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

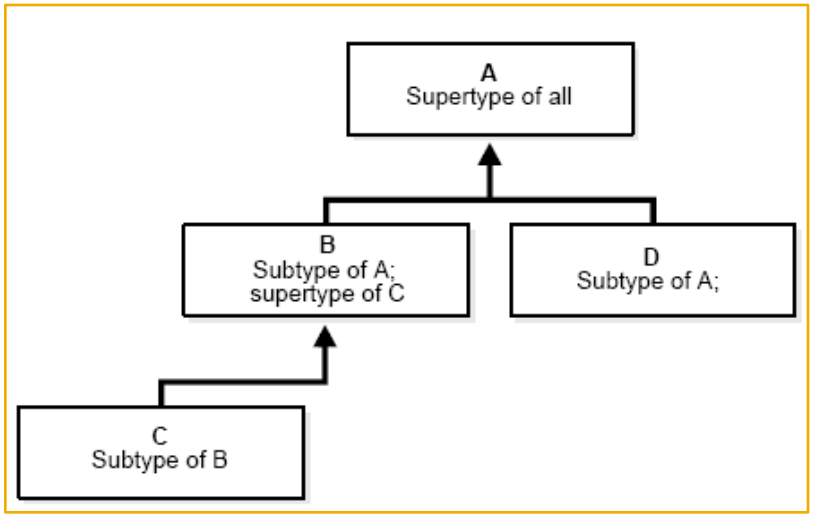
Exercício (proposta)

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

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;
```

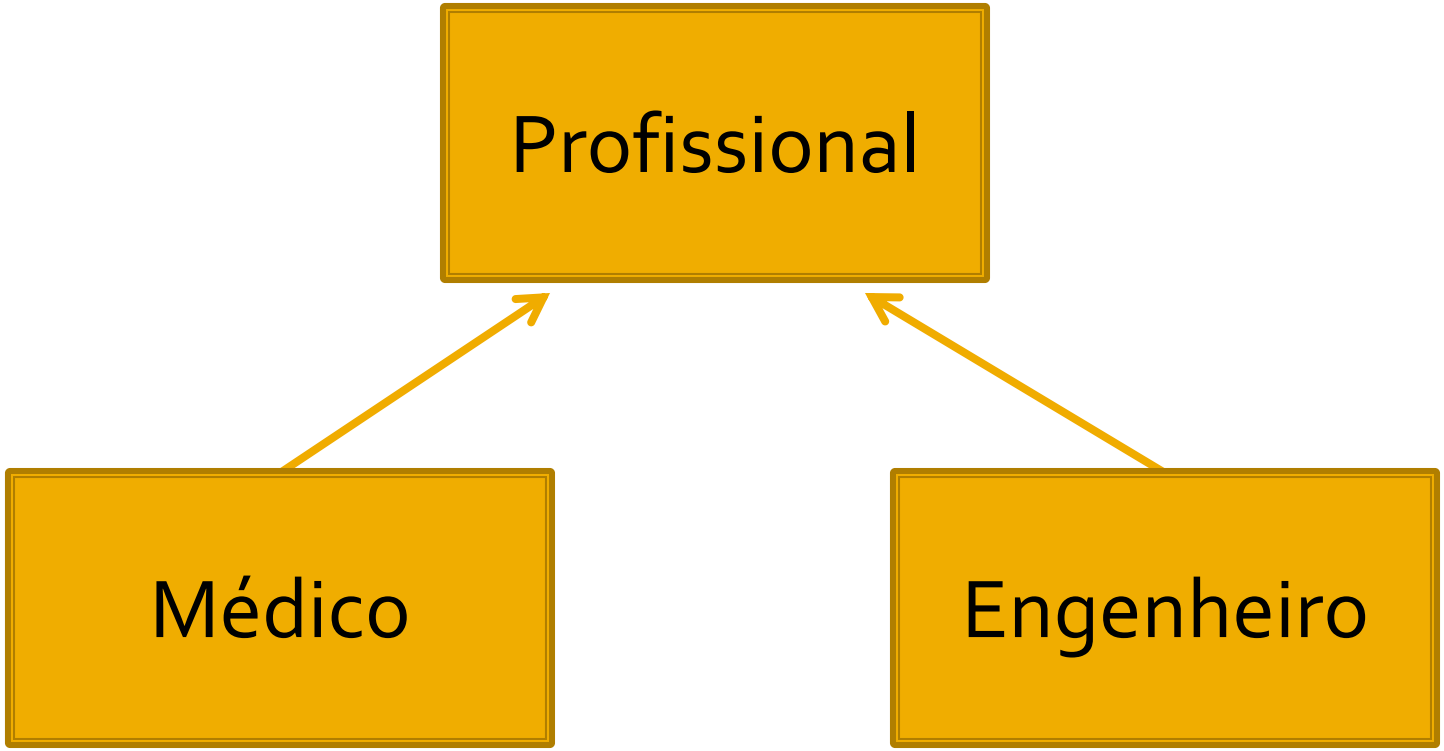
Herança

- 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:



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)

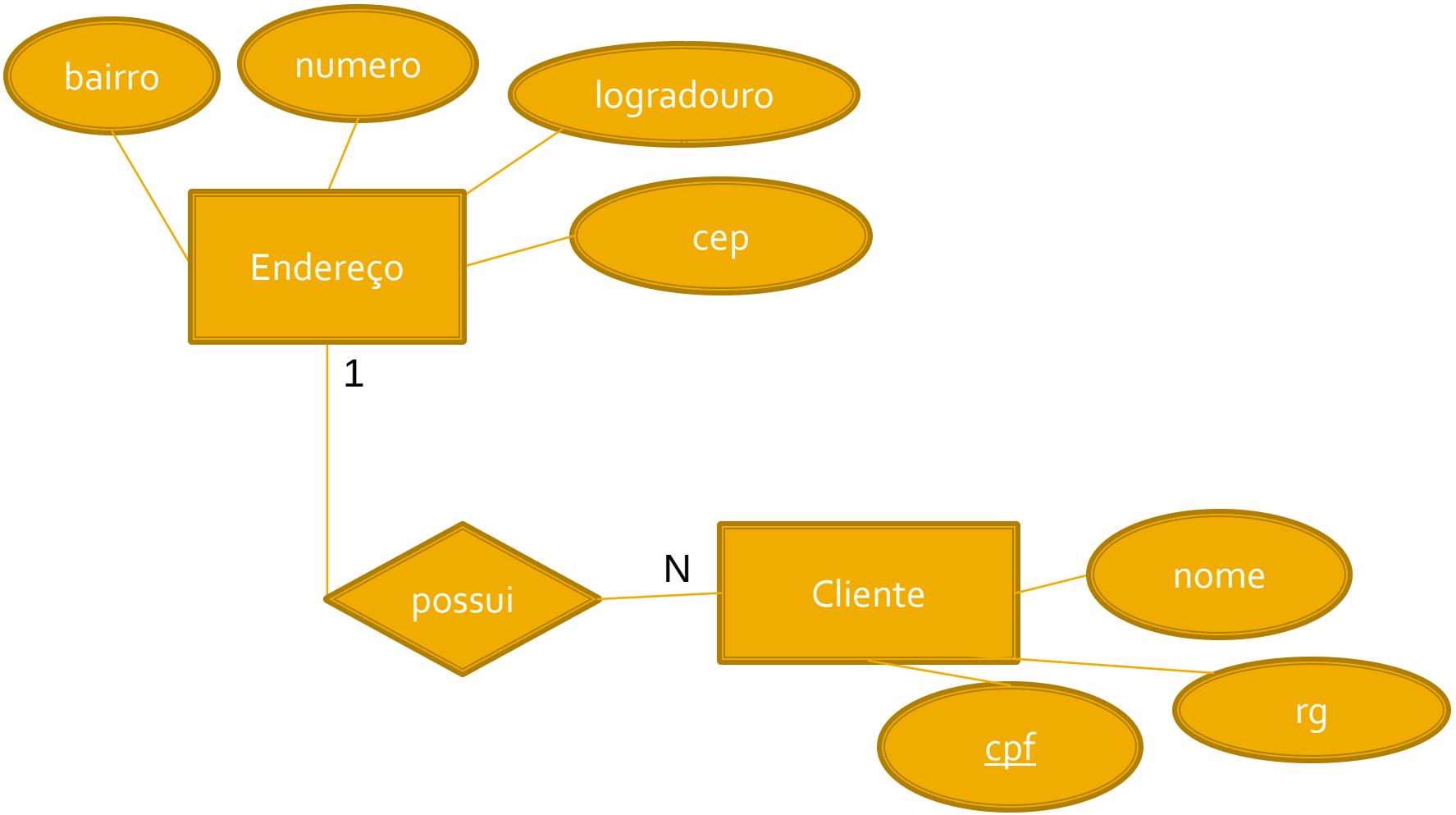
1. Crie um tipo TP_QUADRILATERO que possui como atributos id, altura e largura.
2. Possui os seguintes métodos:
    1. Um construtor
    2. Um outro que retorna a área do quadrilátero
