

ODMG - *Object Database Management Group*

- (Tentativa de) Padronização para modelo de dados e acesso para SGBDOO
- Consórcio de pesquisadores e fabricantes
- Componentes principais do padrão
 - modelo de objetos
 - linguagem de definição de dados (ODL)
 - linguagem de consulta (OQL)

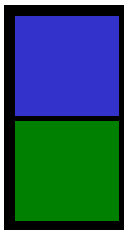
Modelo de Objetos

- Características
 - Formas de definição de dados (ODL)
 - Interfaces
 - define apenas comportamento (assinatura)
 - não possui instâncias
 - Classes
 - define atributos, relacionamentos e comportamento
 - possui instâncias (com OID – chamados **objetos**)
 - implementação do comportamento em alguma LPOO
 - Herança
 - OID e chave
- Exemplo de definição de interface

```
interface Pessoa {  
    short idade();  
    boolean ehMenor();  
    ...  
};
```

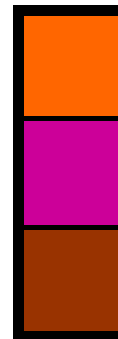
Definição de Classe em ODL

```
class Departamento (extent Departamentos) {  
  attribute string nome;  
  attribute short código;  
  attribute struct Endereço{  
    string rua, short número,  
    string cidade} localização;  
  attribute struct atendimento{  
    time horaInício, time hora Término} horário;  
  attribute Empregado chefe;  
  relationship set<Empregado> empregados  
    inverse Empregado:: depto;  
  void adicionaEmp(short RG) raises  
    (jahTrabalha, RGInexistente);  
  ...  
};
```



conjunto de instâncias

atributo atômico



atributo estruturado

atributo de referência a objeto

relacionamento

Herança

- Duas formas
 - herança */S-A*
 - herança *extends*
- Herança */S-A*
 - herança de interface
 - pode ser *interface*←*interface* ou *interface*←*classe*
 - permite herança múltipla
- Herança *extends*
 - herança de propriedades
 - ocorre somente entre classes (*classe*←*classe*)
 - não permite herança múltipla

Herança em ODL

```
Classe Empregado (extent Empregados) {  
    attribute short RG;  
    attribute string nome;  
    attribute enum gênero{M,F} sexo;  
    attribute Date DN;  
    attribute float salário;  
    relationship Departamento depto  
        inverse Departamento:: empregados;  
    ...  
};
```

herança extends

```
Classe Professor (extent Professores)  
    extends Empregado : Pessoa ← { herança IS-A  
    attribute string titulação;  
    attribute string areaAtuação;  
    ...  
};
```

OID e Chave

- **OID**
 - identificador do objeto
- **Chave**
 - uma ou mais propriedades cujos valores devem ser únicos

```
class Departamentos
(extent todosDeptos
  key código)
{
  attribute string nome;
  attribute short código;
  ...
};
```

```
class Cidades
(key (estado,nome))
{
  attribute string estado;
  attribute string nome;
  ...
}
```

OQL

- Linguagem de consulta declarativa
 - violação de encapsulamento
 - maior flexibilidade para formulação de consultas
- Não há suporte para operações de atualização de dados (I,A,E)
 - métodos devem ser implementados
- Extensão da linguagem SQL com suporte ao tratamento de
 - objetos complexos
 - junções por valor ou por OID
 - invocação de métodos
 - buscas em hierarquias de herança


Consultas e Resultados

- Ponto de partida de uma consulta

- extensão de uma classe (*extent*)

```
select e.*  
from e in Empregados
```

variável de iteração



- Resultados de consultas

- atributos de objetos e/ou novas estruturas

```
select struct (  
    nome:          d.nome  
    empsRicos:     (select e.*  
                    from e in d.empregados  
                    where e.salário > 5000) )  
from d in Departamentos
```


Expressões de Caminho

- Permitem a navegação em estruturas complexas e objetos associados
 - objetos associados
 - atributos de referência e relacionamentos
 - utiliza-se a **notação de ponto** (“.”)
- Exemplo

```
select p.nome, p.titulação
from p in Professores
where p.depto.código = 'INE'
```

Expressões de Caminho

- **Variáveis de iteração** são definidas para a navegação em coleções de objetos referenciados (referências 1:N)
 - a variável de iteração associa-se com cada elemento da coleção referenciada
- Exemplo

```
select e.nome  
from d in Departamentos, e in d.empregados  
where d.código = 'INE'  
and e.salário > 5000
```

Junções

- Junções entre conjuntos de objetos são permitidas, como em BDRs
- Junções tanto por valor quanto por OID são permitidas
- Exemplo

```
select e2.nome  
from e1 in Empregados, e2 in Empregados  
where e1.nome = 'Pedro Campos Souza'  
and e2.salário = e1.salário ← junção por valor  
and e2.depto = e1.depto ← junção por OID
```

Invocação de Métodos

- Métodos podem ser declarados em consultas da mesma forma que atributos
- Exemplos

```
select e.nome  
from e in Empregados  
where e.idade > 50
```

```
select d.código, d.nroHorasAtendimento  
from d in Departamentos
```

Consultas em Hierarquias de Classes

- Consultas aplicadas a uma classe processam automaticamente objetos da classe e de suas subclasses
- Restrições sobre subclasses alvo podem ser especificadas
- Exemplo

```
select (Professores, Pesquisadores) e.nome  
from e in Empregados  
where e.salário > 3000
```

BD Objeto-Relacional - Motivação

- SGBDs Relacionais (SGBDRs)
 - sistemas já consolidados no mercado
 - boa performance
 - muitos anos de pesquisa e aprimoramento
 - eficiência: otimização de consultas, gerenciamento de transações
 - não atendem adequadamente os requisitos de dados de novas categorias de aplicações

BD Objeto-Relacional - Motivação

- SGBDs Orientado a Objetos (SGBDOO)
 - modelo de dados mais rico
 - adequado ao mercado de aplicações não-convencionais
 - pior desempenho, se comparado com SGBDR
 - heterogeneidade a nível de modelo e de capacidades de consulta e atualização
- SGBDs Objeto-Relacional (SGBDOR)
 - combina as melhores características do modelo de objetos no modelo relacional
 - modelo rico + eficiência no gerenciamento de dados
 - tecnologia presente em vários SGBDRs
 - exemplos: Oracle, DB2, Postgres

