

# Banco de Dados I

## 05 - Transformação entre os Modelos Conceitual e Relacional

---

Marcos Roberto Ribeiro



Instituto Federal Minas Gerais - Campus Bambuí

2018

# Introdução

- A abordagem ER é voltada para a modelagem de dados de forma independente do SGBD considerado, neste caso construímos o *modelo conceitual* ou *esquema conceitual* (que pode ser representado graficamente pelo *DER*);
- Já a abordagem relacional modela os dados a nível de SGBD relacional, desta maneira teremos o *modelo lógico* ou *esquema lógico*;
- Apesar de ser possível projetar um banco de dados diretamente no modelo lógico, o correto é obtê-lo através de uma transformação sobre o modelo conceitual;
- Outra transformação possível é do modelo lógico para o modelo conceitual, esta transformação é chamada de *engenharia reversa* e é utilizada para se ter um modelo conceitual de um banco de dados existente;
- Um DER pode ser transformado de várias maneira em um esquema lógico, porém vamos apresentar regras a serem seguidas para a obtenção de um modelo de banco de dados otimizado.

# Transformação de DER para Esquema Lógico

A transformação de um modelo ER em um modelo relacional dá-se nos seguintes passos:

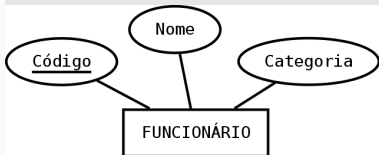
1. Tradução inicial de entidades e respectivos atributos;
2. Tradução de relacionamentos e respectivos atributos;
3. Tradução de generalizações/especializações.

# Tradução Inicial de Entidades

- Este é o passo mais simples da transformação:
  - Cada entidade é traduzida para uma tabela;
  - Cada atributo da entidade define uma coluna desta tabela;
  - Os atributos identificadores da entidade correspondem às colunas que compõem a chave primária da tabela.
- Esta é apenas uma transformação inicial nos próximos passos as tabelas poderão sofrer modificações.

# Exemplo de Transformação

## Entidade Funcionário



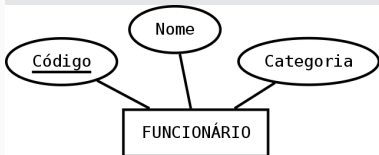
## Tabela funcionario

```
funcionario(  
  codigo_funcionario integer,  
  nome_funcionario varchar(60),  
  categoria varchar(30)  
)
```

- Os nomes de tabelas e atributos devem obedecer às mesmas regras de nomes de variáveis da maioria das linguagens de programação;
- Uma recomendação importante é colocar o nome da tabela no nome da chave primária e na coluna de descrição da tabela, pois muitas vezes estas colunas serão utilizadas em conjunto com colunas de outras tabelas.

# Exemplo de Transformação

## Entidade Funcionário



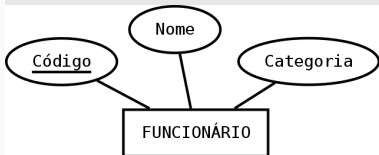
## Tabela funcionario

```
funcionario(  
  codigo_funcionario integer,  
  nome_funcionario varchar(60),  
  categoria varchar(30)  
)
```

- Os nomes de tabelas e atributos devem obedecer às mesmas regras de nomes de variáveis da maioria das linguagens de programação;
- Uma recomendação importante é colocar o nome da tabela no nome da chave primária e na coluna de descrição da tabela, pois muitas vezes estas colunas serão utilizadas em conjunto com colunas de outras tabelas.

# Exemplo de Transformação

## Entidade Funcionário



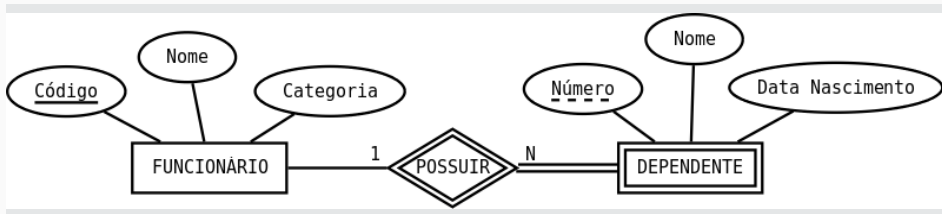
## Tabela funcionario

```
funcionario(  
  codigo_funcionario integer,  
  nome_funcionario varchar(60),  
  categoria varchar(30)  
)
```

- Os nomes de tabelas e atributos devem obedecer às mesmas regras de nomes de variáveis da maioria das linguagens de programação;
- Uma recomendação importante é colocar o nome da tabela no nome da chave primária e na coluna de descrição da tabela, pois muitas vezes estas colunas serão utilizadas em conjunto com colunas de outras tabelas.

# Entidades Fracas

A chave primária da entidade fraca será composta por suas chaves fracas mais os atributos identificadores de sua entidade forte. Por exemplo:

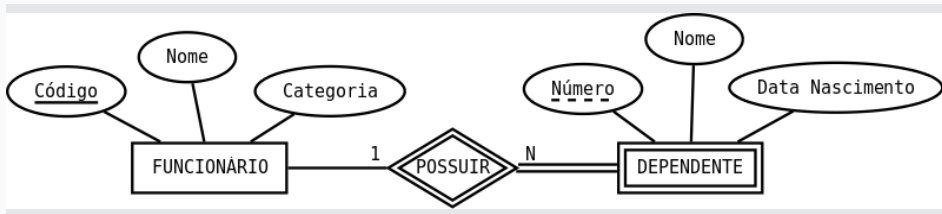


```
funcionario(  
  codigo_funcionario integer, nome_funcionario varchar(60),  
  categoria varchar(30))  
dependente(  
  codigo_funcionario integer, numero_dependente integer,  
  nome_dependente varchar(60), data_nascimento date)
```



# Entidades Fracas

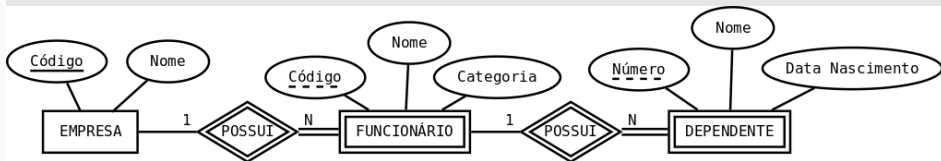
A chave primária da entidade fraca será composta por suas chaves fracas mais os atributos identificadores de sua entidade forte. Por exemplo:



```
funcionario(  
    codigo_funcionario integer, nome_funcionario varchar(60),  
    categoria varchar(30))  
dependente(  
    codigo_funcionario integer, numero_dependente integer,  
    nome_dependente varchar(60), data_nascimento date)
```

