




Persistência de Objetos



Objetos Persistentes

Objetos Persistentes: são objetos que requerem armazenamento persistente.

Exemplo: Instâncias da classe Descrição Produto devem ser armazenadas em uma base de dados.

Mecanismos de Armazenamento de Objetos

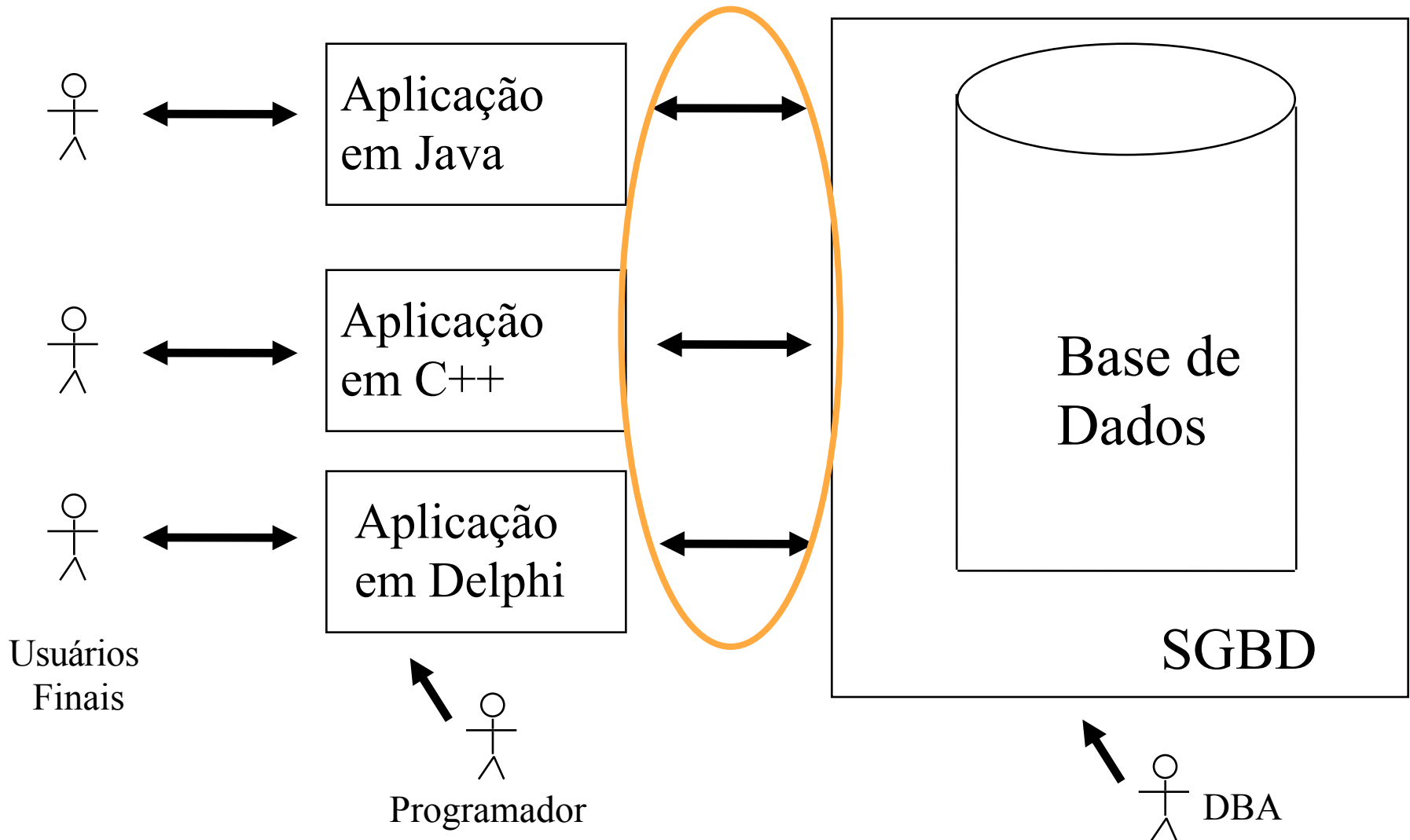
- Banco de Dados Orientado a Objetos

└ não requer mapeamento para armazenar e recuperar objetos

➡ Banco de Dados Relacional

- Outros: Banco de Dados Hierárquico, Arquivo, XML, etc.

Armazenamento em um BD Relacional



Modelo Relacional

No modelo relacional, a base de dados é descrita através de relações.