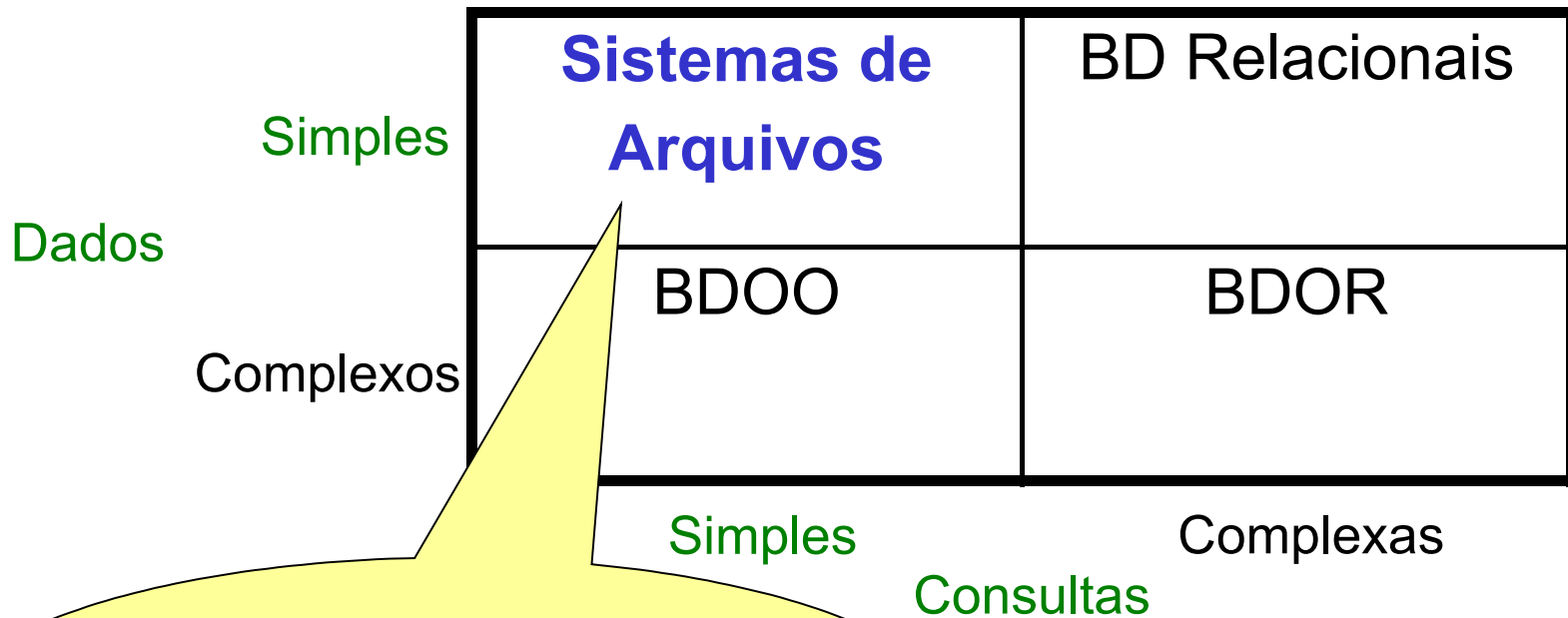
Classificação de *Stonebreaker*

- Pai da tecnologia OR (1997)
- Classifica os principais sistemas gerenciadores de dados em 4 quadrantes

Dados	Simples	Sistemas de Arquivos	BD Relacionais
	Complexos	BDOO	BDOR
		Simples	Complexas
		Consultas	

Classificação de *Stonebreaker*

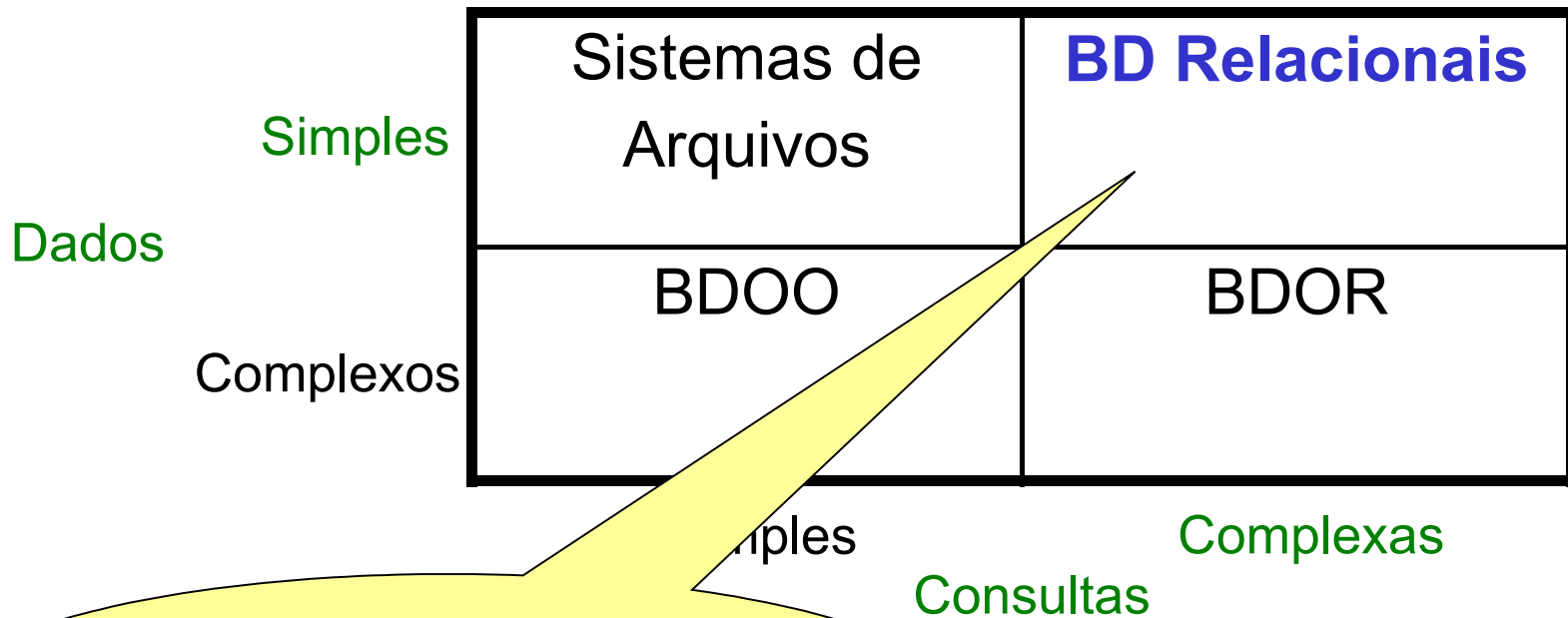
- Quadrantes de tipos de gerenciadores de dados



