Modelagem Objeto Relacional X PostGreSQL

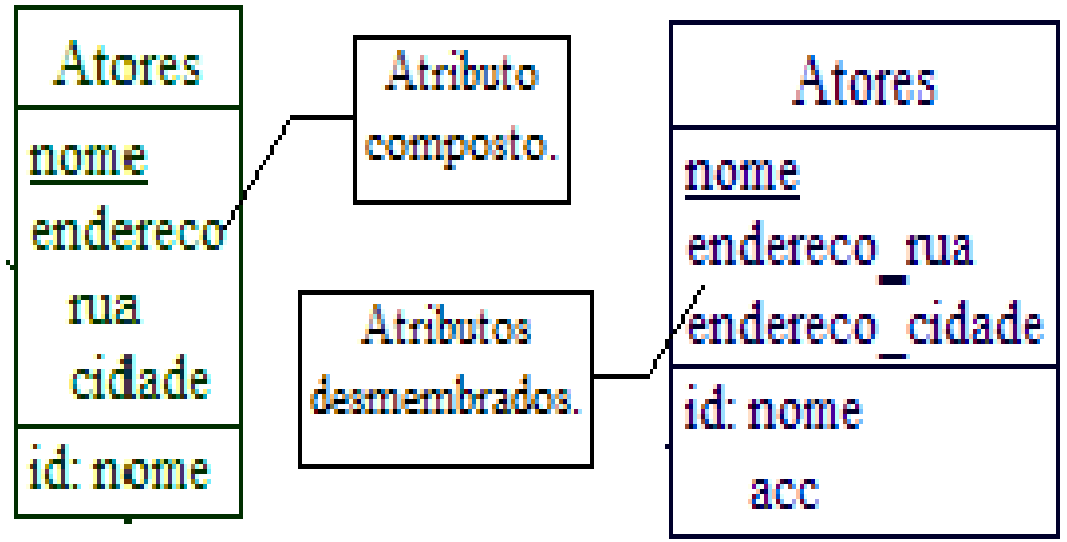
Profa. Dra. Sarajane Marques Peres
Each – USP
each.uspnet.usp.br/sarajane

	Modelo ER	Modelo ODL	Modelo Relacional	Modelo EER
Representação de entidade do mundo real	Conjunto Entidades	Classes	Relações	Conjuntos Entidades
Representação do relacionamento entre entidades do mundo real	Conjuntos Relacionamentos	Relacionamentos	Chaves Estrangeiras	Conjuntos Entidades
Representação de especializações/generalizações	Relacionamentos ISA	Representação Explicita	Representação Implicita	Representação Explicita (em diferentes niveis)
Representação de Restrições	Representações de restrições simples	Métodos	Cláusulas CHECK	Representações de restrições simples
Compatibilidade com o modelo fisico do SGBD relacional	Precisa de mapeamento parcial	Precisa de mapeamento total	Total	Precisa de mapeamento parcial
Compatibilidade com o modelo fisico do SGBD orientado a objetos	Precisa de mapeamento total	Total	Precisa de mapeamento total	Precisa de mapeamento total
Compatibilidade o modelo fisico do SGBD OR	Total	Precisa de mapeamento total	Total	Total
Expressividade de modelagem orientada a objetos	Baixa	Exata	Baixa	Alta

