Banco de Dados Objeto-Relacional

- Prof. Antonio Guardado
- Fatec IPIRANGA

Centro Paula Souza

GOVERNO DO ESTADO SÃO PAULO

Sumário

- Bancos de Dados Objeto-Relacional
  - Modelo de Dados Objeto Relacional
  - Linguagem de Consultas Objeto Relacional
- Visão geral dos conceitos de orientação a objetos disponíveis no ORACLE
- Extensões para criar Objetos Complexos
- Manipulação de objetos com SQL

Banco de Dados Objeto-Relacional

- SGBDs Objeto-Relacional combinam os benefícios do modelo Relacional com a capacidade de modelagem da abordagem OO
- Fornecem suporte para consultas complexas sobre dados complexos
- Atendem aos requisitos das novas aplicações e da nova geração de aplicações de negócios

Banco de Dados Objeto-Relacional
Modelos e Linguagens

- O modelo de dados OR é uma extensão do modelo Relacional
  - As extensões incluem mecanismos para permitir aos usuários estender o banco de dados com tipos e funções específicas da aplicação
- A linguagem de consulta OR é uma extensão da linguagem SQL para suportar o modelo de objetos
  - As extensões incluem consultas envolvendo objetos, atributos multivalorados, TADs, métodos e funções como predicados de busca em uma consulta

Modelo de Dados Objeto-Relacional

❶ Permite especificar e utilizar tipos abstratos de dados(TADs) da mesma forma que os tipos de dados pré-definidos

- TADs são tipos de dados definidos pelo usuário que encapsulam *comportamento* e *estrutura interna (atributos)*

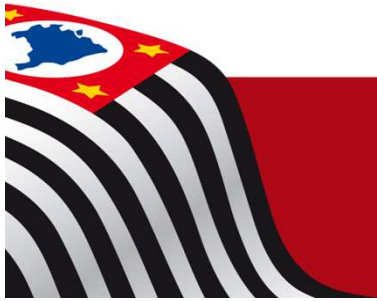
❷ A tabela convencional é estendida para permitir a referencia de objetos (referência de tipos), TADs e valores alfanuméricos como domínio de colunas

Modelo de Dados Objeto-Relacional

❸ Utiliza referências para representar conexões inter-objetos tornando as consultas baseadas em caminhos de referência mais compactas do que as consultas feitas com junção

❹ Herança é implementada organizando todos os tipos em hierarquias

❺ Utiliza os construtores *set*, *list*, *multiset* ou *array* para organizar coleções de objetos



Benefícios do Modelo de Dados Objeto-Relacional

- Nova Funcionalidade
  - Aumenta indefinidamente o conjunto de tipos e funções fornecidas pelo SGBD
- Desenvolvimento de aplicações simplificado
  - Reuso de código
- Consistência
  - Permite a definição de padrões, código reusável por todas as aplicações

Linguagem de Consultas para Bancos de Dados Objeto-Relacional

- O resultado de uma consulta ainda consiste de tabelas
  - Um SGBD Objeto-Relacional ainda é relacional pois suporta dados armazenados em tabelas formadas por linhas e colunas
- A linguagem de consultas para BDOR é uma extensão da linguagem SQL, utilizada para definição e manipulação de dados e consultas

SQL3

- É a base para muitos SGBDs OR (Oracle, Informix's Universal Server, IBM's DB2 Universal Database, entre outros)
- Também chamada de SQL:1999 e tem sido caracterizada como "SQL Orientada a Objetos"
- SQL 3 é muito mais do que SQL-92 incrementada com a tecnologia de OO. Envolve características adicionais que podem ser classificadas em:
  - Relacionais: novos tipos de dados, novos predicados
  - Orientadas a Objetos:tipos de dados definidos pelo usuário, definição de métodos, uso de referências

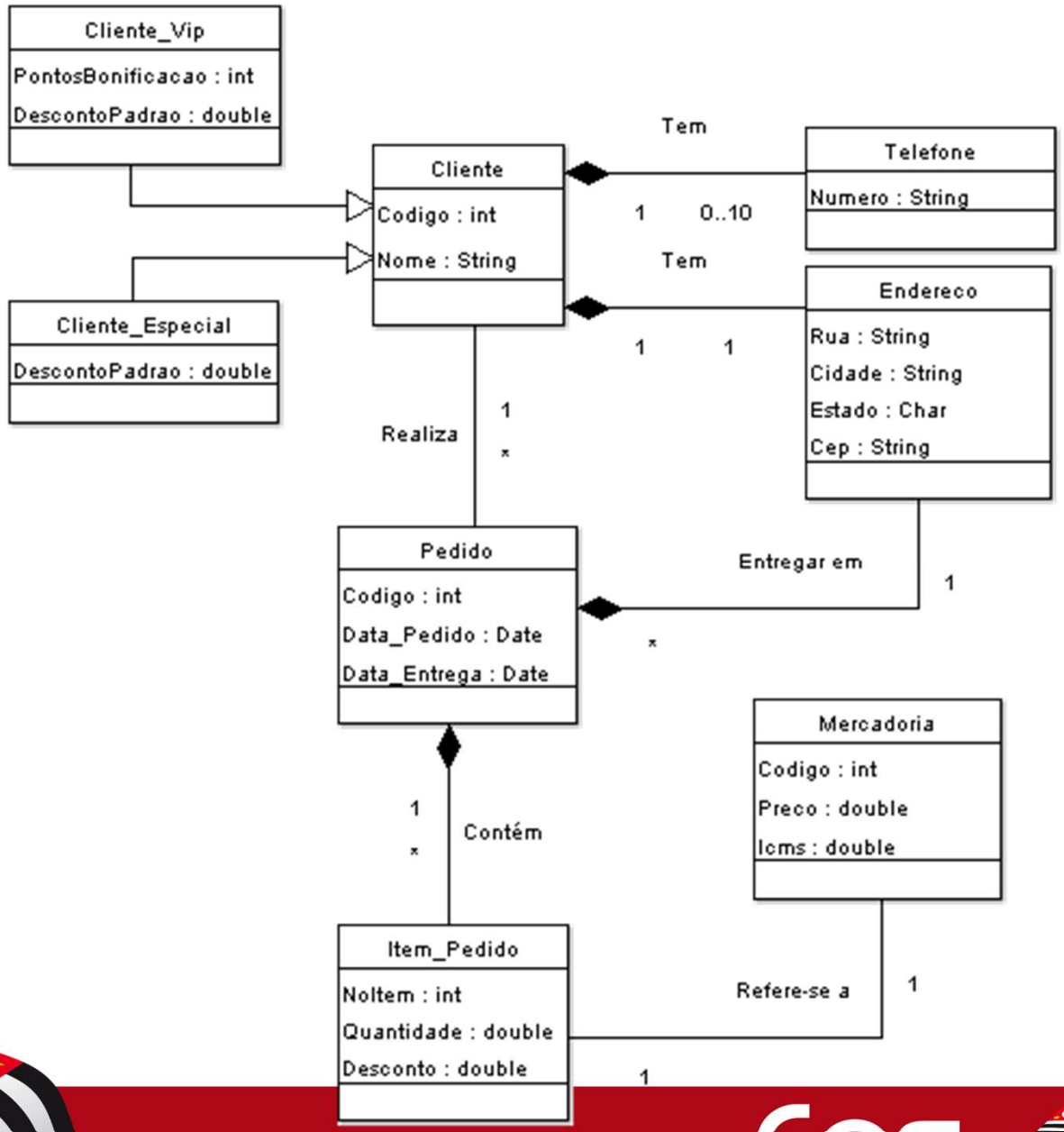
O ORACLE fornece um suporte completo para todos os três diferentes tipos de implementação:

Relacional.