    3. E outro que atualiza apenas a altura do objeto

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)

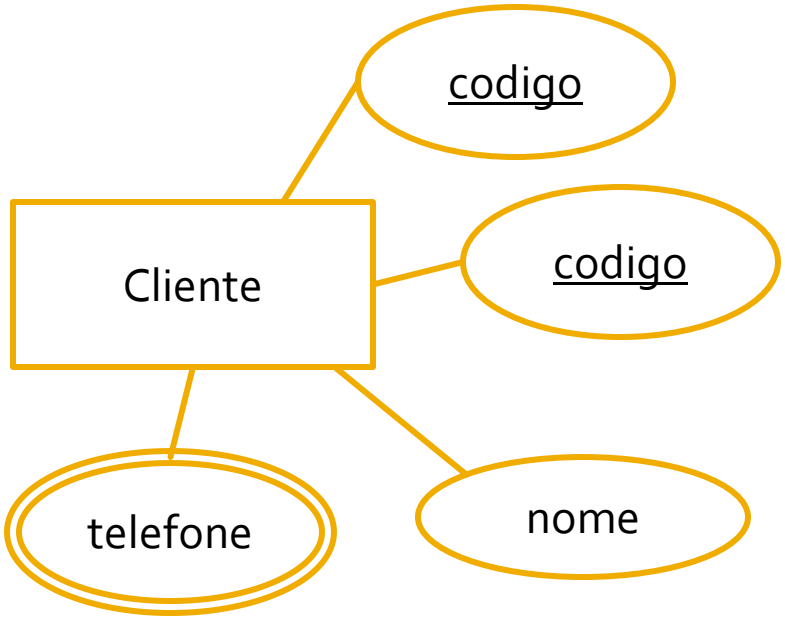
- Se é necessário eficiência na execução de consultas sobre coleções, então é recomendado o uso de nested tables.
- Tanto VARRAY quanto NESTED TABLE podem usar o tipo REF como atributo.