A questão da Primeira Forma Normal

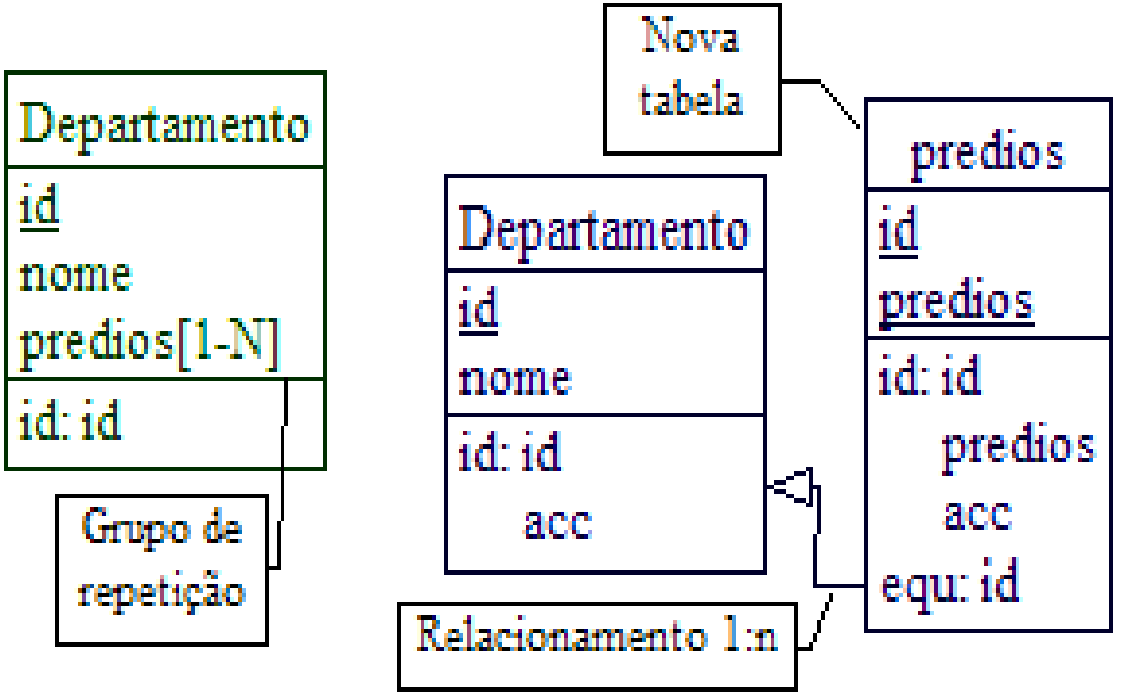
- Esta é uma regra bastante importante do modelo de dados relacional já que é premissa de modelagem.
-
- Todos as tecnologias que suportam, puramente, o modelo relacional, não admitem o uso de atributos compostos ou de grupos de repetição (multivalorados).
- Assim, no momento do mapeamento entre um modelo de dados conceitual do tipo Entidade-Relacionamento para um modelo do tipo Relacional, a normalização:
  - elimina os grupos repetidos e os atributos compostos colocando-os, respectivamente, em tabelas separadas fazendo uso de um relacionamento 1:n e
  - criando um atributo separado para cada sub-atributo de um tipo complexo.

Modelo Entidade-Relacionamento →



← **Modelo Relacional**

Modelo Entidade-Relacionamento →



← **Modelo Relacional**

Esquema Relacional:

Departamento: id, nome
Prédios: prédios, id, endereço_rua, endereço_cidade

Esquema Objeto-Relacional conceitual:

Departamento: id, nome, lista de prédios (prédio, endereço(rua,cidade))

Instância Relacional (somente tabelas *flat*):

Departamento

nro	nome
1	DIN
2	DEQ
3	DAU

Prédios

id	prédios	endereço_rua	endereço_cidade
1	19a	RuaA	CidadeA
1	20a	RuaB	CidadeB
2	09a	RuaC	CidadeC
3	18a	RuaA	CidadeB

Instância Objeto-Relacional (tabelas *aninhadas*):

Departamento

id	nome	Prédios		
		prédio	endereço	
			rua	cidade
1	DIN	19a	RuaA	CidadeA
		20a	RuaB	CidadeB
2	DEQ	09a	RuaC	CidadeC
3	DAU	18a	RuaA	CidadeB

O que é realmente "implementável"?

(a) implementação de uma única tabela com todas as informações, porém, com um sentido semântico diferente – o endereço é do departamento. Veja as definicões de tipo e de tabela, uma instância da tabela criada e alguns exemplos de manipulação de dados.

```
CREATE TYPE t_endereco AS
(
        rua text,
        numero int,
        bairro text,
        complemento text
);
```

```
CREATE TABLE departamento_2
(
        id int,
        nome varchar(15),
        predio varchar(10)[],
        endereco t_endereco,
PRIMARY KEY (id)
);
```

```
--INSERT INTO departamento_2 VALUES
--(1,'DIN','{19a,20a}',('Av Colombo',5720,'Zona 07','Maringá'));
--INSERT INTO departamento_2 VALUES
--(2,'DEQ','{09a}',('Av Colombo',5721,'Zona 07','Maringá'));
INSERT INTO departamento_2 VALUES
(3,'DAU','{18a}',('Av Colombo',5722,'Zona 08','Maringá'))
;
```