# Entidades Fracas Encadeadas

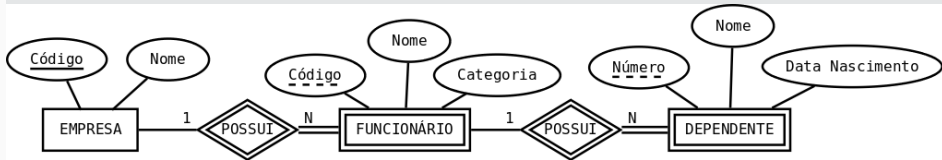
No caso de entidades fracas encadeadas a o atributo da entidade forte se propaga para todas suas as entidades fracas.



```
empresa(  
  codigo_empresa integer, nome_empresa varchar(60),  
funcionario(  
  codigo_empresa integer, codigo_funcionario integer,  
  nome_funcionario varchar(60), categoria varchar(30))  
dependente(  
  codigo_empresa integer, codigo_funcionario integer,  
  numero_dependente integer, nome_dependente varchar(60),  
  data_nascimento date)
```

# Entidades Fracas Encadeadas

No caso de entidades fracas encadeadas a o atributo da entidade forte se propaga para todas suas as entidades fracas.

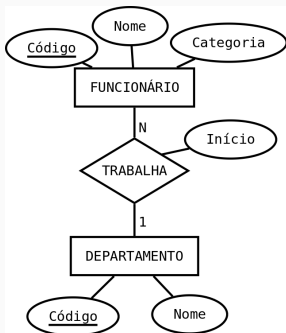


```
empresa(  
  codigo_empresa integer, nome_empresa varchar(60),  
funcionario(  
  codigo_empresa integer, codigo_funcionario integer,  
  nome_funcionario varchar(60), categoria varchar(30))  
dependente(  
  codigo_empresa integer, codigo_funcionario integer,  
  numero_dependente integer, nome_dependente varchar(60),  
  data_nascimento date)
```

- A tradução dos relacionamentos leva em consideração a cardinalidade máxima e as restrições de participação;
- Basicamente existem as seguinte possibilidades de tradução:
  - Adição de colunas;
  - Criação de tabela própria.

# Relacionamentos 1:N

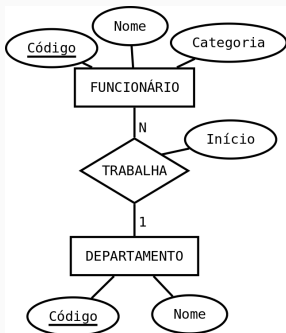
- A tradução de relacionamentos com cardinalidade 1:N é feita com a adição de colunas;
- A chave primária da entidade com cardinalidade 1 aparece como chave estrangeira na tabela gerada pela entidade com cardinalidade N.
- Os atributos do relacionamento acompanham a chave estrangeira. Por exemplo:



```
departamento(  
  codigo_departamento integer,  
  nome_departamento varchar(60))  
funcionario(  
  codigo_funcionario integer,  
  nome_funcionario varchar(60),  
  categoria varchar(30), inicio date  
  *codigo_departamento integer)  
*funcionario.codigo_departamento:  
  departamento:codigo_departamento
```

# Relacionamentos 1:N

- A tradução de relacionamentos com cardinalidade 1:N é feita com a adição de colunas;
- A chave primária da entidade com cardinalidade 1 aparece como chave estrangeira na tabela gerada pela entidade com cardinalidade N.
- Os atributos do relacionamento acompanham a chave estrangeira. Por exemplo:

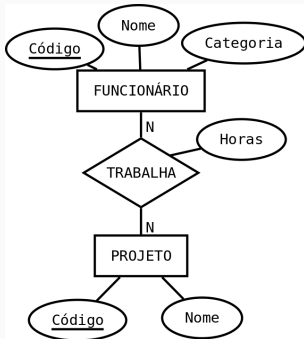


```
departamento(  
  codigo_departamento integer,  
  nome_departamento varchar(60))  
funcionario(  
  codigo_funcionario integer,  
  nome_funcionario varchar(60),  
  categoria varchar(30), inicio date  
  *codigo_departamento integer)  
*funcionario.codigo_departamento:  
  departamento:codigo_departamento
```

## Relacionamentos N:N

- No caso dos relacionamentos com cardinalidade N:N, é necessária a criação de uma tabela própria na tradução;
- A tabela conterá as chaves primárias das entidades participantes transformadas em chaves estrangeiras e os atributos do relacionamento;
- A chave primária da tabela será composta por todas as chaves estrangeiras;
- O nome da tabela criada pode ser o próprio nome do relacionamento ou outro nome mais coerente com o modelo.

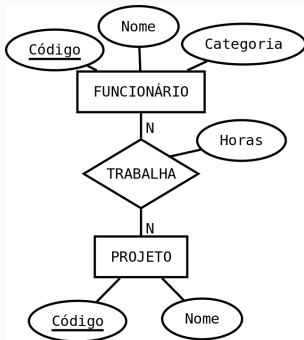
# Exemplo de Tradução de Relacionamento N:N



```
projeto(  
    codigo_projeto integer,  
    nome_projeto varchar(60))  
funcionario(  
    codigo_funcionario integer,  
    nome_funcionario varchar(60),  
    categoria varchar(30))  
funcionario_projeto(  
    codigo_projeto integer,  
    codigo_funcionario integer,  
    horas integer)  
*funcionario_projeto.codigo_projeto:  
    projeto.codigo_projeto  
*funcionario_projeto.codigo_funcionario:  
    funcionario.codigo_funcionario
```



# Exemplo de Tradução de Relacionamento N:N

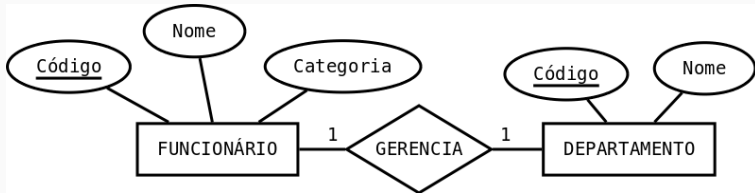


```
projeto(  
    codigo_projeto integer,  
    nome_projeto varchar(60))  
funcionario(  
    codigo_funcionario integer,  
    nome_funcionario varchar(60),  
    categoria varchar(30))  
funcionario_projeto(  
    codigo_projeto integer,  
    codigo_funcionario integer,  
    horas integer)  
*funcionario_projeto.codigo_projeto:  
    projeto:codigo_projeto  
*funcionario_projeto.codigo_funcionario:  
    funcionario.codigo_funcionario
```

# Relacionamentos 1:1

- A tradução dos relacionamentos 1:1 utiliza as mesmas regras da tradução de relacionamentos 1:N;
- Basicamente, deve-se considerar o relacionamento como se possuísse cardinalidade 1:N;
- A principal dificuldade é determinar quais entidades podem ser consideradas com cardinalidade 1 e quais podem ser consideradas com cardinalidade N;
- Em alguns casos é interessante considerar as restrições de participação.
- Se todas ou nenhuma das entidades do relacionamento possuem participação total é preciso usar o bom senso;
- Caso contrário, podemos considerar as entidades com participação total como se tivessem cardinalidade N.

