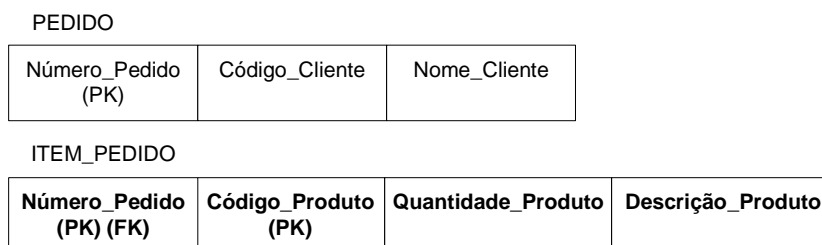
Número_Pedido (PK)	Código_Cliente	Nome_Cliente	Código_Produto	Quantidade_Produto	Descrição_Produto
-----------------------	----------------	--------------	-----------------------	---------------------------	--------------------------

Os atributos **Código_Produto**, **Quantidade_Produto** e **Descrição_Produto** possuem mais de um valor para cada ocorrência da entidade (para cada **Numero_Pedido**), ou seja, um pedido pode ser composto por vários produtos.

2 – Remova os atributos (ou grupo de atributos) com uma cópia da chave primária.

Deve ser criada uma nova entidade que terá uma chave composta (com dois atributos): a cópia da chave da entidade original (que será chave primária e estrangeira ao mesmo tempo) e um dos atributos removidos (que deve ser adequado para essa função).

Obs: Caso nenhum dos atributos removidos seja adequado para a função de compor uma chave composta, deve ser criado um atributo para cumprir essa função.

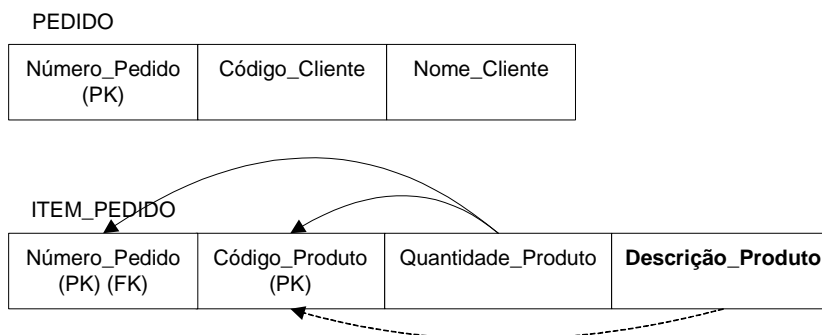


4.1.2. Segunda Forma Normal (2FN)

Uma entidade se encontra na **Segunda Forma Normal (2FN)** quando, já submetida a 1FN, apresente uma chave composta que se relaciona de forma integral com todos os seus atributos, ou seja, todos os atributos dependem da chave composta completa.

Etapas para a normalização (2FN)

3 – Identifique, nas entidades que possuam chave composta, atributos dependentes somente de parte da chave primária composta.



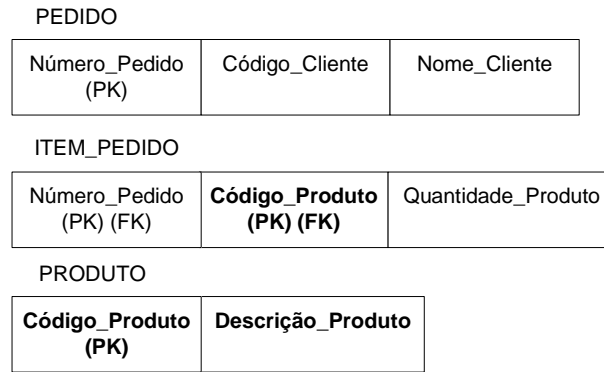
O atributo **Quantidade_Produto** depende dos dois atributos chave, ou seja, essa quantidade é de um produto específico (**Código_Produto**) e foi vendida em um determinado pedido (**Número_Pedido**).

O atributo **Descrição_Produto** depende somente do atributo **Código_Produto**, ou seja, a descrição do produto não muda de acordo com o pedido realizado.

4 – Remova os atributos encontrados com uma cópia da parte da chave primária composta (atributo chave ao qual ele apresenta dependência)

Deve ser criada uma nova entidade composta pela parte da chave primária e pelos atributos removidos.

Obs: A parte da chave composta, do atributo original, se torna também chave estrangeira.



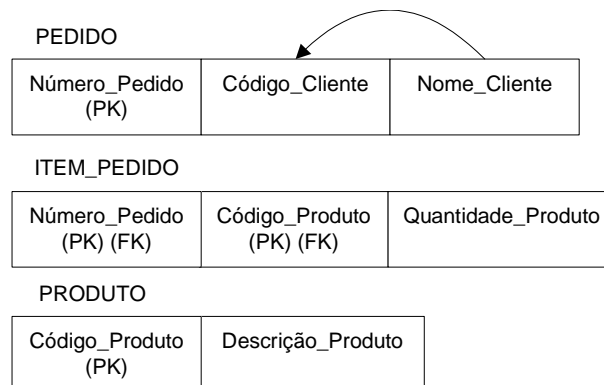
4.1.3. Terceira Forma Normal (3FN)

Uma entidade se encontra na Terceira Forma Normal (3FN) quando não houver atributos associados que tenham Dependência Transitiva com a chave, ou seja, não existam elementos intermediários de ligação entre os dois objetos.

Dependência Transitiva ocorre quando um determinado atributo apresenta dependência de um atributo que não seja chave.

Etapas para a normalização (3FN)

5 – Identifique atributo(s) dependente(s) de outro(s) atributo(s) que não sejam chave.

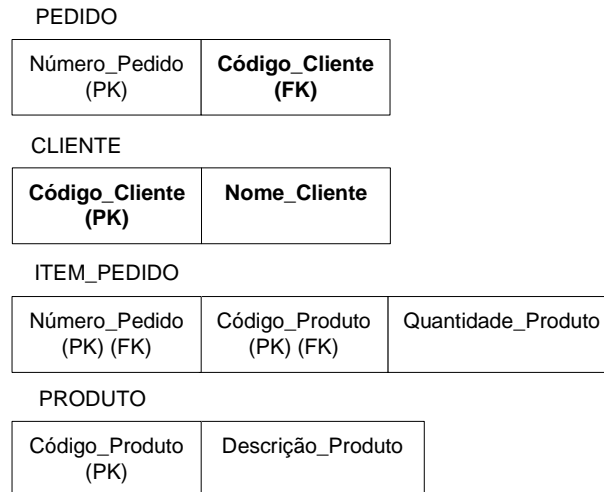


O atributo **Nome_Cliente** é dependente do atributo **Código_Cliente** e não da chave **Número_Pedido**, ou seja, o nome do cliente não se altera em função do pedido no qual ele faz parte.

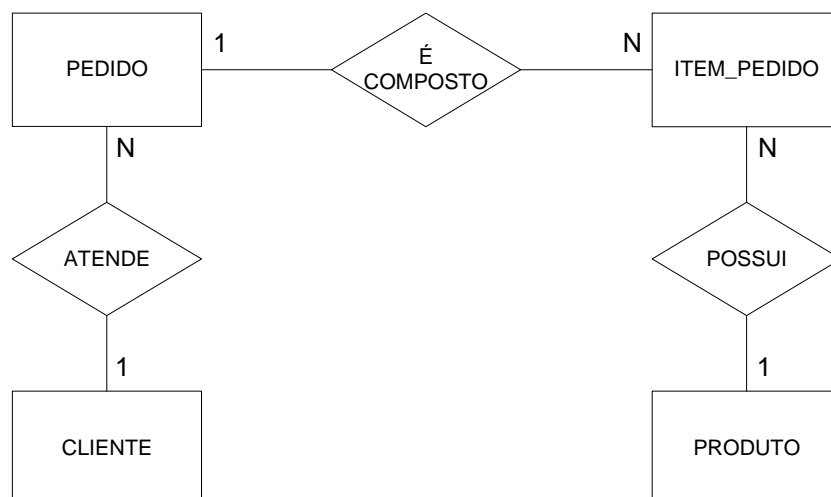
6 – Remova esses atributos com uma cópia do atributo do qual depende.

Deve ser criada uma nova entidade com os atributos dependentes e com uma cópia do atributo do qual dependem (ele é o atributo chave dessa nova entidade).

Obs: Na entidade original, o atributo que foi copiado se torna chave estrangeira.



Após a normalização, o modelo conceitual se altera para refletir as novas entidades que surgiram. Os relacionamentos com a entidade inicial (que foi normalizada) continuam relacionados a ela e não às entidades que foram geradas no processo de normalização.



4.2. Notação “Pé-de-Galinha” (Crow's Foot)

Para apresentar o Modelo Lógico, que não representa formalmente os relacionamentos constantes do Modelo Conceitual, lançamos mão de uma notação um pouco diferente da que vimos até o momento, é a **Notação “Pé-de-Galinha” (Crow's Foot**, no original em inglês).

Essa notação busca apresentar as cardinalidades dos relacionamentos entre as entidades, sem utilizar da notação numérica.

Se considerarmos somente a cardinalidade máxima, a representação acontece da seguinte maneira:

	Modelo de Tabelas	Modelo Lógico – Entidades
Relacionamento 1:1		
Relacionamento 1:N		
Relacionamento M:N		

Já as cardinalidades mínimas são representadas da seguinte maneira:

Cardinalidade	Notação “Pé-de-Galinha”
0,1	ENTIDADE
0,N	ENTIDADE
1,1	ENTIDADE
1,N	ENTIDADE

4.3. Derivação de Estruturas de Generalização-Especialização

A **Derivação de Estruturas de Generalização-Especialização** resultará diretamente em um Modelo Lógico para implementação. No processo de derivação do Modelo Conceitual para o Modelo Lógico, as estruturas de generalização-especialização devem ser eliminadas da representação (apesar de continuarem presentes conceitualmente).

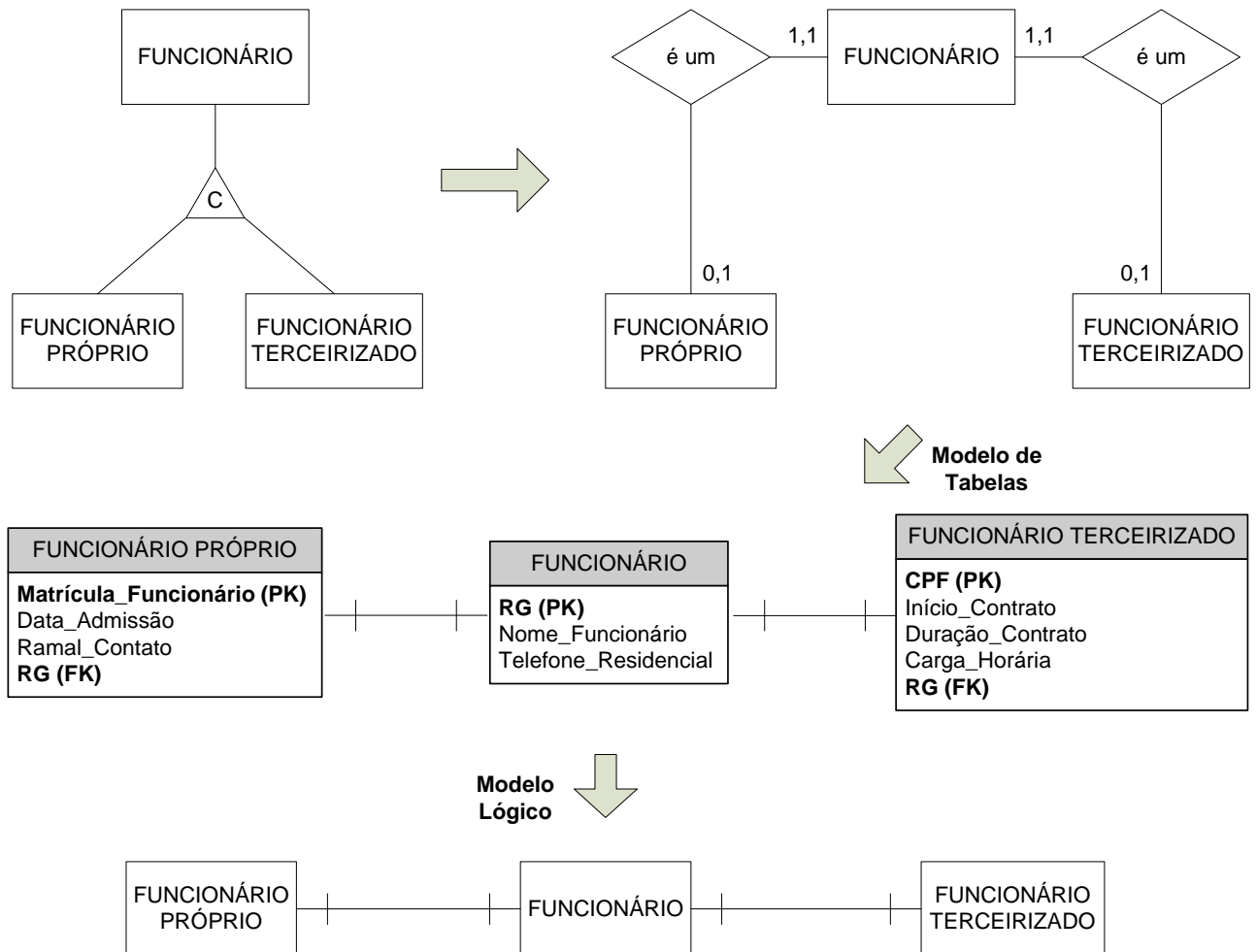
Durante o processo de derivação podem ser adotadas quatro estratégias diferentes, que são apresentadas a seguir.

4.3.1. Estratégia 1

Criar uma tabela para a entidade generalizada e uma tabela para cada entidade especializada, mantendo os relacionamentos associados a cada uma delas.

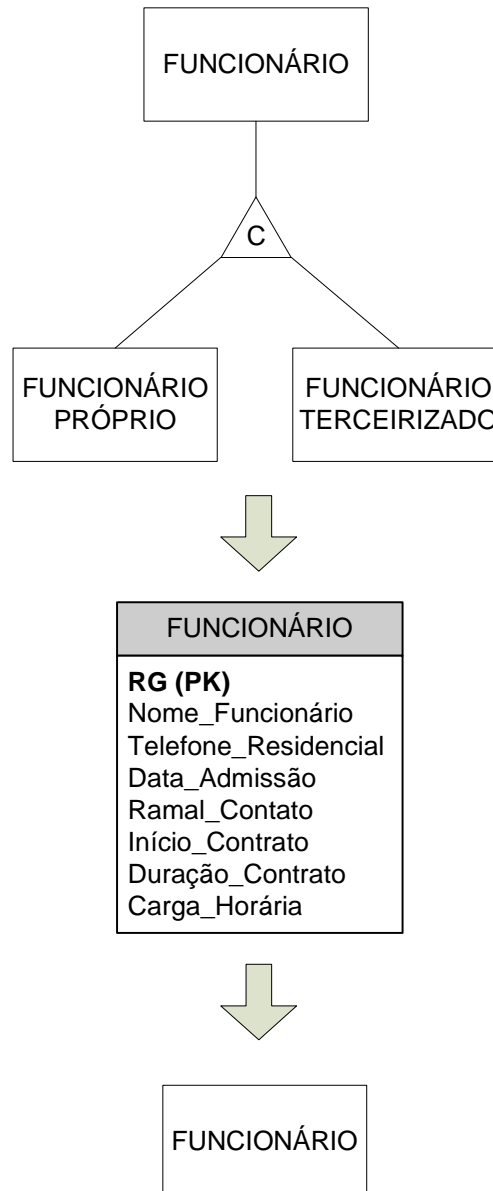
As tabelas das entidades especializadas recebem o atributo Chave Primária da entidade generalizada como Chave Estrangeira.

Observe o diagrama abaixo, onde são apresentados: o Modelo Conceitual do relacionamento e o Modelo Lógico resultante (apresentando o Modelo de Tabelas e o Modelo Lógico propriamente dito).