- dados são registros de tamanho fixo
- poucas consultas predefinidas, em geral buscas por igualdade de campos dos registros

Classificação de *Stonebreaker*

- Quadrantes de tipos de gerenciadores de dados



- dados são linhas de tabelas cujos atributos possuem domínios simples
- flexibilidade de consultas com SQL

Classificação de *Stonebreaker*

- Quadrantes de tipos de gerenciadores de dados



- dados são objetos com estrutura complexa
- capacidade de consulta limitada, baseada em navegação por objetos (poucos usam todos os recursos da OQL, nem todas as cláusulas da SQL estão presentes)

Classificação de *Stonebreaker*

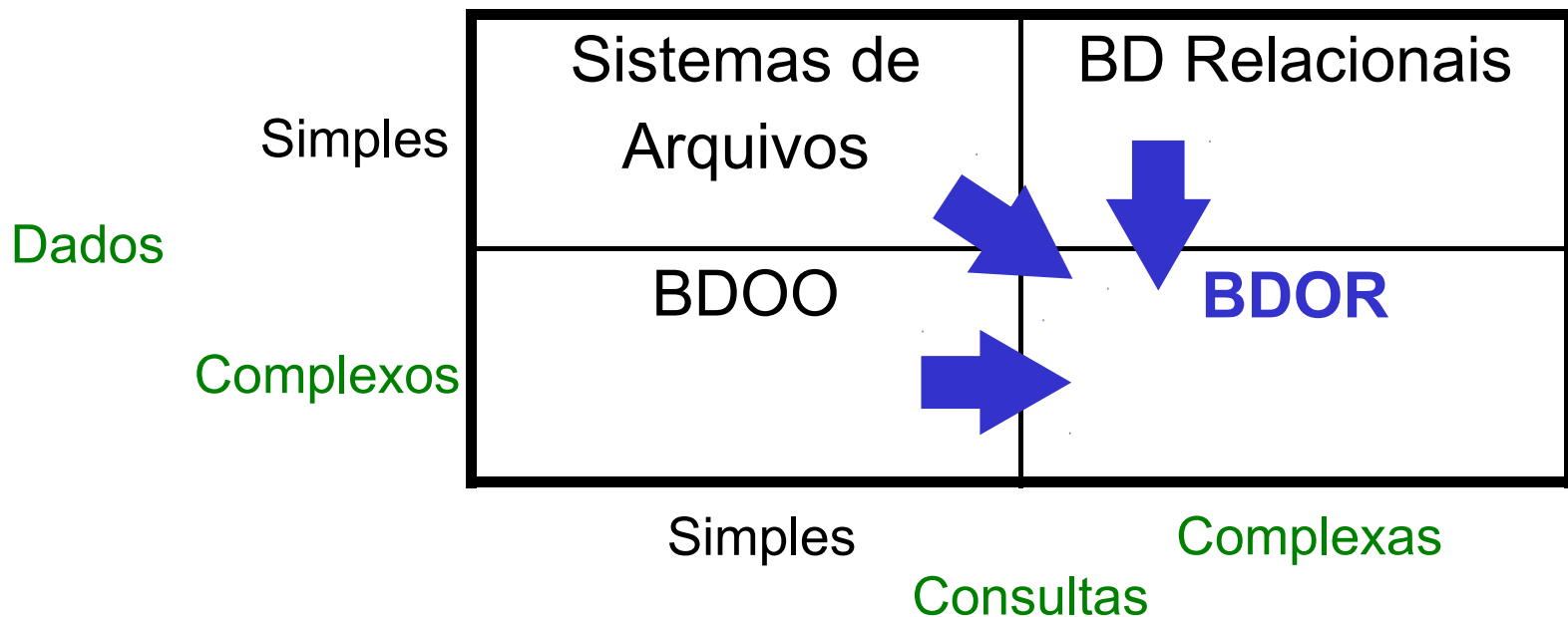
- Quadrantes de tipos de gerenciadores de dados

Dados	Simples	Sistemas de Arquivos	BD Relacionais
	Complexos	BDOO	BDOR
		Sim	Complexas
		Consultas	

- dados são tabelas com estrutura complexa
- uso do padrão SQL estendido (SQL-3) para garantir flexibilidade nas consultas

Classificação de *Stonebreaker*

- Tendência
 - migração para tecnologia OR



SQL-3 (SQL 99)

- Versão mais atual da SQL
 - por enquanto (SQL-4 em andamento) ...
- Extensão da SQL-2 (SQL 92)
 - tratamento de objetos
 - consultas recursivas
 - instruções de programação
 - ...

SQL-3

- Suporte ao tratamento de objetos
 - tabelas aninhadas (objetos linha)
 - tipos abstratos de dados (TADs)
 - referências e OIDs
 - objetos complexos
 - definição de comportamento
 - herança

Definição de Objetos

- Duas formas
 - tipo objeto linha (*row object*)
 - define uma estrutura de tupla (registro)
 - atributos podem ser do tipo objeto linha
 - permite a definição de uma estrutura aninhada
 - tipo abstrato de dado (TAD)
 - define uma estrutura complexa
 - define comportamento e herança

Objeto Linha

- Definição

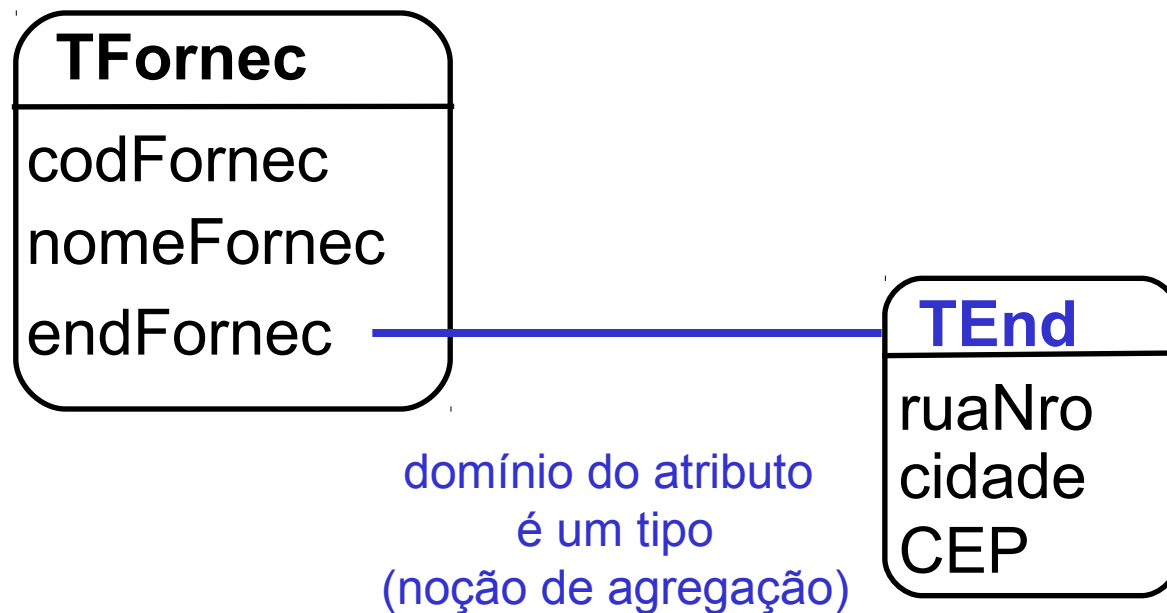
```
CREATE ROW TYPE (<declaração_componentes>)
```