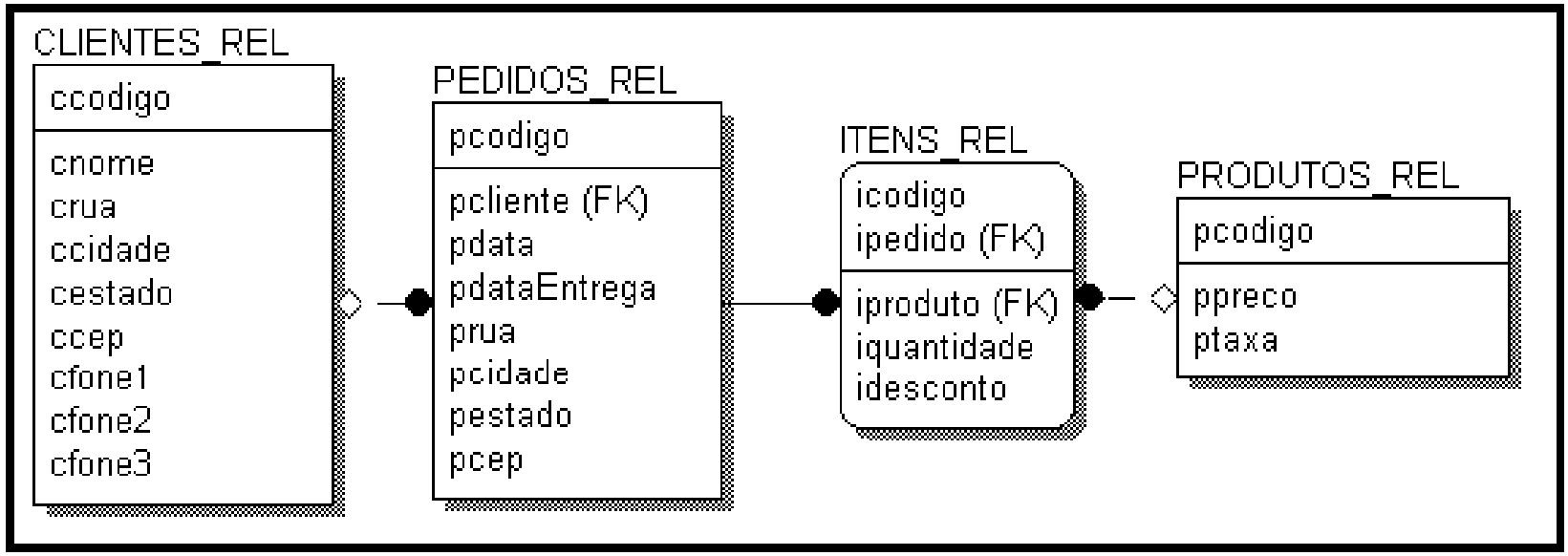
Objeto-relacional - Um banco de dados, tradicionalmente relacional, estendido para incluir os conceitos OO e estruturas como tipos de dados abstratos, nested tables e varying arrays.

Orientado a objetos - Um banco de dados orientado a objetos cujo projeto é, desde o seu início, desenvolvido com análise orientada a objetos.