4.3.2. Estratégia 2

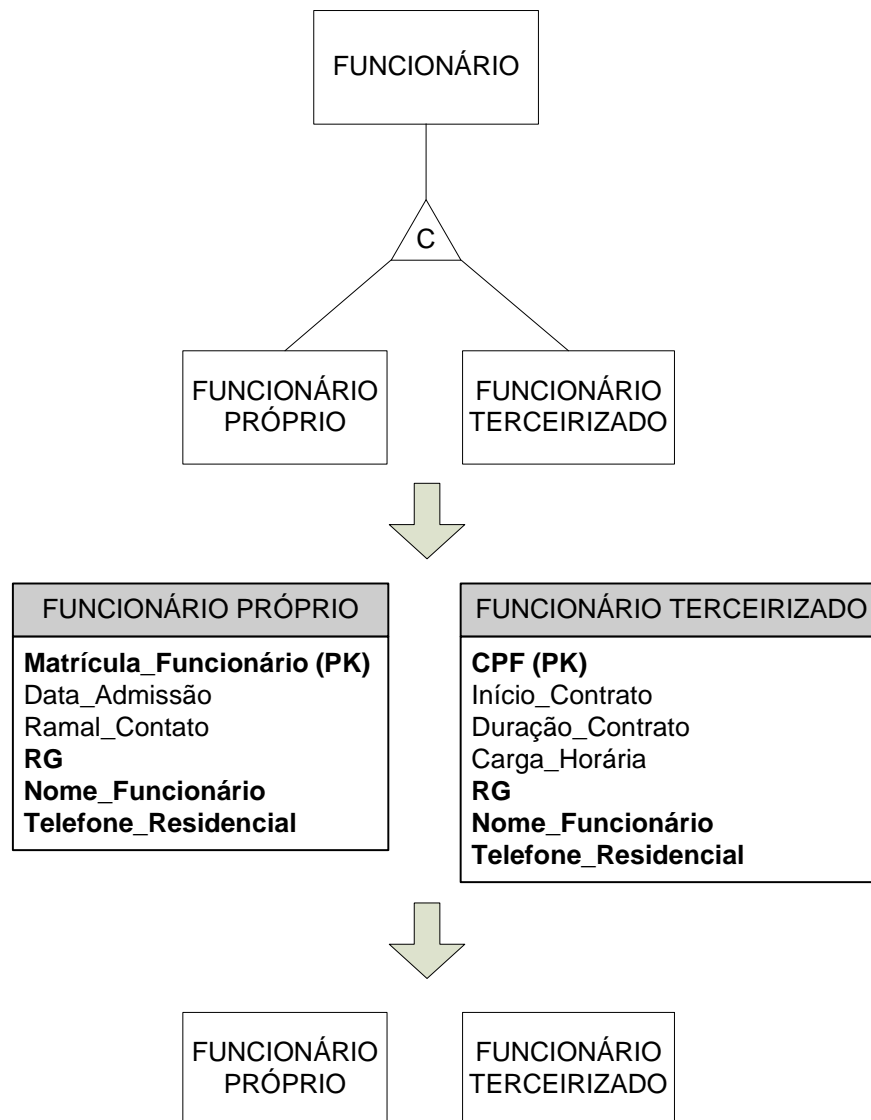
Criar somente uma tabela para a entidade generalizada e agregar os atributos e relacionamentos especializados a essa entidade. Observe o diagrama abaixo:



4.3.3. Estratégia 3

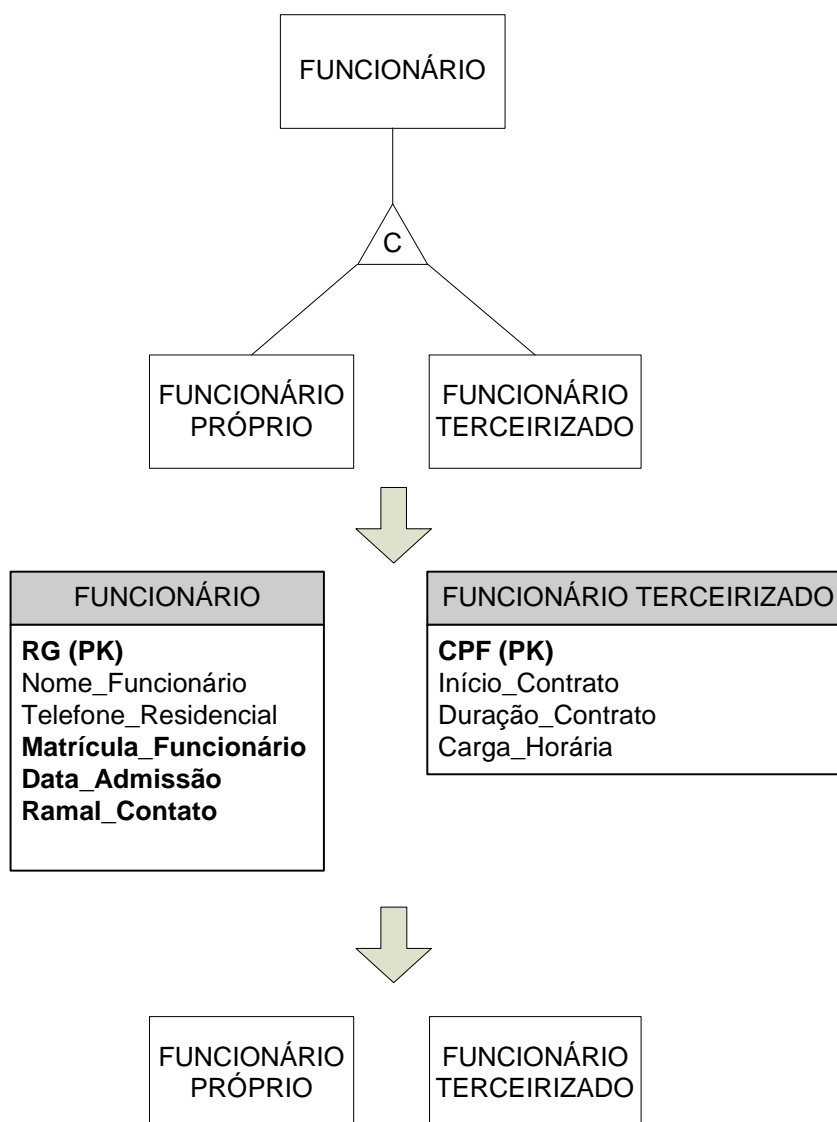
Criar somente tabelas para as entidades especializadas, agregando os atributos e relacionamentos generalizados a essas entidades. Não ocorrerá relacionamento entre as tabelas resultantes, elas serão independentes entre si, mas terão de refletir os relacionamentos existentes anteriormente com a entidade generalizada que foi eliminada.

Isso pode exigir que um relacionamento com a entidade generalizada se transforme em vários relacionamentos, um com cada entidade, antes especializada. Observe o diagrama abaixo:



4.3.4. Estratégia 4

Migrar atributos e relacionamentos de algumas das entidades especializadas (não todas) para a entidade generalizada ou vice-versa. Novamente não ocorrerá relacionamento entre as tabelas resultantes, elas serão independentes entre si, apresentando o mesmo problema de relacionamento descrito anteriormente na **Estratégia 3**. Observe o diagrama abaixo:



4.4. Derivação de Relacionamentos

A **Derivação de Relacionamentos** resultará diretamente em um Modelo Lógico para implementação. No processo de derivação do Modelo Conceitual para o Modelo Lógico, os relacionamentos devem ser eliminados da representação (apesar de continuarem presentes conceitualmente).

Para isso é necessário observar a cardinalidade máxima do relacionamento, e de acordo com ela, realizar uma operação de derivação. Esse processo levará em conta relacionamentos 1:1, 1:N e M:N, com ou sem atributos no relacionamento.

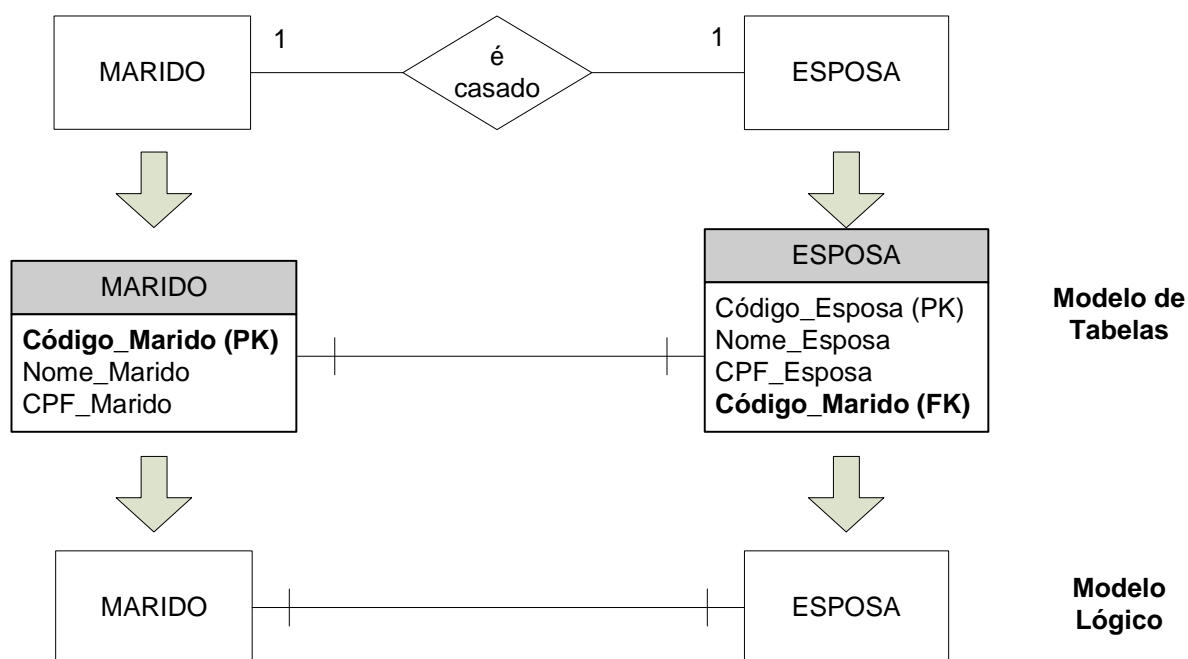
4.4.1. Relacionamentos 1:1 sem atributos

Para derivar esse tipo de relacionamento, existem duas possibilidades.

Processo 1

Para derivar esse tipo de relacionamento acrescentar a Chave Primária (PK) de uma das tabelas como Chave Estrangeira (FK) da outra.

O critério para escolha de quem deve fornecer a Chave Estrangeira é simples. A entidade que tiver os dados alimentados primeiro deve fornecer a Chave Estrangeira.



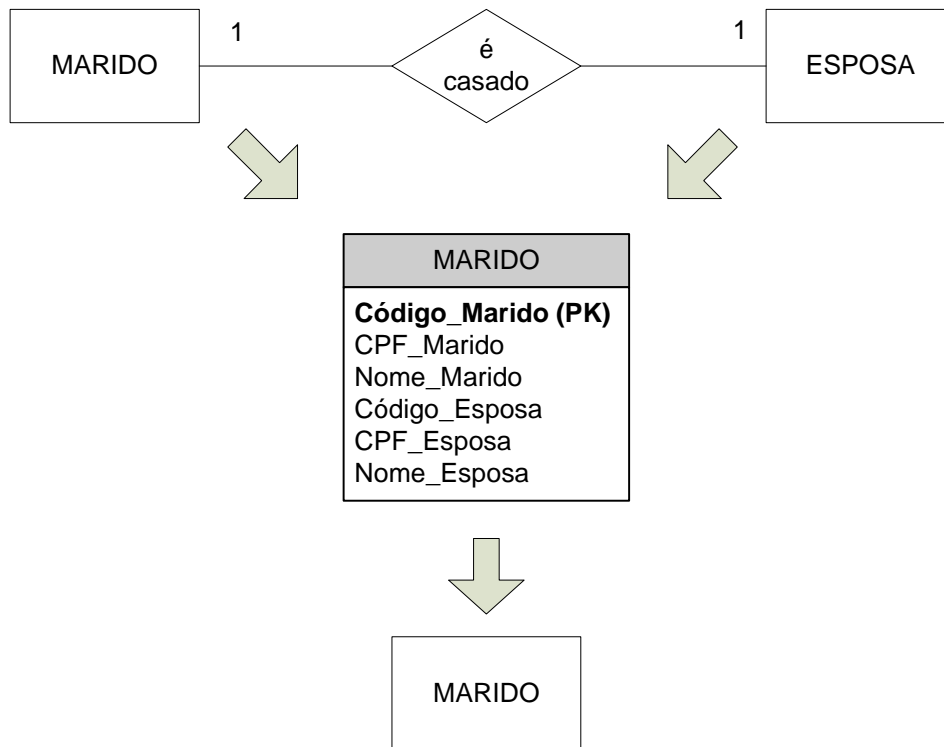
Processo 2

Migrar todos os atributos de uma das tabelas para a outra, sendo que somente a Chave Primária da tabela que recebeu os atributos é mantida. O critério para escolher qual tabela deve receber os atributos é o mesmo do Processo 1.

Alguns cuidados devem ser tomados quando se escolhe esse processo de derivação:

- A tabela que irá desaparecer deve ser completamente dependente da outra; e
- Os relacionamentos passarão a valer para as duas tabelas unidas.

Observe o exemplo abaixo:

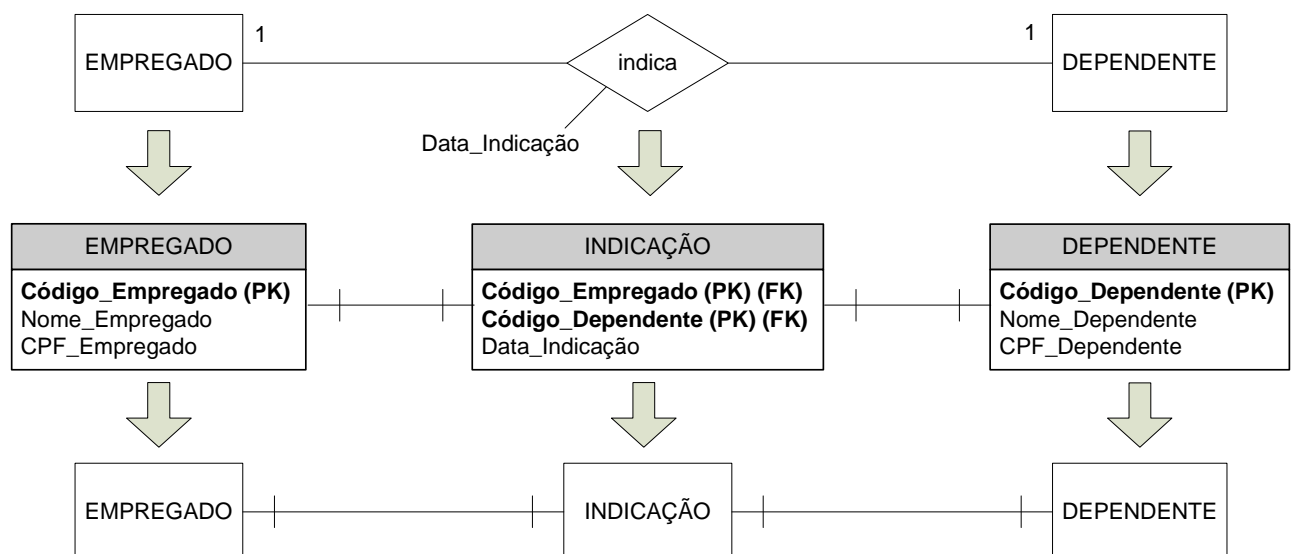


4.4.2. Relacionamentos 1:1 com atributos

Para derivar esse tipo de relacionamento, existem duas possibilidades.