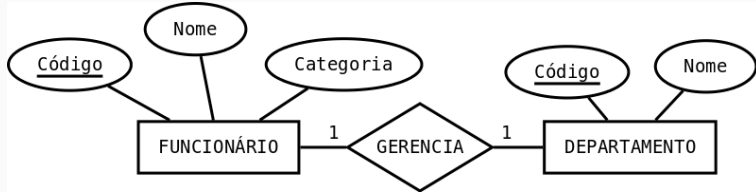
## Exemplo de Tradução de Relacionamento 1:1



```
funcionario(  
    codigo_funcionario integer,  
    nome_funcionario varchar(60),  
    categoria varchar(30))  
departamento(  
    codigo_departamento integer,  
    nome_departamento varchar(30),  
    *codigo_funcionario integer)  
*departamento.codigo_funcionario:  
funcionario.codigo_funcionario
```

```
departamento(  
    codigo_departamento integer,  
    nome_departamento varchar(30))  
funcionario(  
    codigo_funcionario integer,  
    nome_funcionario varchar(60),  
    categoria varchar(30),  
    *codigo_departamento integer)  
*funcionario.codigo_departamento:  
departamento.codigo_departamento
```

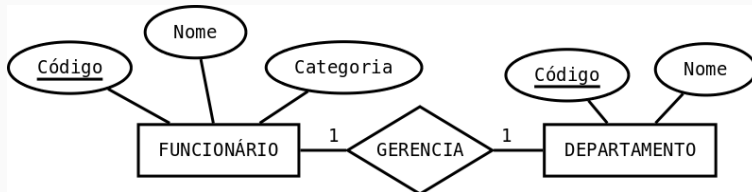
# Exemplo de Tradução de Relacionamento 1:1



```
funcionario(  
    codigo_funcionario integer,  
    nome_funcionario varchar(60),  
    categoria varchar(30))  
departamento(  
    codigo_departamento integer,  
    nome_departamento varchar(30),  
    *codigo_funcionario integer)  
*departamento.codigo_funcionario:  
funcionario.codigo_funcionario
```

```
departamento(  
    codigo_departamento integer,  
    nome_departamento varchar(30))  
funcionario(  
    codigo_funcionario integer,  
    nome_funcionario varchar(60),  
    categoria varchar(30),  
    *codigo_departamento integer)  
*funcionario.codigo_departamento:  
departamento.codigo_departamento
```

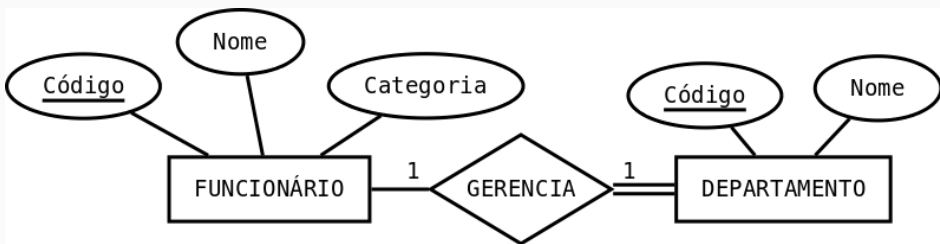
## Exemplo de Tradução de Relacionamento 1:1



```
funcionario(  
    codigo_funcionario integer,  
    nome_funcionario varchar(60),  
    categoria varchar(30))  
departamento(  
    codigo_departamento integer,  
    nome_departamento varchar(30),  
    *codigo_funcionario integer)  
*departamento.codigo_funcionario:  
funcionario.codigo_funcionario
```

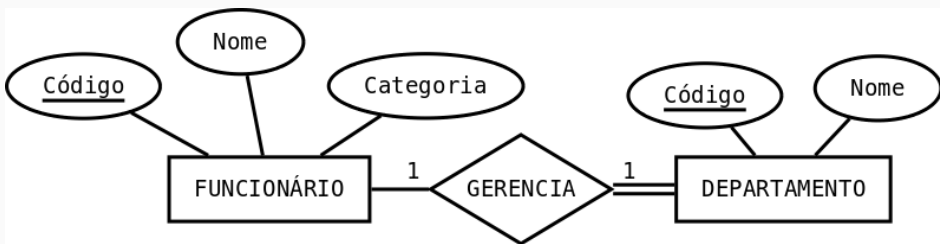
```
departamento(  
    codigo_departamento integer,  
    nome_departamento varchar(30))  
funcionario(  
    codigo_funcionario integer,  
    nome_funcionario varchar(60),  
    categoria varchar(30),  
    *codigo_departamento integer)  
*funcionario.codigo_departamento:  
departamento.codigo_departamento
```

## Exemplo de Tradução de Relacionamento 1:1 com Participação Total



```
funcionario(  
    codigo_funcionario integer, nome_funcionario varchar(60),  
    categoria varchar(30))  
departamento(  
    codigo_departamento integer, nome_departamento varchar(30),  
    *codigo_funcionario integer)  
*departamento.codigo_funcionario:  
    funcionario.codigo_funcionario
```

## Exemplo de Tradução de Relacionamento 1:1 com Participação Total



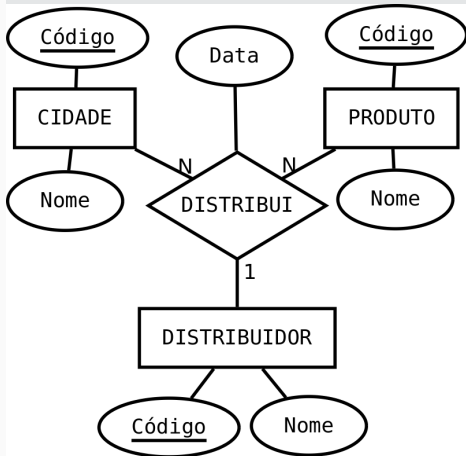
```
funcionario(  
    codigo_funcionario integer, nome_funcionario varchar(60),  
    categoria varchar(30))  
departamento(  
    codigo_departamento integer, nome_departamento varchar(30),  
    *codigo_funcionario integer)  
*departamento.codigo_funcionario:  
    funcionario.codigo_funcionario
```

