Gerenciamento de Dados e Informação

Práticas dos conceitos objeto-relacional

Equipe de monitoria

Aula prática 4

Roteiro

- Tipos
- Tabela de Objetos
- Herança
- Métodos
- Referências
- Coleções
- Composição de coleções
- Conectividade

Tipos e tabela de objetos

Tipos 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.

Tipos de Objetos (sintaxe)

```
CREATE [OR REPLACE] TYPE <nome do tipo>
   AS OBJECT (
    <lista de atributos e métodos>
);
```

```
DROP TYPE <nome do tipo> [FORCE];
```

```
SELECT * FROM user_types;
```

Tabelas de Objetos

- Objetos são diferentes de tabelas
- Tipos de Objetos apenas definem uma estrutura lógica, contendo nome, métodos e atributos.
  - Não obrigatoriedade da presença de métodos
- Tabelas armazenam espaço físico
- Cria-se tabelas de objetos previamente definidos
- Cada tabela recebe instâncias de objetos de apenas um tipo

Tabelas de Objetos (sintaxe)

```
CREATE TABLE  <nome da tabela>
   OF <nome do tipo> (
    <lista de propriedades dos atributos>
);
```

```
DROP TABLE <nome da tabela>;
```

```
INSERT INTO <nome da tabela>
   (<nomes dos atributos>)
   VALUES  (<valores>);
```

```
DELETE FROM <nome da tabela>
   WHERE <condição>;
```

Tipos vs. Tabelas de Objetos

- Tipos não permitem restrições de valores para os seus atributos;

- Restrições devem ser feitas nas tabelas:
  - NOT NULL
  - UNIQUE
  - PRIMARY KEY
  - CHECK

Exercício (proposta)

- Construir um tipo Endereço com os seguintes atributos:
  - Rua
  - Cidade
  - Estado
  - CEP
- E um tipo Pessoa, que possui:
  - Id
  - Nome
  - Endereço.

Exercício (resolução – tipos e tabelas)

```
CREATE OR REPLACE TYPE
endereco_tp AS OBJECT
  (Rua VARCHAR2(50),
   Cidade VARCHAR2(25),
   Estado CHAR(2),
   Cep NUMBER);
/
CREATE OR REPLACE TYPE
pessoas_tp AS OBJECT
  (id NUMBER,
   Nome VARCHAR2(25),
   Endereco endereco_tp);
```

Endereco_tp é usado para definir o tipo (domínio) do atributo coluna Endereço de outro tipo.

```
CREATE TABLE pessoa_tab OF
pessoas_tp
  (id PRIMARY KEY);
```

Exercício (resolução – inserções)

- Inserindo uma Pessoa:

```
INSERT INTO pessoa_tab VALUES
(1, 'João', endereco_tp('Rua Simão Mendes', 'Recife', 'PE', '53050110'));
```

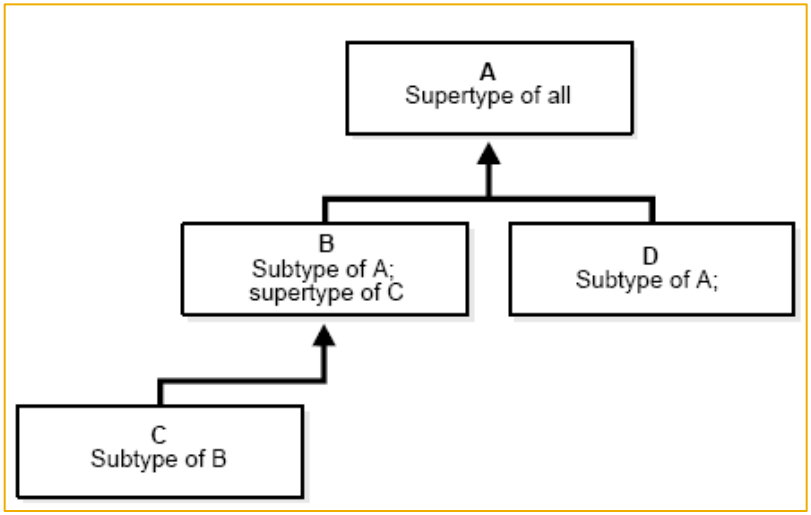
```
INSERT INTO pessoa_tab VALUES
(2, 'Maria', endereco_tp('Rua Padre Faustino', 'Jaboatão', 'PE', '45879362'));
```

```
INSERT INTO pessoa_tab VALUES
(3, 'José', endereco_tp('Rua Ernesto Ribeiro', 'Olinda', 'PE', '15469781'));
```

Herança

Herança

- Apenas herança simples é permitida no ORACLE



Herança

- Controle do usuário sobre a definição de tipos e métodos "herdáveis" - FINAL e NOT FINAL.
  - Tipos abstratos

```
CREATE [OR REPLACE] TYPE <nome do tipo>
    AS OBJECT (...) NOT INSTATIABLE;
```

  - Para permitir criação de subtipos

```
CREATE [OR REPLACE] TYPE <nome do tipo>
    AS OBJECT (...) NOT FINAL;
```