id [PK] integer	nome character var	predio character var	endereco t_endereco
1	DIN	{19a,20a}	("Av Colombo",5720,"Zona 07",Maringá)
2	DEQ	{09a}	("Av Colombo",5721,"Zona 07",Maringá)
3	DAU	{18a}	("Av Colombo",5722,"Zona 08",Maringá)

```
select predio[1]
from departamento_2
where id=1;
```

predio character varying
19a

```
select (endereco).rua
from departamento_2
where id=1;
```

rua text
Av Colombo

```
select id
from departamento_2
where (endereco).rua = 'Av Colombo'
```

id integer
1
2
3

O que é realmente “implementável”?

(b) implementação de duas tabelas para representar o sentido semântico do exemplo.

```
CREATE TABLE departamento_3
(
id integer,
nome character varying(15),
primary key (id)
);
```

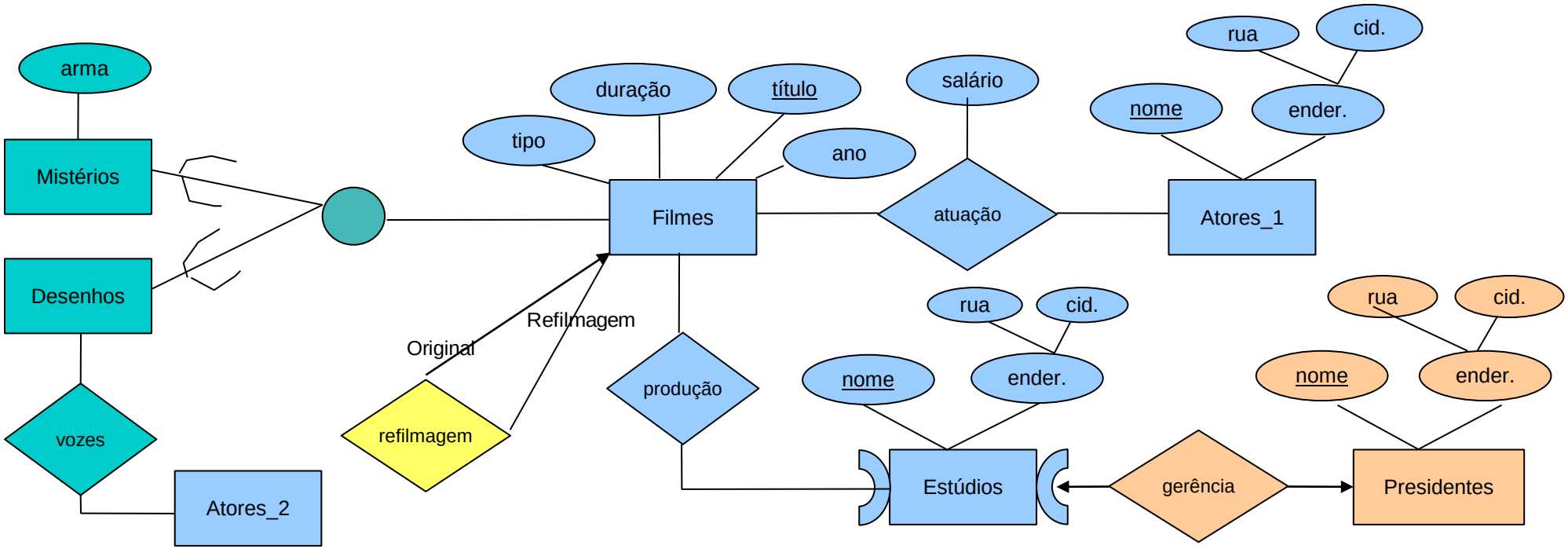
```
CREATE TABLE predio
(
nome varchar(15),
endereco t_endereco,
id integer references departamento_3,
primary key (nome)
);
```

id [PK] integer	nome character var
1	DIN
2	DEQ
3	DAU

nome [PK] characte	endereco t_endereco	id integer
09a	("Av. Colombo",333,"Zona 07",Maringá)	2
18a	("Av. Colombo",666,"Zona 07",Maringá)	3
19a	("Av. Colombo",222,"Zona 07",Maringá)	1
20a	("Av. Colombo",223,"Zona 07",Maringá)	1