## Relacionamentos com Grau Maior que 2

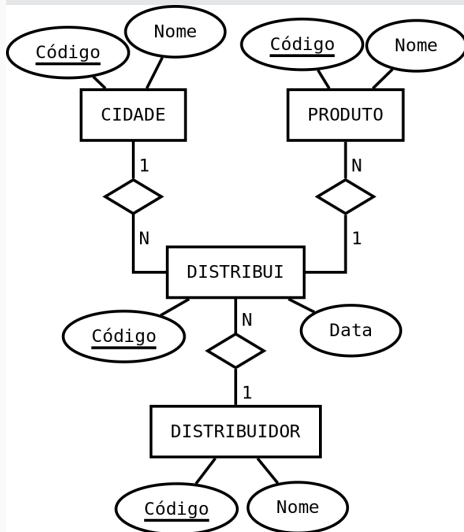
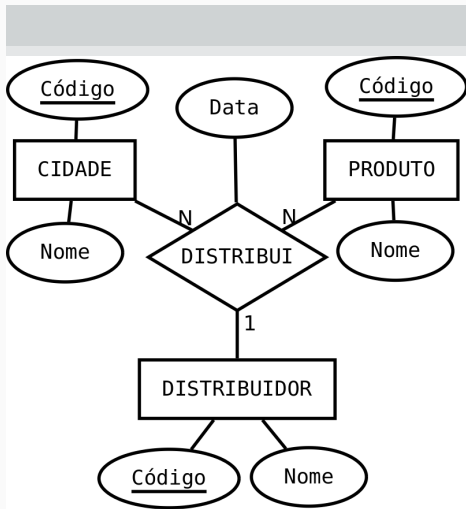
- As regras apresentadas até este ponto, aplicam-se somente à relacionamentos binários. Para relacionamentos de grau maior que dois a transformação dá-se da seguinte maneira:
  1. O relacionamento é transformado em uma entidade. Esta nova entidade é ligada através de relacionamentos binários a cada uma das entidades que participavam do relacionamentos original;
  2. As regras de implementação de entidades e relacionamentos binários apresentadas anteriormente são aplicadas às entidades e aos relacionamentos binários assim criados.



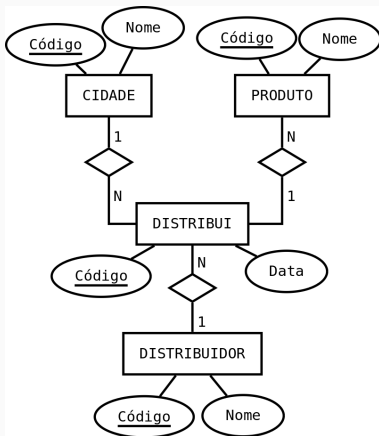
# Exemplo



# Exemplo

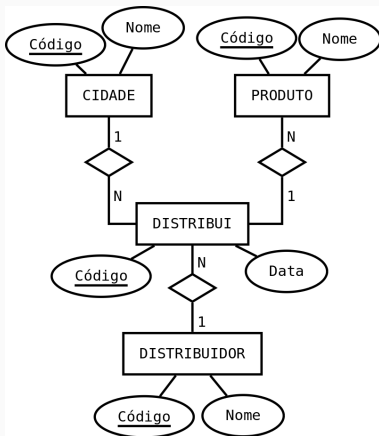


# Exemplo



```
cidade(codigo_cidade integer,  
      nome_cidade varchar(50))  
produto(codigo_produto integer,  
        nome_produto varchar(50))  
distribuidor(  
  codigo_distribuidor integer,  
  nome_distribuidor varchar(50))  
distribui(codigo integer,  
          data date,  
          *codigo_cidade integer,  
          *codigo_produto integer,  
          *codigo_distribuidor integer)  
*distribui.codigo_cidade:  
  cidade.codigo_cidade  
*distribui.codigo_produto:  
  produto.codigo_produto  
*distribui.codigo_distribuidor:  
  distribuidor.codigo_distribuidor
```

# Exemplo

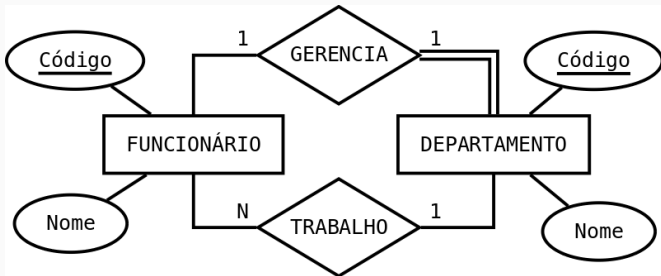


```
cidade(codigo_cidade integer,  
      nome_cidade varchar(50))  
produto(codigo_produto integer,  
        nome_produto varchar(50))  
distribuidor(  
  codigo_distribuidor integer,  
  nome_distribuidor varchar(50))  
distribui(codigo integer,  
          data date,  
          *codigo_cidade integer,  
          *codigo_produto integer,  
          *codigo_distribuidor integer)  
*distribui.codigo_cidade:  
  cidade.codigo_cidade  
*distribui.codigo_produto:  
  produto.codigo_produto  
*distribui.codigo_distribuidor:  
  distribuidor.codigo_distribuidor
```

## Participação Total

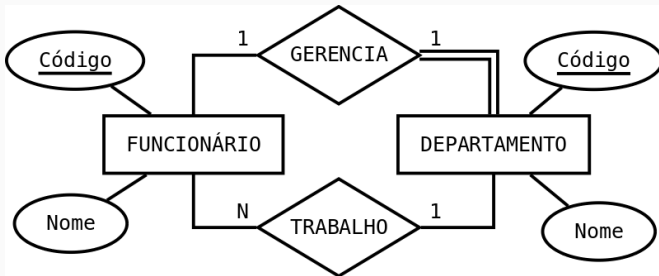
- Até o momento não nos preocupamos com a obrigatoriedade das colunas, mas isto deve constar no modelo relacional;
- Para isto deve ser observado se a entidade possui participação total no relacionamento, neste caso a chave estrangeira criada na tabela deverá ser obrigatória.