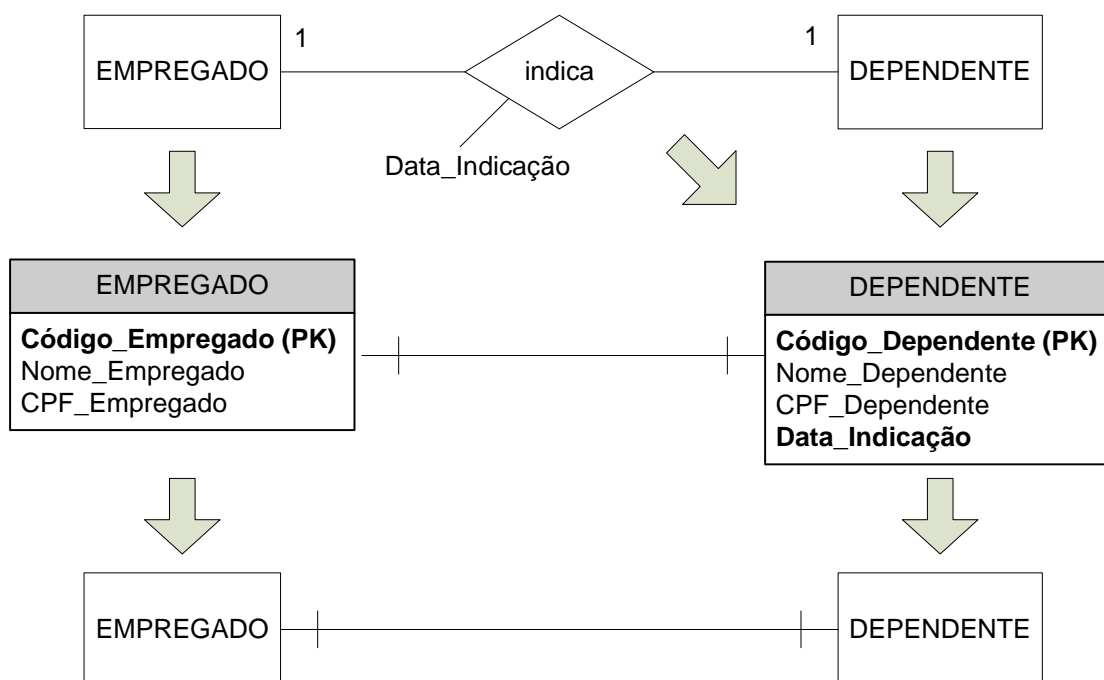
Processo 1

Criar uma terceira tabela que contenha as chaves das duas outras tabelas, sendo que essas chaves serão Chaves Primárias (PK) e Chaves Estrangeiras (FK) simultaneamente. Os atributos do relacionamento são relacionados normalmente nesta terceira tabela. Observe o exemplo abaixo:



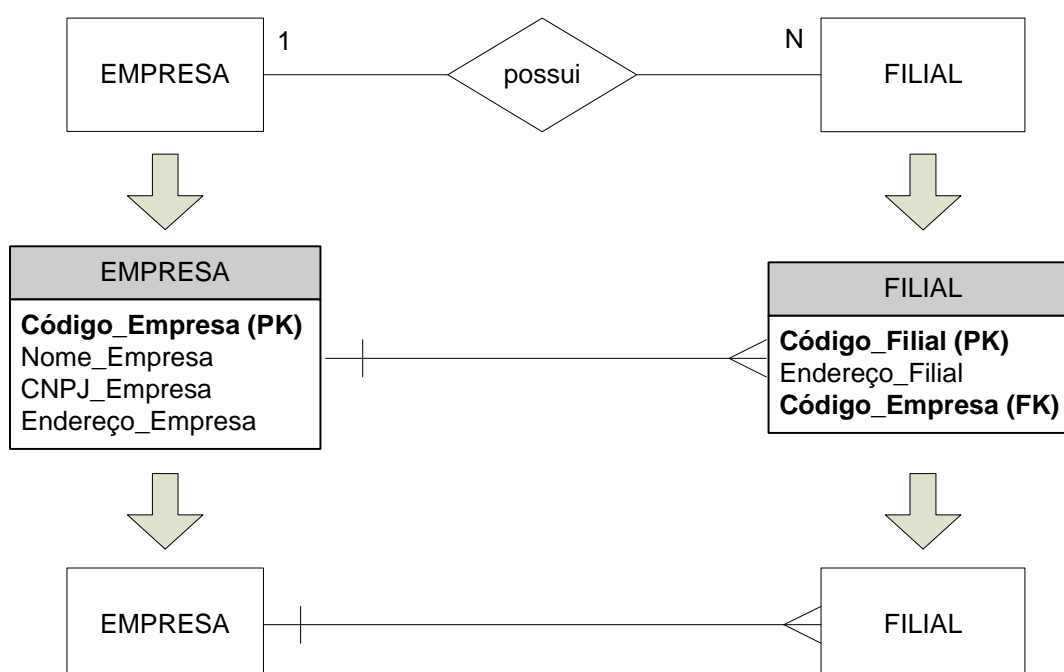
Processo 2

Migrar os atributos do relacionamento para uma das tabelas. A escolha da tabela que deve receber esses atributos depende do domínio do problema (problema que está sendo modelado). Observe o exemplo abaixo:



4.4.3. Relacionamentos 1:N sem atributos

Para realizar essa derivação basta acrescentar a Chave Primária da tabela com cardinalidade 1 na tabela com cardinalidade N, como Chave Estrangeira. Observe o exemplo abaixo:

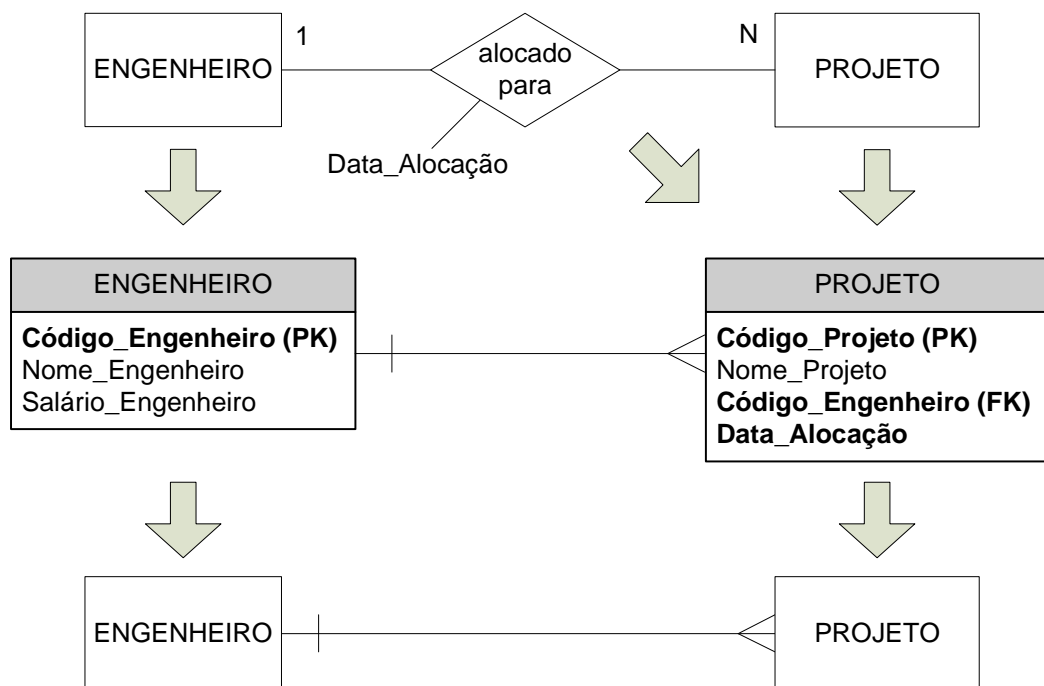


4.4.4. Relacionamentos 1:N com atributos

Para derivar esse tipo de relacionamento, existem duas possibilidades.

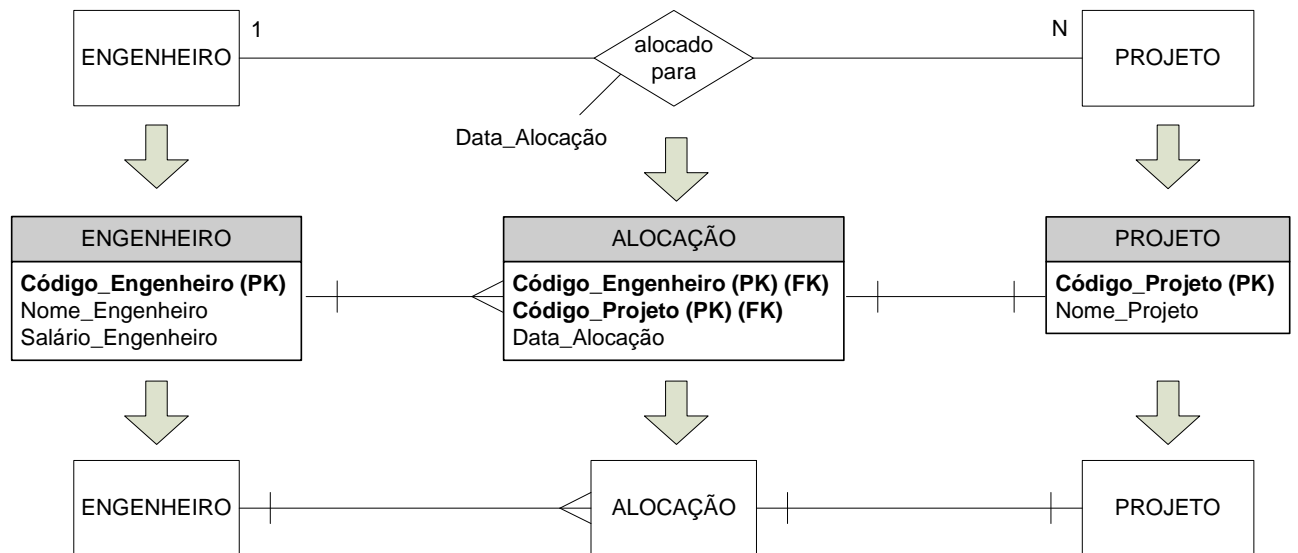
Processo 1

Em primeiro lugar, seguir o processo para relacionamentos 1:N sem atributos, apresentado no item anterior, e então migrar os atributos do relacionamento para a tabela com cardinalidade N. Observe o exemplo abaixo:



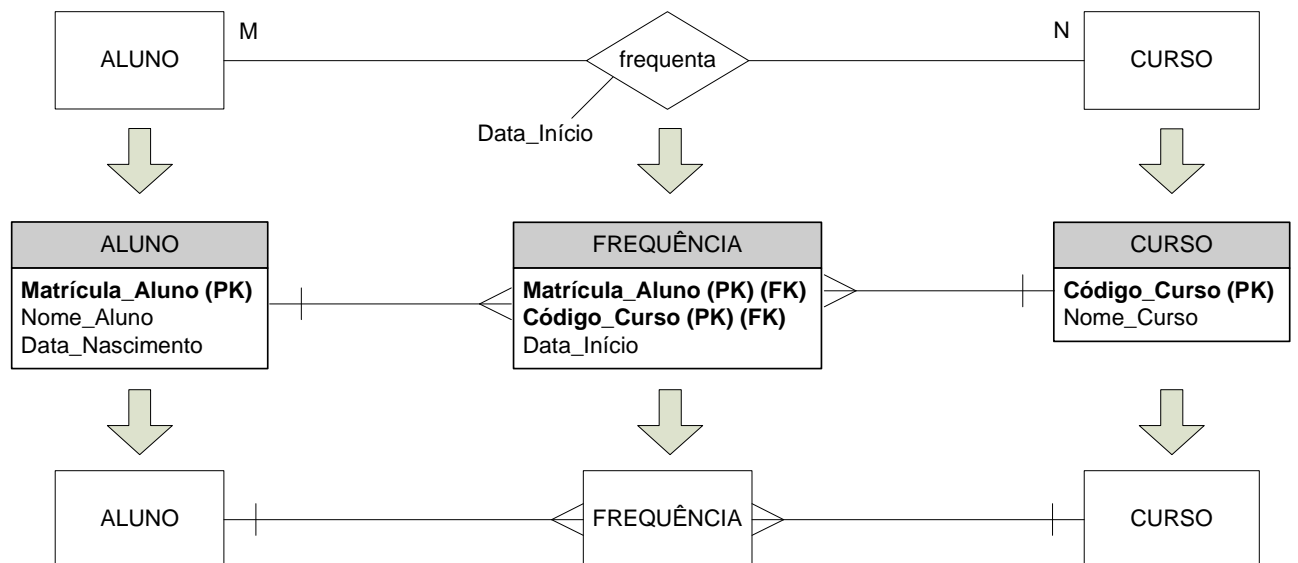
Processo 2

Criar uma terceira tabela que contenha as chaves das duas outras tabelas, sendo que essas chaves serão Chaves Primárias (PK) e Chaves Estrangeiras (FK) simultaneamente. Os atributos do relacionamento são relacionados normalmente nesta terceira tabela. Observe o exemplo abaixo:



4.4.5. Relacionamentos M:N

Para derivar esse tipo de relacionamento criar uma terceira tabela que contenha as chaves das duas outras tabelas, sendo que essas chaves serão Chaves Primárias (PK) e Chaves Estrangeiras (FK) simultaneamente. Se existirem atributos do relacionamento, eles são relacionados normalmente nesta terceira tabela. Observe o exemplo abaixo:



CAPÍTULO VII – Criação do Banco de Dados utilizando o *Enterprise Manager* do *Microsoft SQL Server 2000*

Na prática, um Banco de Dados pode ser entendido como um conjunto de tabelas interligadas. Sempre tendo em mente que estamos trabalhando com o modelo relacional de bancos de dados.

Para criar esse banco de dados é necessário utilizar uma ferramenta conhecida como **Sistema Gerenciador de Banco de Dados (SGBD)**. Atualmente, existem vários fornecedores, oferecendo diferentes SGBDs, como por exemplo:

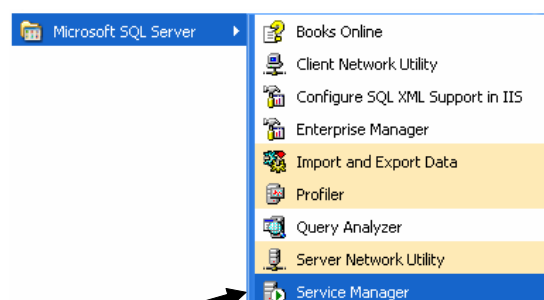
Fornecedor	Site	Produto	Detalhes
Microsoft	www.microsoft.com.br	SQL Server 2008	
		SQL Server 2008 Express	Gratuito
		MSAccess	
Oracle	www.oracle.com	Oracle Database 11g Enterprise Edition	
		Oracle 10g Release 2 Express Edition	Gratuito
IBM	www.ibm.com	IBM DB2 UDB Express	Gratuito
		IBM DB2 Enterprise	
Firebird	www.firebirdsql.org	Firebird	Gratuito
MySQL	www.mysql.com	MySQL 5.0	Gratuito
Postgre	www.postgre.com	PostgreSql	Gratuito
Caché	www.intersystems.com.br	Caché 5.0	

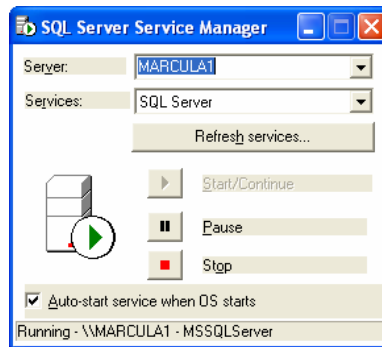
4.1. Microsoft SQL Server 2000

Inicialmente devemos verificar se o serviço está ativado (servidor de banco de dados ativado). Para isso, devemos observar o **Service Manager**. Existem duas formas de acessar esse recurso.

A primeira forma é utilizar o menu Iniciar, seguindo a sequência: *Iniciar* → *Programas* → *Microsoft SQL Server* → *Service Manager*.

Quando o serviço estiver ativado, no ícone do *Service Manager*, aparecerá uma seta verde. Nesse caso, nem é necessário clicar na opção. Caso a seta verde não apareça, clicar na opção e surgirá a janela *SQL Server Service Manager*.





Então clicar na opção *Start/Continue* na janela do *Service Manager*.

A outra opção para verificar a ativação e acessar a mesma janela acima é observar a “bandeja” (*System Tray*) no canto inferior direito da tela. O ícone do *Service Manager* estará presente, onde podemos observar o ícone, ou até mesmo, dar duplo clique para acessar a janela acima.



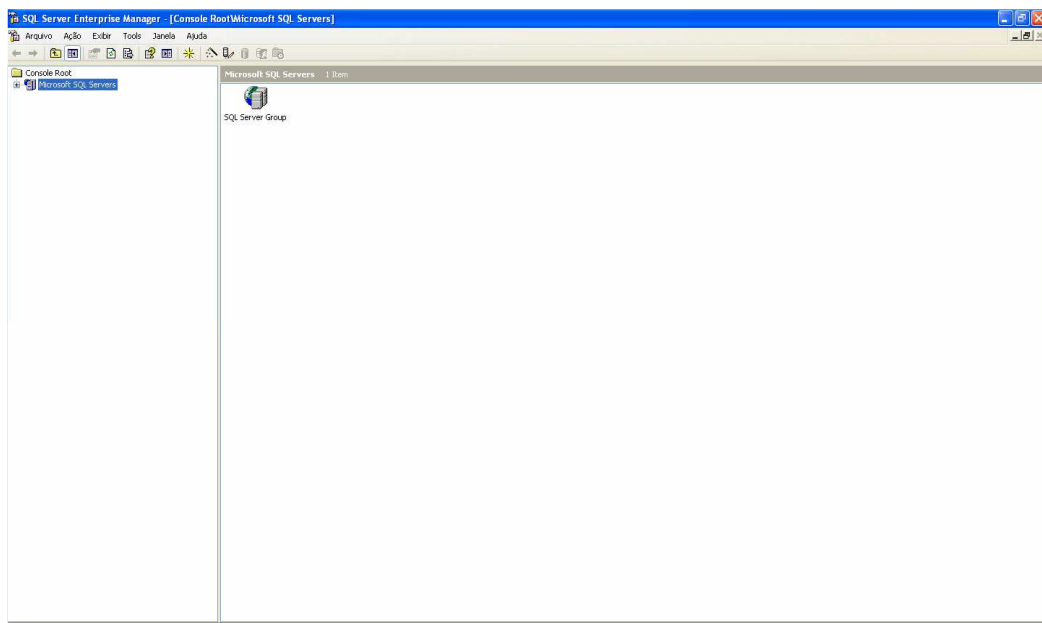
Os outros dois componentes front-end importantes do SQL Server 2000 são:

- *Enterprise Manager* – utilizado para criar e gerenciar os objetos do banco de dados; e
- *Query Analyzer* – utilizado para criar e executar instruções SQL (*Transact-SQL*). Este será assunto futuro de nossas aulas.