Informalmente, uma relação pode ser vista como uma tabela.

Exemplo: Tabela de Empregados

Cód.	Nome	Salário
110	Benedito Gomes	567,43
200	Alberto Silva	345,00
835	João Santos	280,00
902	Cláudia Vieira	200,00
005	Walter Souza	1456,86
711	Flávia Costa	450,00

Exemplo de Banco de Dados Relacionais

Tabela Empregado

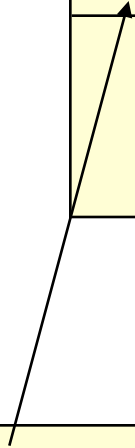
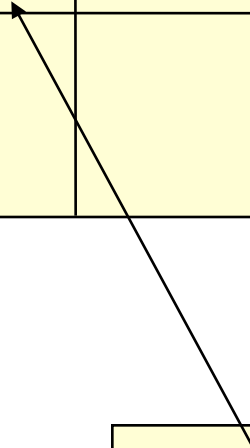
<u>Código</u>	Nome	Salário

Tabela Projeto

<u>Código</u>	Descrição

Tabela Emp-Proj

<u>Cód. Empregado</u>	<u>Cod. Projeto</u>





Persistência de Objetos em SGBD Rel.

Problema da Persistência de objetos em SGBDs relacionais: modelo e linguagem distintos

- Linguagens OO
 - tipos de dados complexos (i.e. classes)
 - hierarquias de herança
 - linguagem imperativa (i.e. métodos)
- SGBDs relacionais
 - tipos de dados simples (domínios atômicos)
 - dados armazenados em tabelas
 - DML declarativa (i.e. SQL)



Persistência de Objetos

Serviço de Persistência:

Traduz objetos em registros e os salva em uma base de dados, e traduz os registros em objetos quando estes são recuperados de uma base de dados.

Como implementar o serviço de persistência?

1. Através da própria classe de objeto persistente
2. Através de uma camada de persistência

Classe de Objeto Persistente

1. Serviços de Persistência através de uma Classe de Objeto Persistente:

A classe de objeto persistente define o código para salvar e carregar os objetos em uma base de dados.



Mapeamento Direto

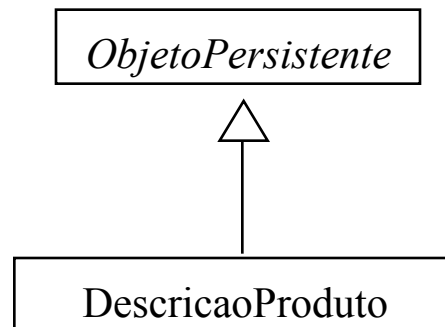
➔ O mapeamento direto precisa ser adicionado e mantido manualmente.

Problemas:

- Alto acoplamento entre a classe e o mecanismo de persistência.
- Serviços técnicos misturados com a lógica da aplicação (baixa coesão).

Classe de Objeto Persistente

- Pode ser criada uma superclasse abstrata ObjetoPersistente, da qual todos os objetos persistentes devem herdar.



A classe ObjetoPersistente define um atributo OID e métodos abstratos para salvar, carregar e remover de uma base de dados.

Camada de Persistência

2. Serviços de Persistência através de uma Camada de Persistência:

Outras classes são responsáveis pelo mapeamento dos objetos persistentes.