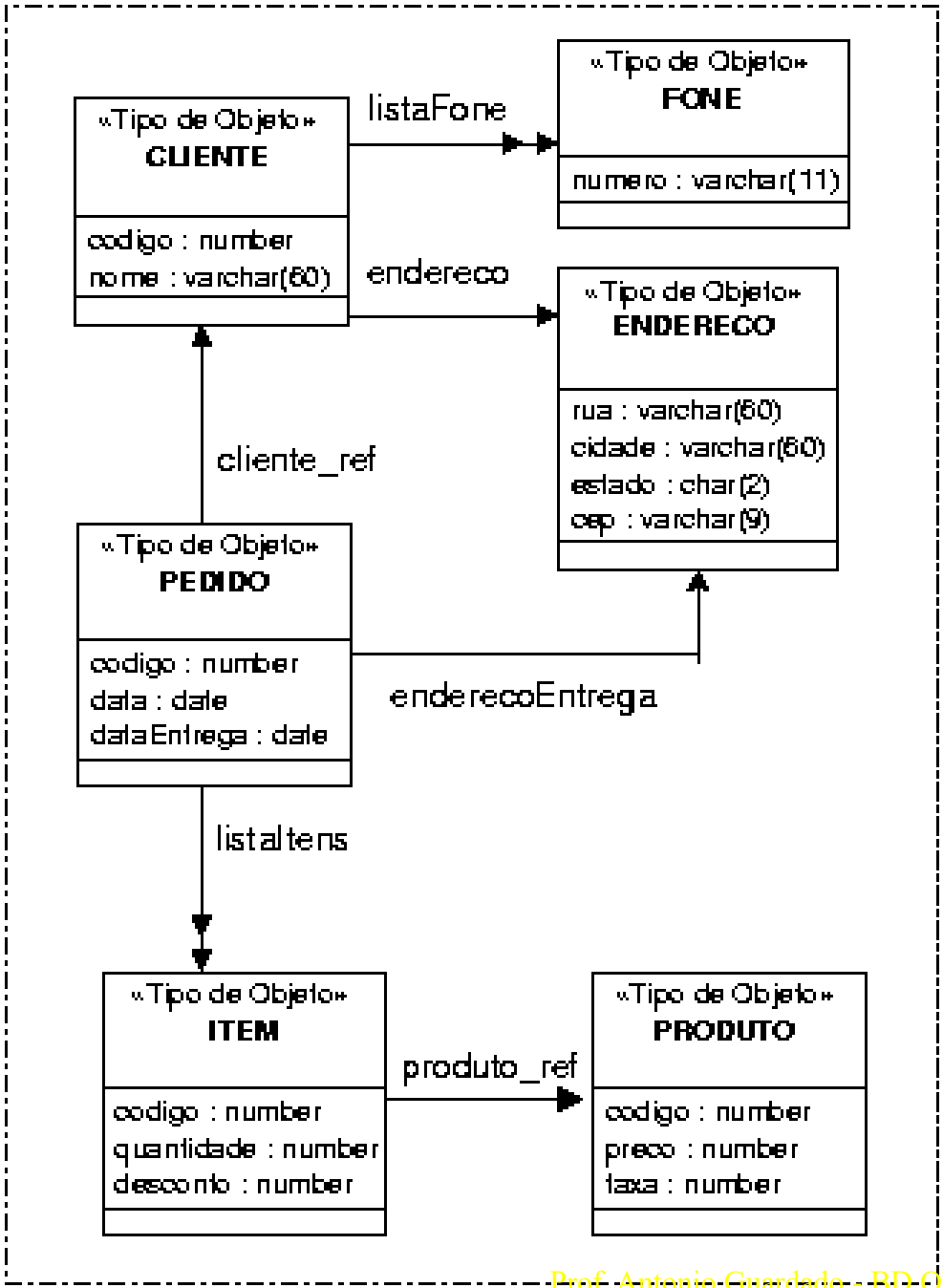
Exemplo - UML


Cliente_Vip
PontosBonificacao : int
DescontoPadrao : double
Cliente
Codigo : int
Nome : String
Tem
1
0..10
Telefone
Numero : String
Tem
1
1
Endereco
Rua : String
Cidade : String
Estado : Char
Cep : String
Cliente_Especial
DescontoPadrao : double
Realiza
1
*
Pedido
Codigo : int
Data_Pedido : Date
Data_Entrega : Date
Entregar em
1
*
Mercadoria
Codigo : int
Preco : double
Icms : double
1
*
Contém
Item_Pedido
NoItem : int
Quantidade : double
Desconto : double
Refere-se a
1
1

Esquema Relacional



Esquema Objeto Relacional



Definindo os Tipos

Definindo os tipos

```
CREATE TYPE PEDIDO AS OBJECT (
 codigo            NUMBER,
 cliente_ref       REF CLIENTE,
 data              DATE,
 dataEntrega       DATE,
 listaItens        ITEM_LISTA,
 enderecoEntrega ENDERECO );

CREATE TYPE ENDERECO AS OBJECT
(      rua     VARCHAR2(20),
       cidade VARCHAR2(10),
       estado CHAR(2),
       cep     VARCHAR2(10)        );

CREATE TYPE CLIENTE AS OBJECT
(    codigo        NUMBER,
       nome       VARCHAR2(200),
       endereco   ENDERECO,
       listaFone   FONE_LISTA   );
```

```
CREATE TYPE ITEM AS OBJECT
(   codigo   NUMBER,
    produto_ref  REF PRODUTO,
    quantidade   NUMBER,
    desconto     NUMBER    );

CREATE TYPE ITEM_LISTA AS TABLE
OF ITEM

CREATE TYPE PRODUTO AS OBJECT (
 codigo    NUMBER,
 Preco     NUMBER,
 Taxa      NUMBER  ) ;

CREATE TYPE FONE_LISTA AS
VARRAY(10) OF VARCHAR2(20);
```

Orientação a objetos no ORACLE a partir da 9i

O ORACLE versão 9i em diante oferece diferentes tipos de objetos:

Tipos de Objetos (TADs)

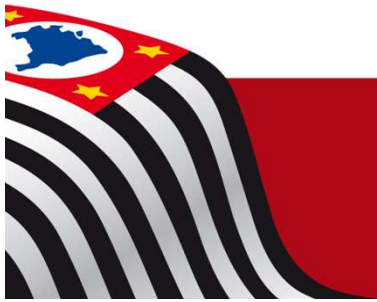
Nested Tables (Tabelas aninhadas)

VArrays (Varying Arrays)

Large Objects (LOBs)

References (REF)

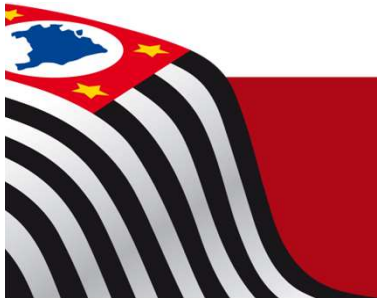
Object View (Visão de Objetos)



Tipos de Objetos (Object Types)

Objetos são abstrações de entidades do mundo real, como por exemplo, uma ordem de compra, um cliente, um produto.

Um tipo de objeto funciona como um molde para criação de objetos, através da atribuição de valores a essa estrutura de dados (classe).



Tipos de objetos

- Um Tipo de Objeto é um esquema de objeto com 3 componentes:
  - Nome
  - Atributos
  - Métodos
- Um tipo de objeto pode ser usado para:
  - Definir o domínio de atributos (“column object”) de tabelas
  - Definir o tipo dos atributos de TADs (“embedded object”)
  - Criar uma tabela de objetos

Tipos de objetos

```
create type ENDERECO_TY as object
(Rua         VARCHAR2(50),
 Cidade      VARCHAR2(25),
 Estado      CHAR(2),
 Cep         NUMBER);

create table PESSOAS
(Nome        VARCHAR2(25),
 Endereço    ENDERECO_TY);
```

ENDEREÇO_TY é usado para definir o tipo (domínio) da coluna Endereço da tabela PESSOAS

Tipos de objetos

```
create type ENDERECO_TY as object
(Rua         VARCHAR2(50),
 Cidade      VARCHAR2(25),
 Estado      CHAR(2),
 Cep         NUMBER);

create type PESSOA_TY as object
(Nome        VARCHAR2(25),
 Endereco    ENDERECO_TY);
```

ENDEREÇO_TY é usado para definir o tipo do atributo Endereco do tipo PESSOA_TY

Não é possível ocorrer uma inserção de dados em PESSOA_TY. Isso porque um tipo de objeto descreve dados, mas não os armazena.

Tipos de objetos

Para armazenar dados é necessário a criação de uma tabela a partir de um tipo de objeto.

```
create type PESSOA_TY as object
(Nome       VARCHAR2(25),
 CPF        NUMBER,
 Endereco   ENDERECO_TY);

create table PESSOAS of PESSOA_TY
(CPF       primary key );
```

A tabela **PESSOAS** irá armazenar dados com a estrutura do tipo **PESSOA_TY**

Tabelas no ORACLE a partir da versão 9i

Oracle suporta 2 tipos de tabelas:

- Tabela Relacional
- Tabela de Objetos

Uma tabela de objetos é um tipo especial de tabela que lida com objetos (“row objects”) e fornece uma visão relacional dos atributos desses objetos.

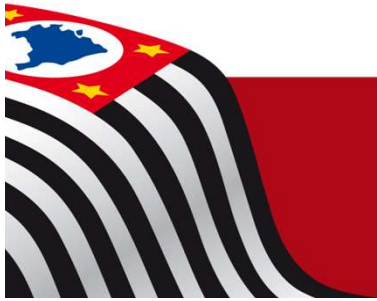


Tabela de objetos versus Tabela relacional

Uma tabela de objetos difere de uma tabela relacional em vários aspectos:

- **Cada linha de uma tabela de objetos possui um identificador de objeto (OID), definido pelo ORACLE quando a linha é inserida na tabela;**
- **Um OID é um ponteiro para um objeto "linha" (ROW Object);**
- **As linhas (row objects) de uma tabela de objetos podem ser referenciadas por outros objetos do banco de dados.**

Column Objects versus Row Objects

“Row Objects”

- Objetos armazenados em tabelas de objetos

“Column Objects”

- Objetos armazenados em colunas de tabelas relacionais ou como atributos de outros objetos (“embedded object”)

Quando objetos devem ser criados como row object ?