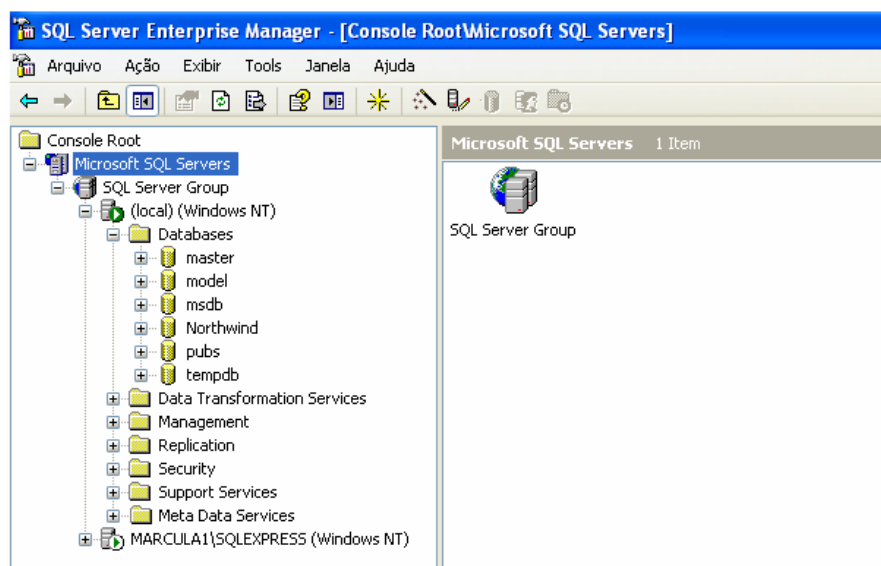
4.2. Enterprise Manager

Utilizaremos o *Enterprise Manager* para criar o banco de dados (*database*) e as tabelas que compõem o banco de dados (*tables*).

Para acessar o *Enterprise Manager* seguir a sequência: *Iniciar* → *Programas* → *Microsoft SQL Server* → *Enterprise Manager*. Surgirá a seguinte janela:



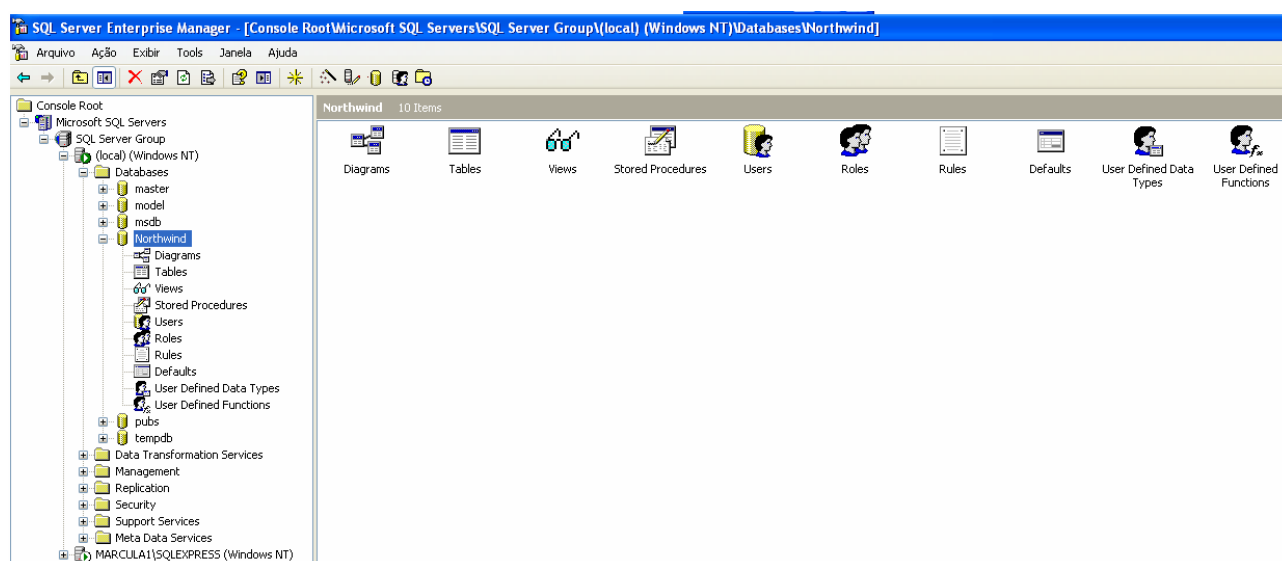
Se formos clicando nos sinais [+] ao lado de *Microsoft SQL Servers* e nas opções que forem aparecendo, até a opção *Databases*, encontraremos o seguinte:



Sob *Databases* temos listados os bancos de dados padrão do *SQL Server 2000* (aqueles que são instalados junto com o programa).

Obs: Quando realizar essa operação no laboratório, pode ser que sejam encontrados outros bancos de dados na lista.


Se clicarmos em *Northwind* (banco de dados de exemplo), obteremos o seguinte:

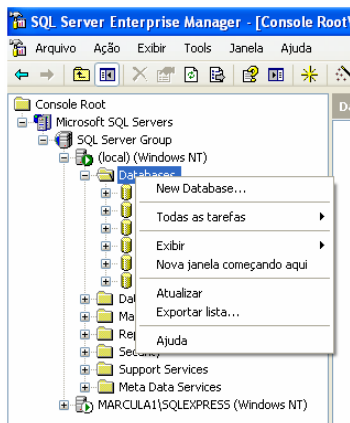


Temos aí todos os objetos pertencentes ao banco de dados *Northwind*. Esses objetos aparecerão para todos os bancos de dados que forem criados. Os mais importantes são:

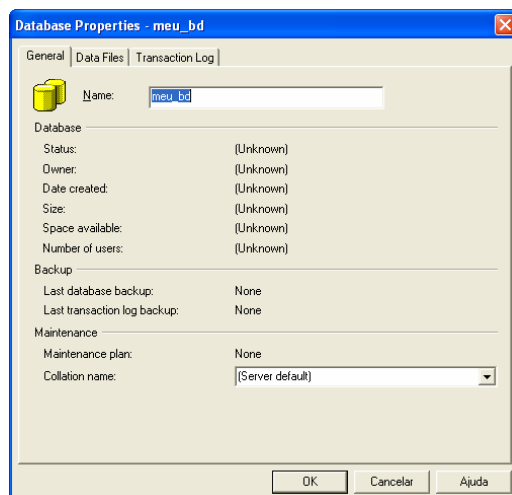
- *Diagrams* – utilizado para criar diagramas de relacionamento;
- *Tables* – utilizado para criar, alterar, excluir e incluir dados em uma tabela;
- *Views* – utilizado para criar visões (tabelas com consultas de dados).

4.3. Criando um novo banco de dados (Database)

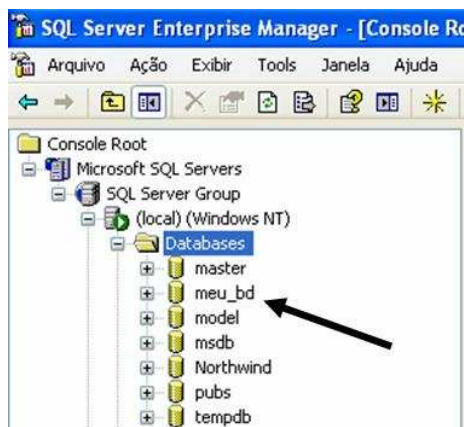
Para criar um novo banco de dados você pode clicar no botão *New Database*  da barra de ferramentas ou clicar com o botão direito do mouse sobre *Databases* e o seguinte menu surgirá:




Clicar na opção *New Database*, quando surgirá a janela *Database Properties* com um campo para você digitar o nome do novo banco de dados. No exemplo, o nome dado foi *meu_bd*.

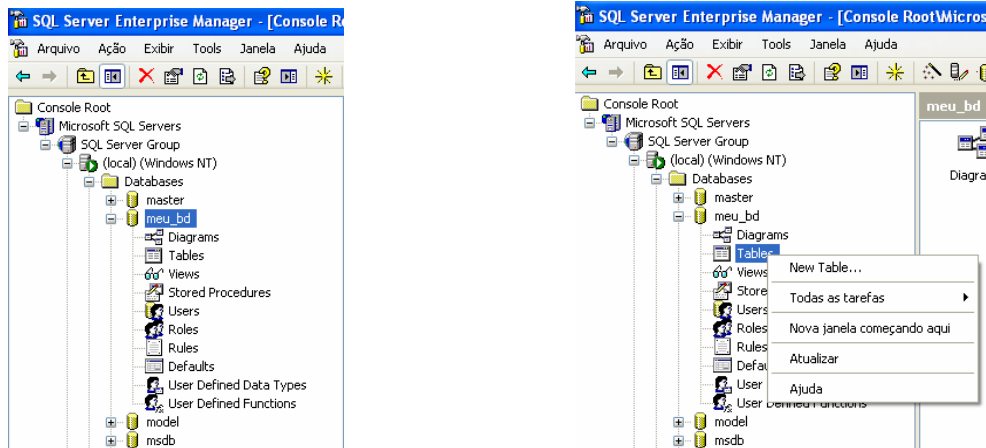


Confirme clicando em *OK* e o novo banco de dados será criado.

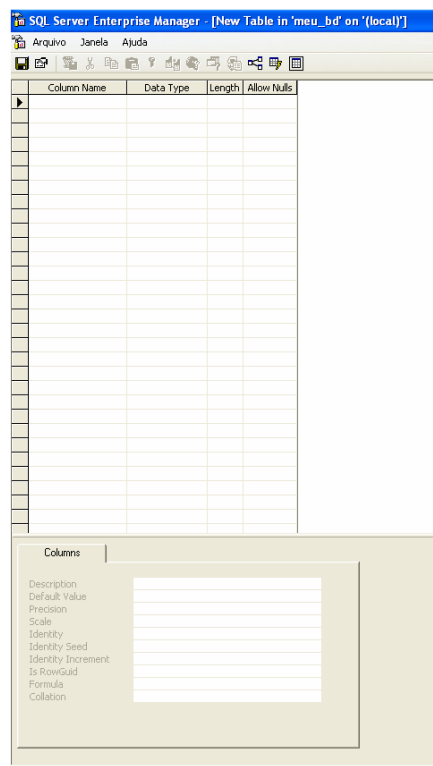


4.4. Criando uma nova tabela

Criado o banco de dados *meu_bd*, precisamos criar as tabelas farão parte desse banco de dados. Para isso, abra as suas opções de objetos de *meu_bd* (botão direito do mouse) e clique sobre *Tables*. Você pode clicar diretamente no botão *New*  da barra de ferramentas ou clicar com o botão direito do mouse sobre *Tables* e surgirá o menu da figura abaixo à direita.



Clique na opção *New Table* e surgirá a janela para criação da tabela (determinação das colunas que farão parte dessa tabela).



Antes de criarmos a tabela é importante conhecermos alguns detalhes importantes sobre esse processo. Durante a criação de uma tabela, serão determinados os nomes das colunas que compõem essa tabela.

Além dos nomes, para cada coluna, deve ser determinado o tipo de dados que será armazenado nela. As tabelas abaixo apresentam os tipos de dados (*Data Type*) mais comumente usados no *SQL Server 2000*.

Tipo de dados: Texto (<i>String</i>)	
Tipo de coluna	Descrição
char(n)	String de caracteres com tamanho fixo (determinado por n). Máximo de 8.000 caracteres. Tamanho: n bytes.
varchar(n)	String de caracteres com tamanho variável. Máximo de 8.000 caracteres
varchar(max)	String de caracteres com tamanho variável. Máximo de 1.073.741.824 caracteres
text	String de caracteres de tamanho variável. Máximo de 2GB de dados de texto
nchar(n)	String de caracteres (Unicode) com tamanho fixo (determinado por n). Máximo de 4.000 caracteres. Tamanho: n bytes.
nvarchar(n)	String de caracteres (Unicode) com tamanho variável. Máximo de 4.000 caracteres
nvarchar(max)	String de caracteres (Unicode) com tamanho variável. Máximo de 536.870.912 caracteres
ntext	String de caracteres (Unicode) de tamanho variável. Máximo de 2GB de dados de texto
Tipo de dados: Binário	
Tipo de coluna	Descrição
bit	Permite somente 0, 1 ou NULL
binary	Dados binários com tamanho fixo. Máximo de 8.000 Bytes.
Tipo de dados: Numéricos	
Tipo de coluna	Descrição
tinyint	Permite números inteiros de 0 a 255. Tamanho: 1 Byte.
smallint	Permite números inteiros de -32.768 a 32.768. Tamanho: 2 Bytes.
int	Permite números inteiros entre -2.147.483.648 e 2.147.483.647. Tamanho: 4 Bytes.
bigint	Permite números inteiros entre -9.223.372.036.854.775.808 e 9.223.372.036.854.775.807. Tamanho: 8 Bytes.
decimal(p,s)	Números de precisão fixa e não inteiros. Permite números de $-10^{38} + 1$ até $10^{38} - 1$. O parâmetro <i>p</i> indica a quantidade total de dígitos que podem ser armazenados (contando a parte inteira e a parte decimal do número). O parâmetro <i>p</i> deve ser um valor entre 1 e 38 (padrão é 18). O parâmetro <i>s</i> indica a quantidade de dígitos armazenados na parte decimal do número (à direita do ponto decimal – “casas decimais”) e representa a precisão fixa do número. O parâmetro <i>s</i> deve ser um valor entre 0 e <i>p</i> (padrão é 0). Tamanho: 5 – 17 Bytes.
smallmoney	Dados monetários de -214.748,3648 até 214.748,3647. Tamanho: 4 Bytes.
money	Dados monetários de -922.337.203.685.477,5808 até 922.337.203.685.477,5807. Tamanho: 8 Bytes.
real	Dados numéricos com precisão flutuante de $-3,40 \times 10^{+38}$ até $3,40 \times 10^{+38}$. Tamanho: 4 Bytes.
float(n)	Dados numéricos com precisão flutuante de $-1,79 \times 10^{+308}$ até $1,79 \times 10^{+308}$. O parâmetro <i>n</i> indica se o campo deve manter 4 ou 8 Bytes. A declaração <i>float(24)</i> mantém um campo de 4 Bytes e <i>float(53)</i> um campo de 8 Bytes. O valor padrão de <i>n</i> é 53. Tamanho: 4 ou 8 Bytes.

Tipo de dados: Data e Hora	
Tipo de coluna	Descrição
datetime	Armazena data e hora (juntas), a partir de 1 de janeiro de 1753 até 31 de dezembro de 9999, com uma precisão de 3,33 milissegundos. Tamanho: 8 Bytes.
date	Armazena somente data, a partir de 1 de janeiro de 0001 até 31 de dezembro de 9999. Tamanho: 3 Bytes.
time	Armazena somente hora com precisão de 100 nanosegundos. Tamanho: 3 – 5 Bytes.

A tabela que vamos criar possui as seguintes colunas:

Nome do campo	Tipo de dado
id_pessoa (PK)	int
nome_pessoa	varchar(50)
cpf_pessoa	char(11)
endereco_pessoa	varchar(100)
cidade_pessoa	varchar(40)
renda_pessoa	decimal(10,2)

Na janela de criação de tabelas, devemos inserir as informações sobre as colunas nos locais correspondentes.

Para a coluna *id_pessoa* digitar o nome em *Column Name*, pressionar *ENTER*. O foco passa para *Data Type* que é uma caixa de seleção, onde devemos clicar e escolher o tipo de dado (nesse caso *int*). Para esse tipo de coluna não devemos nos preocupar com *Length*.

Já o último parâmetro é importante (*Allow Null*). O padrão dele é sempre estar selecionado, ou seja, a coluna aceita o valor NULL¹ (nada armazenado). No caso dessa tabela não queremos que nenhuma coluna aceite o valor NULL, portanto devemos clicar no sinal para tirar essa seleção.

