



Laboratório de Bases de Dados

Prof. José Fernando Rodrigues Júnior

Aula 11 – Objeto Relacional

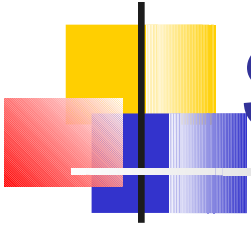
Material: Profa. Elaine Parros Machado de Sousa





SGBD Relacional

- Dados e relacionamentos
 - coleções de tabelas
- Vantagens
 - bom desempenho
 - processamento de transações
 - otimização de consultas
- Desvantagem
 - não atende adequadamente os requisitos de dados de aplicações inerentemente orientadas a objetos



SGBD Orientado a Objetos

- Dados e relacionamentos
 - coleções de objetos
- Objeto
 - propriedades + métodos
- Desvantagem
 - desempenho baixo, quando comparado a SGBDs relacionais
- Exemplos:
 - O2, Objectivity/DB, ObjectStore, GemStone, Versant,



Relacional vs Orientado a Objetos

- Principais diferenças:
 - SGBDs relacionais usam chaves estrangeiras - SGBDS OO usam ponteiros (ou links)
 - SGBDS OO suportam modelos de dados mais complexos
- O uso de links dispensa a necessidade de junção
 - no entanto, faz com que a busca pela informação seja aleatória – isto é, uma tupla referenciada pode estar em qualquer lugar
 - para operações de alta cardinalidade isto faz com que o desempenho caia
 - quando se usa junção, todos os dados são processados, mesmo os não necessários, mas o processamento é sequencial, o que em disco é mais eficiente do que acesso aleatório



SGBD Orientado a Objetos

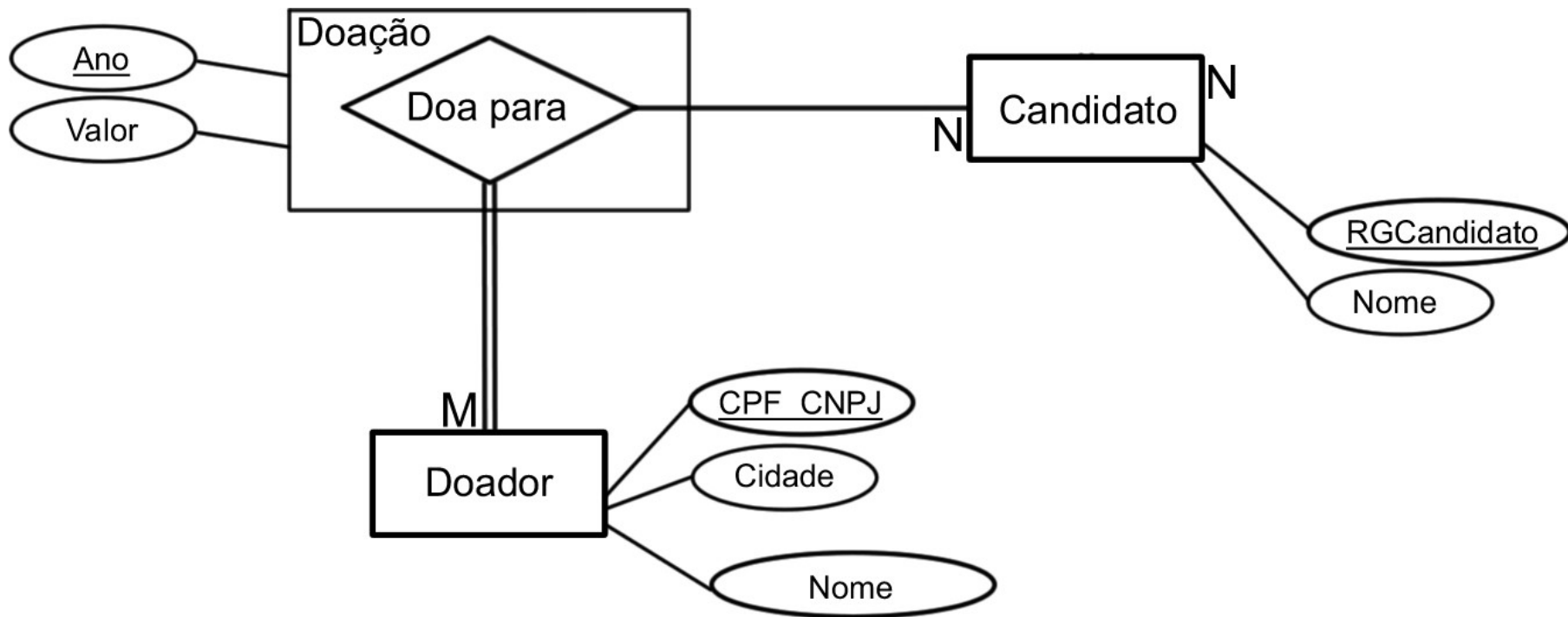
■ Porque não vingou?