- Exemplos

```
CREATE ROW TYPE TFornec (  
    codFornec          CHAR(4) ,  
    nomeFornec          VARCHAR(40) ,  
    endFornec          TEnd ) ;
```

```
CREATE ROW TYPE TEnd (  
    ruaNro              VARCHAR(60) ,  
    cidade              VARCHAR(40) ,  
    CEP                 INTEGER ) ;
```

Modelagem OR – Tipo Objeto Linha

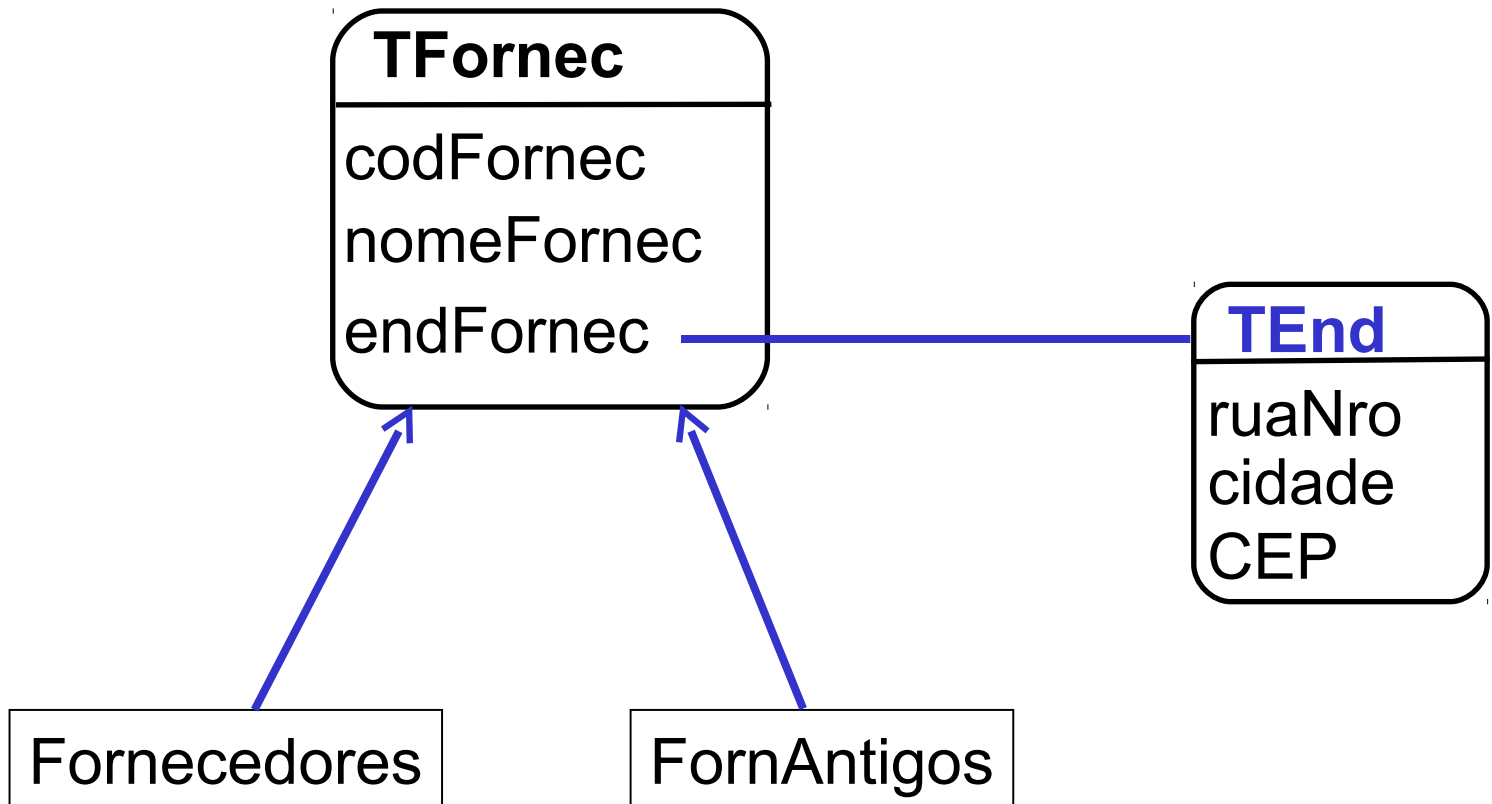


Criação de Tabelas

- Indicação do tipo a que pertence
- Várias tabelas podem ser de um mesmo tipo
- Exemplos

```
CREATE TABLE Fornecedores OF TYPE TFornece;  
CREATE TABLE FornAntigos OF TYPE TFornece;
```

Modelagem OR – Tabela Baseada em Tipo



Acesso a Atributos Aninhados

- Notação de ponto (“dois pontos”) para navegação em atributos que fazem parte de uma estrutura aninhada
- Exemplo

```
SELECT codFornec, endFornec..ruaNro  
FROM Fornecedores  
WHERE endFornec..cidade = 'Florianopolis'
```

Criação de Objetos Linha

- Indicação de valores para todos os níveis de aninhamento
- Exemplo

```
INSERT INTO Fornecedores  
VALUES ('F102', 'João Silva', TEnd('rua A,  
120', 'Florianópolis', 88000));
```