-objetos contidos em outros objetos e que possuem existência própria no banco de dados,

-Objetos compartilhados por mais de um objeto do banco de dados .

Tabela de objetos

```
create type PESSOA_TY as object
(Nome        VARCHAR2(25),
 CPF         NUMBER,
 Endereco    ENDERECO_TY);

create table PESSOAS of PESSOA_TY
(CPF       primary key );
```

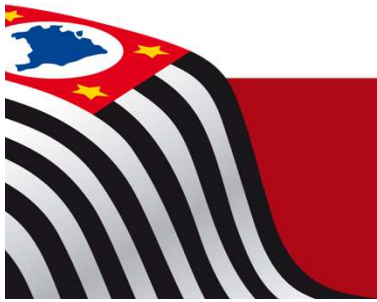
A tabela de objetos PESSOAS pode ser vista como:

- **Uma Tabela com uma única coluna:**
  - **cada linha é um objeto do tipo PESSOA.**
- **Uma Tabela com múltiplas colunas**
  - **Uma coluna para cada atributo do tipo PESSOA_TY**

Manipulando tabelas de objetos

Existem diferenças significativas no modo de utilização de uma tabela de objetos.

Cada linha dentro de uma tabela de objetos possuirá um OID, e essas linhas poderão ser referenciadas como objetos.



Inserção em tabelas de Objetos

- **Inserção em PESSOAS como uma tabela de uma única coluna.**

- Usa o método construtor ***PESSOA_TY*** que constrói novos objetos do tipo PESSOA_TY.

```
INSERT INTO PESSOAS VALUES
(PESSOA_TY('Ricardo Cabral' , 543875432,
        ENDERECO_TY('Cruz 57','Fortaleza', 'CE',  60160230)))
```

Métodos construtores para os tipos **PESSOA_TY** e **ENDERECO_TY**. O nome do método construtor tem o mesmo nome do tipo.

Inserção em tabelas de Objetos

- **Inserção em PESSOAS como uma tabela de uma múltiplas colunas.**

- - Oracle permite também fazer a inserção em PESSOAS como uma tabela de múltiplas colunas

```
INSERT INTO PESSOAS VALUES
('Ricardo Cabral',543875432,
    ENDERECO_TY('rua Cruz 57','Fortaleza','CE',60160230));
```

Método construtor para o tipo ENDERECO_TY

Seleção em tabelas

- Seleção em PESSOAS como uma tabela de múltiplas colunas.

```
select  *

 from PESSOAS;
```

```
==========================================================

NOME                CPF          ENDERECO(RUA, CIDADE, ESTADO, CEP)

------------------  -----------  ----------------------------------------------------

Ricardo Cabral      543875432    ENDERECO_TY('rua Cruz 57', 'Fortaleza',
'CE', 60160230)
```

Seleção em tabelas

- Seleção em PESSOAS como uma tabela de uma de uma única coluna.

```
SELECT VALUE (p)
FROM PESSOAS p
WHERE p.nome = 'Ricardo Cabral';
```

==========

```
VALUE(P)(NOME, CPF, ENDERECO(RUA, CIDADE, ESTADO, CEP))
```

```
PESSOA_TY('Ricardo Cabral', 543875432, ENDERECO_TY('rua Cruz 57',
                                        'Fortaleza', 'CE', 60160230))
```

Seleção em tabelas

```
SELECT  p.CPF
 FROM PESSOAS p
WHERE  p.nome = 'Ricardo Cabral';
```

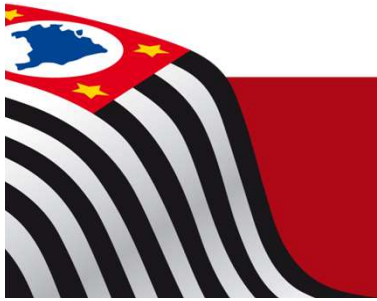
```
 CPF
-----------

543875432
```

Seleção em tabelas

```
SELECT p.endereco

FROM PESSOAS p

WHERE p.nome = 'Ricardo Cabral';

==========

ENDERECO(RUA, CIDADE, ESTADO, CEP)
----------
ENDERECO_TY('rua Cruz 57', 'Fortaleza', 'CE', 60160230)
```

Seleção em tabelas

```
select p.endereco.cidade

from PESSOAS p

where p.nome = 'Ricardo Cabral';

==========================================

ENDERECO.CIDADE
----------------------
Fortaleza
```



Atualização e Remoção em tabelas de Objetos

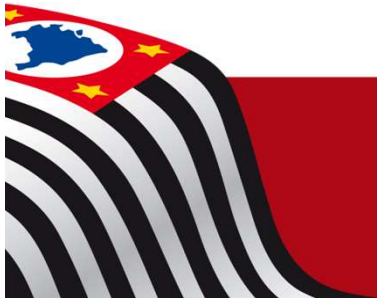
```
update PESSOAS p
Set p.endereco= ENDERECO_TY('rua Rios 57',
                            'Fortaleza', 'CE', 60160230)
where p.nome = 'Ricardo Cabral';
```

```
Delete from PESSOAS p
where p.nome = 'Ricardo Cabral';
```

Orientação a objetos no ORACLE

Extensões para criar objetos complexos

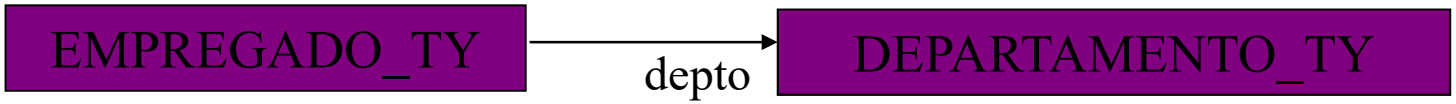
- TADs (Tipos de Objetos)
- Tipo Referência (REF)
- Tipo Coleção
  - Nested Tables (Tabelas aninhadas)
  - Varying Arrays
- Large Objects (LOBs)



Identificadores de objetos

- Uma tabela de objetos contém uma coluna gerada pelo SGBD contendo o OID do "row object". O oid de um objeto é único e imutável.
- Sobre essa coluna de OID é também criado automaticamente um índice para prover acesso eficiente sobre o objeto através do OID. A coluna de OID é equivalente a se ter uma coluna extra de 16 bytes para chave primária.
- Um OID permite que um "row object" seja referenciado em atributos de outros objetos ou em colunas de tabelas relacionais.
- Um tipo pré-definido **REF** é capaz de representar tais referências.