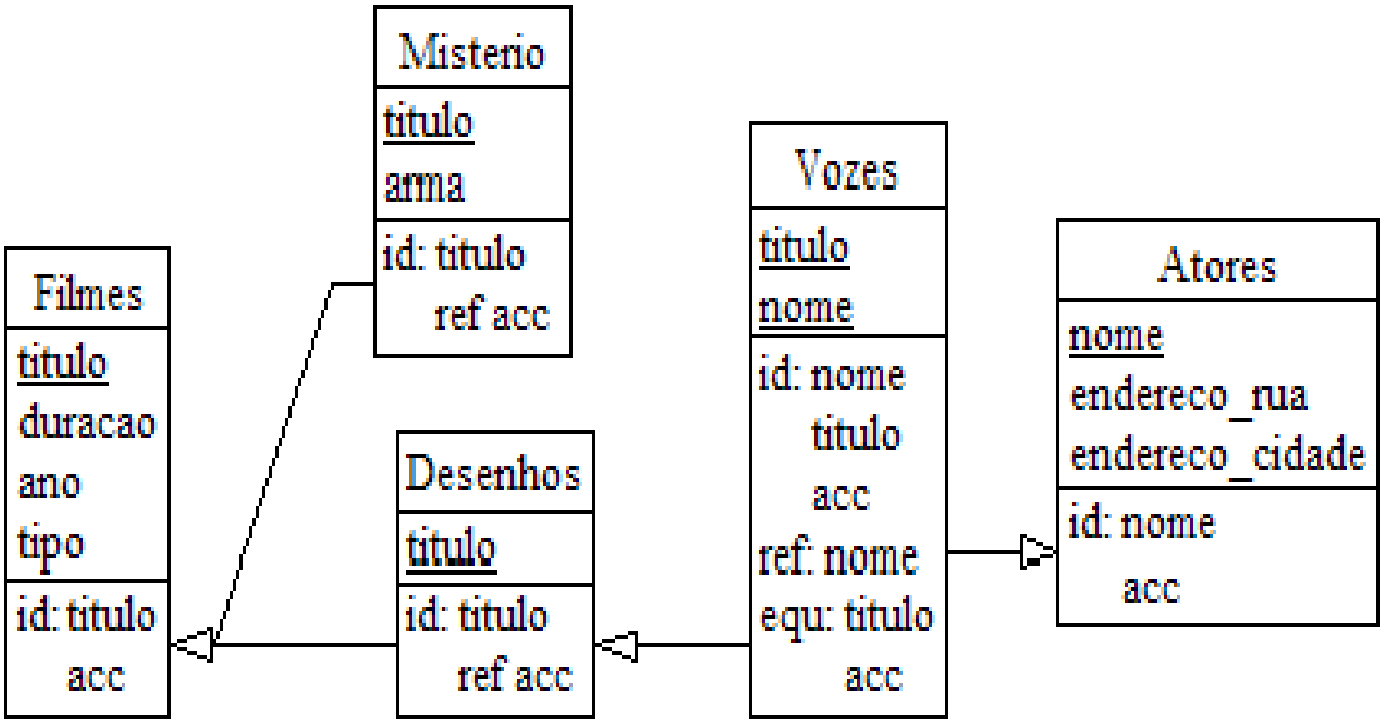
- Insiram os valores!!!!!!!!

Trabalhando com herança



Modelo Entidade
Relacionamento Estendido

Trabalhando com herança



Modelo Relacional

Script (parcial)

```
create table Atores (
     nome varchar(50) not null,
     endereco_rua varchar(1) not null,
     endereco_cidade varchar(1) not null,
     constraint ID_Atores primary key (nome));

create table Atuacao (
     titulo varchar(50) not null,
     nome varchar(50) not null,
     salario float(1) not null,
     constraint ID_Atuacao primary key (nome, titulo));

create table Desenhos (
     titulo varchar(50) not null,
     constraint FKFil_Des_ID primary key (titulo));

create table Misterio (
     titulo varchar(50) not null,
     arma char(1) not null,
     constraint FKFil_Mis_ID primary key (titulo));

create table Filmes (
     titulo varchar(50) not null,
     duracao float(1) not null,
     ano date not null,
     tipo char(1) not null,
     constraint ID_Filmes_ID primary key (titulo));

create table Vozes (
     titulo varchar(50) not null,
     nome varchar(50) not null,
     constraint ID_Vozes primary key (nome, titulo));
```