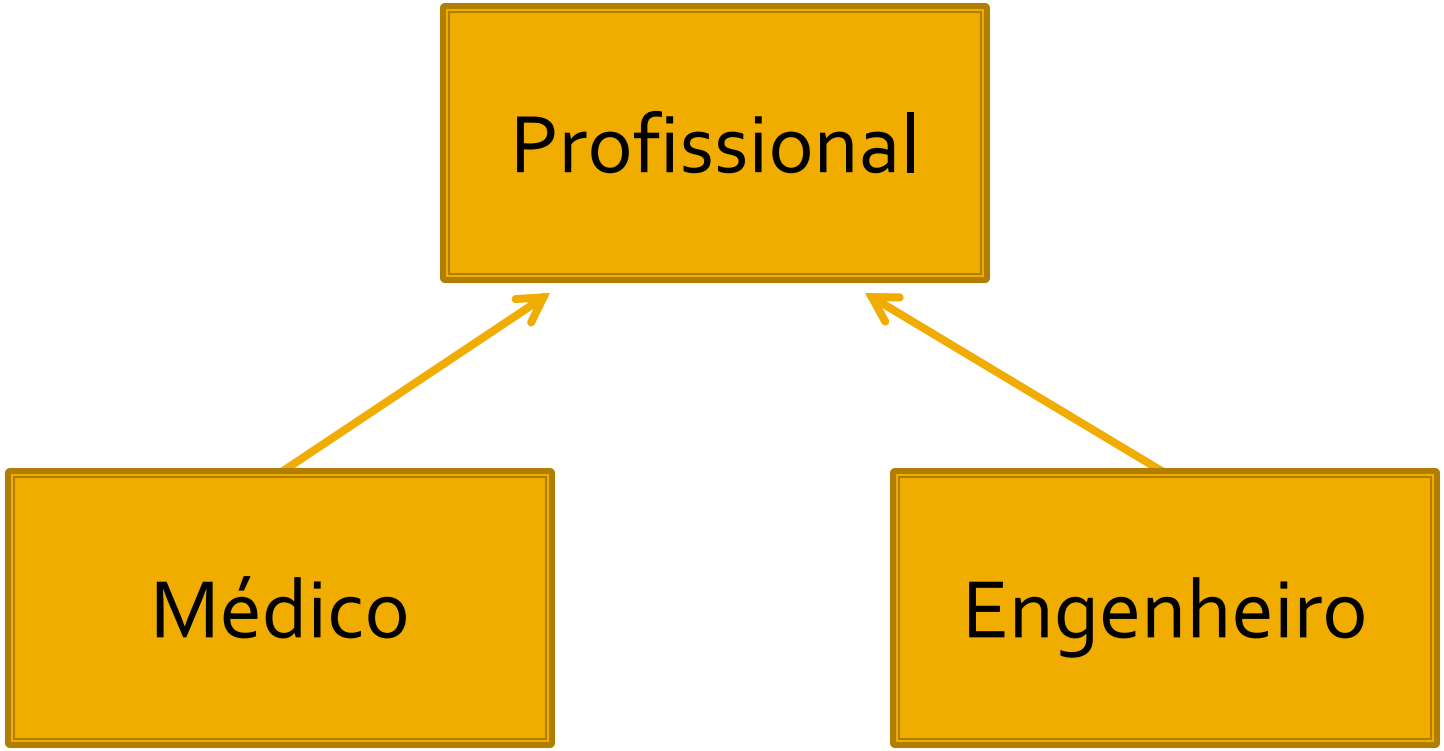
Herança

- Para criar um subtipo (sintaxe):

```
CREATE [OR REPLACE] TYPE
    <nome do subtipo> UNDER <nome do tipo> (
        [definição dos atributos específicos]
);
```

Exercício (proposta)

- Implementar o modelo, criar as tabelas necessárias, realizar inserções:



Exercício (resposta - tipos)

```
CREATE OR REPLACE TYPE tp_profissional as object(
        nome VARCHAR2(100),
        data_nascimento DATE
)NOT FINAL NOT INSTANTIABLE;
/
CREATE OR REPLACE TYPE tp_medico
    UNDER tp_profissional(
        cadastro_crm NUMBER,
        especialidade VARCHAR2(30)
);
/
CREATE OR REPLACE TYPE tp_engenheiro
     UNDER tp_profissional(
        cadastro_crea NUMBER
);
```

Exercício (resposta - tabelas)

```
CREATE TABLE tb_medico of tp_medico(
cadastro_crm PRIMARY KEY);
/
CREATE TABLE tb_engenheiro of tp_engenheiro(
cadastro_crea PRIMARY KEY);
```

Exercício (resposta - inserções)

```
INSERT INTO tb_medico VALUES
    (tp_medico('Jose', to_date('05/04/2009', 'dd/mm/yyyy'),
            12345 , 'Cardiologista'));
/
INSERT INTO tb_medico VALUES
   (tp_medico('Ana', to_date('05/04/2009', 'dd/mm/yyyy'),
            54321, 'Neurologista'));
/
INSERT INTO tb_engenheiro VALUES
   (tp_engenheiro('Luiz', to_date('05/04/2009', 'dd/mm/yyyy'),
            34567));
```

Métodos

Métodos

- Programas associados aos tipos que fazem computações e podem ter acesso aos atributos do tipo
- Na declaração de um tipo são definidas as assinaturas dos métodos, depois são implementados
- Tipos de Métodos
  - Member Method
  - Static Method
  - Constructor Method
  - Comparison Methods

Métodos

- Métodos podem ser FINAL ou NOT FINAL
  - Para permitir que um método não possa ser sobrescrito nos subtipos, este deve ser definido como FINAL
  - Por padrão, um método é definido como NOT FINAL

```
CREATE [OR REPLECE] type <nome do tipo> as object (
   <lista de atributos>[,
   <lista de assinaturas dos métodos>
);
```

```
CREATE [OR REPLECE] type body <nome do tipo> as (
      <lista de implementação dos métodos>
);
```

Métodos

- Exemplo

```
CREATE OR REPLACE TYPE TP_PERIODO AS OBJECT (
  dtInicio DATE,
  dtFim DATE,                                                    -- Atributos

  CONSTRUCTOR FUNCITION TP_PERIODO (di DATE, df DATE)
  RETURN SELF AS RESULT,                                         -- Constructor Method

  MEMBER FUNCTION dt_pertence  (pData DATE) RETURN INTEGER,
  MEMBER PROCEDURE set_DataInicio (pData DATE),                  -- Member Method

  ORDER MEMBER FUNCTION match (p tp_periodo) RETURN INTEGER,
  MAP MEMBER FUNCTION compara RETURN INTEGER                     -- Comparison Method
);
```

Métodos

Atenção!

Um objeto só pode ter **UM** método MAP OU **UM** método ORDER. O código utilizado como exemplo anteriormente não funcionará pois possui um método **MAP e um ORDER**.

Métodos

- Member Functions
  - Podem ser chamados através de um SELECT como em funções de PL/SQL.
- Member Procedures
  - Só é possível chamá-los em Blocos Anônimos, Functions, Procedures ou Triggers, pois não diferentemente das Member Function possui retorno.

Métodos

- Comparison Method
  - Permite a comparação de dois objetos
  - Torna possível utilizar as cláusulas DISTINCT, GROUP BY, ORDER BY, UNION entre outras.
  - Sem definir o MAP ou ORDER só é possível verificar se dois objetos são iguais
  - São funções chamadas implicitamente pelo SGBD quando é realizada a comparação entre dois tipos.