- SGBDs OO não são escaláveis – isto é, não mantém o desempenho para grandes volumes de dados
- SGBDs OO apresentam alta complexidade para a otimização de consultas
- Impõe mudanças (maior complexidade) no uso do SQL, um padrão fortemente já consolidado
- SGBDs relacionais passaram a oferecer funcionalidades OO nos chamados SGBD objeto-relacionais oferecendo ambas as possibilidades ao mesmo tempo
- SGBDs são sistemas caros, o custo da migração para SGBDs OO não seria compensado pelas vantagens, que não são tão grandes
- Empresas de SGBDs relacionais já estão consolidadas e tem mais recursos para orientar o mercado via marketing e novos produtos



SGBD Objeto-Relacional

- Modelo de dados
 - fundamentado no modelo relacional
 - estendido com características do modelo OO
- Vantagens
 - modelo semanticamente mais rico
 - eficiência no gerenciamento de dados
 - possibilita o armazenamento direto de instâncias de objetos advindos de aplicações orientadas a objetos
- SQL3
 - objetos, herança, métodos
- Exemplos
 - Oracle, Informix, DB2, PostgreSQL, ...





ORACLE

- Recursos Objeto-Relacionais
 - *object types*
 - *object tables*
 - métodos
 - herança
 - *object views*
 - REF *datatype*
 - coleções



Object Type

- elemento do esquema da base de dados
- tipo de dado definido pelo usuário
 - semântica
- instanciado em objetos
- atributos
 - estrutura
- métodos
 - comportamento

Object Type

```
CREATE TYPE PESSOA AS OBJECT (  
    nome VARCHAR2(30),  
    data_nascimento DATE,  
    endereco VARCHAR2(100)  
);
```

```
CREATE TABLE alunos (  
    NUSP VARCHAR2(7) PRIMARY KEY,  
    dados_pessoais PESSOA);
```

```
INSERT INTO alunos VALUES (1111,  
    PESSOA('Ana', '04-04-1980', 'São Paulo'));
```

Object Type – Valores NULL

```
INSERT INTO alunos VALUES (1111,  
                             PESSOA(NULL, NULL, NULL) );
```

```
INSERT INTO alunos VALUES (1111, NULL) ;
```

Qual a diferença????

Object Type – Constraints

```
CREATE TABLE alunos (  
    NUSP VARCHAR(7) PRIMARY KEY,  
    dados_pessoais PESSOA  
        DEFAULT PESSOA (null, null, null),  
    UNIQUE (dados_pessoais.nome));
```

```
INSERT INTO alunos VALUES (1111, DEFAULT);
```



Object Table

- cada linha armazena um objeto

```
CREATE TABLE tabela_pessoas OF  
PESSOA;
```



Object Table

- pode ser vista como tabela com múltiplas colunas (**operações relacionais**)

```
INSERT INTO tabela_pessoas VALUES ('Lia', '05-05 1985', 'RJ');
```

```
SELECT * FROM tabela_pessoas;
```

- pode ser vista como tabela com coluna única armazenando um objeto (**operações OO**)

```
INSERT INTO tabela_pessoas  
VALUES (PESSOA('Lia', '05-05-1985', 'RJ'));
```

```
-- função VALUE - usa variável de correlação (ex: p)
```

```
SELECT VALUE(p) FROM tabela_pessoas p  
WHERE p.nome = 'Lia';
```



Object Table

- um **OID** é criado automaticamente para cada objeto da tabela, e usado como chave primária
- como alternativa, pode ser definida explicitamente uma chave primária

```
CREATE TABLE tabela_pessoas OF PESSOA (  
    nome PRIMARY KEY );
```

Object Tables – Constraints

```
CREATE TABLE tabela_pessoas OF PESSOA (  
    nome PRIMARY KEY,  
    CHECK (data_nascimento IS NOT NULL)  
);
```