## Exemplo



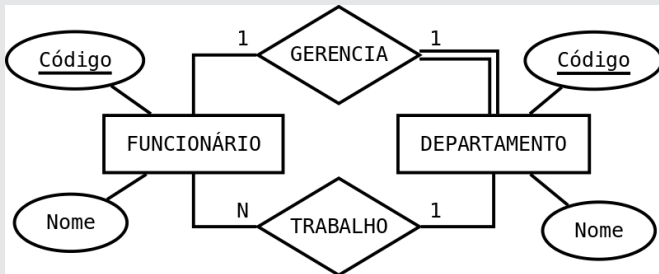
```
funcionario(+codigo_funcionario integer,  
            +nome_funcionario varchar(60),*codigo_departamento integer)  
*funcionario.codigo_departamento:  
    departamento.codigo_departamento  
departamento(+codigo_departamento integer,  
              +nome_departamento varchar(30),+*codigo_funcionario integer)  
*departamento.codigo_funcionario:  
    funcionario.codigo_funcionario
```

## Exemplo



```
funcionario(+codigo_funcionario integer,  
            +nome_funcionario varchar(60),*codigo_departamento integer)  
*funcionario.codigo_departamento:  
    departamento.codigo_departamento  
departamento(+codigo_departamento integer,  
               +nome_departamento varchar(30),+*codigo_funcionario integer)  
*departamento.codigo_funcionario:  
    funcionario.codigo_funcionario
```

## Exemplo



Qual seria o problema se a entidade **FUNCIONÁRIO** tivesse participação total no relacionamento **TRABALHO**?

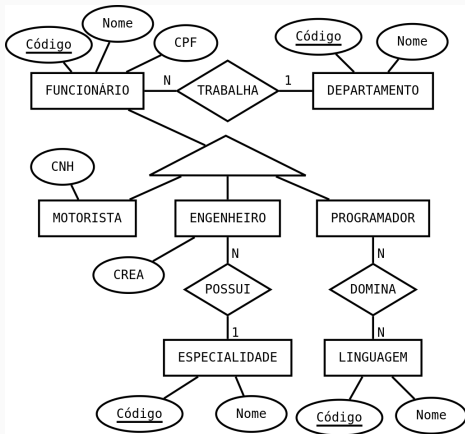


- Para a implementação de hierarquias de generalização e especialização na abordagem relacional, há duas alternativas:
  - Uso de uma tabela para cada entidade;
  - Uso de uma única tabela para toda hierarquia.

# Uma Tabela por Hierarquia

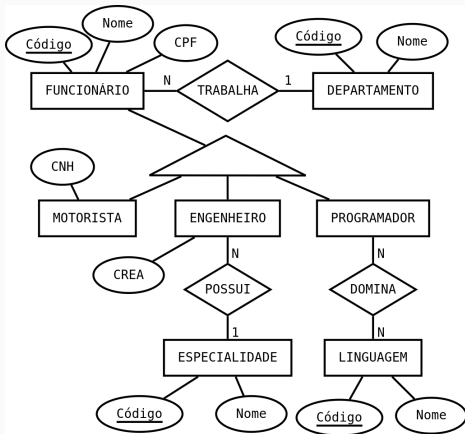
- Nesta alternativa, todas tabelas referentes às especializações de uma entidade genérica são fundidas em uma única tabela. Esta tabela terá:
  - Chave primária para entidade mais genérica;
  - Caso não exista, uma coluna **Tipo**, que identifica o tipo de entidade que está sendo representada por cada linha da tabela;
  - Uma coluna para cada atributo da entidade genérica;
  - Colunas referentes aos relacionamentos dos quais participa a entidade genérica e que sejam implementados através da alternativa de adicionar colunas à tabela da entidade genérica;
  - Uma coluna para cada atributo de cada entidade especializada, *estas colunas devem ser opcionais*;
  - Colunas *opcionais* referentes aos relacionamentos dos quais participa cada entidade especializada e que sejam implementados através da alternativa de adicionar colunas à tabela da entidade.
- Uma entidade especializada pode não gerar nenhuma coluna, caso não tenha atributos e seus relacionamentos sejam implementados através de tabelas próprias.

# Exemplo



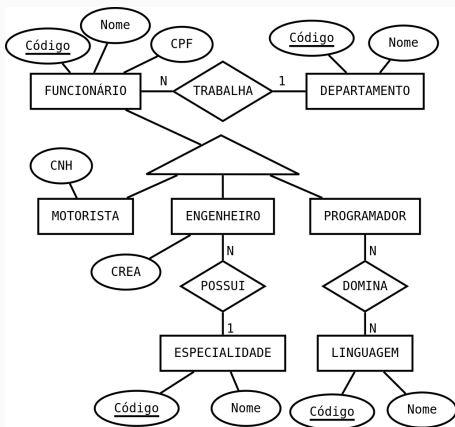
```
departamento(  
    +codigo_departamento integer,  
    +nome_departamento varchar(30))  
especialidade(  
    +codigo_especialidade integer,  
    +nome_especialidade varchar(30))  
linguagem(  
    +codigo_linguagem integer,  
    +nome_linguagem varchar(30))
```

# Exemplo



```
departamento(  
    +codigo_departamento integer,  
    +nome_departamento varchar(30))  
especialidade(  
    +codigo_especialidade integer,  
    +nome_especialidade varchar(30))  
linguagem(  
    +codigo_linguagem integer,  
    +nome_linguagem varchar(30))
```

# Exemplo

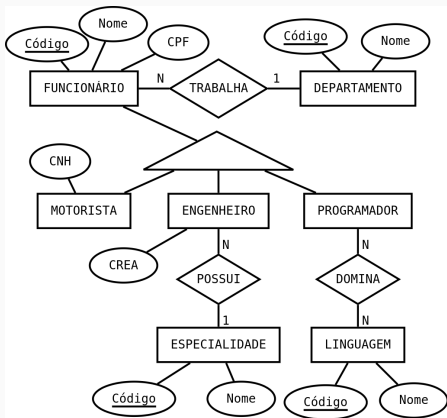


```
funcionario(  
    +codigo_funcionario integer,  
    +nome_funcionario varchar(60),  
    cpf char(11),  
    *codigo_departamento integer,  
    +tipo char(1),  
    cnh varchar(11),  
    crea varchar(10),  
    *codigo_especialidade integer,  
)  
  
*funcionario.codigo_departamento:  
    departamento.codigo_departamento  
*funcionario.codigo_especialidade:  
    especialidade.codigo_especialidade  
domina(+*codigo_linguagem integer,  
    +*codigo_funcionario integer)  
*domina.codigo_linguagem:  
    linguagem.codigo_linguagem  
*domina.codigo_funcionario:  
    funcionario.codigo_funcionario
```

## Uma Tabela por Entidade Especializada

- Nesta implementação deve-se criar uma tabela para cada entidade que compõe a hierarquia, aplicando as regras correspondentes à implementação de entidades e relacionamentos já apresentadas;
- O único acréscimo é a inclusão da chave primária da tabela correspondente à entidade genérica, em cada tabela correspondente a uma entidade especializada.