Métodos

- MAP
  - Não possui parâmetros, retorna um valor escalar (CHAR, DATE, VARCHAR, NUMBER) que será comparado com o valor de outro objeto
- ORDER
  - Recebe sempre um objeto do mesmo tipo como parâmetro. É possível realizar comparações entre os objetos e retorna um número inteiro (negativo, zero, positivo). Semelhante a interface de Java java.util.Comparator

Exercício (proposta)

1. Crie um tipo TP_QUADRILATERO que possui como atributos id, altura e largura.
2. Possui os seguintes métodos:
    1. Um construtor
    2. Um outro que retorna a área do quadrilátero
    3. E outro que atualiza apenas a altura do objeto

Exercício (resposta – declaração)

```
create or replace type tp_quadrilatero as object (
   id number,
   altura number,
   largura number,
   constructor function tp_quadrilatero
           (i number, a number, l number)
               return self as result,
   member function get_area return number,
   member procedure set_altura (a number)
);
```

Exercício (resposta – implementação)

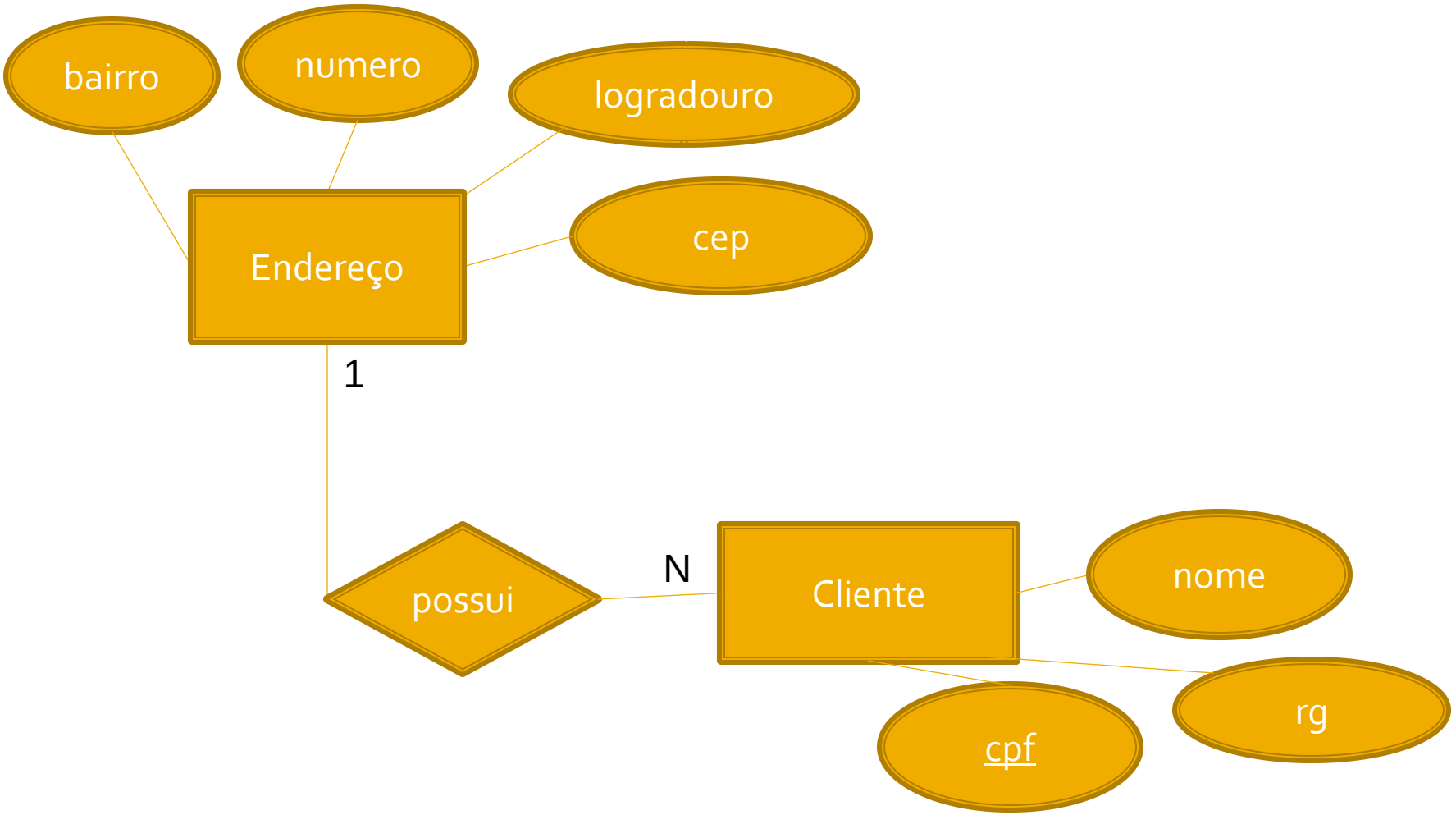
```
CREATE OR REPLACE TYPE BODY tp_quadrilatero as
        constructor function tp_quadrilatero(i number, a number, l number)
        return self as result is
        begin
        id := i;
        altura := a;
        largura := l;
        end;

        member function get_area return number is
        begin
        return altura * largura;
        end;

        member procedure set_altura(a number) is
        begin
        altura := a;
        end;
end;
```

Referência

Referência (Tipo REF)

- Retorna referência OID(object id) a uma instância de uma object table
- 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.

Exemplo (proposta)



Exemplo (proposta)

1. Implementar os tipos, usando os conceitos de referência
2. Criar as tabelas necessárias
3. Realizar inserções

   Observação: será necessário o uso de consulta aninhada.