	Column Name	Data Type	Length	Allow Nulls
►	id_pessoa	int	4	✓

Observe que na parte inferior da lista de colunas que serão criadas existe uma área para que sejam inseridos outros parâmetros sobre cada coluna (a lista de parâmetros muda de acordo com a coluna selecionada e com o tipo de dado escolhido para a coluna). Para a nossa primeira coluna os parâmetros são os seguintes:

Columns	
Description	
Default Value	
Precision	10
Scale	0
Identity	No
Identity Seed	
Identity Increment	
Is RowGuid	No
Formula	
Collation	

¹ **Cuidado!** Valor **NULL** significa que nada foi inserido na coluna. Não confundir com valor 0 (zero), quando um valor foi inserido (o valor zero).

- **Description** – permite colocar um texto de descrição para a coluna (auxilia muito quando temos que criar uma grande quantidade de tabelas e colunas).
- **Default Value** – permite definir um valor padrão para a coluna, ou seja, se nada for digitado, a coluna assume o valor padrão (default).
- **Identity** – permite definir que seja gerado um número único para cada novo registro que é inserido na tabela (auto-incremento). Isso normalmente é utilizado em colunas que sejam chave primária (PK) e que desejamos que o programa crie os códigos dessa coluna automaticamente.
- **Formula** – permite definir alguma operação matemática que deve ser realizada sobre o dado inserido na coluna.

Para a coluna *nome_pessoa*, o procedimento é semelhante ao anterior, somente escolhemos como tipo de dados *varchar(50)*. O parâmetro 50 do tipo de dado será inserido em *Length* (observe que 50 já aparece com valor padrão). E a coluna não deve aceitar NULL como valor.

	Column Name	Data Type	Length	Allow Nulls
	id_pessoa	int	4	
▶	nome_pessoa	varchar	50	<input type="checkbox"/>

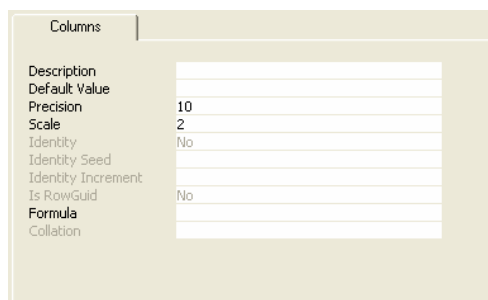
Para as colunas *cpf_pessoa*, *endereco_pessoa* e *cidade_pessoa*, o procedimento é exatamente o mesmo apenas com atenção para o tipo de variável e o tamanho escolhido (em caso de textos).


	Column Name	Data Type	Length	Allow Nulls
	id_pessoa	int	4	
	nome_pessoa	varchar	50	
	cpf_pessoa	char	11	
	endereco_pessoa	varchar	100	
	cidade_pessoa	varchar	40	
▶				


Para a próxima coluna, *renda_pessoa*, devemos prestar atenção em alguns detalhes. Para essa coluna especificamos o seu tipo de dados como sendo *decimal(10,2)*, então, depois de inserido o nome da coluna e o tipo de dados escolhido, precisamos determinar essa precisão nos parâmetros que ficam no quadro mais abaixo na tela. Os parâmetros são:

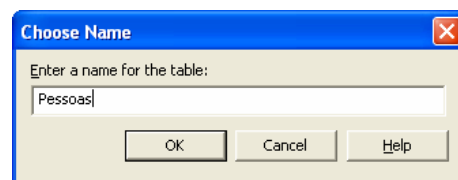
- **Precision** – especifica a quantidade máxima de dígitos que podem ser armazenados, tanto a direita, quanto a esquerda do ponto decimal;
- **Scale** – especifica a quantidade máxima de dígitos que podem ser armazenados à direita do ponto decimal (são as “casas decimais”).

Dessa forma, precisamos configurar o parâmetro *Precision* para 10 e o parâmetro *Scale* para 2. Observe a figura abaixo.

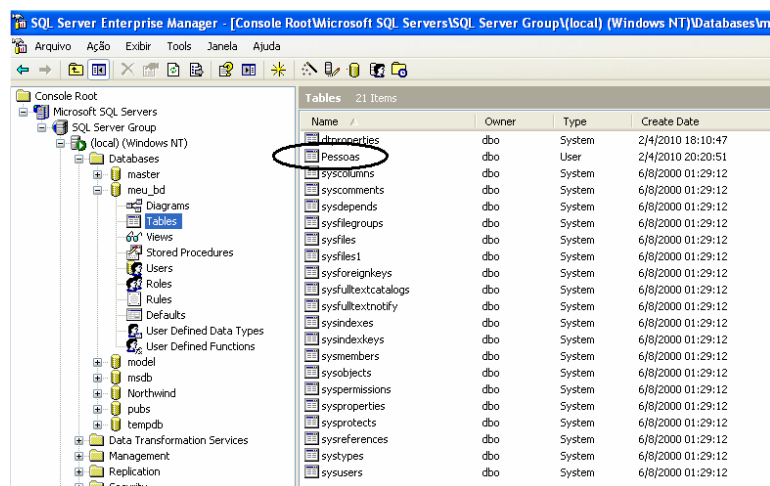


Terminamos a determinação de todas as colunas, mas ainda falta um detalhe muito importante: determinar qual das colunas será a chave primária da tabela (Primary Key – PK). A chave primária dessa tabela deve ser a coluna *id_pessoa*. Para determinar isso, basta selecionar a linha correspondente ao nome da coluna que será a chave primária (clique no botão à esquerda do nome da coluna) e clique no botão *Set Primary Key*  na barra de ferramentas. Aparecerá o mesmo ícone (uma chave) na coluna *id_pessoa*, indicando que agora ela é a chave primária.

Terminada a criação das colunas, clique no botão *Save*  da barra de ferramentas para salvar a tabela criada. Aparecerá a janela *Choose Name* (apresentada ao lado) onde deverá ser digitado o nome da tabela a ser criada (no nosso exemplo *Pessoas*). Clique em *OK* para confirmar a criação.

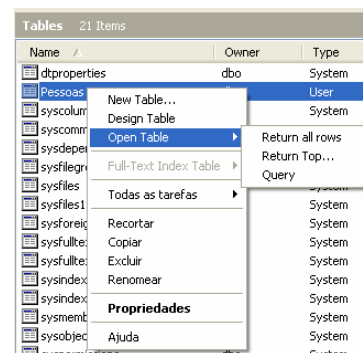


Ao fechar essa janela de criação de tabelas, voltamos para a janela inicial do *Enterprise Manager*. Se clicarmos sobre *Tables* do banco de dados *meu_bd*, podemos conferir que a tabela foi criada.



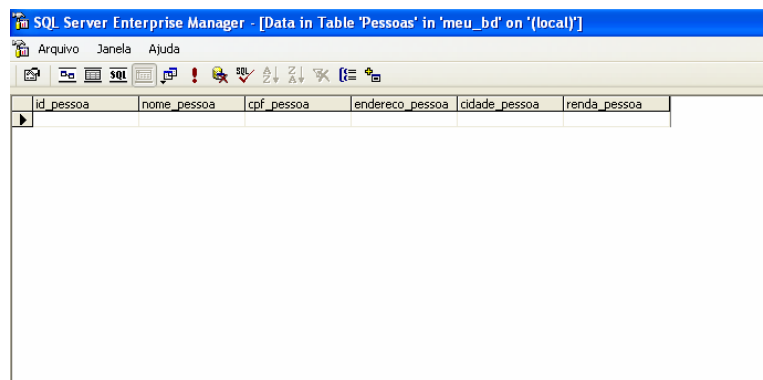
Se clicarmos com o botão direito sobre o nome da tabela, surgirá um menu (apresentado ao lado) que permite algumas operações sobre a tabela. As operações mais importantes são:

- **Design Table** – permite retornar à janela de criação da tabela para alterar algum dos parâmetros da tabela;
- **Open Table** – permite abrir a tabela para que dados sejam inseridos nela (opção *Return all rows*).



4.5. Inserindo dados na tabela

Como apresentado anteriormente, para inserir dados na tabela, clicar com o botão direito do mouse sobre o nome da tabela que desejamos trabalhar (no nosso exemplo *Pessoas*). No menu que aparece escolher a opção *Open Table* e clicar em *Return all rows*. Feito isso aparecerá a janela do *Enterprise Manager* para apresentação e inserção de dados na tabela *Pessoas*.



Observe que agora temos realmente a aparência de uma tabela, com cada coluna representando um tipo de dados que deve ser inserido.

Para inserir dados nessa tabela basta ir digitando os dados e pressionando *ENTER* até que todos os dados estejam armazenados. Uma linha (registro) só terá os seus dados armazenados quando os dados tiverem sido digitados e o cursor passado para a próxima linha vazia.

Ao terminar a entrada de dados, a aparência será a seguinte:

id_pessoa	nome_pessoa	cpf_pessoa	endereco_pessoa	cidade_pessoa	renda_pessoa
1	Tom Cruise	11122233344	Magnolia St., 2	Nova York	4600000
2	Angelina Jolie	55566677788	Lara Croft Av., 99	Los Angeles	3400000
3	Sean Connery	99900011122	Highlander St., 389	Nova York	2750000
4	Julia Roberts	33344455566	Pretty Woman Rd. 2230	Nova York	3450000
5	Brad Pitt	77788899900	Fight Club Av., 760	Washington	2100000
6	Meryl Streep	22233344455	Mamma Mia St., 890	Nova York	3400000
7	Will Smith	66677788899	Robot Av., 460	Boston	4100000
8	Penelope Cruz	00011122233	Volver St., 65	Nova York	3790000
9	Denzel Washington	44455566677	Eli Av., 188	Los Angeles	5300000
10	Sigourney Weaver	88899900011	Avatar St., 54	Washington	3200000

4.6. Criando relacionamentos entre as tabelas

Agora vamos observar como podemos criar relacionamentos entre as tabelas, utilizando o *Enterprise Manager*. Para isso vamos criar uma nova tabela, chamada *Pedidos*, com as características abaixo:


Nome do campo	Tipo de dado
id_pedido (PK)	int
data_pedido	date
id_pessoa (FK)	int
valor_pedido	decimal(10,2)

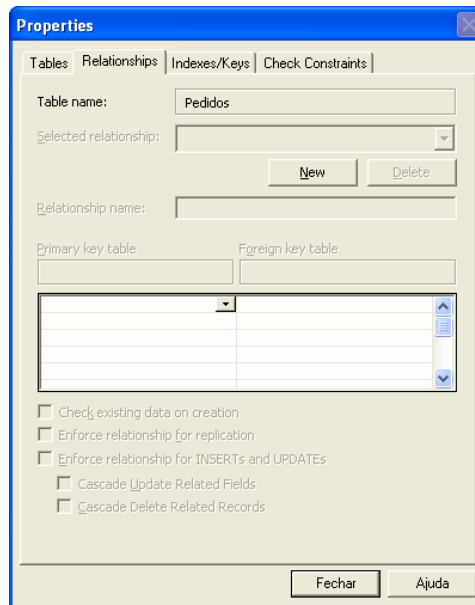
Observe que agora temos uma coluna (*id_pessoa*) que é uma chave estrangeira (Foreign Key – FK), ou seja, ela é chave primária em outra tabela (a tabela *Pessoas*).

Em primeiro lugar, devemos tomar cuidado no momento da criação da tabela, pois como essa coluna tem relacionamento com uma coluna de outra tabela, ela deve ser do mesmo tipo (nesse caso *int*).

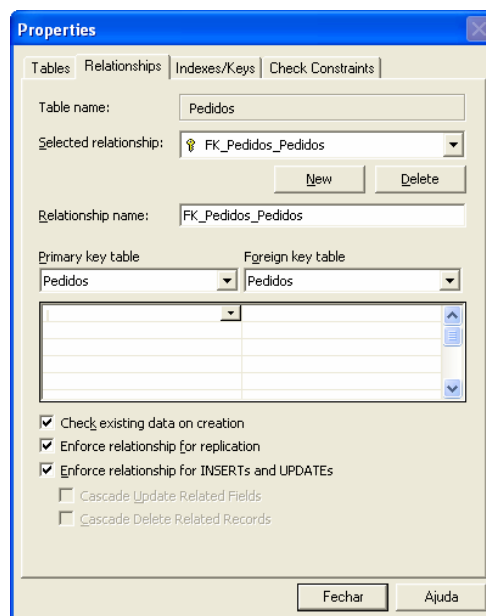
Em segundo lugar, criamos a tabela sem nos preocuparmos com a chave estrangeira (ainda não). O resultado disso será:

Column Name	Data Type	Length	Allow Nulls
id_pedido	int	4	
data_pedido	datetime	8	
id_pessoa	int	4	
valor_pedido	decimal	9	

Terminada a criação das colunas, devemos nos preocupar com a criação da chave estrangeira. Para isso, com a janela de criação da tabela *Pedidos* aberta, clicar no botão *Manage Relationships*  da barra de ferramentas. Surgirá a janela *Properties* apresentada abaixo.



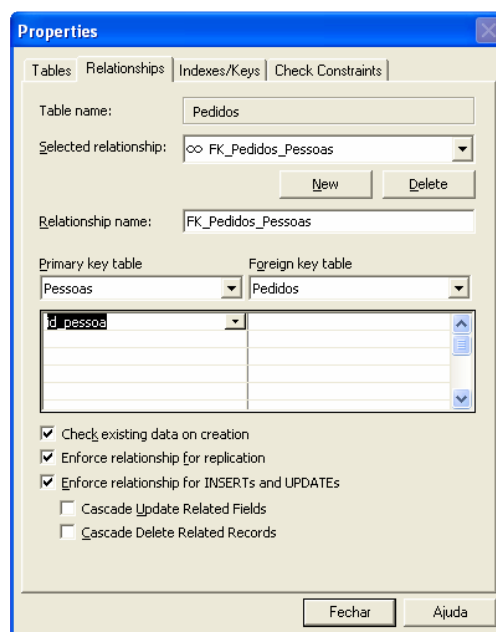
Clicar no botão *New* para criar um novo relacionamento. No campo *Relationship Name* surgirá um nome para o relacionamento que está sendo criado (observe que o nome é *FK_Pedidos_Pedidos*, portanto uma chave estrangeira).



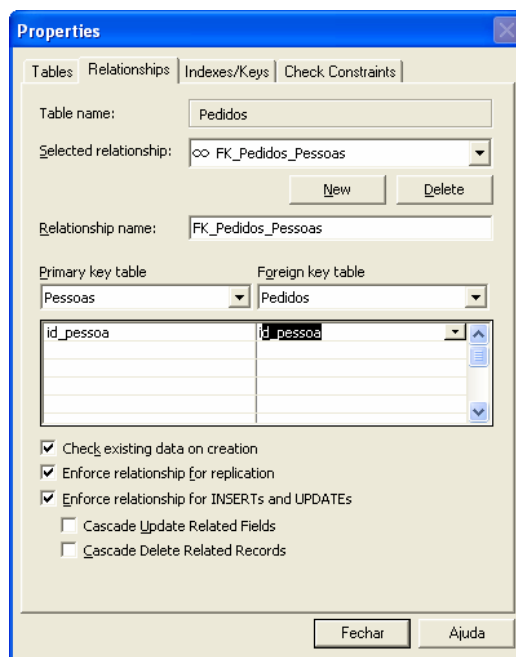
Agora devemos determinar a ligação entre os campos das tabelas. No nosso caso devemos relacionar: chave primária da tabela *Pessoas* (*id_pessoa*) com a chave estrangeira da tabela *Pedidos* (*id_pessoa*).

Para realizar essa operação devemos:

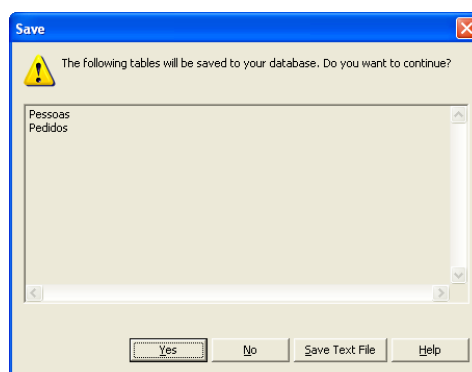
1. Escolher no campo *Primary key table* qual tabela possui a chave primária no relacionamento (no nosso exemplo *Pessoas*). Para isso basta clicar no botão à direita do campo e escolher na lista de tabelas.
2. Com a tabela escolhida, devemos escolher (no campo logo abaixo) qual coluna é a chave primária desse relacionamento. No nosso exemplo, devemos escolher *id_pessoa*.



3. Agora temos que escolher no campo *Foreign key table* qual tabela possui a chave estrangeira no relacionamento (no nosso exemplo *Pedidos*).
4. Com a tabela escolhida, devemos escolher (no campo logo abaixo) qual coluna é a chave estrangeira desse relacionamento. No nosso exemplo, devemos escolher *id_pessoa*.