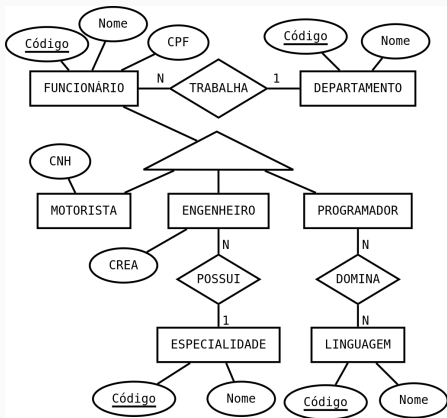
# Exemplo



```
departamento(  
    +codigo_departamento integer,  
    +nome_departamento varchar(30))  
especialidade(  
    +codigo_especialidade integer,  
    +nome_especialidade varchar(30))  
linguagem(  
    +codigo_linguagem integer,  
    +nome_linguagem varchar(30))  
funcionario(  
    +codigo_funcionario integer,  
    +nome_funcionario varchar(60),  
    cpf char(11), +tipo char(1),  
    *codigo_departamento integer)  
*funcionario.codigo_departamento:  
departamento.codigo_departamento
```

E a coluna **tipo**?

# Exemplo

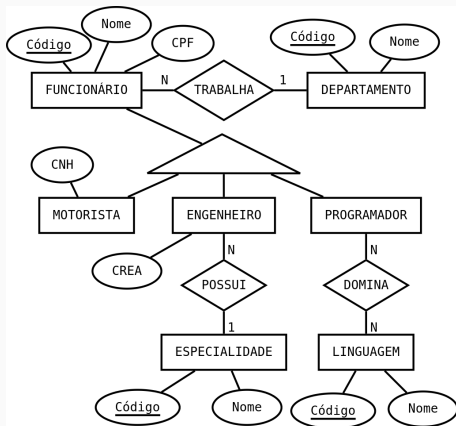


```
departamento(  
    +codigo_departamento integer,  
    +nome_departamento varchar(30))  
especialidade(  
    +codigo_especialidade integer,  
    +nome_especialidade varchar(30))  
linguagem(  
    +codigo_linguagem integer,  
    +nome_linguagem varchar(30))  
funcionario(  
    +codigo_funcionario integer,  
    +nome_funcionario varchar(60),  
    cpf char(11), +tipo char(1),  
    *codigo_departamento integer)  
*funcionario.codigo_departamento:  
departamento.codigo_departamento
```

E a coluna **tipo**?



# Exemplo



```
motorista(  
    +*codigo_funcionario integer,  
    cnh varchar(11))  
*motorista.codigo_funcionario:  
    funcionario.codigo_funcionario  
engenheiro(  
    +*codigo_funcionario integer,  
    crea varchar(10),  
    *codigo_especialidade integer)  
*engenheiro.codigo_funcionario:  
    funcionario.codigo_funcionario  
*engenheiro.codigo_especialidade:  
    especialidade.codigo_especialidade  
domina(+*codigo_linguagem integer,  
    +*codigo_funcionario integer)  
*domina.codigo_linguagem:  
    linguagem.codigo_linguagem  
*domina.codigo_funcionario:  
    funcionario.codigo_funcionario
```

# Comparação entre as Duas Alternativas

## Vantagem da implementação com tabela única

- Todos os dados da entidade genérica e os dados de suas especializações, estão em uma única linha. Não há necessidade de junções quando desejamos obter dados da entidade genérica juntamente a entidade especializada.

## Vantagem da implementação com uma tabela por entidade especializada

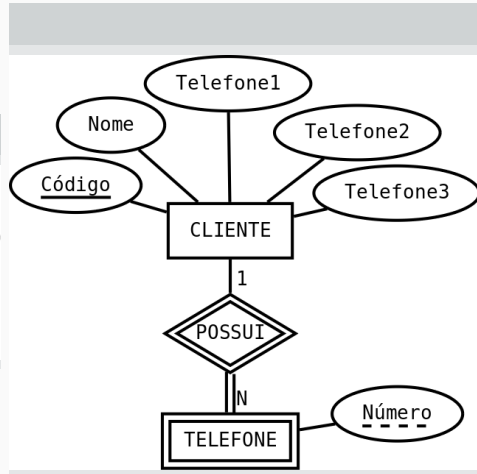
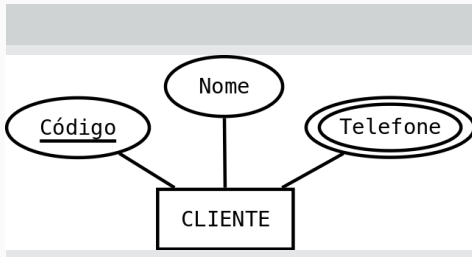
- Redução do número de colunas opcionais e controle das especializações via aplicação.

# Simulação de Atributos Multi-Valorados

- Nas aulas anteriores já discutimos que atributos multi-valorados não são desejáveis em DER;
- Contudo na permanência deste tipo de atributo há duas alternativas para a transformação no esquema lógico:
  - Transformar o atributo multi-valorado em vários atributos mono-valorados;
  - Criar um entidade para o atributo multi-valorado.

# Atributos Multi-Valorados como Entidade

- Abaixo temos um exemplo de uma transformação de atributo multi valorado em entidade:



# Atributos Multi-Valorados como Entidade

Entretanto, esta implementação pode trazer problemas de performance, tais como:

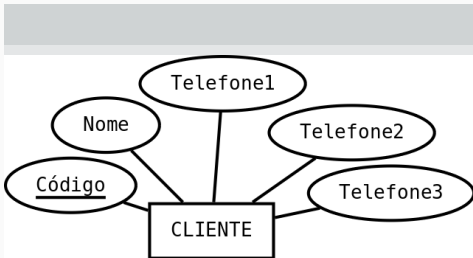
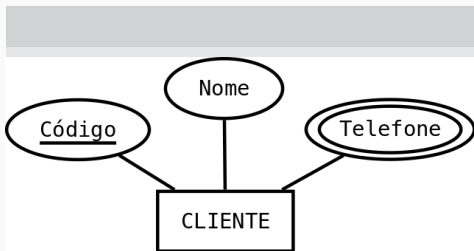
- São raros os clientes que possuem mais que três telefones. Quando isso ocorrer, é suficiente armazenarmos apenas três números;
- Não há consultas ao banco de dados usando o número de telefone como critério de seleção. Os números de telefone são apenas exibidos ou impressos juntos às demais informações de cliente.

Nesta situação, por uma questão de performance, pode ser mais viável manter uma tabela desnormalizada<sup>1</sup>.