Referenciando Objetos



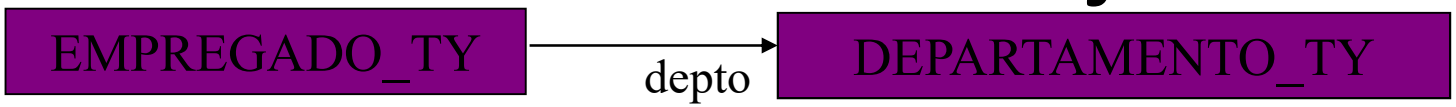

```
create type DEPARTAMENTO_TY as object
 (Nome        VARCHAR2(25),
   … );
```

```
create table DEPARTAMENTOS of DEPARTAMENTO_TY
  (
    … );
```

Os objetos do tipo **DEPARTAMENTO_TY** podem ser referenciados em *colunas de tabelas relacionais* ou em *atributos de outros objetos*.

```
create type EMPREGADO_TY as object
(Nome        VARCHAR2(25),
 CPF         NUMBER,
 depto       REF DEPARTAMENTO_TY);
```

Referenciando Objetos




```
create type DEPARTAMENTO_TY as object
 (Nome        VARCHAR2(25),
   …);
```

```
create table DEPARTAMENTOS of DEPARTAMENTO_TY
  (
    …);
```

Os objetos do tipo **DEPARTAMENTO_TY** podem ser referenciados em *colunas de tabelas relacionais* ou em *atributos de outros objetos*.

```
create table EMPREGADOS
(Nome        VARCHAR2(25),
 CPF         NUMBER,
 depto       REF DEPARTAMENTO_TY);
```

Tipo REF

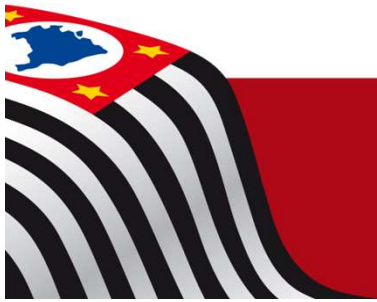
```
create type EMPREGADO_TY as object
(Nome        VARCHAR2(25),
 CPF         NUMBER,
 depto       REF DEPARTAMENTO_TY);
```

- Um objeto do tipo REF encapsula uma **referência para um "row object"** de um tipo de objeto especificado;
- O valor de um objeto do tipo REF é um "ponteiro lógico" para um row object.
- REFs e coleções de REFs são utilizados na modelagem de associações entre objetos. Por ex. o relacionamento entre uma ordem de compra e um cliente
- REFs constituem um mecanismo simples para navegar entre objetos. Pode-se utilizar a notação estendida de "pontos" para seguir os ponteiros **sem a necessidade de junções explícitas**

Tipo REF

```
create table EMPREGADOS
(Nome       VARCHAR2(25),
 CPF        NUMBER,
 Depto      REF DEPARTAMENTO_TY);
```

```
Nome                                      Nulo?   Tipo
----------------------------------------- ------- ------------------------
NOME                                              VARCHAR2(25)
CPF                                               NUMBER
DEPTO                                             REF OF DEPARTAMENTO_TY
```



Obtendo REFs

Como obter o valor do oid de um row object?

- Seleciona o objeto e aplica o operador REF

```
select REF(d)
  from DEPARTAMENTOS  d
 where d.Nome = 'pessoal';
==========================
```

O operador REF recebe como argumento um "row object" e retorna um valor de referência para esse objeto.

```
REF(D)
--------------------------------------------------------------------
0000280209FDC21397E1F846C1A6F503F32B03638AEEB71435C3214
B2687096513EAA861830040DBB20000
```

Obtendo REFs

```
insert into EMPREGADOS values (
   'Ricardo Cabral',
     543875432 ,
   (Select REF(d) from DEPARTAMENTOS d
      where   d.nome='pessoal')
)
```

Obtendo REFs

```
select * from EMPREGADOS;

NOME                 CPF         DEPTO

-------------------- ----------- ------------------------------------
Ricardo Cabral   543875432   0000220208 FDC21 397E1
F846C1A6F503F32B03638AEEB71435C3214B2687096513
EAA86183
```

Desreferenciando o “REF”

Como Acessar o valor do objeto referenciado por um REF?

O operador **DEREF** executa a função oposta de **REF** — recebe um valor de referência e retorna o valor de um “row object”. O **DEREF** toma como argumento o OID gerado para uma referência.

```
select DEREF(e.depto)
 from EMPREGADOS e  where e.nome = 'Ricardo Cabral';
==========================================

DEREF(E.DEPTO)(NOME, ...)
------------------------------------------------------------
DEPARTAMENTO_TY('pessoal',...)
```

Desreferenciando o “REF”

Desreferenciamento Implícito

```
select e.depto.nome
  from EMPREGADOS e
  where e.nome = 'Ricardo Cabral';

DEPTO.NOME
------------------------
pessoal
```

Operador VALUE

VALUE mostra os dados no mesmo formato que **DEREF** irá usá-los.

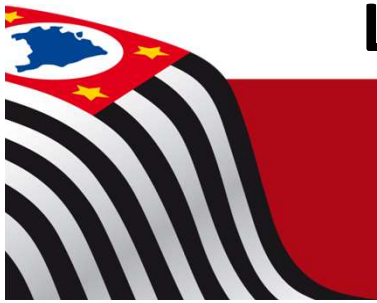
```
select value(p)
from PESSOAS p
where p.nome = 'Ricardo Cabral';
```

```
VALUE(P)(NOME, CPF, ENDERECO(RUA, CIDADE, ESTADO, CEP))
--------------------------------------------------------------------------------
PESSOA_TY('Ricardo Cabral', 543875432, ENDERECO_TY('rua Rios 57',
                                                   'Fortaleza', 'CE', 60160230))
```

Orientação a objetos no ORACLE

Extensões para criar objetos complexos

- **TADs (Tipos de Objetos)**
- **Tipo Referência (REF)**
- **Tipo Coleção**
  - **Nested Tables (Tabelas aninhadas)**
  - **Varying Arrays**
- **Large Objects (LOBs)**



Tipo Coleção

Coleções modelam:

- atributos multivalorados

- relacionamentos 1-m

O ORACLE oferece dois tipos de coleções:

- VARRAYS

- NESTED TABLES.

Coleções

- **Varrays vs. Nested tables**

- "Varrays" são coleções ordenadas e "limitada"; "nested tables" são coleções não ordenadas e que não tem limite no número de linhas.

- Varrays são armazenadas como objetos "contínuos" . Nested tables são armazenadas em uma tabela onde cada elemento é mapeado em uma linha na tabela de armazenamento.

- Se é necessário eficiência na execução de consultas sobre coleções, então é recomendado o uso de nested tables.