5. Clicar em *Fechar*, salvar a tabela e o relacionamento está criado. Observe que aparecerá a janela *Save* abaixo, indicando que existem alterações nas duas tabelas (*Pessoas* e *Pedidos*) que devem ser salvas. Clicar em *Yes*.



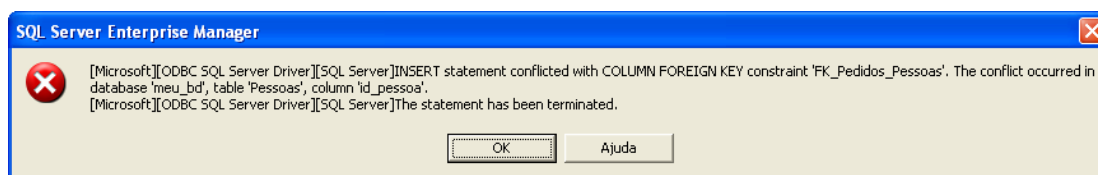
Para confirmar que o relacionamento foi criado, observe o seguinte. Se a tabela *Pedidos* tem uma chave estrangeira relacionada à tabela *Pessoas*. Isso significa que a coluna *id_pessoa* de *Pedidos* só poderá receber valores que estejam presentes (armazenados) na coluna *id_pessoa* de *Pessoas* (é para isso que serve uma chave estrangeira).

Vamos comprovar isso, inserindo dados na tabela *Pedidos* (exatamente como fizemos com a tabela *Pessoas*). Observando a tabela *Pessoas*, temos valores para *id_pessoa* de 1 até 10, ou seja, qualquer valor fora dessa faixa, inserido na tabela *Pedidos* deveria ser rejeitado.

Tentaremos inserir os seguintes dados na tabela *Pedidos*:

SQL Server Enterprise Manager - [Data in Table 'Pedidos' in 'meu...]			
Arquivo Janela Ajuda			
id_pedido data_pedido id_pessoa valor_pedido			
1	30/3/2009	13	2300

Ao pressionar ENTER para confirmar os dados da linha (registro), surgirá a seguinte mensagem:



Essa mensagem indica que houve um conflito com a restrição de chave estrangeira na tabela *Pessoas*, pois tentamos utilizar um valor para *id_pessoa* que não existe naquela tabela. Os dados não serão armazenados dessa forma, portanto teremos que modificar o valor de *id_pessoa* para se adequar ao determinado.

Obs: A tabela *Pedido* apresenta outro detalhe interessante. Observe que a coluna *data_pedido* deve ser preenchida com uma data. Devemos preenchê-la com uma data no formato *dd/mm/aaaa* e caso essa data seja inválida (por exemplo, dia 31 em um mês de fevereiro) também será gerada uma mensagem de erro impedindo o armazenamento dos dados da linha.

4.7. Exclusão de bancos de dados e tabelas

Para excluir qualquer banco de dados ou tabela que tenha sido criada, utilizando o *Enterprise Manager*, basta clicar com o botão direito do mouse sobre o que se deseja excluir e escolher a opção *Excluir*. Os menus são um pouco diferentes, dependendo do que foi selecionado, mas sempre existirá a opção de exclusão do objeto.


Aparecerá uma janela para confirmação de exclusão. Lembre que depois de excluído o objeto não poderá mais ser recuperado (a não ser que exista uma cópia de segurança dele).

4.8. Criando Índices

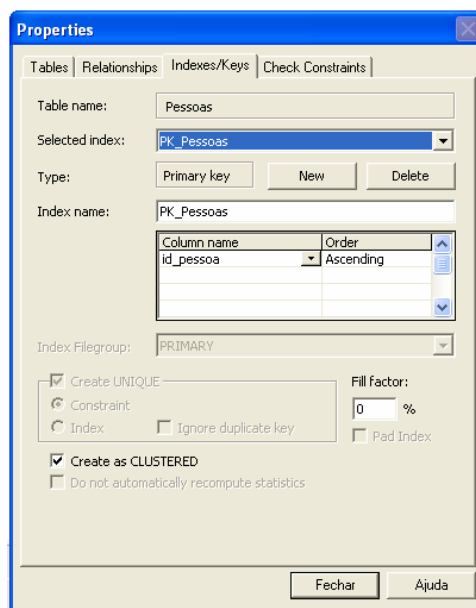
Um índice pode ser criado em uma tabela para permitir que a aplicação de banco de dados encontre os dados mais rapidamente e eficientemente, sem a necessidade de ler toda a tabela. Os usuários não vêem os índices, apenas percebem o aumento da velocidade nas suas consultas aos dados. Os índices são criados naquelas colunas onde o usuário (por meio dos programas que ele executa) frequentemente realiza buscas de dados.

Obs: A atualização dos dados em tabelas com índices leva mais tempo do que em tabelas sem eles (porque os índices também precisam ser atualizados). Dessa forma, somente devemos criar índices em colunas (e tabelas) que serão frequentemente utilizadas em buscas (consultas) de dados.

Vamos criar um índice para *renda_pessoa* da tabela *Pessoas*. Para isso navegue até a tabela, clicar com o botão direito sobre o nome da tabela *Pessoas* e escolher a opção *Design Table*.

Com isso voltaremos para a nossa janela de criação/alteração de características das colunas. Clicar no botão *Manage Relationships*  da barra de ferramentas. Surgirá a janela *Properties*, como vimos anteriormente.

Na janela *Properties* clicar na guia *Indexes/Keys*. A janela terá essa aparência:

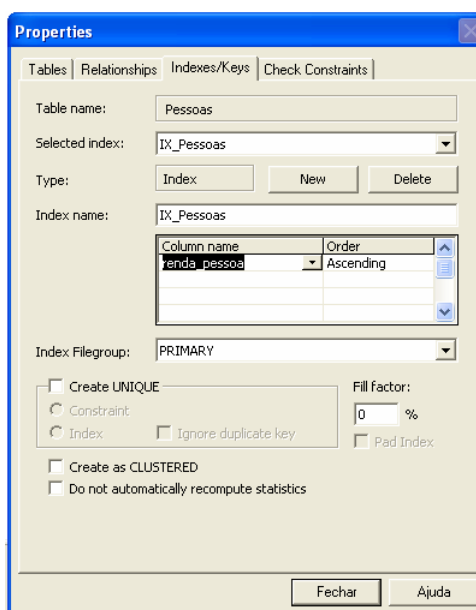


Nessa guia é possível criar índices para as tabelas. Observe que já temos um índice criado: a chave primária da tabela (*id_pessoa*), que como padrão, sempre é considerada um índice.

Para criar um novo índice clicar no botão *New*. No campo *Selected Index* surgirá uma sugestão de nome para a índice que está sendo criado (*IX_Pessoas*). Caso você queira alterar o nome, basta digitar um novo nome no campo *Index Name*.

Escolhido o nome precisamos escolher a coluna que será indexada. No nosso caso é a coluna *renda_pessoa*. Para escolher a coluna, logo abaixo do campo *Index Name* temos uma caixa de seleção, basta clicar e escolher o nome da coluna.

Ao lado é possível escolher se a indexação ocorrerá crescentemente (*Ascending*) ou decrescentemente (*Descending*). A janela ficará com a seguinte aparência:



Terminada a escolha, basta fechar a janela (botão *Fechar*) que o índice estará criado. Não esqueça de salvar as alterações na tabela.

CAPÍTULO VIII – Linguagem SQL – Comandos DDL

SQL (Structured Query Language) é uma linguagem de pesquisa de dados declarativa que é utilizada em conjunto com bancos de dados relacionais. Essa linguagem foi desenvolvida no início dos anos 70 pela IBM, para demonstrar a viabilidade da implementação do Modelo Relacional proposto por E. F. Codd naquela época.

O seu nome original era **SEQUEL (Structured English Query Language)**, modificado para que se transformasse em um padrão mundial. Isso aconteceu em 1986 quando a ANSI (*American National Standard Institute*) padronizou a linguagem e depois em 1987, quando a.

Em 1987 ISO (*International Standard Organization*) a tornou o SQL um padrão mundial. Isso aconteceu também devido a sua simplicidade e facilidade de uso. É uma linguagem declarativa, onde se determina a forma do resultado esperado e não o caminho para chegar até o resultado (isso diminui o ciclo de aprendizado).

Apesar dessa padronização em nível mundial, existem muitas variações e extensões que dependem do fabricante do sistema gerenciador de banco de dados. Normalmente, os comandos básicos são sempre os mesmos.

Uma outra característica importante do SQL é que ela é uma *linguagem hospedeira*, ou seja, para acessar dados dos bancos de dados relacionais, as linguagens de programação permitem que comandos na linguagem SQL sejam “embutidos” no programa que está sendo criado.

Até o momento, vimos como criar e configurar bancos de dados e tabelas, além de inserir e alterar dados em tabelas, utilizando os recursos do *Microsoft SQL Server 2000* (usando o *Enterprise Manager*). Isso é bastante interessante, porque conseguimos realizar todas essas operações por meio de interfaces gráficas que facilitam bastante o trabalho. Mas isso apresenta um problema.

Normalmente, a criação dos bancos de dados e das tabelas fica a cargo de uma determinada pessoa (o **DBA – Database Administrator**), que tem acesso direto aos recursos do sistema gerenciador de bancos de dados. Já a inclusão e o acesso (consulta) aos dados ficam a cargo de outra pessoa (o desenvolvedor do sistema), que normalmente não poderá pedir para que o usuário acesse diretamente o sistema gerenciador de bancos de dados para realizar os seus trabalhos. Sendo assim, o desenvolvedor necessita “embutir” comandos no seu programa para realizar essas tarefas. Esses comandos são criados utilizando a linguagem SQL.

4.1. Comandos da Linguagem SQL

Os comandos da linguagem SQL são divididos em categorias, de acordo com o tipo de operação que realizam. Os tipos são:

- **DDL (Data Definition Language)** – comandos que são usados para definir (criar e alterar) bancos de dados e tabelas.
- **DML (Data Manipulation Language)** – comandos que são usados para inserir dados e para realizar consultas aos dados dos bancos de dados.

4.2. Comandos DDL

Comandos de criação e exclusão de bancos de dados (*database*), tabelas (*table*) e índices (*index*). Além disso, especificam ligações entre as tabelas e impõem restrições entre as tabelas. Os comandos mais importantes são:

- **CREATE DATABASE** – cria um novo banco de dados;
- **ALTER DATABASE** – modifica um banco de dados;
- **CREATE TABLE** – cria uma nova tabela;
- **ALTER TABLE** – modifica uma tabela;
- **DROP TABLE** – exclui uma tabela;
- **CREATE INDEX** – cria um índice (chave de busca);
- **DROP INDEX** – exclui um índice.

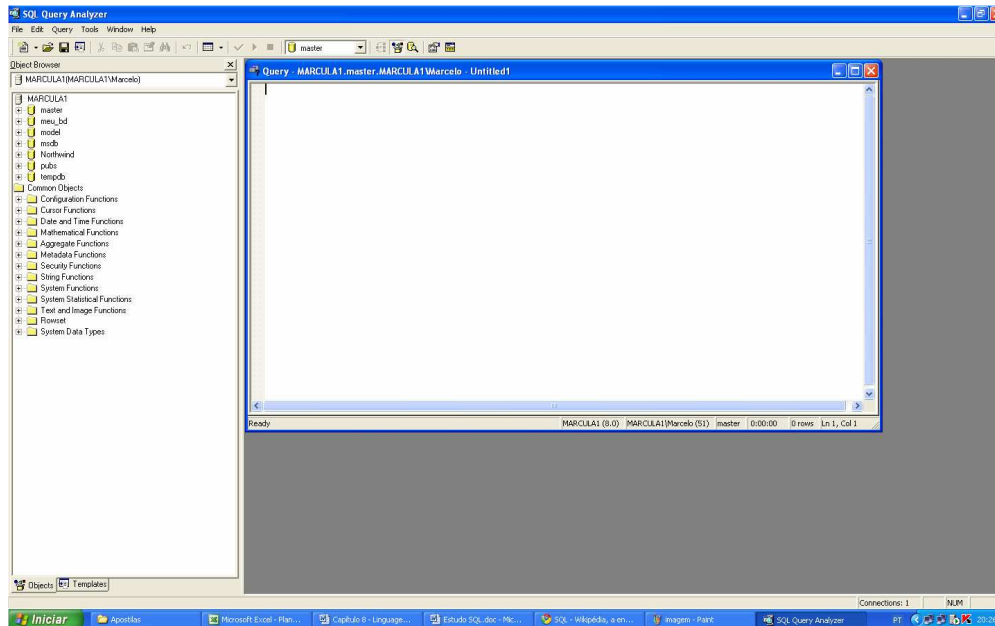
Para trabalhar com os comandos SQL dentro do *Microsoft SQL Server 2000* utilizaremos os recursos do *Query Analyzer*. Devemos apenas lembrar que dentro do SQL Server, a linguagem de consulta é conhecida como *Transact-SQL*.

4.3. Query Analyzer

Para acessar o *Query Analyzer* seguir a sequência: *Iniciar* → *Programas* → *Microsoft SQL Server* → *Query Analyzer*. Surgirá a seguinte janela:



Nessa janela deverá ser escolhido o servidor ativo (no campo *SQL Server*) e o tipo de autenticação (em *Windows authentication*). Escolhidas as opções adequadas, clicar em *OK* e surgirá a janela do *Query Analyzer*.



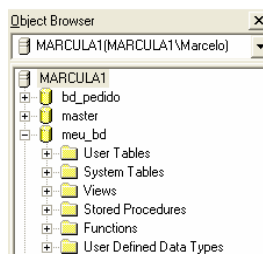
Na janela à esquerda (chamada *Object Browser*) temos os bancos de dados que podemos utilizar (observe que o nosso banco de dados *meu_bd* está nessa lista). Na janela em branco, à direita, temos a nossa área de trabalho, onde serão digitados os comandos SQL.

Obs: Devemos ter sempre em mente que TODAS as operações que realizamos até aqui, podem ser reproduzidas utilizando-se de comandos SQL.

Obs: A linguagem *Transact-SQL* não é *case sensitive*, ou seja, podemos digitar os comandos com letras maiúsculas ou minúsculas. O resultado é o mesmo.

4.4. Usando o *Object Browser*

No *Object Browser* podemos “navegar” pelos bancos de dados. Observando os bancos de dados listados, vemos o nosso banco de dados *meu_bd* criado anteriormente. Podemos observar a sua configuração. Primeiramente clicamos no sinal [+] ao lado do nome do banco de dados e uma lista é expandida.

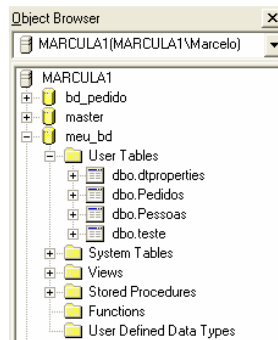


Nessa lista podemos observar alguns objetos:

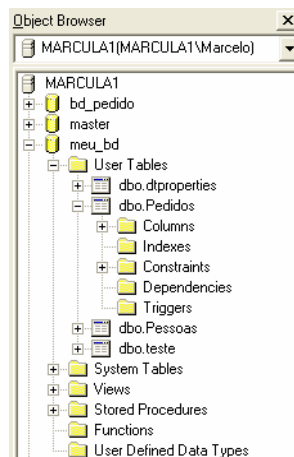
- **User Tables** – tabelas que pertencem a esse banco de dados;
- **System Tables** – tabelas geradas pelo gerenciador de banco de dados para dar suporte ao funcionamento do banco de dados criado;
- **Views** – visões (consultas) criadas no banco de dados;

- **Stored Procedures** – rotinas armazenadas no banco de dados (detalharemos mais futuramente);
- **Functions** – funções relacionadas ao banco de dados;
- **User Defined Data Types** – tipos de dados criados pelo usuário no banco de dados.

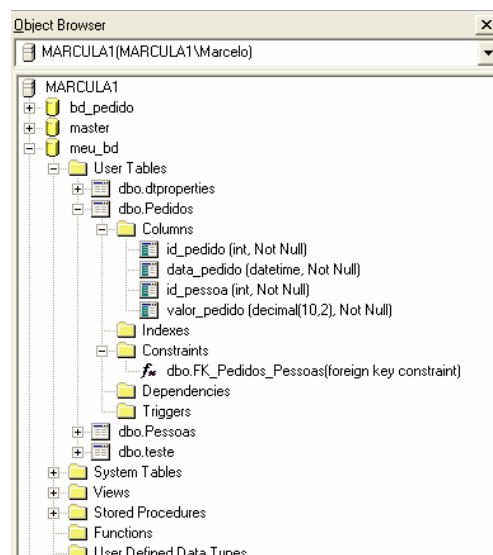
No momento, nos interessa observar os detalhes das tabelas criadas. Para isso clicar no [+] ao lado de *User Tables*. Surgirá o seguinte:



São as tabelas que foram criadas no banco de dados *meu_bd* (*dbo.Pedidos* e *dbo.Pessoas*). Para observar os detalhes da tabela, clicar no [+] ao lado do nome dela. Clicando em *dbo.Pedidos*, teremos:



Observe na figura abaixo que após expandirmos os itens *Columns* e *Constraints* podemos observar as colunas que compõem a tabela (com os seus detalhes) e a restrição de chave estrangeira criada.