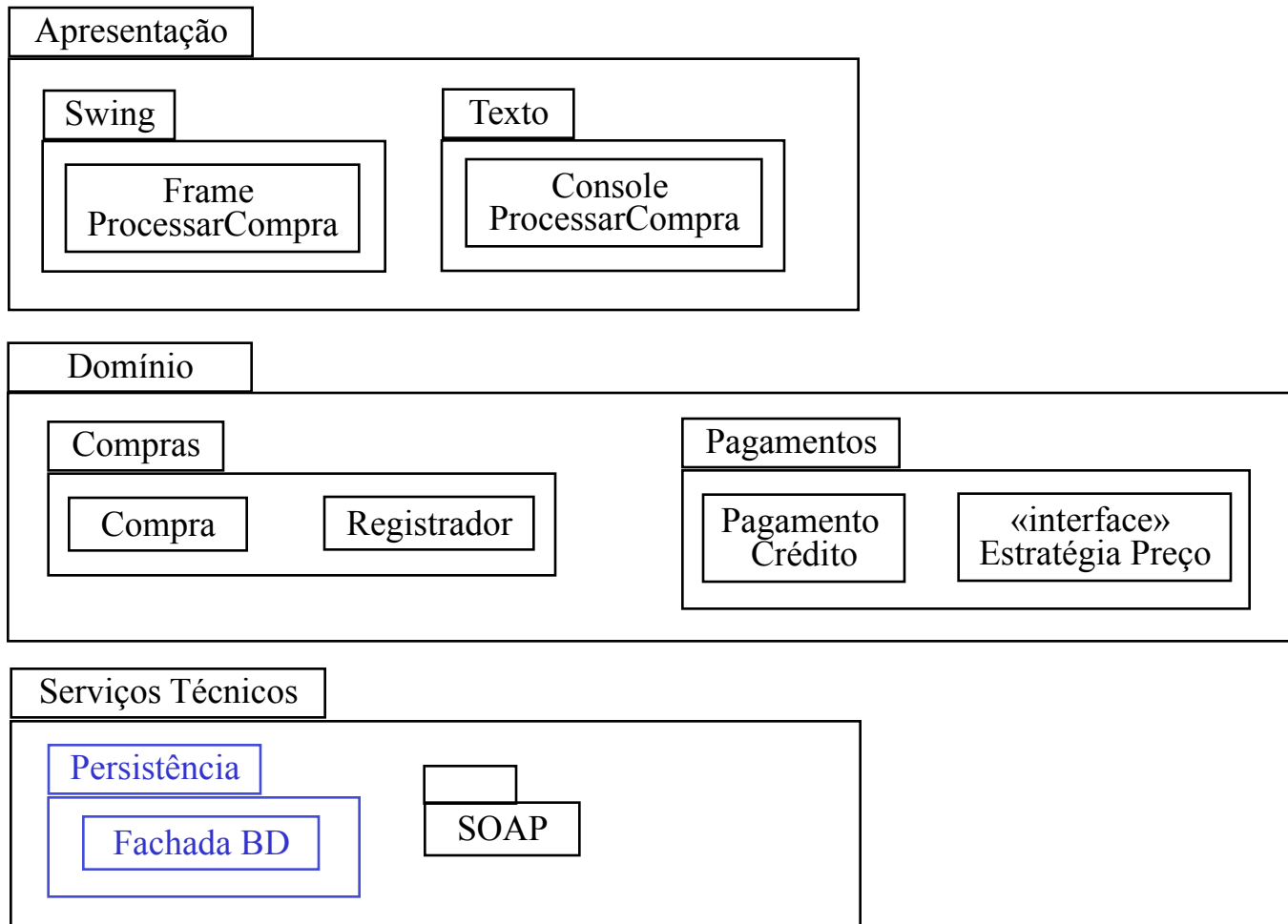



Mapeamento Indireto

Para cada classe é definido um mapeador que é responsável por salvar e carregar os objetos da classe na base de dados.

Camada de Persistência

Exemplo: Arquitetura em Camadas (Layers)