Nested Table

- É uma tabela que é representada como uma coluna dentro de outra tabela.
- É um conjunto não ordenado de elementos do mesmo tipo.
- Tem uma única coluna e o tipo da coluna é um tipo pré-definido ou um tipo de objeto.

```
create type  TELEFONES_NT as table of VARCHAR2(14)

create table  EMPREGADOS
(Nome         VARCHAR2(25),
 CPF          NUMBER,
 Telefones    TELEFONES_NT )
NESTED TABLE Telefones store as TELEFONES_ST;
```

Nested Table *Tipo Tabela*

Nested Table

- Representando Associações 1-m


```
create type EMPREGADO_TY as object
(Nome      VARCHAR2(25),
 CPF       NUMBER);

create type  EMPREGADOS_NT as table of EMPREGADO_TY
create type DEPARTAMENTO_TY as object
(Nome      VARCHAR2(25),
  Empregados    EMPREGADOS_NT );

create table DEPARTAMENTOS of DEPARTAMENTO_TY
NESTED TABLE Empregados store as EMPREGADOS_ST;
/
```

Tipo "TABLE"

- Pode ser usado como:
  - Tipo de uma coluna de uma tabela relacional
  - Tipo de um atributo de outro tipo de objeto.

```
create type  EMPREGADOS_NT as table of EMPREGADO_TY
create type DEPARTAMENTO_TY as object
(Nome          VARCHAR2(25),
 Empregados  EMPREGADOS_NT );
```

O valor do atributo **Empregados** *é uma tabela com um única coluna e o tipo da coluna é* **EMPREGADO_TY**

A "nested table" Empregados também pode ser vista com uma tabela de múltiplas colunas (nome, CPF, telefones)

"Storage Table" (Tabela de Armazenamento)

- Uma definição de um tipo "TABLE" não aloca espaço de armazenamento para os objetos da tabela.
- *Quando uma nested table é criada?*

```
create type  EMPREGADOS_NT as table of EMPREGADO_TY

create type DEPARTAMENTO_TY as object
(Nome          VARCHAR2(25),
Empregados  EMPREGADOS_NT );

create table DEPARTAMENTOS of DEPARTAMENTO_TY
 nested table Empregados store as EMPREGADOS_ST;
```

Especifica **EMPREGADOS_ST** como tabela de armazenamento de todos as nested tables **Empregados** dos objetos em DEPARTAMENTOS

Armazenamento de Nested Table

DEPARTAMENTOS

Nome	.....	Empregados
		A
		B
		C
		D

Tabela de Armazenamento

NESTED_TABLE_ID	Values
B	
B	
C	
A	
D	
C	
D	
A	

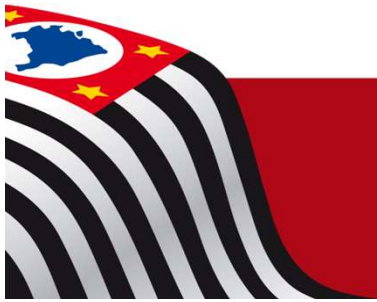
- Oracle armazena as linhas de uma nested table em uma tabela separada (tabela de armazenamento)

- Oracle usa nested_table_id para relacionar linhas na tabela de armazenamento com a nested table correspondente.

Manipulando Nested Tables

```
create type  TELEFONES_NT as table of VARCHAR2(14)

create table  EMPREGADOS
(Nome       VARCHAR2(25),
 CPF        NUMBER,
 Telefones  TELEFONES_NT )
NESTED TABLE Telefones store as TELEFONES_ST;
```

```
insert into EMPREGADOS values
('Jane Souza', 12347645620,
TELEFONES_NT('4561098','99960056'));
```



Consultando Coleções

```
create type  TELEFONES_NT as table of VARCHAR2(14)

create table  EMPREGADOS
(Nome       VARCHAR2(25),
 CPF        NUMBER,
 Telefones  TELEFONES_NT )
NESTED TABLE Telefones store as TELEFONES_ST;
```

```
Select *
From TABLE(Select e.telefones
           from EMPREGADOS e
           where e.nome = 'Jane Souza')

COLUMN_VALUE
--------------
4561098
99960056
```

Nested tables podem ser consultadas usando a expressão **TABLE**

Manipulando Nested Tables

```
create type  TELEFONES_NT as table of  VARCHAR2(14)

create table  EMPREGADOS
(Nome       VARCHAR2(25),
 CPF        NUMBER,
 Telefones  TELEFONES_NT )
NESTED TABLE Telefones store as TELEFONES_ST;
```

```
insert into TABLE(Select e.telefones
                  from EMPREGADOS e
                  where e.nome = 'Jane Souza')
 values ('99450057');
```

Manipulando Nested Tables

Para excluir uma determinada nested table, deve-se atribuir um valor NULL na coluna da linha de sua tabela pai, como no exemplo a seguir:

```
UPDATE EMPREGADOS e
SET e.telefones = NULL
WHERE CPF = 12347645620 ;
```

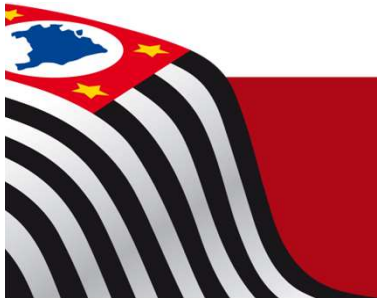
Uma vez excluída a nested table, não se pode inserir valores na mesma antes que seja recriada.

```
UPDATE EMPREGADOS e
SET e.telefones = telefones_NT()
WHERE CPF = 12347645620 ;
```

Atualizações em Nested tables

A tabela pode ser criada já com objetos

```
UPDATE EMPREGADOS e
SET e.telefones = TELEFONESc_NT('2449876', '98877805')
WHERE CPF = 12347645620 ;
```



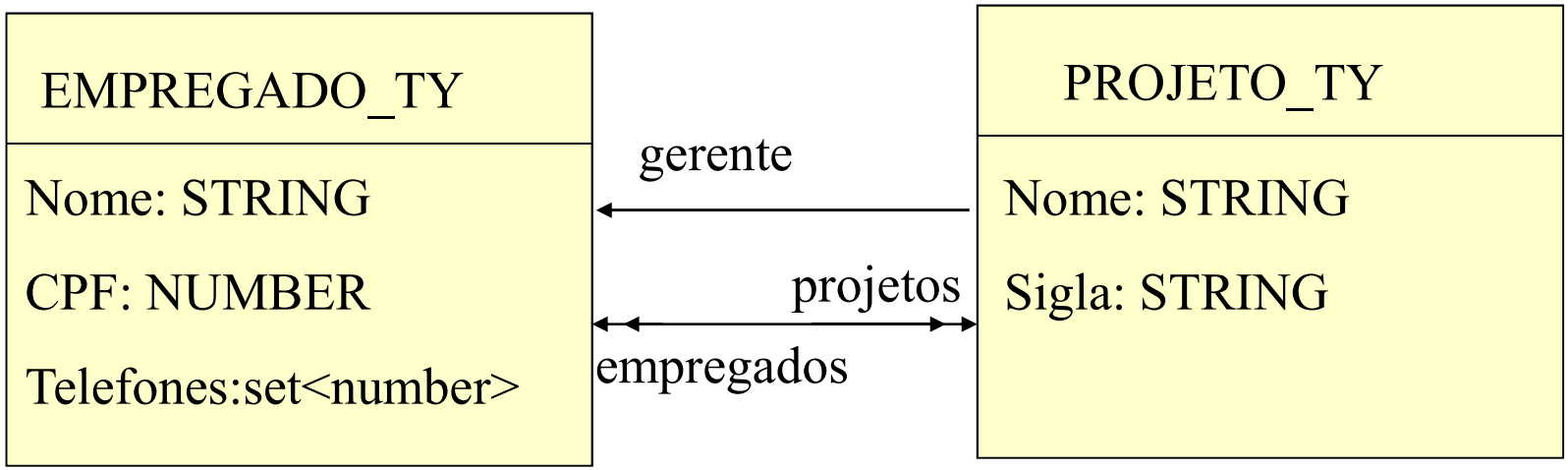
Manipulando Coleções Aninhadas

```
create type  EMPREGADOS_NT as table of EMPREGADO_TY;

create type DEPARTAMENTO_TY as object
(Nome       VARCHAR2(25),
 Empregados    EMPREGADOS_NT );

create table DEPARTAMENTOS of DEPARTAMENTO_TY
NESTED TABLE Empregados store as emp_tab;
```

```
SELECT d.nome, e.*  FROM DEPARTAMENTOS d, TABLE(d.Empregados) e;

SELECT d.*, e.*
FROM DEPARTAMENTOS d, TABLE(d. Empregados)(+) e;
```