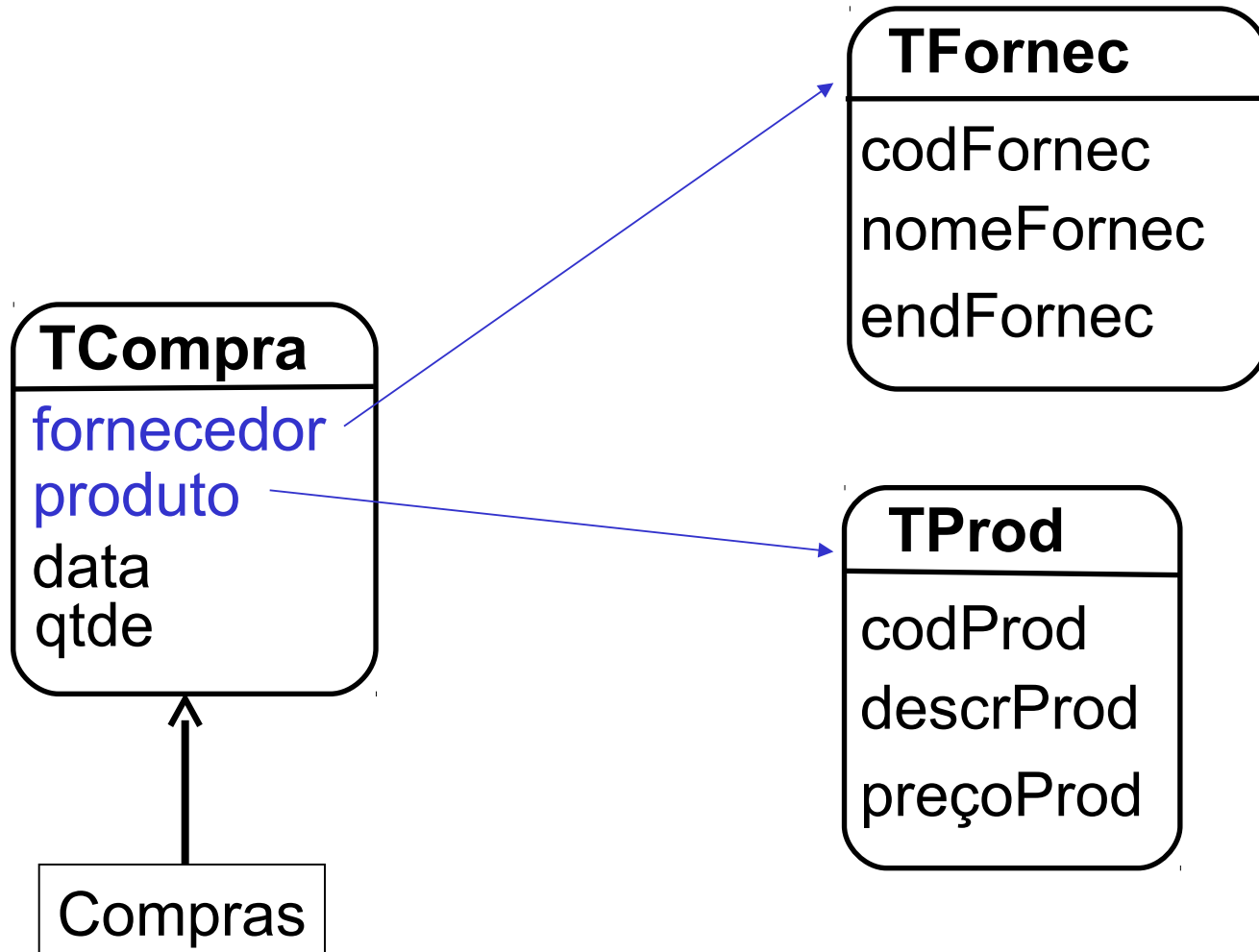
Referência

- Definição de relacionamento entre objetos
- Não é semelhante a uma chave estrangeira
 - chave estrangeira pode ser composta
 - só referencia uma tabela que tenha definido um OID (tabela baseada em um tipo)
- Exemplo

```
CREATE ROW TYPE TCompra (  
    fornecedor      REF (TFor nec) ,  
    produto         REF (TProd) ,  
    data            DATE ,  
    qtde            INTEGER) ;
```

```
CREATE TABLE Compras OF TYPE TCompra ;
```

Modelagem OR – Referências



Acesso a Objetos Relacionados

- Exemplo

```
SELECT fornecedor->nomeFornec  
FROM Compras  
WHERE qtde > 1000  
AND produto->codProd = 45;
```

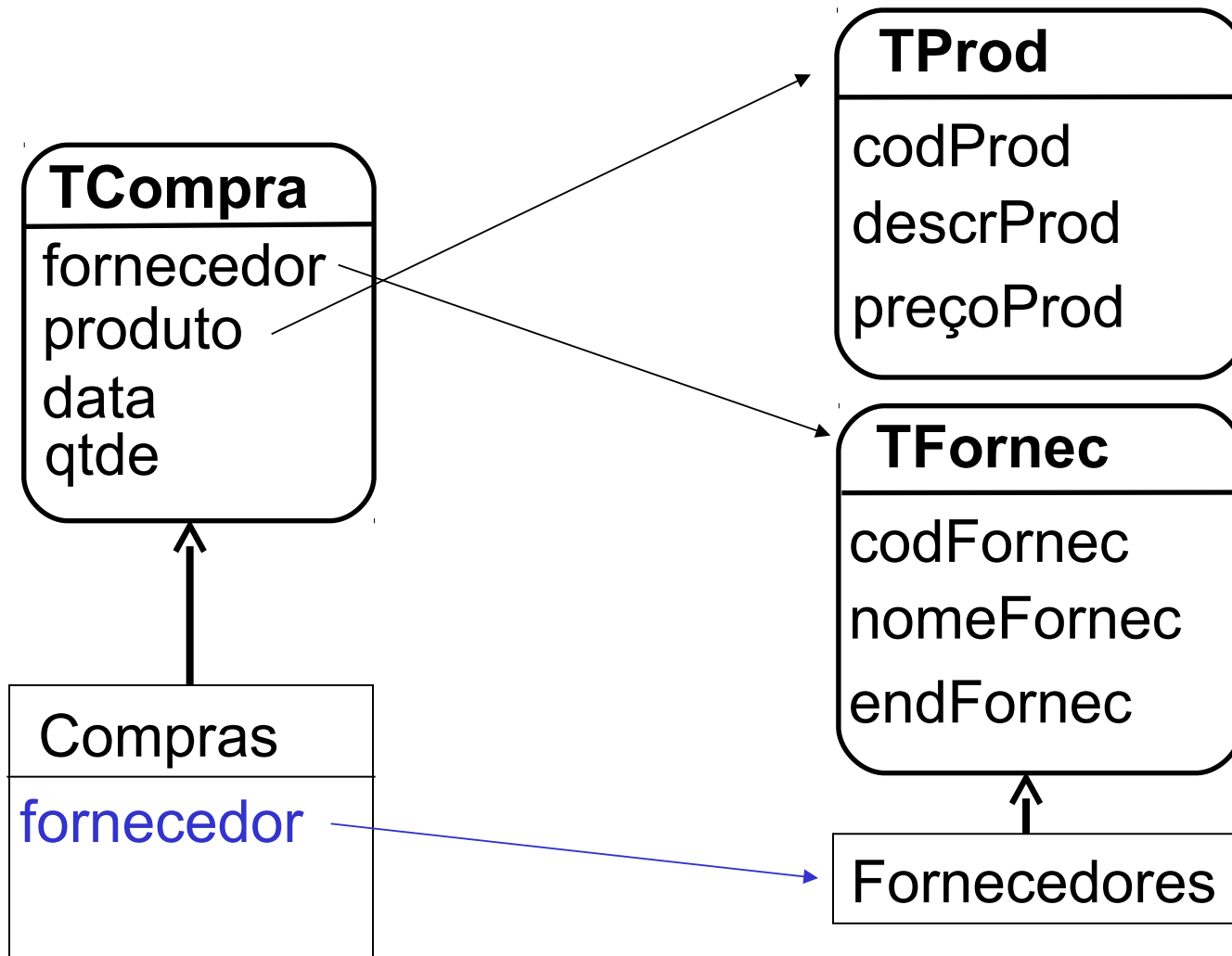
*indica uma referência a
um OID e não a um atributo
de um componente agregado*