Trabalhando com herança

- A partir do *script* gerado na Ferramenta DBMain, do modelo relacional do slide anterior, gerou-se o banco de dados correspondente.
- Nesta estrutura, as seguintes tentativas de inserção de dados foram feitas:
  - inserção na tabela Mistério, de um filme já cadastrado na tabela Filmes; (OK)
  - inserção na tabela Mistério, de um filme ainda não cadastrado na tabela Filmes; (Não permitido, violação de chave estrangeira);

Implementando a Herança

```
CREATE TABLE Misterio
(
 arma character(1),
 PRIMARY KEY (titulo)
) INHERITS (filmes);
```

Apenas o atribuo da própria tabela precisou ser criado. Enquanto que na estrutura anterior, a chave estrangeira precisava estar presente na definição da tabela.

Opcionalmente pode-se estabelecer qualquer restrição de chave utilizando atributos da entidade mãe. Nenhuma delas é herdada na herança.

Crie Desenho_1 !!!!!

Tabela Filmes

titulo [PK] characte	duracao real	ano date	tipo character(1)
Rei Leao	180	2003-12-23	D
Sol e Lua	102	1999-10-08	M

Implementando Herança

- Inserindo tuplas nas tabelas filhas ....

Tabela Misterio

titulo [PK] characte	duracao real	ano date	tipo character(1)	arma character(1)
Rei Leao	180	2003-12-23	D	X

Tabela Desenho

titulo [PK] characte	duracao real	ano date	tipo character(1)
Rei Leao	999	2003-12-23	D

Tabela Filmes

titulo [PK] characte	duracao real	ano date	tipo character(1)
Rei Leao	180	2003-12-23	D
Rei Leao	180	2003-12-23	D
Rei Leao	999	2003-12-23	D
Sol e Lua	102	1999-10-08	M

Herança

```
CREATE TABLE misterio
(
  titulo character varying(50) NOT NULL,
  arma character(1) NOT NULL,
  CONSTRAINT fkfil_mis_id PRIMARY KEY (titulo),
  CONSTRAINT fkfil_mis_fk FOREIGN KEY (titulo)
      REFERENCES filmes (titulo) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITHOUT OIDS;
ALTER TABLE misterio OWNER TO sara;
```

← Sem herança!

Com herança! →

```
CREATE TABLE misterio_1
(
-- Inherited:   titulo character varying(50) NOT NULL,
-- Inherited:   duracao real NOT NULL,
-- Inherited:   ano date NOT NULL,
-- Inherited:   tipo character(1) NOT NULL,
  arma character(1),
  CONSTRAINT misterio_1_pkey PRIMARY KEY (titulo)
) INHERITS (filmes)
WITHOUT OIDS;
ALTER TABLE misterio_1 OWNER TO sara;
```

Implementando ...