Outer-join

Outer-join --> constará no resultados os DEPARTAMENTOS d tais que d.empregados é NULL or vazia. Neste caso as colunas correspondentes a d.Empregados terá valor NULL


EMPREGADO_TY
Nome: STRING
CPF: NUMBER
Telefones:set<number>
gerente
projetos
empregados
PROJETO_TY
Nome: STRING
Sigla: STRING

Manipulando Coleções Aninhadas

```
CREATE TYPE PROJETO_TY;

CREATE TYPE PROJETO_NT AS TABLE OF REF PROJETO_TY;

CREATE TYPE EMPREGADO_TY AS OBJECT (
  id          NUMBER(4),
  nome        VARCHAR2(50),
  endereco    ENDERECO_TY,
  projetos    PROJETO_NT
);


CREATE TABLE empregados OF EMPREGADO_TY
(id PRIMARY KEY)
NESTED TABLE projetos STORE AS PROJETOS_ST;
```

Tipo incompleto

Manipulando Coleções Aninhadas

```
create type  EMPREGADOS_NT as table of REF EMPREGADO_TY;

CREATE TYPE projeto_TY AS OBJECT (
  codigo          NUMBER(4),
  nome            VARCHAR2(50),
  endereco        ENDERECO_TY,
  gerente         REF EMPREGADO_TY,
  empregados      EMPREGADOS_NT
);


CREATE TABLE projetos OF projeto_TY
   (codigo PRIMARY KEY)
    NESTED TABLE empregados STORE AS EMPREGADOS_ST;
```

Manipulando Nested Tables

- Um atributo/coluna do tipo coleção pode ser consultada usando a expressão TABLE.

```
SELECT  *
FROM TABLE(SELECT t.projetos
             FROM empregados t
             WHERE t.id = 1000);
```

```
CREATE TYPE projeto_TY AS OBJECT (
   codigo          NUMBER(4),
   nome            VARCHAR2(50),
   endereco        ENDERECO_TY,
   gerente         REF EMPREGADO_TY,
   empregados      EMPREGADOS_NT
);
```

```
INSERT INTO projetos VALUES
(  1101,
   'Projeto Alfa',
    ENDERECO_TY('rua Cruz 57','Fortaleza', 'CE', 60160230),
   (SELECT REF(p) FROM empregados p WHERE id = 0001),
   EMPREGADOS_NT
   (
      (SELECT REF(p) FROM empregados p  WHERE p.id = 0001),
      (SELECT REF(p) FROM empregados p WHERE p.id = 0002),
      (SELECT REF(p) FROM empregados p WHERE p.id = 0003)
   )
);
```

```
CREATE TYPE projeto_TY AS OBJECT (
   codigo          NUMBER(4),
   nome            VARCHAR2(50),
   endereco        ENDERECO_TY,
   gerente         REF EMPREGADO_TY,
   empregados      EMPREGADOS_NT
);
```

```
INSERT INTO TABLE(SELECT p.empregados
                  FROM  PROJETOS  p
                  WHERE p.codigo= 1101)
VALUES ( (SELECT REF(e)
          FROM EMPREGADOS e
          WHERE e.id = 0004)
        );
```

Manipulando Nested Tables

```
UPDATE TABLE (SELECT p.empregados
              FROM  PROJETOS  p
              WHERE p.codigo= 1101) e

SET e.endereço = endereço_TY('rua Cruz 57''Fortaleza', 'CE',
                             60.160-230)
WHERE e.id = 8976543;
```

```
UPDATE TABLE (SELECT p.empregados
              FROM  PROJETOS  p
              WHERE p.codigo= 1101)

SET VALUE(e) = empregado_ty( .... )
WHERE e.código = 1;
```

NESTE EXEMPO NÃO É POSSÍVEL FAZER ATUALIZAÇÕES NO ATRIBUTOS DOS OBJETOS DA NESTED TABLE, POIS ESTES SÃO DE REFERÊNCIA

Manipulando Nested Tables

```
Select e.column_value.nome

From  PROJETOS  p, TABLE(p.empregados) e

WHERE p.nome = 'Projeto Alfa';
```

```
Select e.column_value.nome

From TABLE( Select p.empregados

                  from  PROJETOS  p

                  WHERE p.nome = 'Projeto Alfa') e;
```

São equivalentes??

Manipulando Nested Tables

```
INSERT INTO TABLE (SELECT e.projetos
                   FROM   empregados  e
                   WHERE  e.id = 001)
VALUES ((SELECT REF(p) FROM projetos p  WHERE p.codigo=1101));
```

```
UPDATE TABLE(SELECT e.projetos
             FROM   empregados e
             WHERE  e.id = 001)  p
  SET VALUE(p) = projeto_ty(1, 'Projeto Jupiter')
  WHERE p.codigo = 1;
```

NESTE EXEMPO NÃO É POSSÍVEL FAZER ATUALIZAÇÕES NO ATRIBUTOS DOS OBJETOS DA NESTED TABLE, POIS ESTES SÃO DE REFERÊNCIA

Manipulando Coleções Aninhadas

```
DELETE FROM TABLE(SELECT e.projetos
                  FROM    empregados e
                  WHERE e.id = 0001) p
WHERE p.column_value.codigo = 1101;
```

Restrições Declarativas em SQL-92

- DOMÍNIO
- NOT NULL
- DEFAULT
- CHECK
- CHAVE PRIMÁRIA
- UNIQUE
- REFERENCIAL

Restriçoes em Tabelas de Objetos

- Restrições de Tabela
- Restrições de Coluna

```
CREATE TABLE EMPREGADOS OF empregado_TY(
    CPF                 PRIMARY KEY,
    nome                NOT NULL,
    endereco            NOT NULL)

    NESTED TABLE projetos STORE AS PROJETOS_ST;
);
```

Restrições sobre REFs

REFs sem restricão podem armazenar referencias para objetos contidos em qualquer tabela cujos objetos sejam do mesmo tipo do REF.

- **Restrições de integridade referencial**
- **Restrições de escopo**

Restrição Referencial

```
CREATE TABLE Projetos OF PROJETO_TY (
   codigo     PRIMARY KEY,
   nome       NOT NULL,
   gerente    REFERENCES empregados ON DELETE SET NULL)
NESTED TABLE empregados STORE AS EMPREGADOS_ST
```

*Especifica que os REFS armazenados em **gerente** devem apontar para um row object em **empregados***

- Essa forma de restrição sobre colunas REF garante que sempre exista um "row object" para o REF (semelhante a chave estrangeira).
- Deve ser usada sempre que possível pois é a única forma de garantir a existência de um "row object" para um REF.
- Contudo, não é possível especificar restrições de integridade referencial sobre REFs que encontram-se em "nested tables".