Escopo de Referência

- Uma referência indica um tipo
- Deve-se definir o escopo da referência quando mais de uma tabela pertence ao tipo
 - `SCOPE FOR <nome_atributo> IS <nome_tabela>`
- Exemplo

```
CREATE TABLE Compras OF TYPE TCompra  
SCOPE FOR fornecedor IS Fornecedores;
```

Modelagem OR – Escopo de Referência



OID

- Em BDOR, um OID
 - é o valor indicado por atributos de referência
 - pode ser uma chave primária
 - pode ser definido pelo usuário ou pelo sistema
- Exemplos

```
CREATE TABLE Fornecedores OF TYPE TFornece  
REF IS codFornece PRIMARY KEY;
```

```
CREATE TABLE Produtos OF TYPE TProd  
REF IS codProd USER GENERATED;
```

atributo definido pelo usuário, mas seu
valor é controlado pelo sistema (OID)

OID gerado e controlado
pelo sistema

```
... REF IS SYSTEM GENERATED;
```

Modelagem OR – Identificadores

Fornecedores
<u>codFornec</u>

Produtos
<u>codProd</u> (OID)

X
(OID)

Comparações de OIDs

- Comparações idênticas às comparações de valores de outros tipos de atributos
- Exemplo

```
SELECT Fornecedores.nomeFornec  
FROM Fornecedores, Compras, Produtos  
WHERE Produtos.tipo = 'Parafuso'  
AND Compras.qtde > 1000  
AND Produtos.codProd = Compras.produto  
AND Fornecedores.codFornec = Compras.fornecedor;
```

Criação de Objetos com Referência

- Indicação dos valores de Olds
- Exemplo

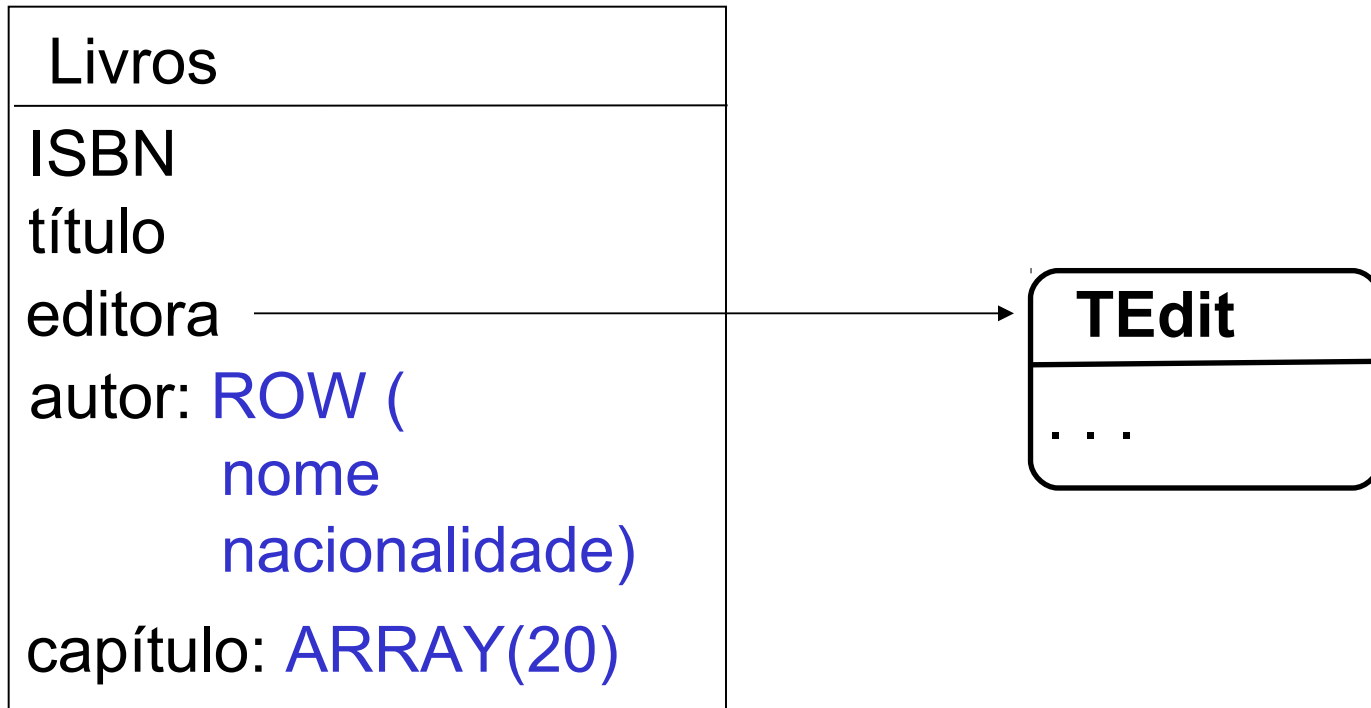
```
INSERT INTO Compras  
VALUES (REF('F102'), REF(1002), '10/09/2013',  
1300);
```