Exemplo (resposta - tipos)

```
CREATE OR REPLACE TYPE tp_endereco AS OBJECT(
        idEndereco NUMBER,
        bairro VARCHAR(30),
        cep VARCHAR(9),
        logradouro VARCHAR(60),
        numero NUMBER
);
/
CREATE OR REPLACE TYPE tp_cliente AS OBJECT(
        cpf VARCHAR(14),
        rg NUMBER,
        nome VARCHAR(120),
        endereco REF tp_endereco
);
```

Exemplo (resposta – tabelas de tipos)

```
CREATE TABLE tb_endereco OF tp_endereco(
        idEndereco PRIMARY KEY
);
/
CREATE TABLE tb_cliente OF tp_cliente(
        cpf PRIMARY KEY,
        endereco WITH ROWID REFERENCES tb_endereco
);
```

Exemplo (inserções)

- Inserção de endereço

```
insert into tb_endereco (idEndereco, logradouro, cep, numero,
bairro) values (1,'Avenida joão de barros','52021-
180',1347,'espinheiro');
```

- Inserção de cliente

```
insert into tb_cliente (cpf,rg, nome, endereco)
values ('123.456.789-54', '6396327', 'Maria Leite Santiago',
(select REF(e) from tb_endereco e where e.idEndereco = 1));
```

```
insert into tb_cliente (cpf,rg, nome, endereco)
values ('422.544.623-88', '9856158', 'Roberto Leite Santiago',
(select REF(e) from tb_endereco e where e.idEndereco = 1));
```

Exemplo (consultas)

- Comando DEREF

```
select DEREF(c.endereco) from tb_cliente c where c.cpf =
'123.456.789-54';
```

- Comando DANGLING

```
SELECT * FROM tb_cliente c WHERE c.endereco
IS NOT DANGLING AND DEREF(c.endereco).bairro = 'espinheiro';
```

Coleções

Coleções

- Coleções modelam:
  - Atributos multivalorados
  - Relacionamentos 1xN
- O ORACLE oferece dois tipos de coleções:
  - VARRAYS
  - NESTED TABLES.

Coleções (varray vs. nested)

- **Varrays** são coleções ordenadas e limitada
  - São armazenadas como objetos contínuos.
- **Nested tables** são coleções não ordenadas e que não tem limite no número de linhas
  - São armazenadas em uma tabela onde cada elemento é mapeado em uma linha na tabela de armazenamento.

Coleções (Varray)

- Armazenam uma série de entradas de dados associadas a uma linha de um banco de dados
- Modelam relacionamento 1-para-muitos e atributos multivalorados

- Sintaxe:

```
CREATE TYPE <nome do conjunto>
   AS VARRAY(<tamanho>) OF <tipo de objeto>;
```

Coleções (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.

- Sintaxe:

```
CREATE TYPE <nome do conjunto>
  AS TABLE OF <tipo de objeto>;
```

Coleções (Quando usar?!)

VARRAY

- Ordem dos elementos é importante
- Número limitado de elementos: armazena mais eficientemente
  - Ex: Telefones

NESTED TABLE

- Fazer consultas SQL em elementos da NT (não é possível em Varrays)
- A ordem não é importante (SQL pode ordenar a saída se necessário)
- Não há limite de elementos
- Adicionar dados na NT (em Varrays não há como)

Coleções (Observações)

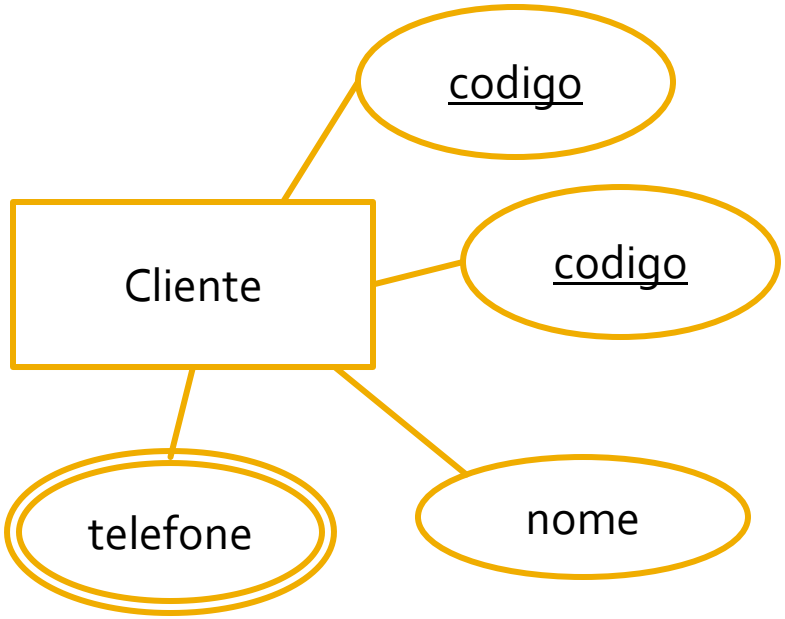
- Se é necessário eficiência na execução de consultas sobre coleções, então é recomendado o uso de nested tables.
- Tanto VARRAY quanto NESTED TABLE podem usar o tipo REF como atributo.

```
CREATE TYPE <nome do conjunto>
  AS VARRAY(<tamanho>) OF REF <tipo de objeto>;
```

```
CREATE TYPE <nome do conjunto>
  AS TABLE OF REF <tipo de objeto>;
```

Exercício 1 (proposta)