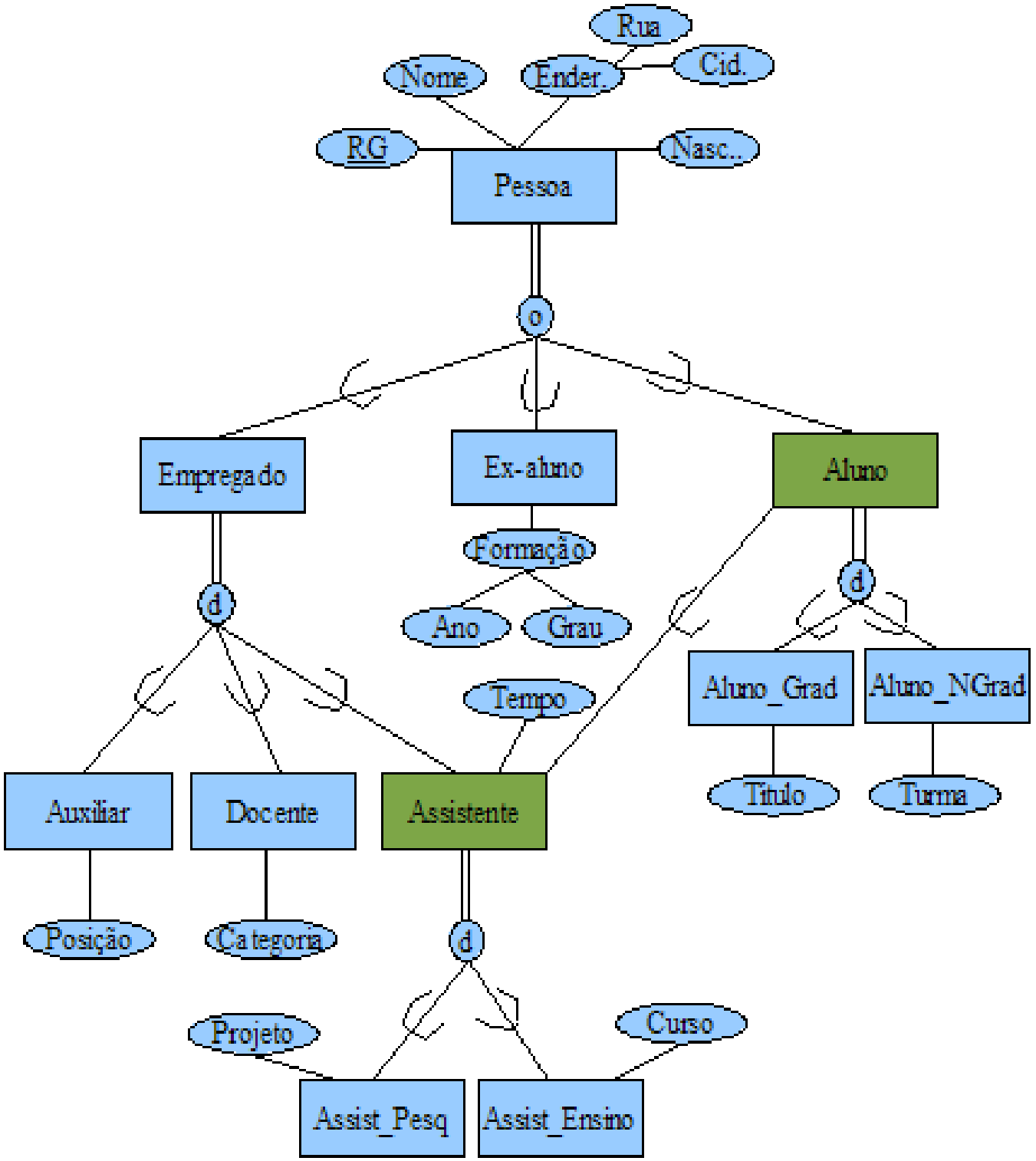
- Restrições do tipo CHECK são herdadas!!!!!
- Restrições de chave não são.

```
ALTER TABLE filmes
    ADD CONSTRAINT duracaominima CHECK
(duracao > 60);
```

Observe os resultados!!!!!!!

Trabalhando com herança

- Reticulados (herança múltipla) e hierarquias



```
create type t_endereco AS (
        rua  character varying(30),
        cidade character varying(15) );

create type t_formacao AS (
        ano integer,
        grau character varying(1) );

create table Ex_aluno (
        formacao t_formacao,
        primary key (RG)
) inherits (Pessoa);

create table Docente (
        categoria char,
        primary key (RG)
) inherits (Empregado);

create table Assist_Pesq (
        projeto character varying(10),
        primary key (RG)
) inherits (Assistente);
```

```
create table Pessoa (
        RG character varying(9),
        nome character varying(30),
        endereco t_endereco,
        nascimento date,
        primary key (RG));

create table Aluno (
        primary key (RG)
) inherits (Pessoa);

create table Assistente
(
        tempo integer,
        primary key (RG)
) inherits (Empregado,Aluno);

create table Assist_Ensino(
        curso character varying(3),
        primary key (RG)
) inherits (Assistente);
```

```
create table Empregado (
        primary key (RG)
) inherits (Pessoa);

create table Aluno_Grad (
        titulo char,
        primary key (RG)
) inherits (Aluno);

create table Aluno_NGrad (
        turma char,
        primary key (RG)
) inherits (Aluno);

create table Auxiliar (
        posicao char,
        primary key (RG)
) inherits (Empregado);
```