Definição de Objetos Complexos

- Novos tipos de dados
 - *Row* (tupla)
 - *Array* (coleção ordenada)
 - arrays n-dimensionais não são permitidos
- Exemplo

```
CREATE TABLE Livros (  
    ISBN          CHAR(10),  
    título        VARCHAR(60) NOT NULL,  
    editora        REF(TEdit),  
    autor          ROW (nome          VARCHAR(50),  
                        nacionalidade VARCHAR(15)),  
    capítulo       VARCHAR(20) ARRAY[20]  
);
```


Modelagem OR – Objetos Complexos



Acesso a Objetos Complexos

```
insert into Livros
values ('65893/186-9', 'Banco de Dados Objeto-
Relacional', REF('Campus'), ('João Souza',
'brasileira'), ARRAY['Introdução', 'OO', 'BD
Objeto-Relacional', 'Conclusão']);

select capitulos[1]
from Livros
where autor..nome = 'João Souza'
```

Objetos Complexos

- Alguns SGBDORs suportam outros tipos coleção
 - Informix
 - *List*, *Set* e *Multiset* (coleção)
 - Oracle
 - **VARRAY** (*array* variável cujos elementos podem ser objetos)
 - **NESTED TABLE** (atributo cujo domínio é uma tabela aninhada)

Tipo Abstrato de Dados (TAD)

- Define comportamento para os objetos
 - encapsulamento de atributos e métodos
- Permite herança de um tipo para um subtipo
- Definição

```
CREATE TYPE <nomeTAD> (  
    <listaAtributos>  
    [<declaraçãoAssinMétodos>] )
```

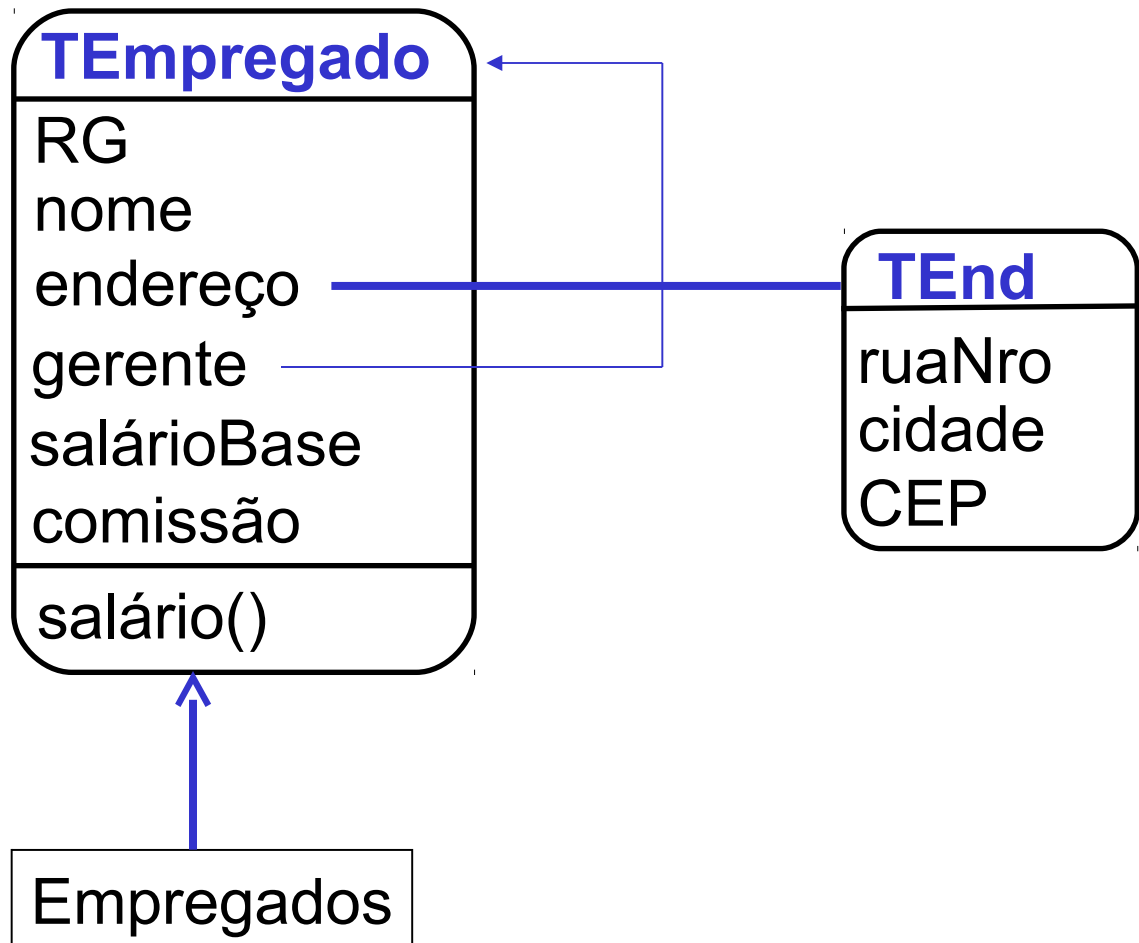
[INSTANTIABLE]  *pode gerar tabelas*

[[NOT] FINAL]  *pode ou não ser especializado*

TAD - Exemplo

```
CREATE TYPE TEmpregado (  
    RG                INTEGER,  
    nome              VARCHAR(40) ,  
    endereço          Tend,  
    gerente            REF (TEmpregado) ,  
    salárioBase        DECIMAL (7,2) ,  
    comissão          DECIMAL (7,2) ,  
  
    METHOD salário() RETURNS DECIMAL (7,2) ;  
    ... )  
INSTANTIABLE  
NOT FINAL;  
  
CREATE TABLE Empregados OF TYPE TEmpregado;
```

Modelagem OR – TAD



TAD - Comportamento

- SQL-3 permite a definição de métodos, funções e procedimentos
- Implementação de método

```
CREATE METHOD salário()  
FOR Tempregado  
RETURN REAL  
BEGIN  
    RETURN salarioBase + comissão*0.8;  
END
```

Métodos x Funções/Proc

- Métodos

- só podem ser definidos dentro de um TAD
- identificação do método a ser executado é determinado em tempo de execução (*late binding*)
 - depende do objeto que o invoca ou de parâmetros