- Implementar os tipos necessários
- Realizar inserções



Observação: Cada cliente possui no máximo 5 telefones

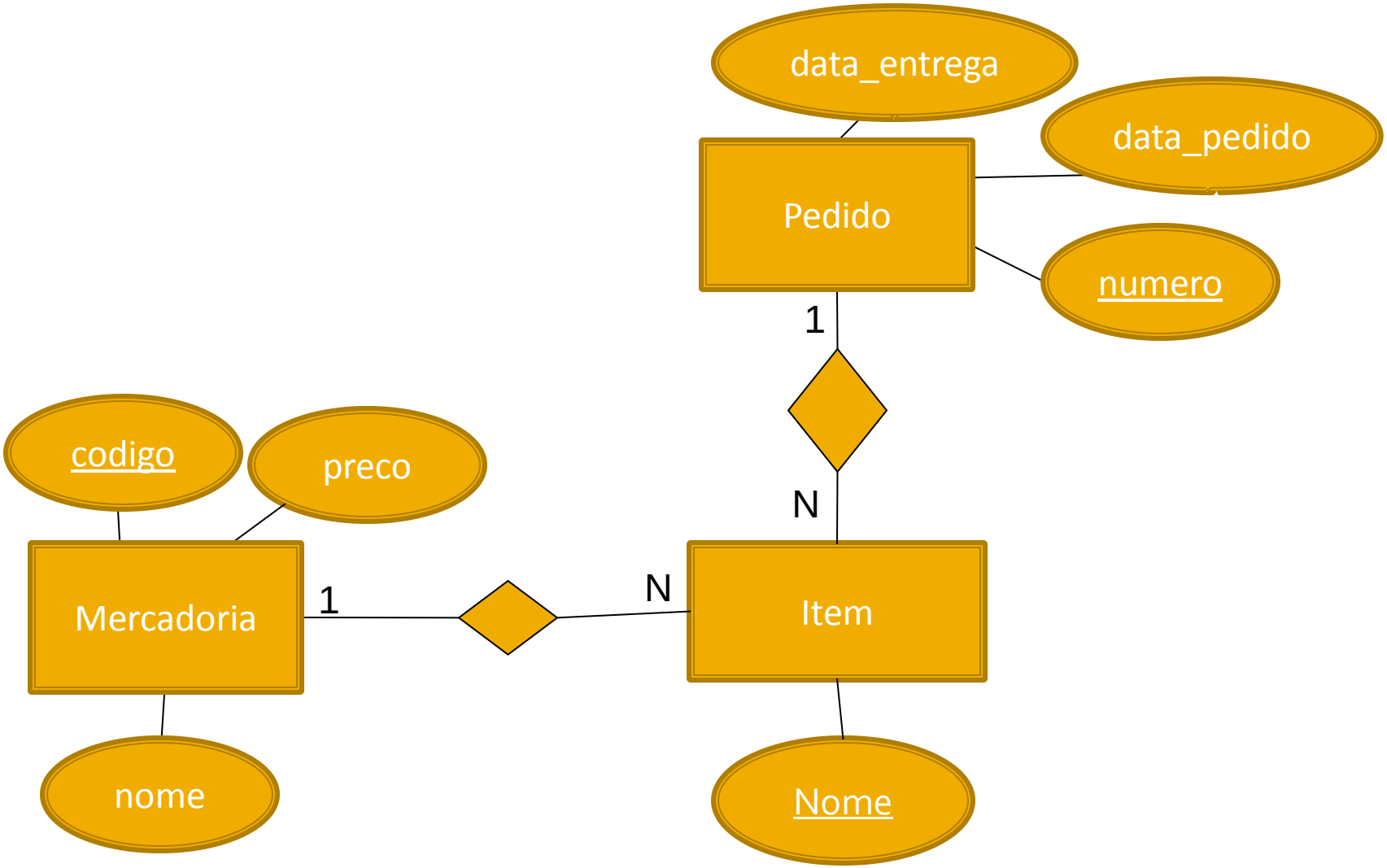
Exercício 1 (resposta)

```
CREATE OR REPLACE TYPE ListaFones_ty
AS VARRAY(5) OF VARCHAR2(10);
/
CREATE OR REPLACE TYPE Cliente_ty AS OBJECT(
                codigo number(11),
                nome      VARCHAR2(25),
                telefones  ListaFones_ty);
```

```
CREATE TABLE Clientes OF Cliente_ty;
```

```
INSERT INTO Clientes VALUES(
  1,  'Fernando Fumagalli',
 ListaFones_ty('8534224433', '8546778899'));
```

Exercício 2 (proposta)



Exercício 2 (resposta - tipos)

```
CREATE OR REPLACE TYPE Mercadoria_ty AS OBJECT (
            codigo     NUMBER,
            nome       VARCHAR2(50),
            preco      FLOAT);
/
CREATE OR REPLACE TYPE Item_ty AS OBJECT(
            numero      NUMBER,
            quantidade   NUMBER,
            mercadoriaRef REF Mercadoria_ty);
```

```
CREATE OR REPLACE TYPE ListaItens_ty AS TABLE OF Item_ty;
/
CREATE OR REPLACE TYPE Pedido_ty AS OBJECT(
            codigo  NUMBER,
            data_pedido   DATE,
            data_entrega  DATE,
            itens  ListaItens_ty);
```

Exercício 2 (resposta - tabelas)

```
CREATE TABLE Mercadorias OF Mercadoria_ty;

CREATE TABLE Pedidos OF Pedido_ty
        NESTED TABLE itens STORE AS Itens_ST;
```

- A criação da tabela Pedidos não aloca espaço de armazenamento para os objetos da tabela aninhada itens.
- Oracle armazena as linhas da tabela itens em uma tabela separada que deve ser definida (Itens_ST).

Exercício 2 (resposta - inserções)

```
INSERT INTO Mercadorias VALUES (2001,  'Mouse',  56.99);
/
INSERT INTO Mercadorias VALUES (2002,  'Teclado',  67.99);
/
INSERT INTO Mercadorias VALUES (2003,  'Monitor',  395.99);
```

```
INSERT INTO Pedidos VALUES(
  101, to_date('02/05/2008', 'dd/mm/yyyy'),to_date('10/05/2008', 'dd/mm/yyyy'),
    ListaItens_ty(
        Item_ty(1, 15, (SELECT REF(M) FROM Mercadorias M WHERE M.NOME = 'Mouse')),
        Item_ty(2, 20, (SELECT REF(M) FROM Mercadorias M WHERE M.NOME = 'Teclado'))

    )
);
/
INSERT INTO
  TABLE(SELECT P.ITENS FROM Pedidos P WHERE P.CODIGO = 101) I
     VALUES (Item_ty(3, 2, (SELECT REF(M) FROM Mercadorias  M
          WHERE M.NOME = 'Monitor'))
);
```

Exercício 2 (resposta - consultas)

```
SELECT P.itens FROM Pedidos P WHERE P.CODIGO = 101 ;
```

```
LISTAITENS_TY(...)

--------------------------------------------------------------------------------------------------
LISTAITENS_TY(
    ITEM_TY(1, 15, 317C5F38034A0F93D1A598569C098C010008),
    ITEM_TY(2, 20, 34A0F93D1A598569C09317C5F3808C010008)
)
```

```
SELECT * FROM TABLE(SELECT P.ITENS FROM Pedidos P WHERE P.CODIGO = 101);
```

N UMERO	QUANTIDADE	MERCADORIAREF
1	15	317C5F38034A0F93D1A598569C098C010008
2	20	34A0F93D1A598569C09317C5F3808C010008
3	2	36D784A0F93D8569C093174A587S8C010008