Implementando ...

```
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "pessoa_pkey" for table "pessoa"
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "empregado_pkey" for table "empregado"
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "ex_aluno_pkey" for table "ex_aluno"
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "aluno_pkey" for table "aluno"
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "aluno_grad_pkey" for table "aluno_grad"
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "aluno_ngrad_pkey" for table "aluno_ngrad"
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "auxiliar_pkey" for table "auxiliar"
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "docente_pkey" for table "docente"
NOTICE:  merging multiple inherited definitions of column "rg"
NOTICE:  merging multiple inherited definitions of column "nome"
NOTICE:  merging multiple inherited definitions of column "endereco"
NOTICE:  merging multiple inherited definitions of column "nascimento"
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "assistente_pkey" for table "assistente"
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "assist_pesq_pkey" for table "assist_pesq"
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "assist_ensino_pkey" for table "assist_ensino"
```

O destaque diz respeito à questão de herdar duas vezes a mesma coisa (situação comum na herança múltipla). Quando isto acontece, as colunas (ou quaisquer outras propriedades) repetidas não são inseridas duas vezes na especificação da tabela filha.

Tipo Referência

- Aparentemente não suportado no PostGreSQL.
- Suportado no Oracle, com algumas limitações.

- Vejamos um exemplo onde queremos referenciar um objeto da tabela Aluno a partir de um objeto da tabela Professor.
  - Limitação: número de objetos numa referência
    - Fazer vetor de referências.

- Convite ao teste!!!

Tipo Referência

```
CREATE TYPE pessoa_type_2 AS OBJECT(
      nome          VARCHAR(40),
      sexo          CHAR,
      cpf           INT,
      rg            INT,
      idade         INT
) NOT FINAL;


CREATE TYPE aluno_type UNDER pessoa_type_2 (
     RA             INT);

CREATE TYPE professor_type UNDER pessoa_type_2 (
     aluno       REF aluno_type
     );

CREATE TABLE aluno_table OF aluno_type (primary key(ra))

CREATE TABLE professor_table OF professor_type (primary key
(cpf))
```

Tipo Referência

```
INSERT INTO aluno_table VALUES ('João Augusto Gastão', 'm',
11122233345, 666777888, 18, 33845)
```

```
INSERT INTO professor_table
      SELECT 'José da Silva', 'm', 44455566678, 333444555,
      45,
      REF(aluno)
      FROM aluno_table aluno
      WHERE aluno.ra = 33845
```

```
SELECT professor.aluno.nome AS nome_aluno
FROM professor_table professor
WHERE professor.cpf = 44455566678
```

Métodos

- Tem-se a herança de CHECK que se aproximam, num nível de abstração bastante alto.
- No Oracle tem-se herança em nível de tipo e a herança de método é possível.

Métodos (Oracle)

```
CREATE TYPE  pessoa_type AS object(
        nome_pessoa VARCHAR(40),
        sexo        CHAR,
        cpf         INT,
        rg          INT,
        data_nasc   DATE,
        endereco    Endereco_type_table,
        cidade      VARCHAR(40),
        estado      CHAR(2),
        fone        INT,
        email       VARCHAR(40),
MEMBER PROCEDURE muda_telefone (novo_telefone IN INT)
MEMBER FUNCTION recupera_idade RETURN NUMBER) NOT FINAL;
```

Métodos (Oracle)

```
CREATE OR REPLACE TYPE BODY pessoa_type AS
        MEMBER PROCEDURE muda_telefone (novo_telefone IN INT)
IS
        BEGIN
             fone := novo_telefone;
        END muda_telefone;
END;


CREATE OR REPLACE TYPE BODY pessoa_type AS
        MEMBER FUNCTION recupera_idade RETURN NUMBER IS
        BEGIN
             RETURN sysdate - data_nasc;
        END recupera_idade;
END;

CREATE TABLE pessoas_tab OF pessoa_type_teste (PRIMARY KEY
(cpf))
NESTED TABLE endereco STORE AS endereco_tab;
```

Extending SQL

- O PostgreSQL estoca mais informações em seu catálogo do que os gerenciadores relacionais convencionais.
  - Informações sobre tipos de dados, funções e métodos de acesso (entre outras coisas – a pesquisar)
  - Estas tabelas podem ser modificadas pelos usuários (operações orientada a catálogo) possibilitando sua extensão

- O servidor incorpora o cógido escrito pelo usuários por meio de um “ dynamic loading”, e este código se torna objeto do BD (gerenciado pelo SGBD).