```
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>;
```

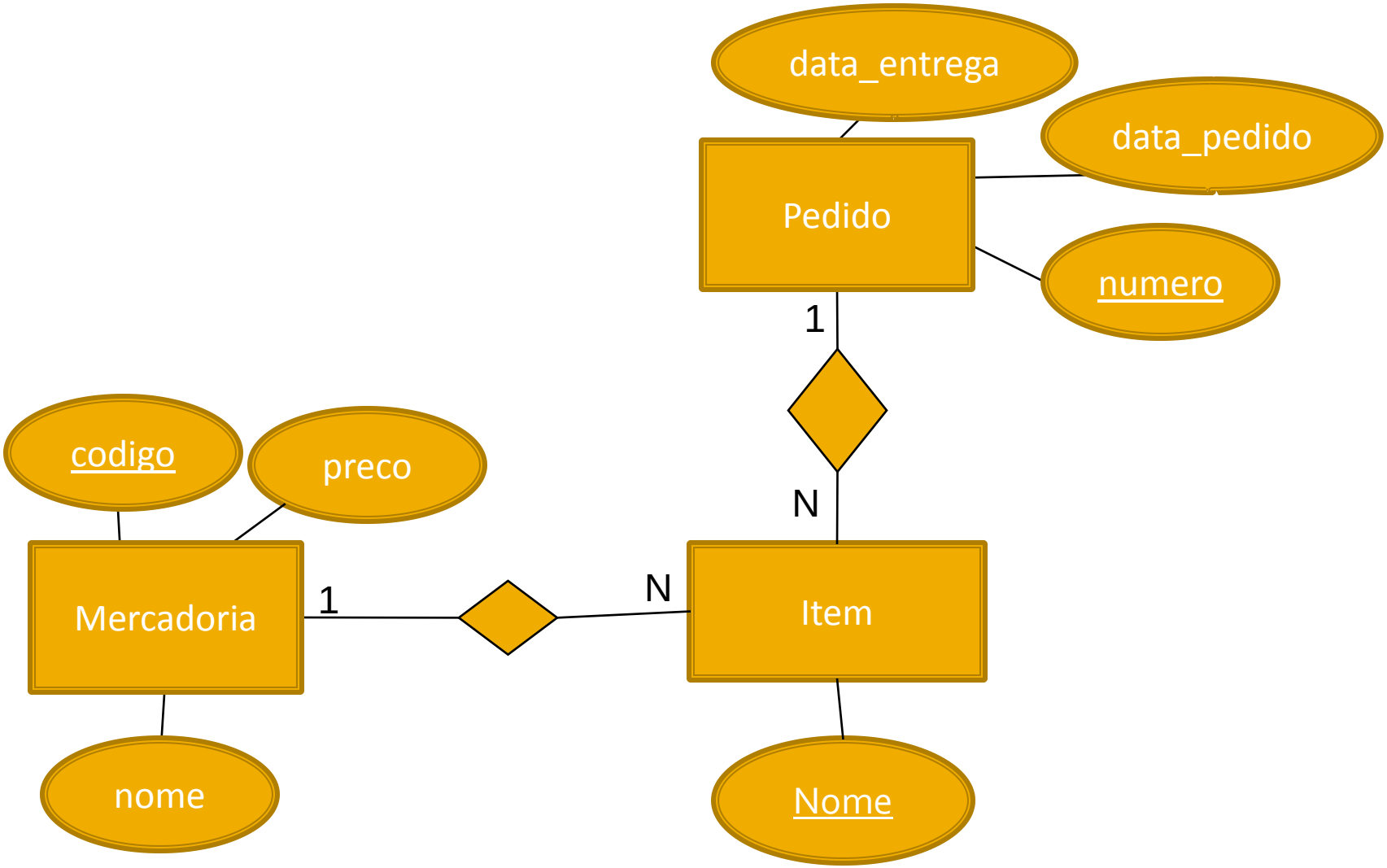
Exercício 1 (proposta)