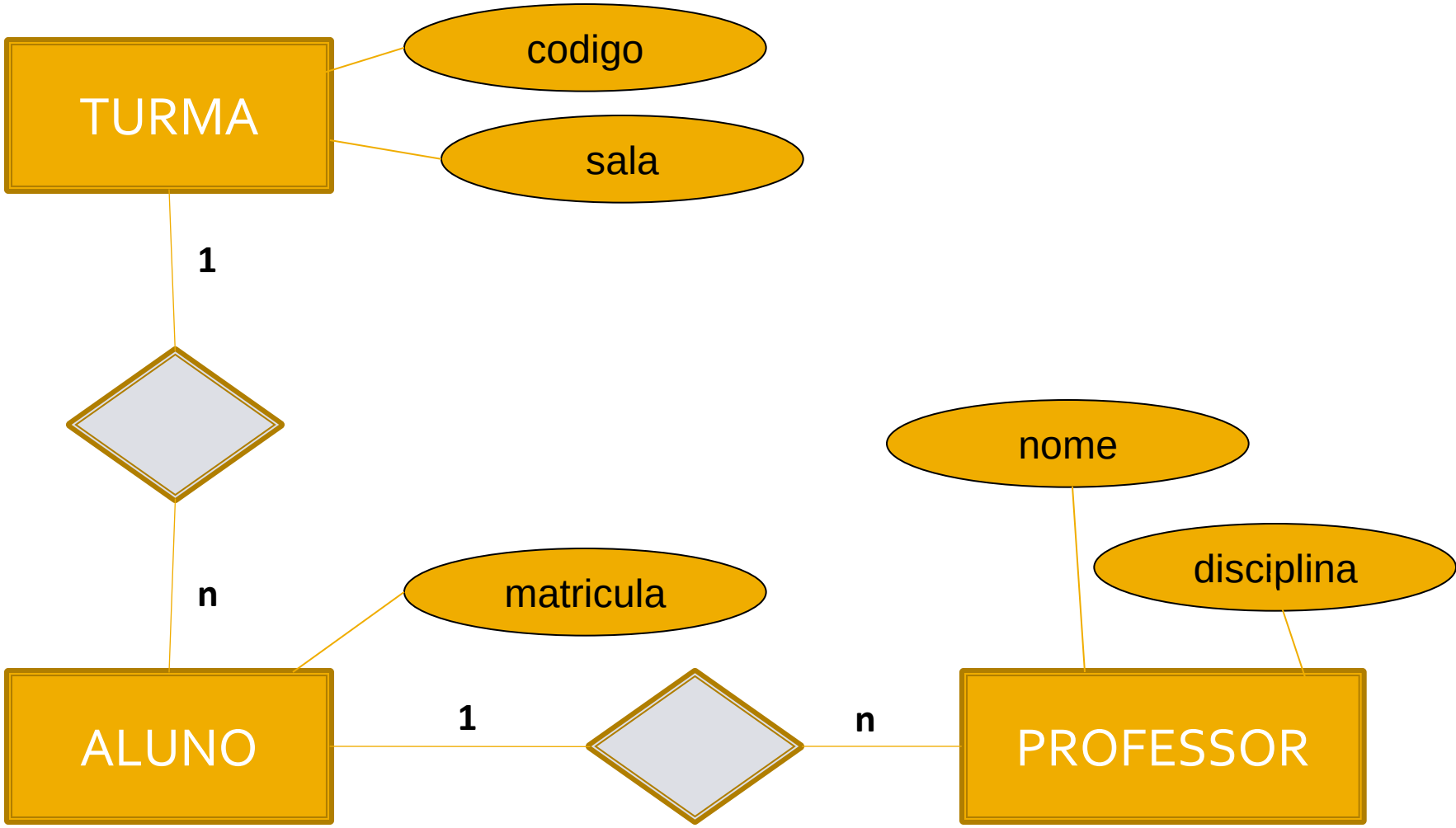
Exercício 2 (resposta - consultas)

```
SELECT VALUE(I) FROM
    TABLE(SELECT P.ITENS FROM Pedidos P WHERE P.CODIGO = 101) I;
```

```
ITEM_TY(NUMERO, QUANTIDADE, MERCADORIA_REF)
--------------------------------------------------------------------------------
ITEM_TY(1, 15, 317C5F38034A0F93D1A598569C098C010008)
ITEM_TY(2, 20, 34A0F93D1A598569C09317C5F3808C010008)
ITEM_TY(3, 2, 36D784A0F93D8569C093174A587S8C010008)
```

Nested de nested

Exercício (proposta)



Exercício (resposta)

```
CREATE TYPE PROFESSOR_TYPE AS OBJECT (
        NOME VARCHAR2(15),
        DISCIPLINA VARCHAR2(15)
);
```

```
CREATE TYPE NT_PROF_T AS TABLE OF PROFESSOR_TYPE;
```

```
CREATE TYPE ALUNO_TYPE AS OBJECT(
        MATRICULA NUMBER,
        PROFESSORES NT_PROF_T
);
```

```
CREATE TYPE NT_ALUNO_T AS TABLE OF ALUNO_TYPE;
```

Exercício (resposta)

```
CREATE TYPE TURMA_TYPE AS OBJECT (
        CODIGO VARCHAR2(3),
        SALA VARCHAR2(3),
        ALUNOS NT_ALUNO_T
);
```

```
CREATE TABLE TURMA_TAB OF TURMA_TYPE (
        CODIGO PRIMARY KEY
) NESTED TABLE ALUNOS STORE AS ALUNOS_T
  (NESTED TABLE PROFESSORES STORE AS PROF_T);
```

Exercício (resposta)

- Inserção de uma turma

```
INSERT INTO TURMA_TAB VALUES
      ('I5A', 'D05',
      NT_ALUNO_T(
        ALUNO_TYPE(210141500, NT_PROF_T(
            PROFESSOR_TYPE('Manoel','Sist. Digitais'),
            PROFESSOR_TYPE('Ruy','Logica')
        )),
        ALUNO_TYPE(210141750, NT_PROF_T(
            PROFESSOR_TYPE('Silvio','HFC'),
            PROFESSOR_TYPE('Hermano','PLP')
        ))
      )--nt_aluno_t
);--insert
```

Exercício (resposta)

- Inserção de um professor

```
INSERT INTO TABLE (SELECT A.PROFESSORES
   FROM TABLE (SELECT T.alunos FROM TURMA_TAB T
      WHERE T.codigo = 'I5A')  A
   WHERE A.MATRICULA = 210141750) Z
VALUES ('Fernando','GDI');
```

