Tipos específicos da “objects option” de Oracle8

- **Além dos tipos básicos de SQL (“built-in data types”), Oracle8 oferece dois tipos definidos pelo usuário (“user defined data types”)**
- **Tipos definidos pelo usário:**
  - tipos de coleções
  - tipos de objetos

Tipo de objeto

- **Conceito equivalente ao de classe em OO**
- **Definição envolve:**
  - o *nome* do tipo de objeto
  - a lista de *atributos* do objeto

    um atributo pode ser de um tipo definido pelo usuário ou um tipo básico de SQL
  - a lista de *métodos* do objeto
    - método poder ser uma função ou uma prcedure
    - método pode ser escrito em
      - PL/SQL (linguagem específica da Oracle) e armazenado na BD
      - C++ e armazenado externamente a BD

Definição de tipo

- **Um tipo é criado pela instrução SQL CREATE TYPE**
- **Exemplo**

```
CREATE TYPE pessoa_externa AS OBJECT
   (nome VARCHAR2(30),
    fone VARCHAR2(20)
   ) ;
```

Objeto de coluna

- **Um tipo definido pelo usuário pode ser usado como tipo de coluna em uma tabela**

```
CREATE TABLE contatos (
  contato     pessoa_externa
  data        DATE ) ;
```

Referência a objetos coluna

- **Usa-se a notação com pontos**

```
SELECT contato.nome
FROM contatos
WHERE data < '01/01/1998';
```

- **Notação pode ser ambígua, caso exista esquema com nome "contato"**
  - Neste caso

    `contato.nome`

    poderia ser a tabela "nome" dentro do esquema "contato"

Referência a nomes de colunas com objetos

- Exemplo de ambiguidade entre nome de esquema e de tabela
- Exemplo:

```
CREATE TYPE type1 AS OBJECT (a NUMBER)
/
CREATE TABLE tab1 (tab2 type1)
/
CREATE TABLE tab2 (x NUMBER)
/
SELECT * FROM tab1 s
   WHERE EXISTS (SELECT * FROM s.tab2
                      WHERE x = s.tab2.a)
/
```

Este alias têm o mesmo nome de um esquema definido para a BD

Referencia o componente a de tab2 dentro de tab1

Ambiguidade de nomes

- **Problema:**
  - o que acontece quando uma coluna denominada “a” é adicionada a tabela “tab2”

```
CREATE TYPE type1 AS OBJECT (a NUMBER)
/
CREATE TABLE tab1 (tab2 type1)
/
CREATE TABLE tab2 (x NUMBER)
/
SELECT * FROM tab1 s
   WHERE EXISTS (SELECT * FROM s.tab2
                 WHERE x = s.tab2.a)
/
```

Evitando ambiguidades de nomes

- Recomendação

  sempre trabalhar com redefinição de nomes de tabelas (aliases)

```
SELECT * FROM s.tab1 p1
   WHERE EXISTS (SELECT * FROM s.tab2 p2
                 WHERE p2.x = p1.tab2.a);
```

Métodos

- **Oracle8 permite a definição de métodos**

```
CREATE TYPE ordem_compra AS OBJECT
   ( id NUMBER,
     contato     pessoa_externa,
     ...,
     MEMBER FUNCTION calc_valor RETURN NUMBER ) ;
```

Chamada de métodos

- Métodos são chamados com a mesma sintaxe de linguagens de POO como java ou C++
- Caso “x” seja uma variável que representa um objeto do tipo “ordem_compra”

```
x.calc_valor()
```

é a chamada do método “calc_valor” sobre este objeto

Construtor

- **Todo objeto tem implicitamente definido o método *construtor***
- **Construtor leva o mesmo nome que o tipo de objeto**

```
ordem_compra
   ( 127,
     pessoa_externa
      ('José Silva','051-3333333')
     ...)
```

é a chamada do construtor de "ordem_compra"

```
CREATE TYPE ordem_compra AS OBJECT
   ( id NUMBER,
     contato     pessoa_externa,
     ...,
     MEMBER FUNCTION calc_valor RETURN NUMBER ) ;
```