- Implementar os tipos necessários
- Realizar inserções



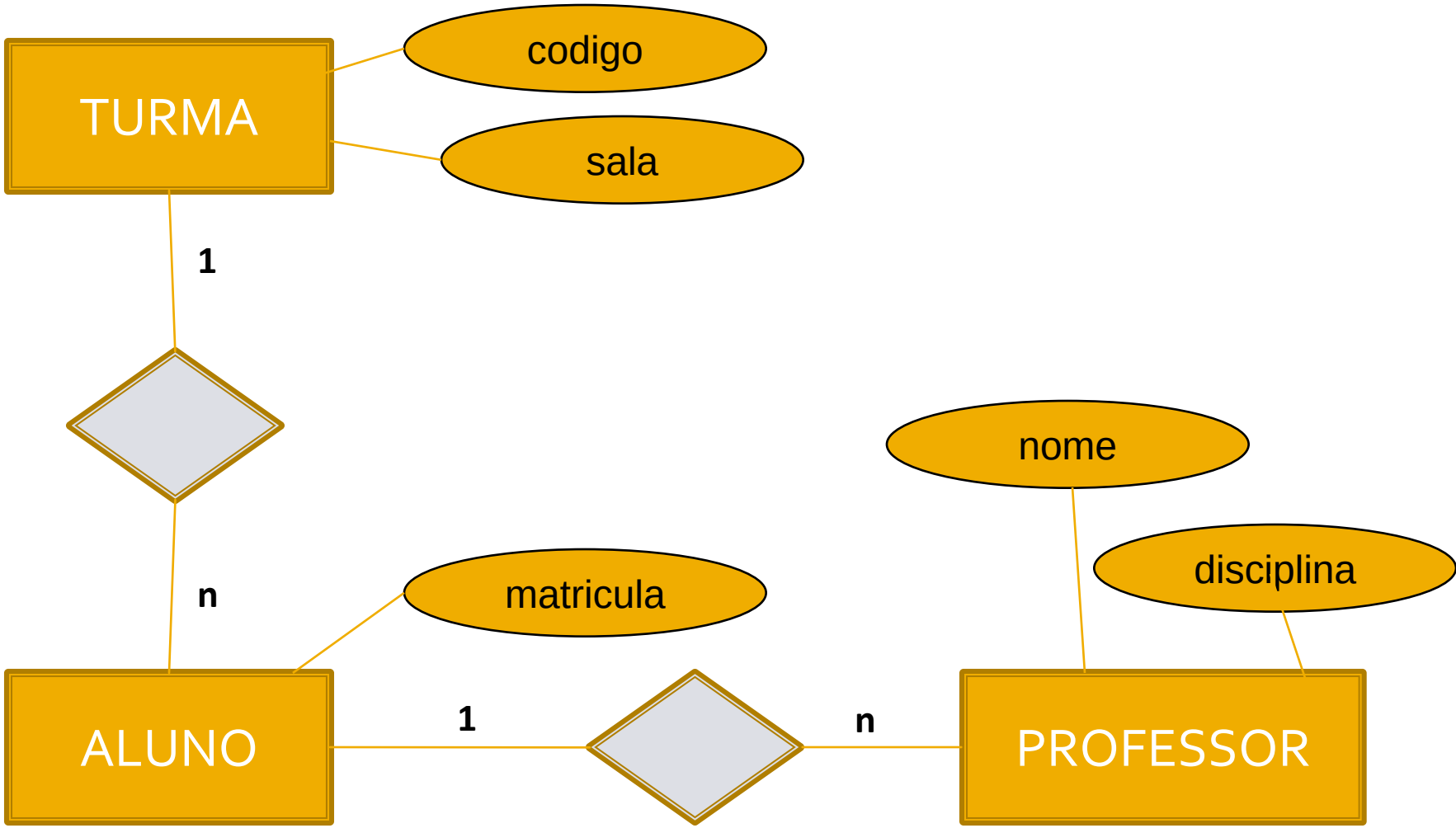
Observação: Cada cliente possui no máximo 5 telefones