Outros detalhes que não trabalhamos ainda podem ser observados também:

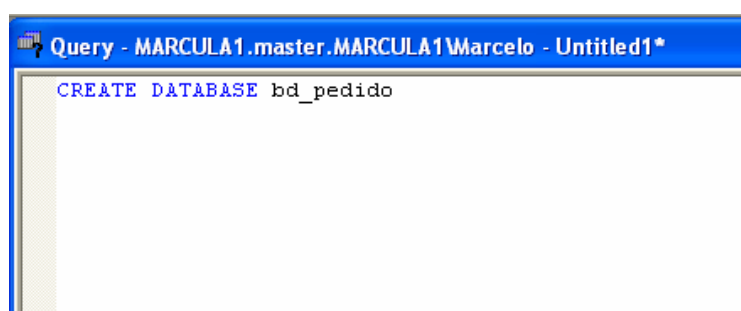
- **Indexes** – apresenta os índices criados para colunas da tabela;
- **Dependencies** – apresenta as dependências dessa tabela;
- **Triggers** – apresenta os *triggers* relacionados a essa tabela (esse assunto será abordado futuramente).


4.5. Criar um banco de dados (CREATE DATABASE)

O comando **CREATE DATABASE** é usado para criar um banco de dados.

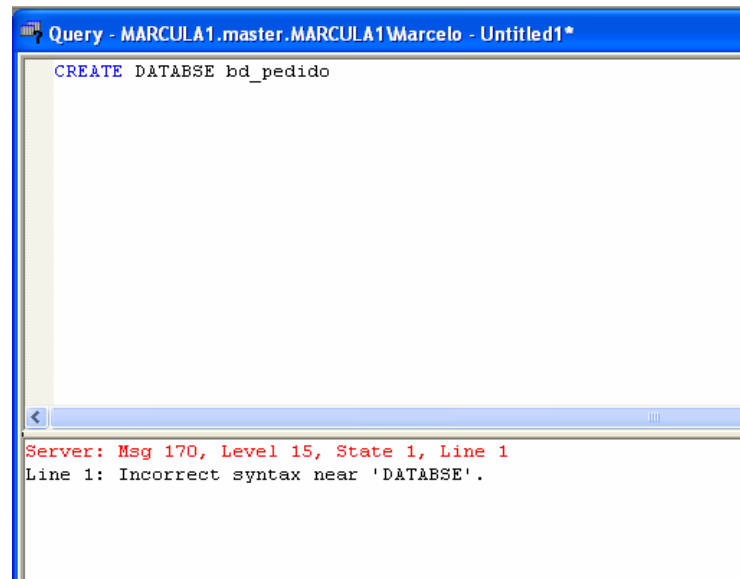
CREATE DATABASE nome_bancomedados;

Como exemplo, vamos criar o banco de dados bd_pedido, para tanto devemos digitar o seguinte comando na área de trabalho do *Query Analyzer*:




Precisamos executar o comando, mas antes podemos pedir para que o *Query Analyzer* analise os comandos digitados para conferir se existe algum erro de sintaxe. Para realizar isso, podemos clicar no botão *Parse Query*  na barra de ferramentas, pressionar **CTRL+F5** no teclado ou acessar o menu *Query* da barra de menus e clicar na opção *Parse*.

Obs: Abaixo uma janela com o exemplo de uma mensagem de erro caso tenhamos digitado o comando de forma errada. Veja que o *Query Analyzer* indica a linha que contém o erro e o tipo de erro (nesse caso um erro de sintaxe incorreta – *Incorrect Syntax*).

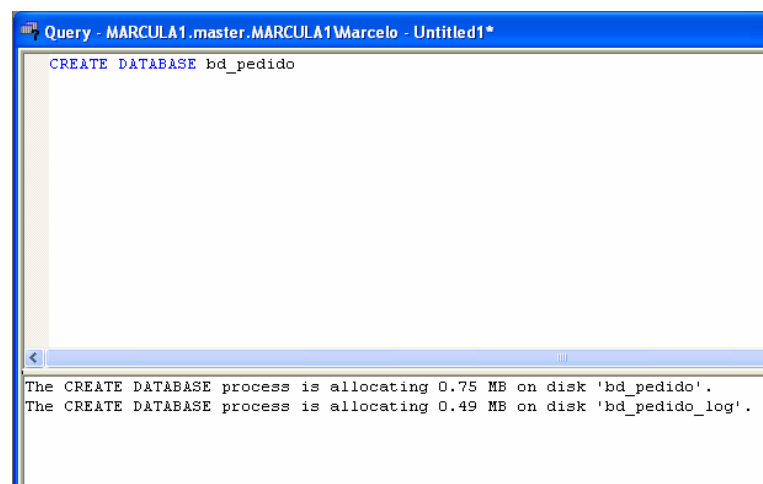


Digitando o comando corretamente, obtemos a mensagem "*The command(s) completed sucessfully*", indicando que não cometemos nenhum erro de sintaxe.

Dessa forma, podemos solicitar que o programa execute o comando. Isso pode ser feito clicando no botão

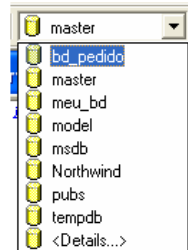
Execute Query  na barra de ferramentas, pressionando *F5* no teclado ou acessando o menu *Query* e clicando na opção *Execute*.

Caso os comandos tenham sido executados sem problemas e o banco de dados criado, surgirá a seguinte mensagem:



Obs: O SQL Server criou dois arquivos. O primeiro *bd_pedido* é o arquivo do banco de dados propriamente dito (arquivo *bd_pedido.mdb*) e o arquivo *bd_pedido_log* é o arquivo de *log* que vai armazenar informações durante o funcionamento do banco de dados (arquivo *bd_pedido_log.ldf*). Além disso, é informado o tamanho inicial dos bancos de dados.

Algumas vezes, o banco de dados criado não aparece imediatamente no *Object Browser*, então como conferir se ele foi criado mesmo? Acima da janela de trabalho, temos uma caixa de seleção que apresenta o nome dos bancos de dados disponíveis para o uso. Basta clicar nela e conferir o banco de dados criado (*db_pedido*).

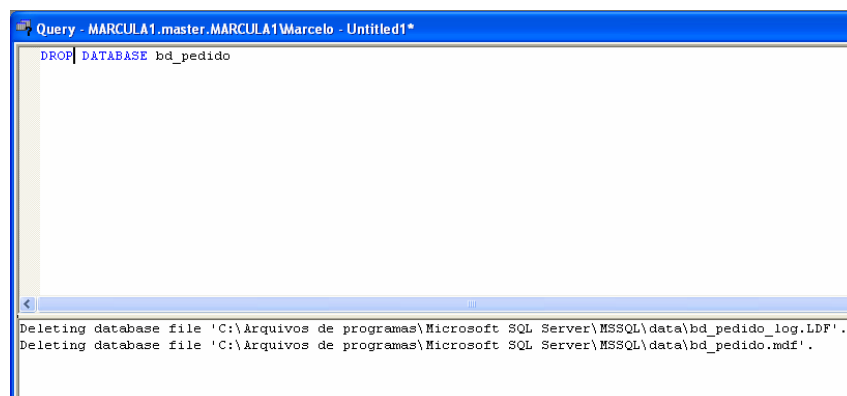


4.6. Excluir um banco de dados (DROP DATABASE)

O comando **DROP DATABASE** é utilizado para excluir um banco de dados:

DROP DATABASE *nome_bancomedados*

Para executar esse comando, basta digitar o comando seguido do nome do banco de dados a ser excluído e mandar executar o comando. Esse comando não tem como ser revertido, ou seja, os dados do banco de dados estarão perdidos. Realizada a operação a seguinte mensagem será apresentada:



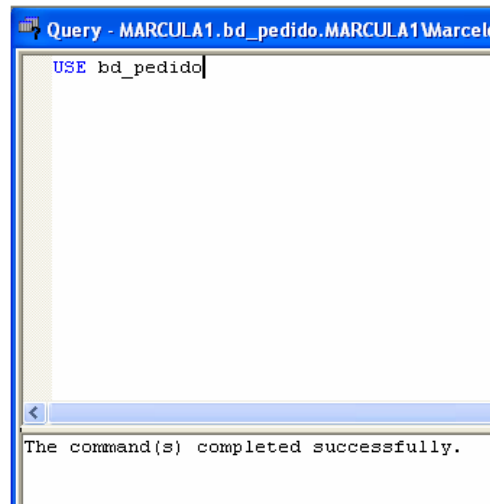
4.7. Habilitar um banco de dados para o uso (USE)

Para que possamos trabalhar com um banco de dados é necessário habilitar o seu uso. Para isso utilizamos o comando **USE**.

USE *nome_bancomedados*

Cuidado, pois se esquecermos de determinar qual banco de dados está em uso no momento, podemos criar tabelas dentro de bancos de dados errados.

Para habilitar o nosso banco de dados *db_pedido*, devemos digitar o seguinte comando e executá-lo:



Obs: outro caminho para realizarmos a mesma operação é clicar na caixa de seleção com os nomes dos bancos de dados e clicar em **bd_pedido**. O resultado é o mesmo.

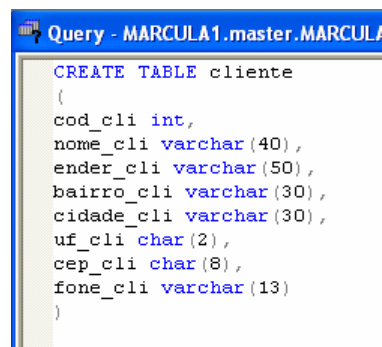
4.8. Criar uma tabela (CREATE TABLE)

O comando **CREATE TABLE** é usado para criar uma tabela em um banco de dados. Esse comando comporta vários parâmetros que vamos analisar em seguida. Abaixo uma sintaxe padrão utilizando a maioria dos parâmetros:

```
CREATE TABLE nome_tabela
(
    nome_coluna1 tipo_dado,
    nome_coluna2 tipo_dado,
    nome_coluna3 tipo_dado,
    ....
    nome_colunaN tipo_dado
);
```

O parâmetro *tipo_dado* especifica que tipo de dado a coluna pode armazenar.

Com isso podemos criar uma tabela denominada *cliente* com os seguintes comandos:



Ainda não vamos executar os comandos SQL. Faltam alguns detalhes importantes.

4.9. Restrições SQL

As restrições SQL são utilizadas para criar limitações à operação da tabela. Mas essas restrições vão se mostrar muito importantes para o funcionamento do banco de dados da forma como desejamos. Essas restrições podem ser especificadas quando uma tabela é criada (comando CREATE TABLE) ou depois da tabela criada (comando ALTER TABLE que será visto posteriormente).

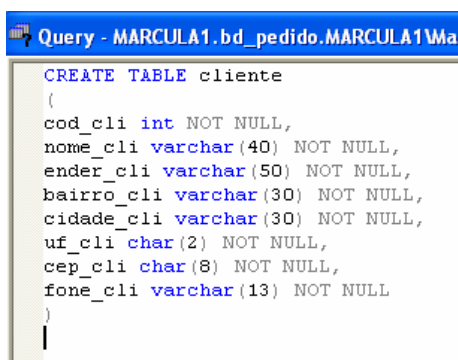
4.9.1. Restrição NOT NULL

A restrição **NOT NULL** força uma coluna a não aceitar valores *NULL* (nulos), ou seja, o campo sempre deve conter algum valor. Isso significa que você não pode inserir um novo registro, ou atualizar um registro, sem adicionar algum valor para esse campo.

A restrição NOT NULL deve ser determinada durante a criação da tabela e para cada coluna que desejamos impor a restrição. Caso nada seja determinado, o valor padrão para uma coluna, quando ela é criada é *NULL*, ou seja, ela aceita não ter qualquer valor armazenado nela.

```
CREATE TABLE nome_tabela
(
  nome_coluna1 tipo_dado NOT NULL,
  nome_coluna2 tipo_dado NOT NULL,
  nome_coluna3 tipo_dado,
  ....
  nome_colunaN tipo_dado
);
```

Na tabela que estamos criando, nenhuma coluna pode ser deixada sem valor, portanto a restrição *NOT NULL* deve ser aplicada a todas as colunas.



```
Query - MARCULA1.bd_pedido.MARCULA1Ma
CREATE TABLE cliente
(
  cod_cli int NOT NULL,
  nome_cli varchar(40) NOT NULL,
  ender_cli varchar(50) NOT NULL,
  bairro_cli varchar(30) NOT NULL,
  cidade_cli varchar(30) NOT NULL,
  uf_cli char(2) NOT NULL,
  cep_cli char(8) NOT NULL,
  fone_cli varchar(13) NOT NULL
)
```

4.9.2. Restrição PRIMARY KEY

A restrição **PRIMARY KEY** identifica de forma unívoca cada registro em uma tabela do banco de dados, ou seja, determina que aquela coluna seja a **Chave Primária (PK)** da tabela. É uma restrição fundamental de ser determinada porque toda tabela deve ter uma chave primária.

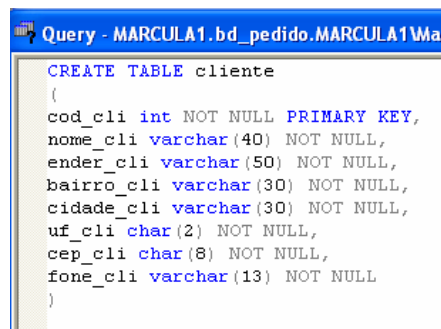
Chaves primárias devem conter valores únicos e não pode conter valores *NULL*.

```
CREATE TABLE nome_tabela
(
  nome_coluna1 tipo_dado NOT NULL PRIMARY KEY,
  nome_coluna2 tipo_dado NOT NULL,
  nome_coluna3 tipo_dado,
  ....
  nome_colunaN tipo_dado
);
```

Opcionalmente é possível criar um nome para a restrição **PRIMARY KEY** e ao mesmo tempo determinar a restrição para várias colunas ao mesmo tempo (basta colocar os nomes das colunas dentro dos parênteses do comando). Os comandos SQL ficariam da seguinte maneira:

```
CREATE TABLE nome_tabela
(
  nome_coluna1 tipo_dado NOT NULL,
  nome_coluna2 tipo_dado NOT NULL,
  nome_coluna3 tipo_dado,
  ....
  nome_colunaN tipo_dado
  CONSTRAINT nome_restricção PRIMARY KEY (nome_coluna_PK)
);
```

Na nossa tabela *cliente* a coluna chave primária será *cod_cli* e os comandos ficarão da seguinte maneira (optamos pelo primeiro método de determinação de chave primária, pois é mais simples):



```
Query - MARCULA1.bd_pedido.MARCULA1Wa
CREATE TABLE cliente
(
  cod_cli int NOT NULL PRIMARY KEY,
  nome_cli varchar(40) NOT NULL,
  ender_cli varchar(50) NOT NULL,
  bairro_cli varchar(30) NOT NULL,
  cidade_cli varchar(30) NOT NULL,
  uf_cli char(2) NOT NULL,
  cep_cli char(8) NOT NULL,
  fone_cli varchar(13) NOT NULL
);
```

4.9.3. Comando IDENTITY

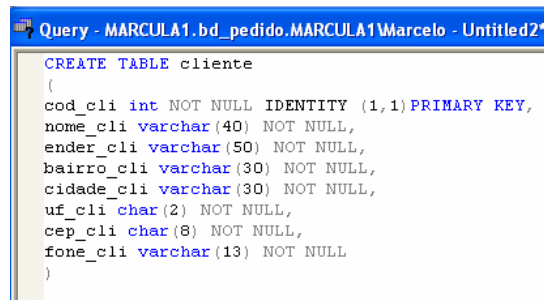
Frequentemente necessitamos que o valor da chave primária da tabela seja criado automaticamente, toda vez que uma nova linha (registro) for inserida. A solução é criar uma coluna de auto-incremento.