Exercício (resposta)

- Atualização de um professor

```
UPDATE TABLE (SELECT A.PROFESSORES FROM
   TABLE(SELECT T.ALUNOS FROM TURMA_TAB T
         WHERE T.CODIGO = 'I5A') A
   WHERE A.MATRICULA = 210141500) P
SET VALUE(P)= PROFESSOR_TYPE('Anjolina','Logica')
WHERE P.DISCIPLINA = 'Logica';
```

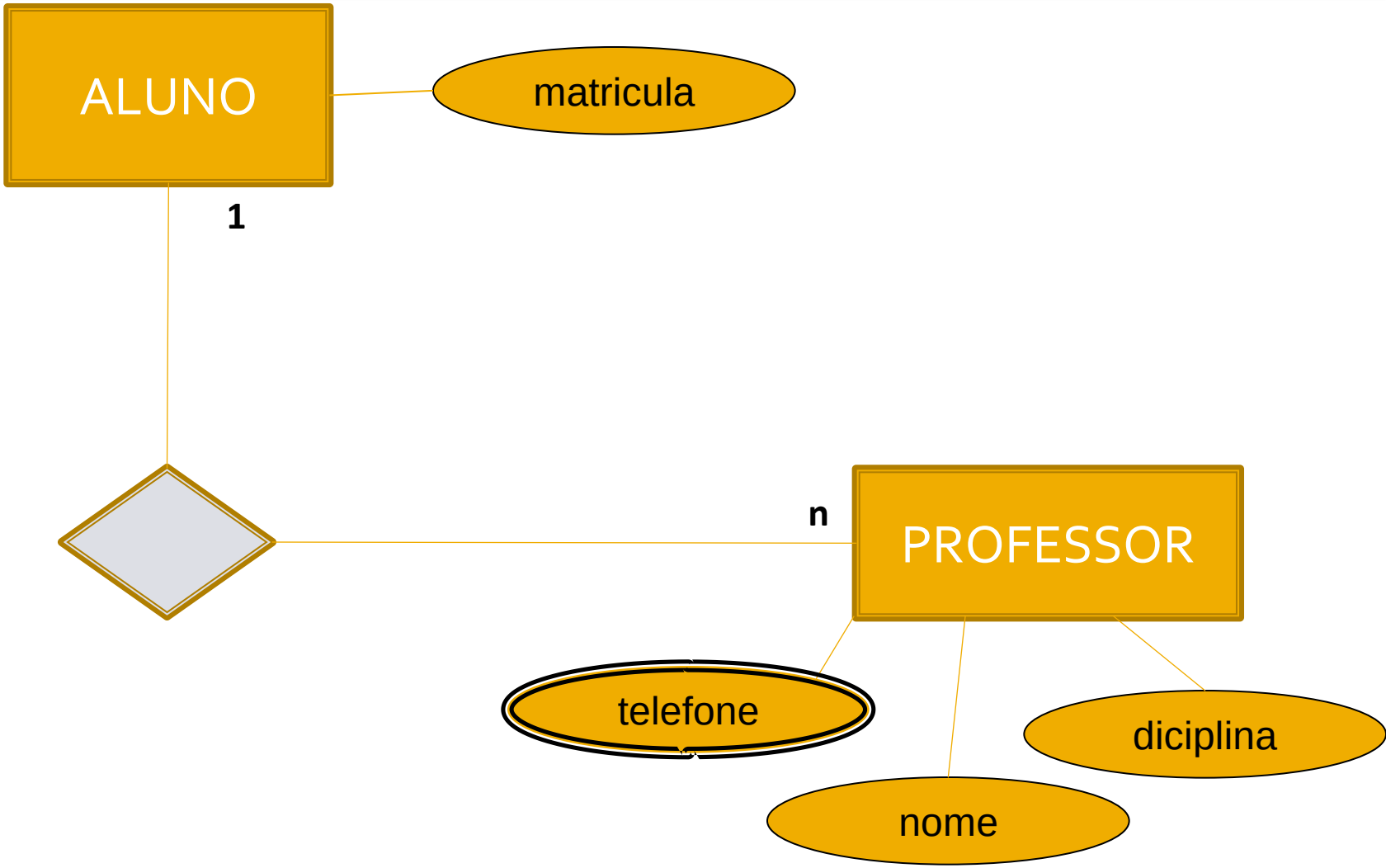
Exercício (resposta)

- Consulta a tabela aninhada

```
SELECT P.NOME, P.DISCIPLINA FROM
   TURMA_TAB T, TABLE(T.ALUNOS) A,
TABLE(A.PROFESSORES) P  WHERE T.CODIGO = 'I5A'
```

Nested de varray

Exercício (proposta)



Exercício (resposta)

```
CREATE OR REPLACE TYPE tp_telefone AS OBJECT (
        cod_area VARCHAR2(15),
        FONE VARCHAR2(15)
);
```

```
CREATE TYPE array_tp_fone AS VARRAY(2) OF tp_telefone;
```

```
CREATE TYPE professor_tp AS OBJECT(
        nome VARCHAR2(15),
        disciplina VARCHAR2(15),
        telefone array_tp_fone
);
```

Exercício (resposta)

```
CREATE TYPE NT_PROFESSOR_T AS TABLE OF PROFESSOR_TP;
```

```
CREATE TYPE tp_aluno  AS OBJECT(
            Matricula NUMBER,
            conj_professores nt_professor_t
);
```

```
CREATE TABLE tb_aluno OF tp_aluno (
            PRIMARY KEY (Matricula)
) NESTED TABLE conj_professores STORE AS prof_nt;
```

Exercício (resposta)

- Inserção com nested vazia

```
INSERT INTO tb_aluno VALUES (21035, nt_professor_t());
```

- Inserção com nested preenchida

```
INSERT INTO tb_aluno VALUES (217896,
   nt_professor_t(
                professor_tp('Manoel', 'SD',
        array_tp_fone(tp_telefone(81, 12345678), tp_telefone(81,87456321))),
      professor_tp('Fernando', 'GDI',
        array_tp_fone(tp_telefone(81, 45879632)))
   )
);
```