---

<sup>1</sup>Posteriormente veremos o conceito de normalização

# Transformar Atributo Multi-Valorado em Atributos Mono-Valorados I



- Nesta implementação, optou-se por simular uma coluna multi-valorada através da criação das colunas Telefones1, Telefones2 e Telefones3;
- Essa alternativa permite que os telefones de um cliente sejam obtidos mais rapidamente, já que encontram-se todos dentro da mesma linha da tabela;

## Transformar Atributo Multi-Valorado em Atributos Mono-Valorados II

- Além disso, implica em menos espaço ocupado, já que o espaço necessário à implementação da chave primária da tabela Telefone, na primeira alternativa, é considerável;
- O inconveniente que esta alternativa apresenta do ponto de vista funcional é que uma eventual consulta usando o número de telefone como critério de busca torna-se mais complicada, já que devem ser referenciados todos os nomes das colunas referentes ao atributo multi-valorado;

# Engenharia Reversa de Modelos Relacionais I

- A engenharia reversa parte de um modelo de implementação e constrói um modelo conceitual que descreve abstratamente a implementação em questão;
- No caso de banco de dados, fala-se de engenharia reversa, quando transforma-se modelos de banco de dados mais ricos em detalhes de implementação em modelos de dados mais abstratos;
- Um caso específico de engenharia reversa de banco de dados é o da engenharia reversa de modelos relacionais. Neste tipo de engenharia reversa, tem-se, como ponto de partida, um modelo lógico de um banco de dados relacional e, como resultado, um modelo conceitual na abordagem ER;



## Engenharia Reversa de Modelos Relacionais II

- A engenharia reversa de modelos relacionais pode ser útil quando não se tem um modelo conceitual para um banco de dados existente. Isso pode acontecer quando o banco de dados foi desenvolvida de forma empírica, sem o uso de uma metodologia de desenvolvimento, ou quando o esquema do banco de dados sofreu modificações ao longo do tempo, sem que as mesmas tenham sido registradas no modelo conceitual.
- O processo de engenharia reversa de um modelo relacional consta dos seguintes passos:
  1. Identificação da construção ER correspondente a cada tabela;
  2. Definição de relacionamentos 1:n e 1:1;
  3. Definição de atributos;
  4. Definição de identificadores de entidades e relacionamentos.

## Exemplo I

```
disciplina(codigo_disciplina integer, nome_disciplina varchar(50))
curso(codigo_curso integer, nome_curso varchar(50))
matriz(*codigo_curso integer, *codigo_disciplina integer,
      obrigatoria char(1))
  *matriz.codigo_curso: curso.codigo_curso
  *matriz.codigo_disciplina: disciplina.codigo_disciplina
sala(*codigo_predio integer, codigo_sala integer,
     capacidade integer)
  *sala.codigo_predio: predio.codigo_predio
predio(codigo_predio integer, local varchar(30))
turma (ano integer, semestre integer, sigla_turma varchar(5),
      *codigo_predio integer, *codigo_sala integer)
  *turma.codigo_disciplina: disciplina.codigo_disciplina
  *turma.(codigo_predio,codigo_sala):
    sala.(codigo_predio,codigo_sala)
```

## Exemplo II

```
turma_disciplina(ano integer, semestre integer,  
                 sigla_turma varchar(5), *codigo_disciplina integer)  
*turma_disciplina.(ano, semestre, sigla_turma):  
    turma.(ano, semestre, sigla_turma)  
*turma_disciplina.codigo_disciplina:  
    disciplina.codigo_disciplina  
tecnico(codigo_tecnico integer, nome_tecnico varchar(50))  
laboratorio (*codigo_predio integer, *codigo_sala integer,  
            equipamento varchar(50), *codigo_tecnico integer)  
*laboratorio.codigo_tecnico: tecnico.codigo_tecnico  
*laboratorio.(codigo_predio, codigo_sala):  
    sala.(codigo_predio, codigo_sala)
```

## Identificação de Elementos do DER

- Na primeira etapa da engenharia reversa de um banco de dados relacional, define-se, para cada tabela do modelo relacional qual seu elemento correspondente no DER;
- Uma tabela pode corresponder a:
  - Uma entidade;
  - Um relacionamento N:N;
  - Uma entidade especializada.
- O fator determinante da construção ER que corresponde a uma tabela é a composição de sua chave primária. Tabelas podem ser classificadas em três tipos de acordo com sua chave primária:
  - Chave primária composta por mais de uma chave estrangeira;
  - Toda chave primária é uma chave estrangeira;
  - Outros Casos.

## Chave primária composta por mais de uma chave estrangeira

- A tabela que possui uma chave primária composta de múltiplas chaves estrangeiras implementa um relacionamento N:N entre as entidades correspondentes às tabelas referenciadas pelas chaves estrangeiras;
- Um exemplo de tabela deste tipo é a tabela **matriz** que tem como chave primária **codigo\_curso** e **codigo\_disciplina**;
- Ambas colunas são chave estrangeira em relação às tabelas **curso** e **disciplina**;
- Portanto, a tabela **matriz** representa um relacionamento entre as entidades **curso** e **disciplina**.

## Toda chave primária é uma chave estrangeira

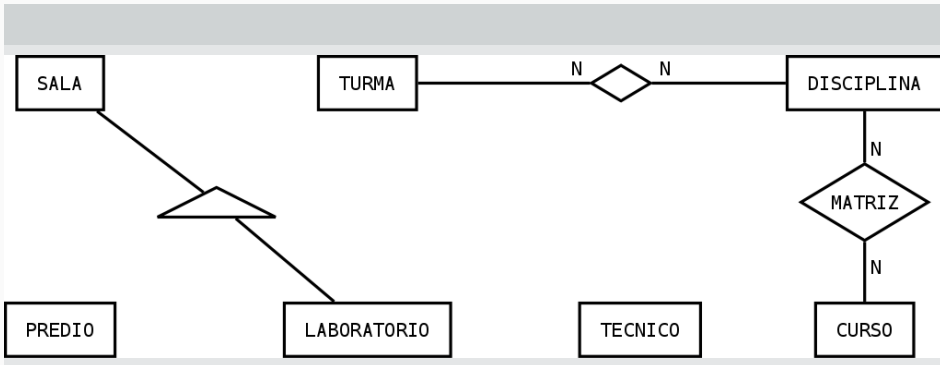
- A tabela cuja chave primária é toda ela uma chave estrangeira representa uma entidade que forma uma especialização da entidade correspondente à tabela referenciada pela chave estrangeira;
- Um exemplo de tabela deste tipo é a tabela **laboratorio** que possui como chave primária as colunas **codigo\_predio** e **codigo\_sala** as quais são chave estrangeira da tabela **sala**;
- A restrição de integridade referencial em questão especifica que uma linha na tabela **laboratorio** somente existe, quando uma linha com a mesma chave existir na tabela **sala**;
- A nível de modelo ER, isso significa que uma ocorrência da entidade laboratório somente pode existir quando a correspondente ocorrência da entidade sala existe, ou seja, significa que a entidade laboratório é uma especialização de sala.