Object Table – nested table

-- tipo coleção

```
CREATE TYPE LISTA_TEL AS TABLE OF VARCHAR2(10);
```

-- object type

```
CREATE TYPE PESSOA AS OBJECT (  
    nome VARCHAR2(30),  
    data_nascimento DATE,  
    endereco VARCHAR2(100),  
    telefones LISTA_TEL  
);
```

-- object table

```
CREATE TABLE tabela_pessoas OF PESSOA  
    NESTED TABLE telefones STORE AS tabela_tel;
```

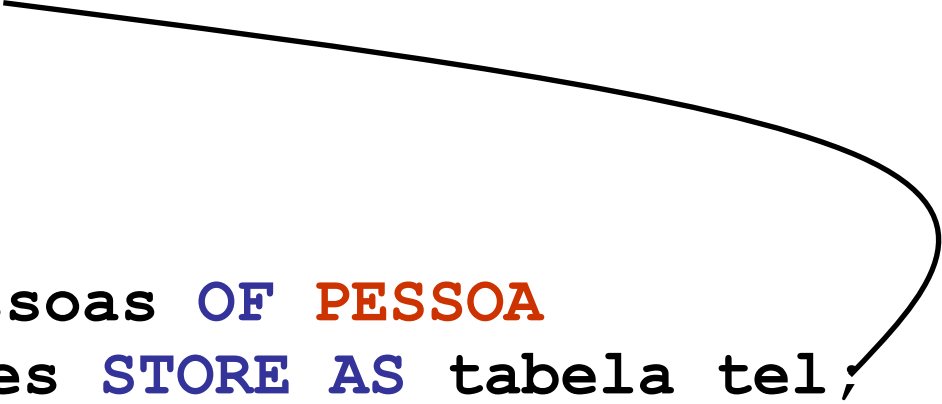
Object Table – nested table

-- tipo coleção

```
CREATE TYPE LISTA_TEL AS TABLE OF VARCHAR2(10);
```

-- object type

```
CREATE TYPE PESSOA AS OBJECT (  
    nome VARCHAR2(30),  
    data_nascimento DATE,  
    endereco VARCHAR2(100),  
    telefones LISTA_TEL  
);
```



-- object table

```
CREATE TABLE tabela_pessoas OF PESSOA  
    NESTED TABLE telefones STORE AS tabela_tel;
```



Métodos

■ Definição

```
CREATE TYPE PESSOA AS OBJECT (  
    nome VARCHAR2(30) ,  
    data_nascimento DATE ,  
    endereco VARCHAR2(100) ,  
    MEMBER FUNCTION idade RETURN NUMBER  
);
```



Métodos

- Implementação

```
CREATE OR REPLACE TYPE BODY PESSOA AS  
MEMBER FUNCTION idade RETURN NUMBER IS  
BEGIN  
    RETURN (SYSDATE - data_nascimento)/365;  
END idade;  
END;
```

Chamada de método

```
declare
```

```
    v_pessoa PESSOA;
```

```
begin
```

```
    v_pessoa := PESSOA('João', '04-06-1978', 'Campinas');
```

```
    INSERT INTO tabela_pessoas VALUES (v_pessoa);
```

```
-- considerando que nome de pessoa é único...
```

```
    SELECT VALUE(v) INTO v_pessoa
```

```
        FROM tabela_pessoas v
```

```
        WHERE v.nome = 'Mariana';
```

```
    dbms_output.put_line('Idade: ' || v_pessoa.idade());
```

```
end;
```



REF Datatype

REF

- “ponteiro” lógico para um objeto (em geral, tupla de uma *object table*)
 - tipo de dado nativo do Oracle
 - permite relacionar objetos – principalmente relacionamentos 1:N e N:N
 - permite fácil navegação entre objetos
 - não é o mesmo que uma chave estrangeira, pois não impõe restrições – é apenas um ponteiro para acesso rápido
-
- Coluna do tipo **REF** – referência a um objeto de um determinado tipo
 - sem escopo – referência a objeto em qualquer *object table*
 - com escopo – referência a objeto em uma *object table* específica



REF Datatype

REF

- “ponteiro” (é um tipo de referência a um objeto em uma *object table*)
- tipo de dado
- permite referências 1:N e N:N
- permite referências
- não é o mesmo que uma chave estrangeira, pois não impõe restrições – é apenas um ponteiro para acesso rápido

Os ponteiros REF se baseiam na chave padrão do Oracle, o ROWID.