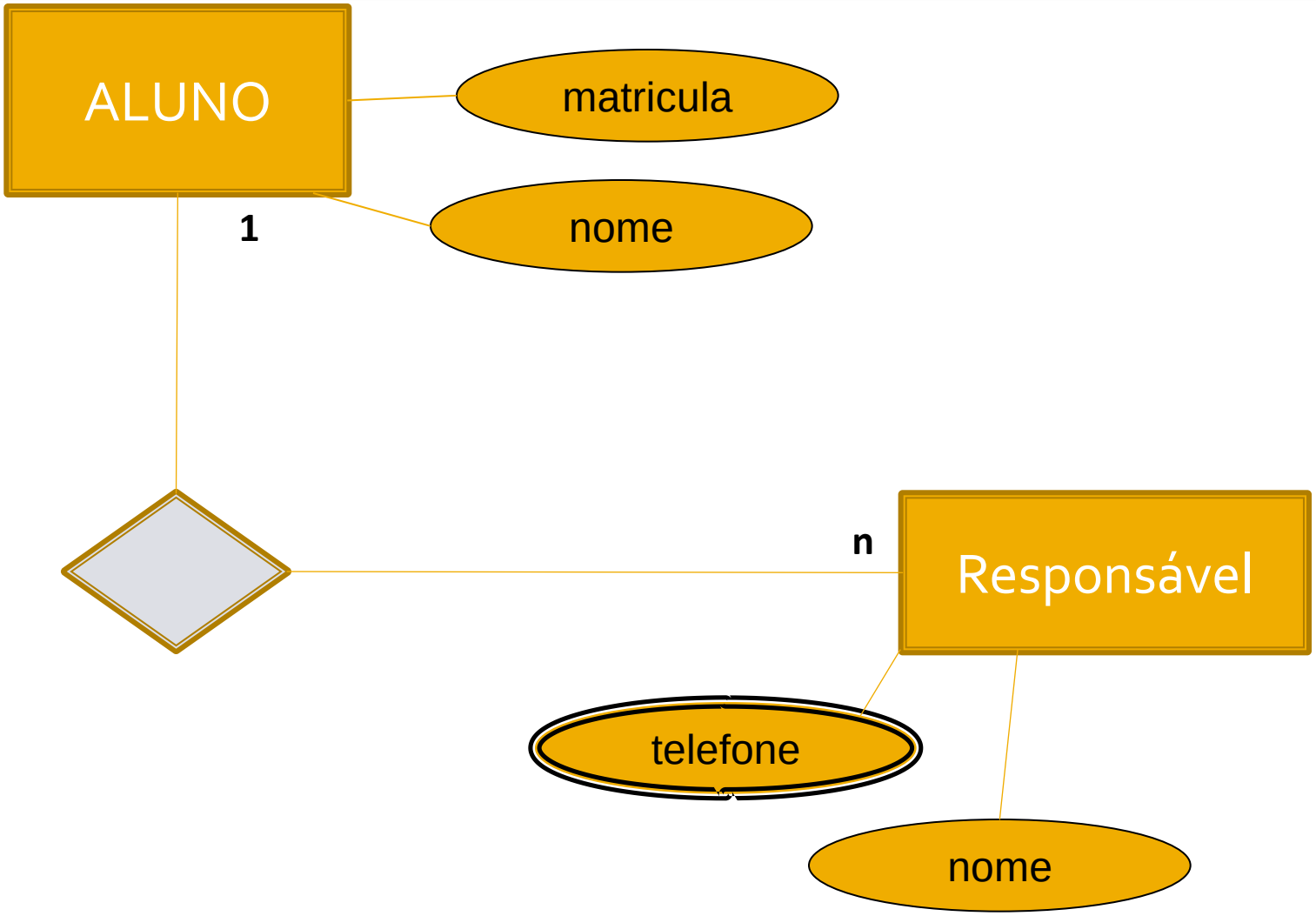
Exercício (resposta)

```
SELECT P.nome FROM
            tb_aluno A,
            TABLE(A.conj_professores) P
   WHERE P.disciplina like 'SD' Group by P.nome;
```

```
SELECT t.cod_area, t.fone FROM
            tb_aluno Al,
            Table (Al.conj_professores) P,
            Table (P.telefone) T
  WHERE P.nome Like 'Manoel' Group by t.cod_area, t.fone;
```

Varray de varray

Exercício (proposta)



Exercício (resposta)

```
CREATE OR REPLACE TYPE tp_telefone AS OBJECT(
      cod_area NUMBER,
      fone NUMBER
);
```

```
CREATE OR REPLACE TYPE array_tp_telefone AS VARRAY(3) OF tp_telefone;
```

```
CREATE OR REPLACE TYPE
tp_responsavel AS OBJECT(
      nome VARCHAR(30),
      telefones array_tp_telefone
);
```

Exercício (resposta)

```
CREATE OR REPLACE TYPE array_tp_responsavel
        AS VARRAY(2) OF tp_responsavel;
```

```
CREATE OR REPLACE TYPE tp_aluno AS OBJECT(
    nome VARCHAR(30),
    matricula NUMBER,
    responsaveis array_tp_responsavel
);
```

```
CREATE TABLE tb_aluno OF tp_aluno(
    PRIMARY KEY(matricula)
);
```

Exercício (resposta)

```
INSERT INTO tb_aluno VALUES ('Joao', 123323,
array_tp_responsavel(
    tp_responsavel(
        'Paulo', array_tp_telefone(tp_telefone(81, 34231234), tp_telefone(81,
92932332), tp_telefone(81, 33221222))),
    tp_responsavel(
        'Barbara', array_tp_telefone(tp_telefone(81, 34231234), tp_telefone (81,
92332552), tp_telefone(81, 33221555)))));
```

```
INSERT INTO tb_aluno VALUES ('Caio', 232342,
array_tp_responsavel(
    tp_responsavel(
        'Bruno', array_tp_telefone(tp_telefone(82, 23232323), tp_telefone(82,
89292922)))));
```

Exercício (resposta)

```
SELECT R.nome FROM tb_aluno A, TABLE(A.responsaveis) R
        WHERE A.matricula = 123323;
```

```
SELECT T.cod_area, T.fone FROM
        tb_aluno A,
        TABLE(A.responsaveis) R,
        TABLE(R.telefones) T
  WHERE R.nome = 'Bruno' GROUP BY T.cod_area, T.fone;
```

Conectividade

Ver anexo!

Perguntas? Sugestões?

Muito obrigado!