- Funções/Procedimentos

- podem ser definidos fora de um TAD
 - `CREATE FUNCTION` / `CREATE PROCEDURE`
- identificação da função/procedimento a ser executado é determinado em tempo de compilação (*early binding*)

Métodos

- Consultas SQL podem invocar métodos ou funções

```
select RG,nome  
from empregados  
where salário() > 1000.00;
```

Herança

- Definição

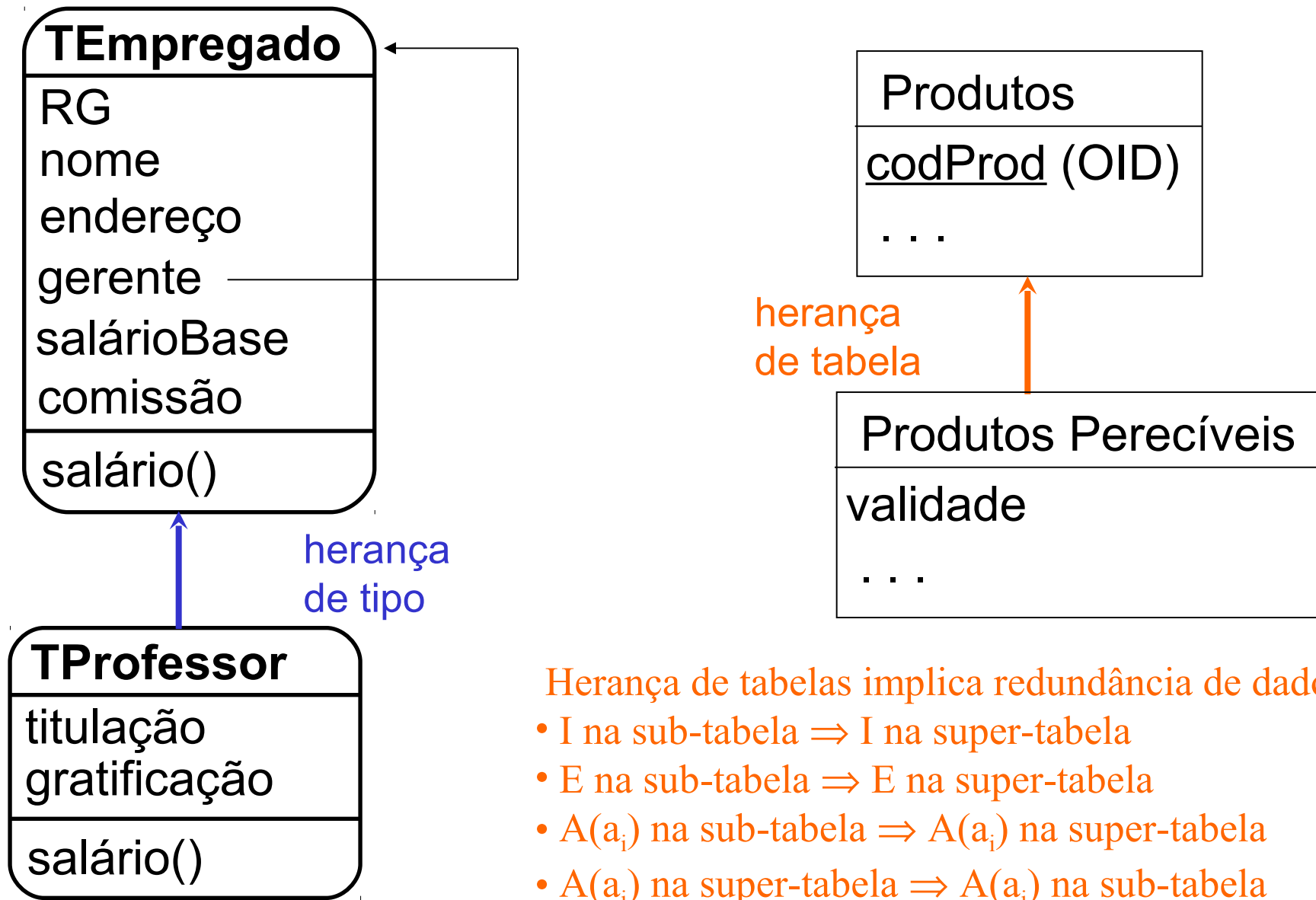
- CREATE TYPE <nomeTAD> UNDER <nomeTAD> (...)
- CREATE TABLE <nomeTab> UNDER <nomeTab> (...)

- Herança múltipla não é permitida

- Exemplo

```
CREATE TYPE Tprofessor UNDER Tempregado (  
    titulação          VARCHAR(15),  
    gratificação       DECIMAL (7,2),  
    OVERRIDING METHOD  salário() RETURNS DECIMAL (7,2);  
    ... )  
INSTANTIABLE  
NOT FINAL
```

Modelagem OR – Herança



Herança de tabelas implica redundância de dados:

- I na sub-tabela \Rightarrow I na super-tabela
- E na sub-tabela \Rightarrow E na super-tabela
- $A(a_i)$ na sub-tabela $\Rightarrow A(a_i)$ na super-tabela
- $A(a_i)$ na super-tabela $\Rightarrow A(a_i)$ na sub-tabela

Projeto Lógico de BDOR

- Combina recomendações de projeto de BDR e BDOO

Esquema ER	Esquema OR
entidade	tabela (pode-se definir adicionalmente um TAD ou um objeto linha para uma entidade, caso haja necessidade ou não de comportamento e/ou reuso de definição)
entidade fraca	<ul style="list-style-type: none">atributo com domínio tupla (<i>ROW</i>) OUatributo de referência fraca -> forte
relacionamento 1:1	<ul style="list-style-type: none">fusão de entidades em uma tabela OUreferências entre tabelas
relacionamento 1:N	atributo de referência na tabela correspondente à entidade do lado N

Projeto Lógico de BDOR

Esquema ER	Esquema OR
relacionamento M:N	<ul style="list-style-type: none">• tabela de relacionamento OU• atributo(s) com domínio(s) <i>ARRAY</i>
atributo monovalorado	atributo atômico
atributo composto	atributo com domínio tupla (<i>ROW</i>)
atributo multivalorado	atributo com domínio <i>ARRAY</i>
especialização	hierarquia de herança entre tipos ou tabelas
entidade associativa	mesmas recomendações para mapeamento de relacionamentos binários

Para resolver

- Apresentar a modelagem lógica objeto-relacional para o domínio da clínica