- Coluna do tipo **REF** – referência a um objeto de um determinado tipo
 - sem escopo – referência a objeto em qualquer *object table*
 - com escopo – referência a objeto em uma *object table* específica

-- object type

```
CREATE TYPE PESSOA AS OBJECT (  
    nome VARCHAR2(30),  
    data_nascimento DATE,  
    endereco VARCHAR2(100)  
);
```

-- object table

```
CREATE TABLE tabela_pessoas OF PESSOA;
```

-- object type com TIPO DE DADO REF

```
CREATE TYPE ALUNO AS OBJECT (  
    NUSP VARCHAR2(7),  
    dados_pessoais REF PESSOA  
);
```

-- object table - escopo do REF é *constraint!*

```
CREATE TABLE tabela_alunos OF ALUNO (  
    dados_pessoais SCOPE IS tabela_pessoas  
);
```


-- object type

```
CREATE TYPE PESSOA AS OBJECT (  
    nome VARCHAR2(30),  
    data_nascimento DATE,  
    endereco VARCHAR2(100)  
);
```

1) Definição do tipo 1

-- object table

```
CREATE TABLE tabela_pessoas
```

2) Instanciação do tipo 1

-- object type com TIPO DE DADO REF

```
CREATE TYPE ALUNO AS OBJECT (  
    NUSP VARCHAR2(7),  
    dados_pessoais REF PESSOA  
);
```

3) Definição do tipo 2,
fazendo referência ao tipo 1

-- object table - escopo do REF é *constraint*!

```
CREATE TABLE tabela  
    dados_pessoais SC  
);
```

4) Instanciação do tipo 2, especificando
detalhes da referência

-- object type

```
CREATE TYPE PESSOA AS OBJECT (  
    nome VARCHAR2(30),  
    data_nascimento DATE,  
    endereco VARCHAR2(100)  
);
```

-- object table

```
CREATE TABLE tabela_pessoas OF PESSOA;
```

-- object type com TIPO DE DADO REF

```
CREATE TYPE ALUNO AS OBJECT (  
    NUSP VARCHAR2(7),  
    dados_pessoais REF PESSOA  
);
```

-- object table - escopo do REF é *constraint*!

```
CREATE TABLE tabela_alunos OF ALUNO (  
    dados_pessoais SCOPE IS tabela_pessoas  
);
```



Funções REF e Deref

-- FUNÇÃO REF

-- considerando que nome de pessoa é único...

```
INSERT INTO tabela_alunos VALUES  
(ALUNO (111,  
      (SELECT REF(p) FROM tabela_pessoas p  
        WHERE p.nome = 'Lia')));
```

-- FUNÇÃO Deref

```
SELECT a.NUSP, Deref(dados_pessoais).nome  
FROM tabela_alunos a;
```



REFERENCES

- REF só determina restrições de integridade quando uma tabela é definida
- Deve-se usar a seguinte definição

`WITH ROWID REFERENCES nome_tabela`

■ Exemplo:

```
CREATE TYPE x_type AS OBJECT(  
    X1 NUMBER,  
    X2 VARCHAR2(100)  
);
```

```
CREATE TYPE y_type AS OBJECT(  
    Y1 NUMBER,  
    Y2 REF x_type  
);
```

```
CREATE TABLE x OF x_type(  
    X1 PRIMARY KEY,  
    X2 NOT NULL  
);
```

```
CREATE TABLE y OF y_type(  
    Y1 PRIMARY KEY,  
    Y2 WITH ROWID REFERENCES x  
);
```



Herança

- criação de hierarquias de tipos – herança simples
- tipos e métodos podem ser definidos como:
 - **não instanciáveis - NOT INSTANTIABLE**
 - tipos: definição de interfaces
 - métodos: definição da assinatura de métodos não implementados
 - devem ser sobrescritos nos subtipos
 - **não finais – NOT FINAL**
 - tipos: podem ser derivados (*default*: **FINAL**)
 - métodos: podem ser sobrescritos (*default*: **NOT FINAL**)

Ex:

```
CREATE OR REPLACE TYPE PESSOA AS OBJECT (  
    nome VARCHAR2(30) ,  
    data_nascimento DATE ,  
    endereco VARCHAR2(100) ,  
    -- método não pode ser sobrescrito  
    FINAL MEMBER FUNCTION idade RETURN NUMBER  
) NOT INSTANTIABLE NOT FINAL; /* tipo pode ser derivado,  
    mas não pode ser instanciado */
```

```
CREATE OR REPLACE TYPE ALUNO UNDER PESSOA(  
    nusp VARCHAR2(7)  
) FINAL; /* tipo pode ser instanciado, mas não pode ser derivado  
    */
```