Restrições de Escopo

```
CREATE TABLE EMPREGADOS OF empregado_TY
    ...
    NESTED TABLE projetos STORE AS PROJETOS_ST;
```

```
ALTER TABLE PROJETOS_ST ADD (
        SCOPE FOR (column_value) IS PROJETOS );
```

- REFs com escopo não garantem que os “row objects” existam(pode haver referências pendentes); apenas garantem que a tabela de objetos referenciada existe.
- REFs sem escopo são úteis se o projeto da aplicação requisita que os objetos referenciados possam estar distribuídos em múltiplas tabelas.
- Em geral, deve-se usar REFs com escopo sempre que possível, pois REFs com escopo são armazenados de forma mais eficiente.

Restrições em Nested Tables

- Restrição de “SCOPE”

- Restrição UNIQUE

```
CREATE UNIQUE INDEX PROJETOS_ST_idx ON
 PROJETOS_ST(nested_table_id, column_value);
```

Tipos e Subtipos

CRIANDO SUBTIPOS

```
CREATE TYPE Person_typ AS OBJECT
( ssn NUMBER,
  name VARCHAR2(30),
  address VARCHAR2(100)) NOT FINAL;
```

Permite que subtipos possam ser derivados deste tipo

```
CREATE TYPE Student_typ UNDER Person_typ
( deptid NUMBER,
  major VARCHAR2(30)) NOT FINAL;
```

- Cria **Student_typ** como um subtipo de **Person_typ**
- **Student_typ** herda todos os atributos e métodos (declarados e herdados) de **Student_typ**
- Novos atributos e métodos podem ser declarados no subtipo

Tipos e Subtipos

Um tipo pode ter vários subtipos

```
CREATE TYPE Employee_typ UNDER Person_typ
( empid NUMBER,
  manager VARCHAR2(30));
```

Um subtipo pode ser subtipo de outro subtipo

```
CREATE TYPE PartTimeStudent_typ UNDER
Student_typ
( numhours NUMBER);
```

Overloading

```
CREATE TYPE MyType_typ AS OBJECT (...,
MEMBER PROCEDURE foo(x NUMBER), ...) NOT FINAL;


CREATE TYPE MySubType_typ UNDER MyType_typ (...,
MEMBER PROCEDURE foo(x DATE),
STATIC FUNCTION bar(...)...
...);
```

Métodos que têm o mesmo nome mas diferentes assinaturas são chamados de *Overload*

MySubType_typ contém duas versões do método foo( ): Uma herdada com parâmetro NUMBER, e a nova com parâmetro DATE.

Overriding Methods

```
CREATE TYPE MyType_typ AS OBJECT (...,
MEMBER PROCEDURE Print(),
FINAL MEMBER FUNCTION foo(x NUMBER)...
) NOT FINAL;
CREATE TYPE MySubType_typ UNDER MyType_typ
(...,
OVERRIDING MEMBER PROCEDURE Print(),
...);
```

Quando um subtipo "overrides" um método, a nova versão é executada quando instâncias do subtipo invocam o método

Substituindo Tipos em uma Hierarquia de Tipos

Substitutability : é a habilidade (polimorfismo) de uma instância de um subtipo poder tomar o lugar de uma instância de um supertipo

```
CREATE TYPE Book_typ AS OBJECT
( title VARCHAR2(30),
  author Person_typ); /* substituível */
```

É substituível por uma instância de **Student_typ** or **Employee_typ.**

```
Book_typ('My Oracle Experience',
                    Employee_typ(12345, 'Joe',
'SF', 1111, NULL))
```

```
SELECT TREAT(author AS Employee_typ).empid
FROM Books_v;
```

Substituindo Tipos em uma Hierarquia de Tipos

```
CREATE TYPE Person_typ AS OBJECT
( ssn NUMBER,
name VARCHAR2(30),
address VARCHAR2(100)) NOT FINAL;
```

```
CREATE TYPE Student_typ UNDER Person_typ
( deptid NUMBER,
  major VARCHAR2(30)) NOT FINAL;
```

```
CREATE TYPE PartTimeStudent_typ UNDER Student_typ
( numhours NUMBER);
```

Substituindo Tipos em uma Hierarquia de Tipos

```
CREATE TABLE persons OF Person_typ;
INSERT INTO persons
VALUES (Person_typ(1243, 'Bob', '121 Front
St'));

INSERT INTO persons
VALUES (Student_typ(3456, 'Joe', '34 View',
12, 'HISTORY'));

INSERT INTO persons
VALUES (PartTimeStudent_typ(5678, 'Tim', 13,
'PHYSICS', 20));
```

Substituindo Tipos em uma Hierarquia de Tipos

```
CREATE TABLE books
   (title varchar2(100),
     author Person_typ);
INSERT INTO books
VALUES('An Autobiography', Person_typ(1243, 'Bob'));

INSERT INTO books
VALUES('Business Rules', Student_typ(3456, 'Joe', 12,
'HISTORY'));

INSERT INTO books
VALUES('Mixing School and Work',
                PartTimeStudent_typ(5678, 'Tim', 13,
'PHYSICS', 20));
```

Substituindo Tipos em uma Hierarquia de Tipos

```
CREATE TABLE catalog (book Book_typ, price NUMBER)
COLUMN book NOT SUBSTITUTABLE AT ALL LEVELS;
```

```
CREATE TABLE Student_books OF Book_typ
COLUMN author IS OF (ONLY Student_typ);
```

```
SELECT name, TREAT(VALUE(p) AS Student_typ).major
FROM persons p;
```

```
NAME      MAJOR
---- ----    -----------
Bob              null
Joe              HISTORY
Tim              PHYSICS
```

Substituindo Tipos em uma Hierarquia de Tipos

```
SELECT VALUE(p) FROM persons p
WHERE VALUE(p) IS OF (Student_typ);
```

```
Value(p)
--------
Student_typ('Joe', 3456, 12, 10000)
PartTimeStudent_typ('Tim', 5678, 13, 1000, 20)
```

```
SELECT b.title title, b.author author FROM books b
WHERE b.author IS OF (ONLY Student_typ);
```

```
TITLE                AUTHOR
-------------------  ------------------------------ ------
BusinessRules        Student_typ('Joe', 3456, 12, 10000)
```

Métodos

São Funções ou Procedimentos que declaramos na definição de um tipo de objeto para implementar o comportamento que você deseja para as instâncias do tipo.

Existem 3 tipos de Método:

-**Member**: implementam as operações das instâncias do tipo

- **Static**: São invocados no tipo de objeto, não para as instâncias. Não tem o parâmetro SELF.

- **Construtores**: São os métodos que fabricam novos objetos. Cada tipo de objeto tem um método construtor definido pelo sistema. O nome do método construtor é o nome do tipo