O comando **IDENTITY** permite que seja gerado um número único para cada nova linha (registro) que é inserida na tabela (coluna de auto-incremento). Esse comando deve ser utilizado sempre em conjunto com a restrição **PRIMARY KEY**.

```
CREATE TABLE nome_tabela
(
  nome_coluna1 tipo_dado NOT NULL IDENTITY (1,1) PRIMARY KEY,
  nome_coluna2 tipo_dado NOT NULL,
  nome_coluna3 tipo_dado,
  ....
  nome_colunaN tipo_dado
);
```

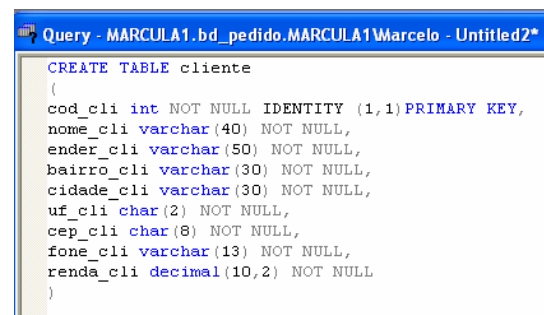
Obs: Como padrão, o valor inicial para **IDENTITY** é 1 e ele incrementará em 1 para cada nova linha inserida. Mas isso pode ser mudado utilizando, por exemplo, o comando **IDENTITY (10,5)**, que tem como valor inicial 10 e incremento de 5 para cada nova linha.

Na tabela cliente vamos determinar que a coluna *cod_cli* tenha os seus valores criados automaticamente (auto-incremento). Para isso, temos o seguinte comando:



```
Query - MARCULA1.bd_pedido.MARCULA1\Marcelo - Untitled2*
CREATE TABLE cliente
(
  cod_cli int NOT NULL IDENTITY (1,1) PRIMARY KEY,
  nome_cli varchar(40) NOT NULL,
  ender_cli varchar(50) NOT NULL,
  bairro_cli varchar(30) NOT NULL,
  cidade_cli varchar(30) NOT NULL,
  uf_cli char(2) NOT NULL,
  cep_cli char(8) NOT NULL,
  fone_cli varchar(13) NOT NULL
)
```

Como ainda não criamos a tabela *cliente*, podemos a qualquer momento acrescentar colunas que tenhamos esquecido. Nesse caso, vamos acrescentar a coluna *renda_cli*, do tipo *decimal (10,2)*.



```
Query - MARCULA1.bd_pedido.MARCULA1\Marcelo - Untitled2*
CREATE TABLE cliente
(
  cod_cli int NOT NULL IDENTITY (1,1) PRIMARY KEY,
  nome_cli varchar(40) NOT NULL,
  ender_cli varchar(50) NOT NULL,
  bairro_cli varchar(30) NOT NULL,
  cidade_cli varchar(30) NOT NULL,
  uf_cli char(2) NOT NULL,
  cep_cli char(8) NOT NULL,
  fone_cli varchar(13) NOT NULL,
  renda_cli decimal(10,2) NOT NULL
)
```

4.9.4. Restrição CHECK

A restrição **CHECK** é utilizada para limitar a faixa de valores que podem ser colocados em uma coluna. Essa faixa de valores será determinada criando-se uma condição associada à coluna que sofrerá a restrição. Essa restrição pode incluir mais do que uma coluna da tabela

```
CREATE TABLE nome_tabela
(
  nome_coluna1 tipo_dado NOT NULL IDENTITY (1,1) PRIMARY KEY,
  nome_coluna2 tipo_dado NOT NULL,
  nome_coluna3 tipo_dado CHECK (condição),
  ....
  nome_colunaN tipo_dado
);
```

Opcionalmente é possível criar um nome para a restrição CHECK e ao mesmo tempo determinar a restrição utilizando várias colunas ao mesmo tempo.

```
CREATE TABLE nome_tabela
(
  nome_coluna1 tipo_dado NOT NULL IDENTITY (1,1) PRIMARY KEY,
  nome_coluna2 tipo_dado NOT NULL,
  nome_coluna3 tipo_dado,
  ....
  nome_colunaN tipo_dado,
  CONSTRAINT nome_restricao CHECK (condição)
);
```

Na tabela *cliente* a coluna *renda_cli* deve sofrer esse tipo de restrição, pois não podemos cadastrar nenhum cliente com valores inferiores a 500 nessa coluna. Os comandos a serem usados são:

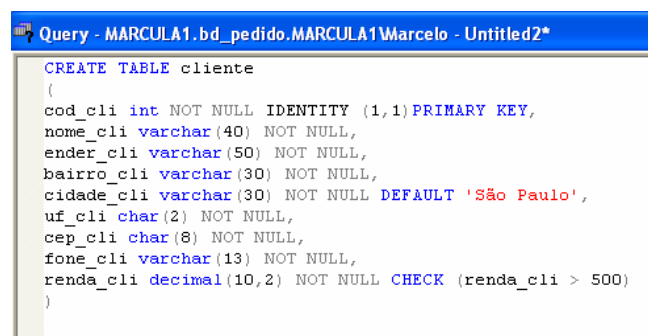
```
Query - MARCULA1.bd_pedido.MARCULA1\Marcelo - Untitled2*
CREATE TABLE cliente
(
  cod_cli int NOT NULL IDENTITY (1,1) PRIMARY KEY,
  nome_cli varchar(40) NOT NULL,
  ender_cli varchar(50) NOT NULL,
  bairro_cli varchar(30) NOT NULL,
  cidade_cli varchar(30) NOT NULL,
  uf_cli char(2) NOT NULL,
  cep_cli char(8) NOT NULL,
  fone_cli varchar(13) NOT NULL,
  renda_cli decimal(10,2) NOT NULL CHECK (renda_cli > 500)
);
```

4.9.5. Restrição DEFAULT

A restrição **DEFAULT** é utilizada para inserir um valor padrão (default) na coluna. Esse valor será inserido caso nenhum outro valor for especificado.

```
CREATE TABLE nome_tabela
(
  nome_coluna1 tipo_dado NOT NULL IDENTITY (1,1) PRIMARY KEY,
  nome_coluna2 tipo_dado NOT NULL DEFAULT (valor_padrão),
  nome_coluna3 tipo_dado CHECK (condição),
  ....
  nome_colunaN tipo_dado
);
```

Na tabela *cliente* vamos utilizar essa restrição para a coluna *cidade_cli*, pois maioria das pessoas a ser cadastrada é da cidade de 'São Paulo'. Os comandos ficarão da seguinte maneira:



```
CREATE TABLE cliente
(
  cod_cli int NOT NULL IDENTITY (1,1) PRIMARY KEY,
  nome_cli varchar(40) NOT NULL,
  ender_cli varchar(50) NOT NULL,
  bairro_cli varchar(30) NOT NULL,
  cidade_cli varchar(30) NOT NULL DEFAULT 'São Paulo',
  uf_cli char(2) NOT NULL,
  cep_cli char(8) NOT NULL,
  fone_cli varchar(13) NOT NULL,
  renda_cli decimal(10,2) NOT NULL CHECK (renda_cli > 500)
)
```

Agora que aplicamos todas as restrições podemos executar os comandos e criar a tabela *cliente*.

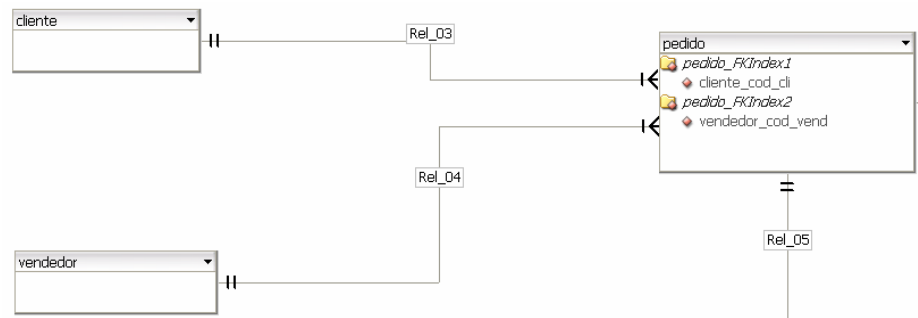
A próxima restrição importante a ser aplicada às tabelas é aquela relacionada com a chave estrangeira, mas para isso, devemos criar uma outra tabela que participará dos relacionamentos que devemos criar. Execute os seguintes comandos SQL para criação das tabelas *vendedor*:

```
CREATE TABLE vendedor
(
  cod_vend int IDENTITY(1,1) PRIMARY KEY,
  nome_vende varchar(40) NOT NULL,
  fone_vende varchar(13) NOT NULL
)
```

Obs: Observe que na coluna designada como chave primária (*cod_vend*) não recebeu a restrição *NOT NULL*. Isso porque é padrão para o SQL Server que a coluna chave primária receba automaticamente essa restrição. Portanto, não é necessário colocar explicitamente a restrição.

4.9.6. Restrição FOREIGN KEY

Para que fique fácil compreender o motivo da criação da tabela *vendedor* observe o Modelo Lógico que estamos implementando. Já temos as tabelas *cliente* e *vendedor* criadas, mas agora precisamos criar a tabela *pedido*, com o detalhe de que ela possuirá dois campos chave estrangeira, ligados às tabelas anteriormente criadas. Para isso utilizaremos a restrição **FOREIGN KEY**.



A restrição **FOREIGN KEY** (Chave Estrangeira - FK) cria uma coluna, em uma tabela, que “aponta” para a chave primária (PK) de outra tabela. A definição de Chave Estrangeira é a seguinte: “É um campo que não é chave primária em uma tabela, mas que é chave primária em outra tabela, com isso criando uma ligação entre as duas tabelas”.

A chave estrangeira é utilizada para evitar ações que possam destruir a ligação entre as tabelas (no modelo acima, excluir um cliente que ainda possui um pedido). Além disso, ela também evita que dados inválidos sejam inseridos na coluna determinada como chave estrangeira, porque os valores devem estar contidos na tabela para a qual ela aponta (no modelo, não é possível criar um pedido para um vendedor que não esteja na tabela *vendedor*).

```
CREATE TABLE nome_tabela
(
    nome_coluna1 tipo_dado NOT NULL IDENTITY (1,1) PRIMARY KEY,
    nome_coluna2 tipo_dado NOT NULL DEFAULT (valor_padrão),
    nome_coluna3 tipo_dado CHECK (condição),
    nome_coluna4 tipo_dado FOREIGN KEY REFERENCES nome_tabela2 (nome_coluna_PK),
    ....
    nome_colunaN tipo_dado
);
```

Onde:

nome_tabela2 – é o nome da tabela para a qual a coluna aponta

nome_coluna_PK – é o nome da coluna chave primária em *nome_tabela2*

Opcionalmente é possível criar um nome para a restrição **FOREIGN KEY** e ao mesmo tempo determinar a restrição para várias colunas ao mesmo tempo (basta colocar os nomes das colunas dentro dos parênteses do comando). Os comandos SQL ficariam da seguinte maneira:

```
CREATE TABLE nome_tabela
(
    nome_coluna1 tipo_dado NOT NULL IDENTITY (1,1) PRIMARY KEY,
    nome_coluna2 tipo_dado NOT NULL DEFAULT (valor_padrão),
    nome_coluna3 tipo_dado CHECK (condição),
    nome_coluna4 tipo_dado,
    ....
    nome_colunaN tipo_dado,
    CONSTRAINT nome_restrição FOREIGN KEY (nome_coluna_FK) REFERENCES nome_tabela2
        (nome_coluna_PK),
);
```

Onde:

nome_coluna_FK – é o nome da coluna chave estrangeira em *nome_tabela* (tabela que está sendo criada)

Vamos criar a tabela *pedido* que possua duas colunas chave estrangeira. Uma delas apontando para a coluna *cod_cli* da tabela *cliente* e a outra apontando para a coluna *cod_vend* da tabela *vendedor*. Os comandos para criar essa tabela são os seguintes:



```
CREATE TABLE pedido
(
    cod_ped int IDENTITY(1,1) PRIMARY KEY,
    cod_cli int NOT NULL FOREIGN KEY REFERENCES cliente(cod_cli),
    cod_vend int NOT NULL FOREIGN KEY REFERENCES vendedor(cod_vend),
    dt_ped datetime NOT NULL,
    vl_ped decimal(10,2) NOT NULL
);
```

4.10. Criando Índices (comando INDEX)

Um índice pode ser criado em uma tabela para permitir que a aplicação de banco de dados encontre os dados mais rapidamente e eficientemente, sem a necessidade de ler toda a tabela. Os usuários não vêem os índices, apenas percebem o aumento da velocidade nas suas consultas aos dados. Os índices são criados naquelas colunas onde o usuário (por meio dos programas que ele executa) frequentemente realiza buscas de dados.

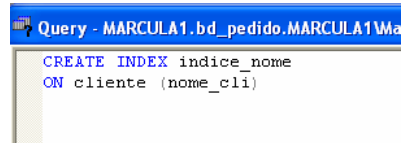
Obs: A atualização dos dados em tabelas com índices leva mais tempo do que em tabelas sem eles (porque os índices também precisam ser atualizados). Dessa forma, somente devemos criar índices em colunas (e tabelas) que serão frequentemente utilizadas em buscas (consultas) de dados.

O comando **CREATE INDEX** é utilizado para criar índices nas tabelas.

```
CREATE INDEX nome_indice
ON nome_tabela (nome_coluna);
```


É possível criar um índice com uma combinação de colunas, listando os seus nomes entre parênteses, após o nome da tabela na qual o índice será criado.

Vamos criar um índice (que receberá o nome *indice_nome*) na coluna *nome_cli* da tabela *cliente*, pois ela será muito utilizada para buscar dados do cliente pelo nome. Os comandos para isso são os seguintes:



```
Query - MARCULA1.bd_pedido.MARCULA1Wa
CREATE INDEX indice_nome
ON cliente (nome_cli)
```

4.11. Excluir um índice (DROP INDEX)

O comando **DROP INDEX** é utilizado para excluir um índice criado em uma tabela de um banco de dados.

DROP INDEX *nome_tabela.nome_indice*

O funcionamento é semelhante ao comando **DROP DATABASE**. Esse comando não tem como ser revertido.

4.12. Excluir uma tabela (DROP TABLE)

O comando **DROP TABLE** é utilizado para excluir uma tabela de um banco de dados:

DROP TABLE *nome_tabela*

O funcionamento é semelhante ao comando **DROP DATABASE**. Esse comando não tem como ser revertido, ou seja, os dados da tabela estarão perdidos.

4.13. Alterando características da tabela ou excluindo objetos da tabela (comando ALTER TABLE)

O comando **ALTER TABLE** é utilizado para adicionar, excluir ou modificar colunas em uma tabela existente, ou para manipular outros objetos criados nas tabelas (por exemplo, índices).

a) Adicionar coluna em uma tabela:

ALTER TABLE *nome_tabela*
ADD *nome_coluna tipo_dado*;

b) Mudar o tipo de dado de uma coluna em uma tabela:

ALTER TABLE *nome_tabela*
ALTER COLUMN *nome_coluna tipo_dado*;

c) Excluir uma coluna em uma tabela:

ALTER TABLE *nome_tabela*
DROP COLUMN *nome_coluna*;

d) Acrescentar uma restrição **PRIMARY KEY** a uma coluna de uma tabela:

```
ALTER TABLE nome_tabela  
ADD PRIMARY KEY (nome_coluna);
```

OU

```
ALTER TABLE nome_tabela  
ADD CONSTRAINT nome_restrição PRIMARY KEY (nome_coluna);
```

e) Acrescentar uma restrição **FOREIGN KEY** a uma coluna de uma tabela:

```
ALTER TABLE nome_tabela_chave_estrangeira  
ADD FOREIGN KEY (nome_coluna_chave_estrangeira)  
REFERENCES nome_tabela_chave_primaria (nome_coluna_chave_primaria);
```

OU

```
ALTER TABLE nome_tabela_chave_estrangeira  
ADD CONSTRAINT nome_restrição  
FOREIGN KEY (nome_coluna_chave_estrangeira)  
REFERENCES nome_tabela_chave_primaria (nome_coluna_chave_primaria);
```

f) Acrescentar uma restrição **CHECK** a uma coluna de uma tabela:

```
ALTER TABLE nome_tabela  
ADD CHECK (condição_verificação);
```

OU

```
ALTER TABLE nome_tabela  
ADD CONSTRAINT nome_restrição CHECK (condição_verificação);
```

g) Excluir restrições **PRIMARY KEY**, **FOREIGN KEY** ou **CHECK** de uma coluna de uma tabela (somente quando a restrição possuir um nome):

```
ALTER TABLE nome_tabela  
DROP CONSTRAINT nome_restrição;
```

h) Acrescentar uma restrição **DEFAULT** a uma coluna de uma tabela:

```
ALTER TABLE nome_tabela  
ALTER COLUMN nome_coluna SET DEFAULT valor_padrão;
```

i) Excluir uma restrição **DEFAULT** a uma coluna de uma tabela:

```
ALTER TABLE nome_tabela  
ALTER COLUMN nome_coluna DROP DEFAULT
```