Ex:

```
CREATE OR REPLACE TYPE PESSOA AS OBJECT (  
    nome VARCHAR2(30) ,  
    data_nascimento DATE ,  
    endereco VARCHAR2(100) ,  
    -- método não pode ser sobrescrito  
    FINAL MEMBER FUNCTION idade RETURN NUMBER  
) NOT INSTANTIABLE NOT FINAL; /* tipo pode ser  
    derivado, mas não pode ser instanciado */
```

```
CREATE OR REPLACE TYPE ALUNO UNDER PESSOA(  
    nusp VARCHAR2(7)  
) FINAL; /* tipo pode ser instanciado, mas não pode ser  
    derivado */
```


Ex:

```
-- object table - OK!
```

```
CREATE TABLE tabela_alunos OF ALUNO;
```

```
-- inserção (instanciação de ALUNO) - OK!
```

```
INSERT INTO tabela_alunos VALUES  
    (ALUNO('Leo', '01-01-1992', 'SP', '111'));
```

```
-- object table - permitido!
```

```
CREATE TABLE tabela_pessoas OF PESSOA;
```

```
-- inserção (instanciação de PESSOA) - NÃO PERMITIDO!!
```

```
INSERT INTO tabela_pessoas VALUES  
    (PESSOA('Leo', '01-01-1992', 'SP'));
```



Herança

- suporte a:
 - sobrescrita (*overriding*) de métodos
 - sobrecarga (*overloading*) de métodos
 - polimorfismo

Ex:

```
CREATE OR REPLACE TYPE retangulo AS OBJECT (  
    . . . . . ,  
    MEMBER FUNCTION area (x NUMBER, y NUMBER) RETURN NUMBER  
    ) NOT FINAL;
```

```
CREATE OR REPLACE TYPE quadrado UNDER retangulo (  
    . . . . . ,  
    -- método sobrecarregado  
    MEMBER FUNCTION area (x NUMBER) RETURN NUMBER  
    );
```

```
CREATE OR REPLACE TYPE losango UNDER retangulo (  
    . . . . . ,  
    -- método sobrescrito  
    OVERRIDING MEMBER FUNCTION area (x NUMBER, y NUMBER) RETURN NUMBER  
    );
```

Ex:

```
CREATE OR REPLACE TYPE retangulo AS OBJECT (  
    . . . . . ,  
    MEMBER FUNCTION area (x NUMBER, y NUMBER) RETURN NUMBER  
    ) NOT FINAL;
```

```
CREATE OR REPLACE TYPE quadrado UNDER retangulo (  
    . . . . . ,  
    -- método sobrecarregado  
    MEMBER FUNCTION area (x NUMBER) RETURN NUMBER  
    );
```

Mesmo método com assinatura diferente

```
CREATE OR REPLACE TYPE losango UNDER retangulo (  
    . . . . . ,  
    -- método sobrescrito  
    OVERRIDING MEMBER FUNCTION area (x NUMBER, y NUMBER) RETURN NUMBER  
    );
```

Mesmo método com mesma assinatura – cláusula overriding



Object View

- Mecanismo para desenvolver aplicações orientadas a objetos sem alterar o esquema relacional
- *Object View* é uma *object table* virtual
 - cada tupla é um “objeto virtual”

Object View

Tabela relacional original:

Aluno = {Nome, Nusp, Idade, DataNasc}

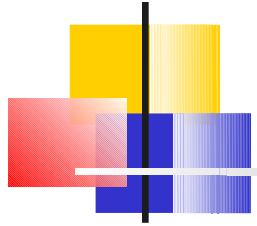
```
CREATE TYPE T_Aluno AS OBJECT (  
    nome VARCHAR2(30) ,  
    nusp NUMBER,  
    idade NUMBER,  
);
```

```
CREATE VIEW V_Aluno OF T_Aluno  
WITH OBJECT IDENTIFIER (nusp) AS  
    SELECT a.nome, a.nusp, a.idade  
    FROM Aluno a;
```



- Referências:

- *Object-Relational Developer's Guide*
- *SQL Reference*



PRÁTICA 11