Exercício 2 (proposta)



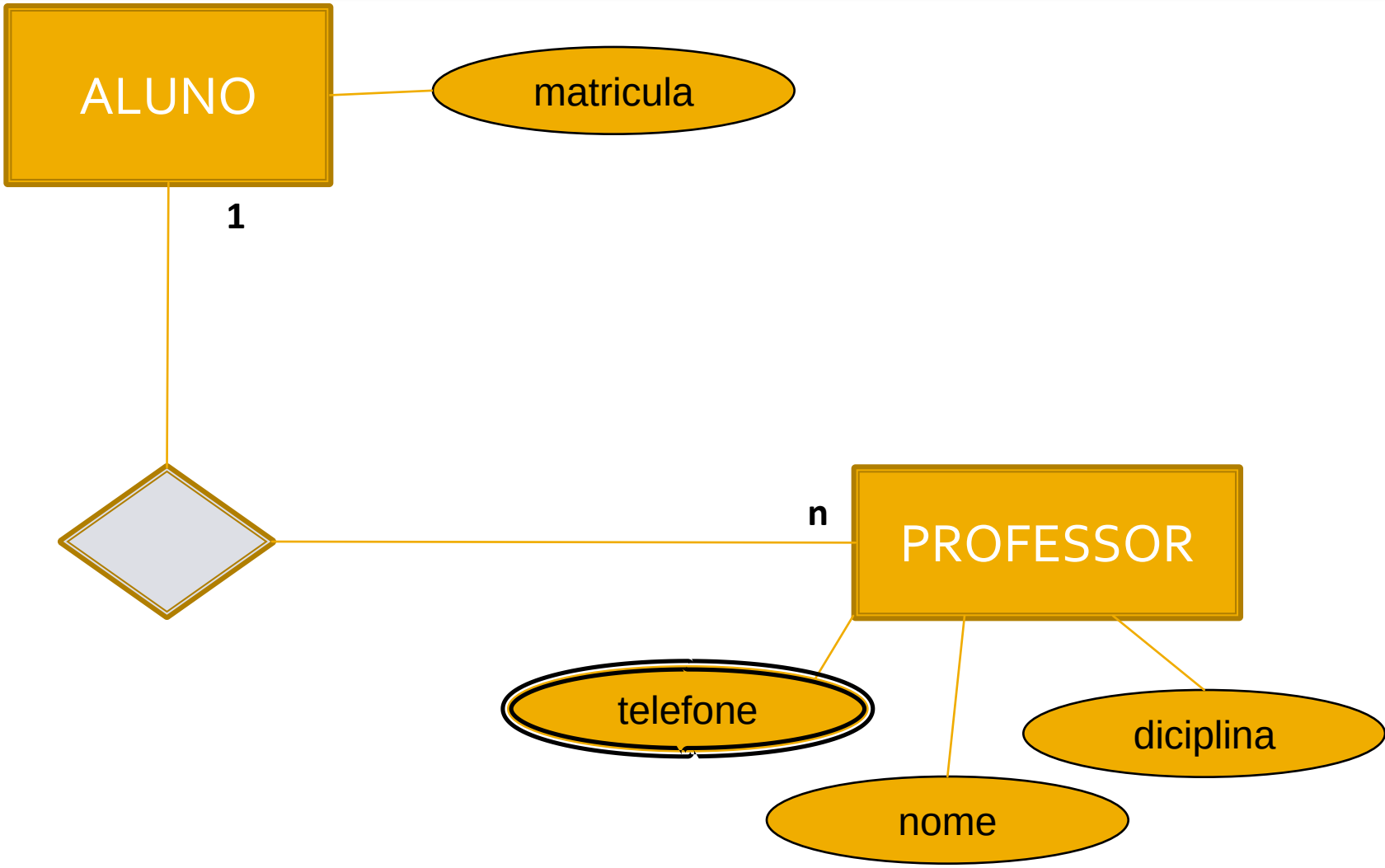
Nested de nested

Exercício (proposta)



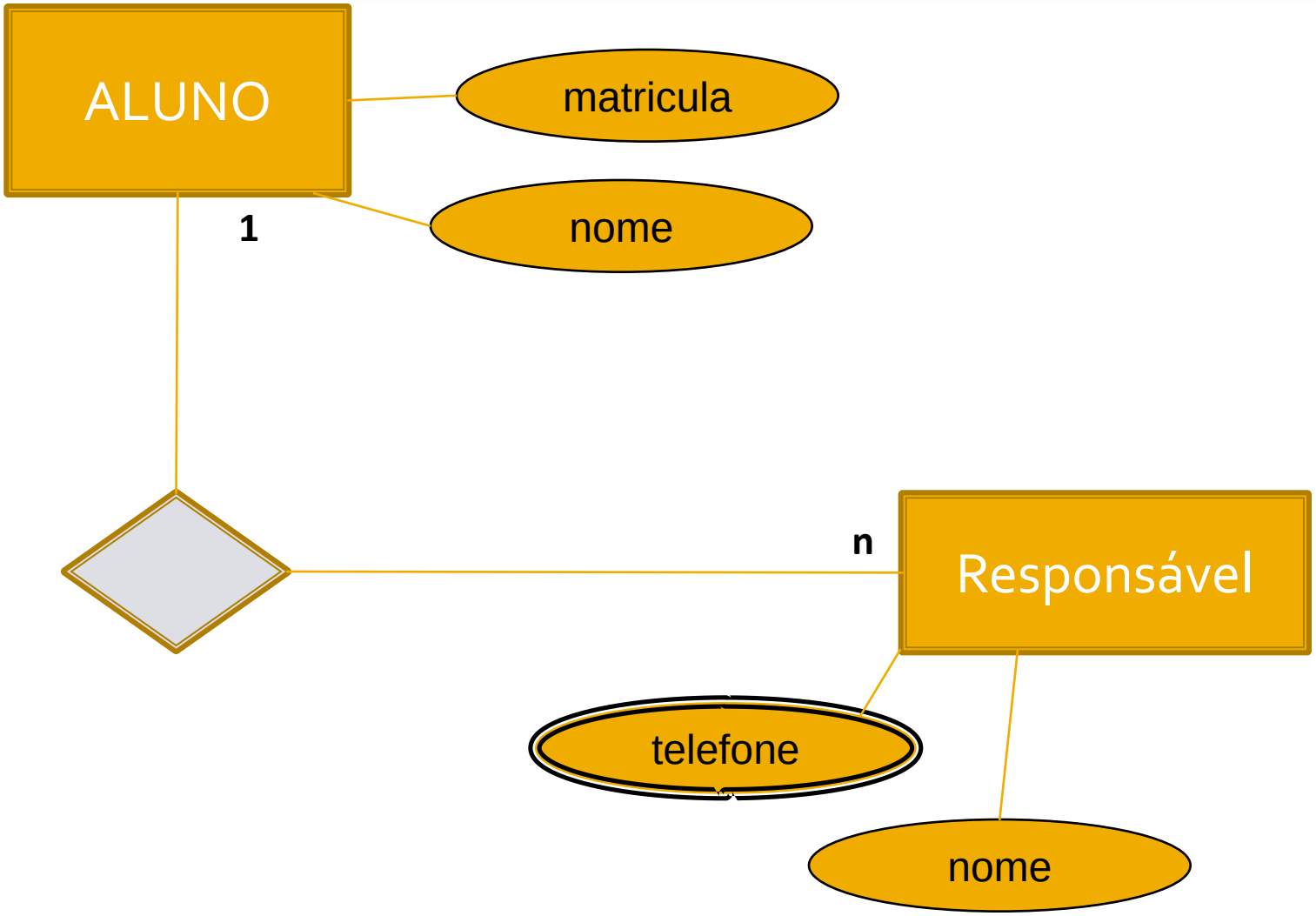
Nested de varray

Exercício (proposta)



Varray de varray

Exercício (proposta)



Conectividade

Ver anexo!

Perguntas? Sugestões?

Muito obrigado!