Extending SQL

- Extensões
  - Definições de tipos, domínios, tipos compostos, pseudo-tipos, polimórficos (anyelemente, anyarray), compostos.
  - Definições de funções
  - Definições de agregados (cada chamada sucessiva de uma função altera o valor (estado) do seu atributo de entrada).
  - Entrada de informação para otimização
  - Definição de índices sobre tipos definidos pelo usuário
  - Suporte a DataBlades (pontos, polígonos, etc – tipos geométricos, e tipos de rede)
  - Algum suporte ao tipo XML (ainda como texto)

Funções - Exemplos

```
-- Function: nova()

-- DROP FUNCTION nova();

CREATE OR REPLACE FUNCTION nova()
  RETURNS integer AS
'select 1 as result;'
  LANGUAGE 'sql' VOLATILE;
ALTER FUNCTION nova() OWNER TO postgres;
```

```
-- Function: teste_function()

-- DROP FUNCTION teste_function();

CREATE OR REPLACE FUNCTION teste_function()
  RETURNS bigint AS
'select count(numero) from departamento;'
  LANGUAGE 'sql' VOLATILE;
ALTER FUNCTION teste_function() OWNER TO postgres;
```

OBS.:
Funções e gatilhos não são herdados.

Exemplo - Gatilho

```
---- Function: inc_emp_depto()

  -- DROP FUNCTION inc_emp_depto();

  CREATE OR REPLACE FUNCTION inc_emp_depto()
    RETURNS "trigger" AS
  $BODY$
      BEGIN
          update departamento set numeroempregados = numeroempregados + 1 where numero = new.numero;
          ---NEW.numeroempregados := numeroempregados + 1;
          --NEW.last_user := current_user;
          RETURN NEW;
      END;
  $BODY$
    LANGUAGE 'plpgsql' VOLATILE;
  ALTER FUNCTION inc_emp_depto() OWNER TO postgres;

-- Trigger: inc_emp_depto on empregado

-- DROP TRIGGER inc_emp_depto ON empregado;

CREATE TRIGGER inc_emp_depto
  AFTER INSERT OR UPDATE
  ON empregado
  FOR EACH ROW
  EXECUTE PROCEDURE inc_emp_depto();
```

Apenas curiosidade, não faz parte das caracterítiscas objeto-relacionais!

Tipos Geométricos

Name	Storage Size	Representation	Description
`point`	16 bytes	Point on the plane	(x,y)
`line`	32 bytes	Infinite line (not fully implemented)	((x1,y1),(x2,y2))
`lseg`	32 bytes	Finite line segment	((x1,y1),(x2,y2))
`box`	32 bytes	Rectangular box	((x1,y1),(x2,y2))
`path`	16+16n bytes	Closed path (similar to polygon)	((x1,y1),...)
`path`	16+16n bytes	Open path	[(x1,y1),...]
`polygon`	40+16n bytes	Polygon (similar to closed path)	((x1,y1),...)
`circle`	24 bytes	Circle	<(x,y),r> (center and radius)

Tipos Geométricos

```
-- Table: geometria

-- DROP TABLE geometria;

CREATE TABLE geometria
(
  ponto point,
  linha line
)
WITHOUT OIDS;
ALTER TABLE geometria OWNER TO postgres;
```

```
insert into geometria
values ('0.5, 1.3','(1.0,2.0),(3.4,5.7)');
```

ponto point	linha lseg
(0.5,1.3)	[(1,2),(3.4,5.7)]
(6,1.4)	[(0.8,2.1),(7.8,2.7)]
(3,3.4)	[(3.8,3.1),(3.8,3.7)]

```
select linha from geometria
       where ponto[0] = 0.5;
```

	linha lseg
1	[(1,2),(3.4,5.7)]