## Outros Casos

- Quando a chave primária da tabela não for composta de múltiplas chaves estrangeiras, nem for toda uma chave estrangeira, a tabela representa uma entidade;
- Exemplificando, a tabela **curso**, cuja chave primária, a coluna **codigo\_curso** não contém chaves estrangeiras, representa uma entidade;
- Da mesma forma, a tabela **sala** também representa uma entidade, sua chave primária (colunas **codigo\_predio** e **codigo\_sala**) contém apenas uma chave estrangeira (coluna **codigo\_predio**);
- O mesmo é válido para as tabelas **disciplina**, **predio** e **turma**.

# DER Inicial

- Até o momento temos o seguinte DER:



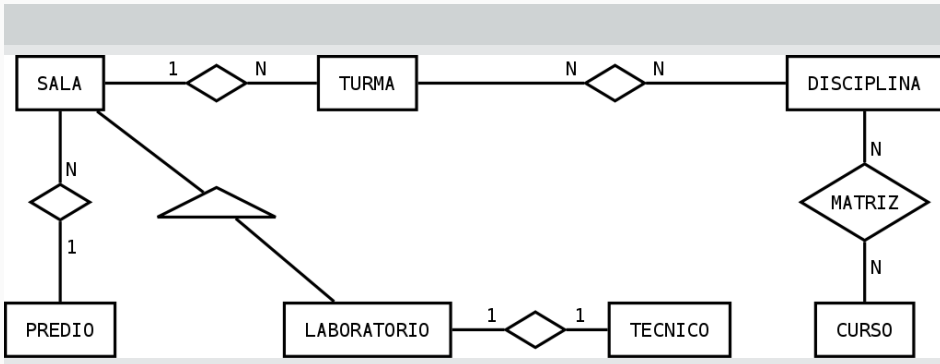


## Identificação de Relacionamentos 1:N e 1:1

- Toda chave estrangeira que não se enquadra nas regras anteriores, ou seja, toda chave estrangeira que não faz parte de uma chave primária composta por múltiplas chaves estrangeiras, nem é toda ela uma chave primária, representa um relacionamento 1:N ou 1:1;
- Em outros termos, toda chave estrangeira que não corresponde a um relacionamento N:N, nem a uma entidade especializada representa um relacionamento 1:N ou 1:1;
- A regra não permite definir se a cardinalidade do relacionamento é 1:N ou 1:1;
- Para definir qual dos dois tipos de relacionamentos está sendo representado pela chave estrangeira, é necessário verificar se há restrições de unicidade sobre a chave estrangeira, se afirmativo o relacionamento será 1:1.

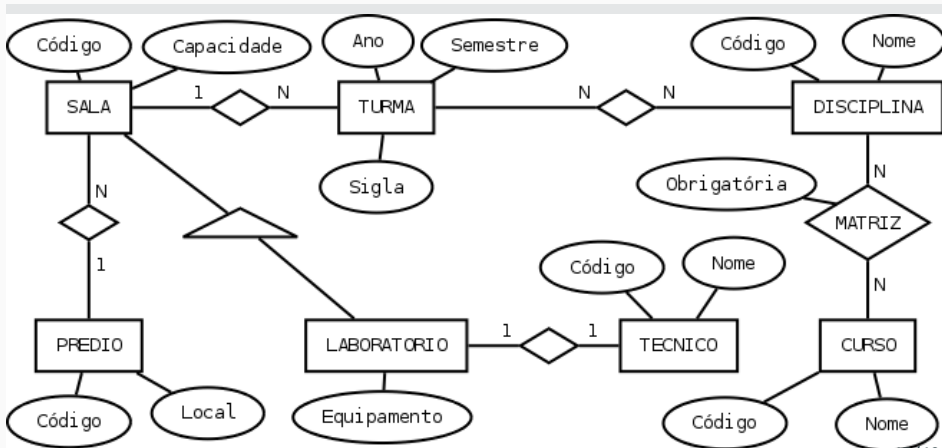
# Identificação de Relacionamentos 1:N e 1:1

- Em nosso exemplo temos o seguinte:



# Definição de Atributos

- Para cada coluna de uma tabela, que não seja chave estrangeira, é definido um atributo na entidade ou relacionamento correspondente;
- As colunas chave estrangeira não correspondem a atributos no diagrama ER, mas sim a relacionamentos, e por isso já foram tratadas.

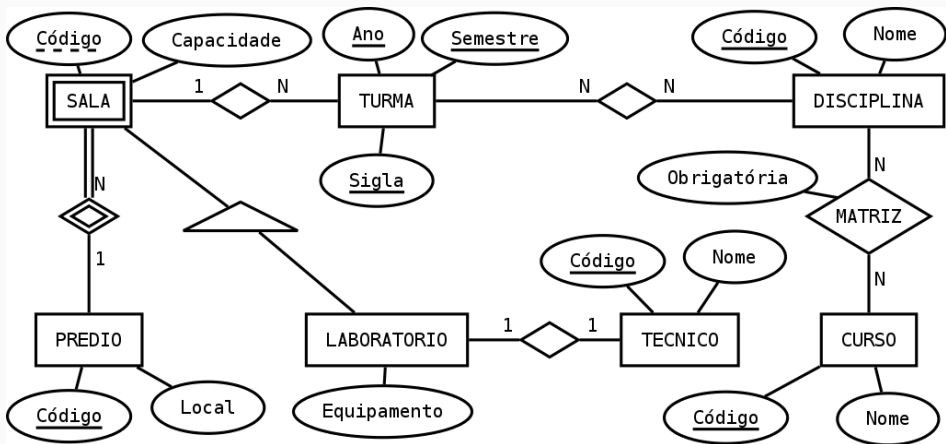





# Identificadores das Entidades

A regra para definição dos identificadores é a seguinte:

- *Coluna da chave primária que não é chave estrangeira* Toda coluna que faz parte da chave primária e que não é chave estrangeira corresponde a um atributo identificador;
- *Coluna da chave primária que é chave estrangeira* Toda coluna que faz parte da chave primária e que é chave estrangeira corresponde a um identificador externo da entidade. Com isto a entidade será uma entidade fraca, por exemplo, a coluna **codigo\_predio**, que é parte da chave primária da tabela **sala** é também chave estrangeira em relação a tabela **predio**. Portanto, a entidade **sala** é uma entidade fraca identificada pelo relacionamento com **predio**.

# DER Final



-  Elmasri, R. and Navathe, S. B. (2011).  
***Sistemas de banco de dados.***  
Pearson Addison Wesley, São Paulo, 6 edition.
-  Heuser, C. A. (2009).  
***Projeto de banco de dados.***  
Bookman, Porto Alegre, 6 edition.
-  Ramakrishnan, R. and Gehrke, J. (2008).  
***Sistemas de gerenciamento de banco de dados.***  
McGrawHill, São Paulo, 3 edition.