Métodos de comparação

- **Para objetos definidos pelo usuário, o SGBD não sabe o que é igualdade (operador =) e ordem (operador > e outros)**
- **Para definir estes operadores há dois métodos:**
  - *map*

    compara dois objetos e define se são iguais ou diferentes (sem estabelecer relação de ordem)
  - *order*

    compara dois objetos e define se são iguais ou diferentes (estabelece relação de ordem)

    e

    retorna

    -1 (primeiro é menor)

    0 (iguais)

    1 (primeiro é maior)

Comparação implícita

- **Caso nenhum método seja definido,**
  - é usado um método *map* com a seguinte funcionalidade

    se todos atributos são diferentes de NULL e iguais, os objetos são iguais

    se há um atributo não vazio diferente nos dois, os objetos são diferentes

    caso contrário a comparação não é determinada (NULL)

Tabelas de objetos

- **Objetos podem ser usados como valores de campos (visto acima)**
- **Isso eqüivale a SQL/3**
- **Objetos também podem ser usados como linhas de uma tabela:**
  - tabela de objetos (*object table*)
- **Criação com sintaxe especial do CREATE TABLE**

```
CREATE TABLE pessoa_externa_t OF pessoa_externa ;
```

Uso de tabelas objeto em SQL

- **Tabelas objetos podem ser referenciadas em SQL**

```
INSERT INTO pessoa_externa_t VALUES (
      'José Silva',
      '051-3333333' ) ;


SELECT VALUE(p) FROM pessoa_externa_t p
       WHERE p.nome = "José Silva" ;
```

OIds e REFs

- **Oracle8 implementa os conceitos de:**
  - OId (*object identifier*)
    - identificador únivoco de cada objeto gerado pelo SGBD
    - não é alterável e não deve ser usado como atributo
  - REF
    - tipo de dado pré-definido
    - implementa uma referência a um OId
    - usado para implementar associações entre objetos
    - operações possíveis
      - acesso ao objeto referenciado
      - atribuição a outro REF
      - atribuição a NULL

REF com escopo

- **Como no SQL/3, um REF pode ter definido um *escopo***
- **Escopo:**
  - restringe um REF para referenciar objetos em uma determinada tabela
  - permite implementação mais eficiente

Dereferenciando

- **Acessar o objeto referenciado por um REF chama-se *dereferenciar***
- **Existe um operador de "dereferencia"**
- **Exemplo:**

```
CREATE TYPE person AS OBJECT (
  name    VARCHAR2(30),
  manager REF person ) ;
```

- **Se x representa um objeto de tipo person, então**

```
x.manager.name
```

representa um string contendo o atributo nome do objeto person referenciado pelo atributo manager de x.

- **Expressão é versão abreviada de:**

```
y.name, onde y = DEREF(x.manager)
```

Obtendo REFs

- **Há um operador para obter o REF de um objeto**
- **Exemplo:**

```
DECLARE OrderRef REF to purchase_order;

SELECT REF(po) INTO OrderRef
               FROM purchase_order_table po
               WHERE po.id = 1000376 ;
```

Tipos de coleções

- **Há dois tipos de coleções**
  - array types
  - table types

- **Quando usados como componentes de objetos, estes tipos geram:**
  - VARRAYs
  - tabelas aninhadas (nested tables)

VARRAY

- **Conjunto *ordenado* de elementos**
- **Todos elementos são de mesmo tipo**
- **Cada elemento possui um *índice* (posição dentro do array)**
- **Tamanho (número de elementos) é variável, mas deve ser definido o limite superior**
- **Exemplo:**

```
CREATE TYPE prices AS VARRAY(10) OF
                                    NUMBER(12,2)
```

- **Um tipo de array pode ser usado como**
  - tipo de dados de uma coluna de uma tabela
  - tipo de atributo de um objeto
  - variável, argumento ou retorno em PL/SQL

Tabelas aninhadas - Nested Tables

- **Conjunto não ordenado de elementos de mesmo tipo**
- **Possui uma única coluna**
  - tipo da coluna é pré-definido de SQL ou object type
- **Exemplo:**

```
CREATE TYPE lineitem_table
                          AS TABLE OF lineitem ;
```

- **Pode ser usado como:**
  - tipo de dados de uma coluna de uma tabela
  - tipo de atributo de um objeto
  - variável, argumento ou retorno em PL/SQL