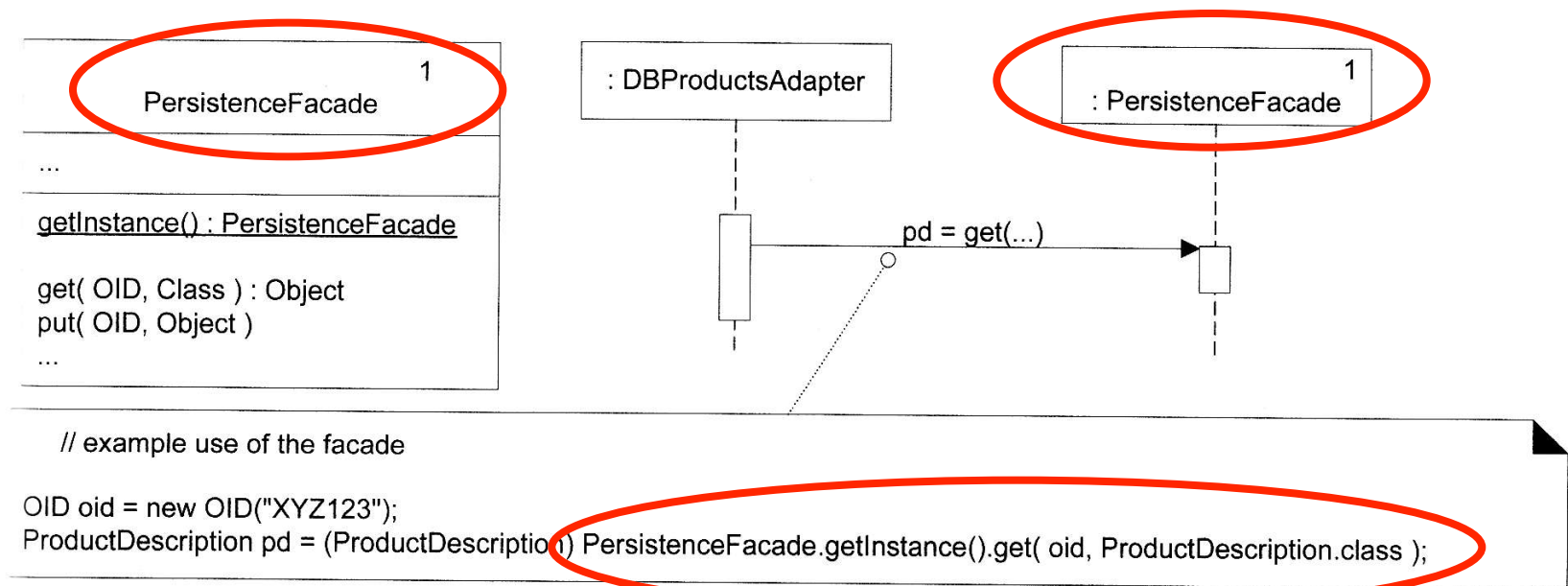
Elementos da Camada de Persistência

1. Fachada
2. Mapeador de Base de Dados
3. Materialização e desmaterialização
4. Caches

Camada de Persistência - Fachada

1. Fachada:

É definida uma fachada para os serviços da camada de persistência.



➡ A operação para recuperar um objeto precisa do OID do objeto e da classe do objeto.

Camada de Persistência - Mapeador

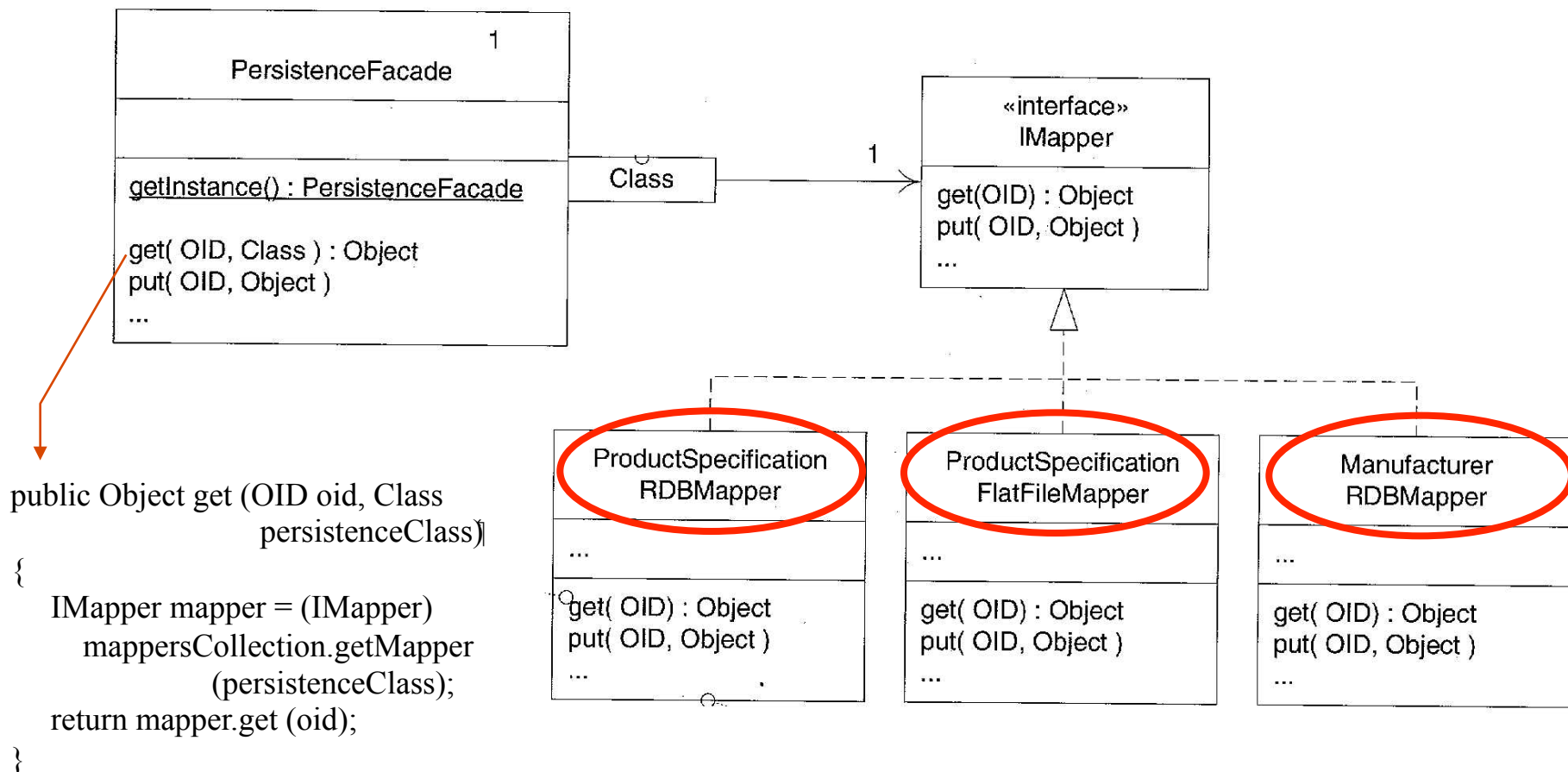
2. Mapeador de Base de Dados:

Como a fachada não faz os serviços de mapeamento, estes serviços são delegados aos mapeadores.

➡ Para cada classe é definido um mapeador que é responsável pela materialização, desmaterialização e caching dos objetos.

Camada de Persistência - Mapeador

Mapeadores das Classes de Objetos Persistentes



Camada de Persistência - Mapeador

Implementação do Mapeadores:

- Mapeadores individuais codificados a mão (código específico)
- 1 Único mapeador baseado em metadados (código genérico)

Gera dinamicamente o mapeamento a partir de um esquema de objeto (metadado que descreve o mapeamento) para a base de dados.

↳ Possível para linguagens com reflexão

Camada de Persistência - Materialização

3. Materialização e desmaterialização:

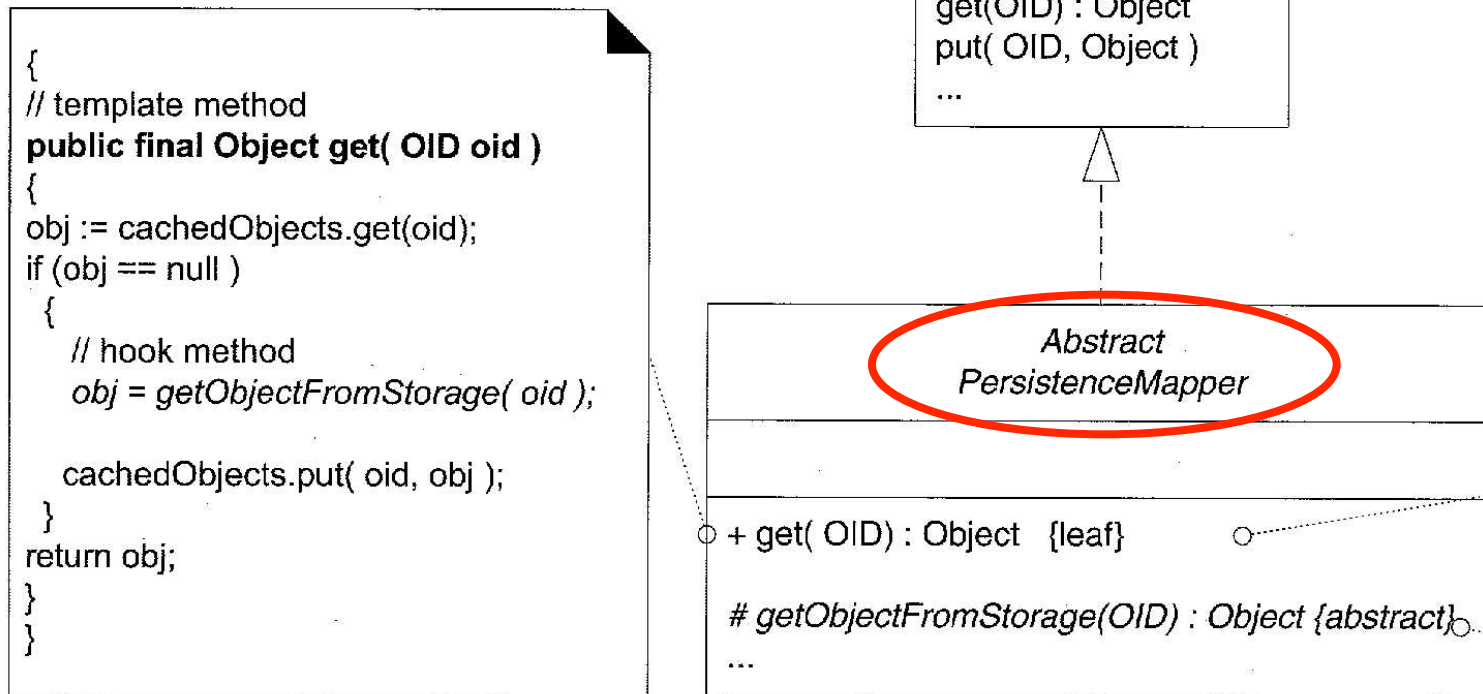
Nos mapeadores são definidos métodos para:

- carregar um objeto da base de dados (método get) - materialização
- salvar um objeto em uma base de dados (método put) - desmaterialização

➡ Como todos os mapeadores apresentam um código comum, este código pode ser definido em uma classe abstrata (método template).

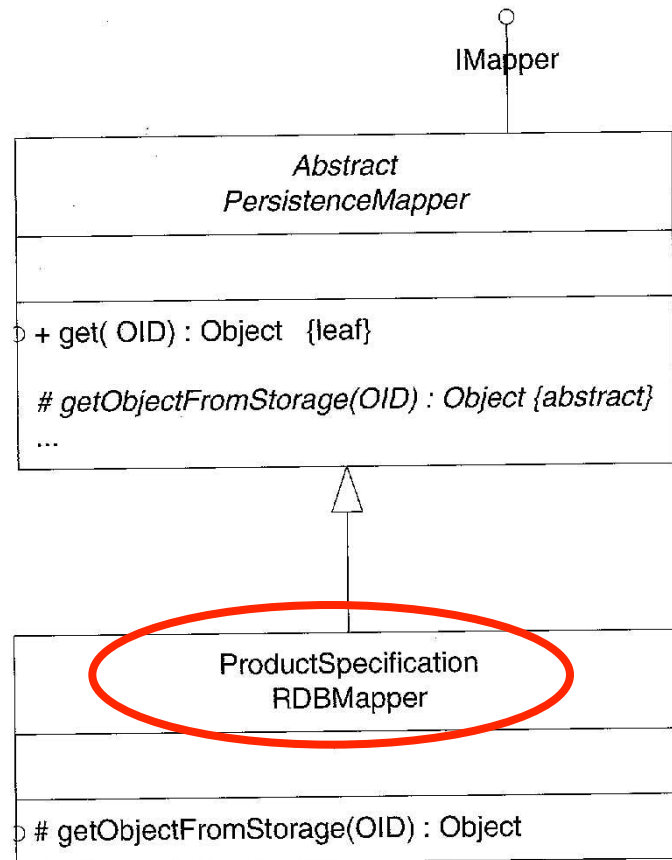
Camada de Persistência - Materialização

Classe Abstrata para o Mapeador



Camada de Persistência - Materialização

Para cada classe de objeto persistente, deve ser definida uma subclasse da classe PersistenceMapper.



Camada de Persistência - Materialização

Exemplo do método getObjectFromStorage do mapeador

```
protected Object getObjectFromStorage (OID oid)
{
    String key = oid.toString();
    dbRec = db.executeSql("SELECT PRICE,ITEM_ID,DESC
                           FROM PROD_DESC
                           WHERE OID = " + key);
    ProductDescription pd = null;
    if (dbRec.next()) {
        pd = new ProductSpecification ();
        pd.setOID (oid);
        pd.setPrice (dbRec. getDouble ("PRICE") );
        pd.setItemID (dbRec. getLong ("ITEM_ID") );
        pd.setDescrip (dbRec. getString ("DESC") );
    }
    dbRec.close(); // fecha o cursor do result set
    return pd;
}
```

Camada de Persistência - Materialização

Exemplo do método executeSql do mapeador


```
import java.sql.*;

public class Database {
    private Connection conn;
    private PreparedStatement stmt;

    public void connect(String driver, String url, String user, String password) throws Exception{
        Class.forName(driver);
        this.conn = DriverManager.getConnection(url,user,password); }

    public ResultSet executeSql(String sqlString) throws Exception{
        try { if (this.stmt != null) this.stmt.close(); } catch (Exception ignore) {}
        this.stmt = this.conn.prepareStatement(sqlString);
        return this.stmt.executeQuery(); }

    public void close() throws Exception {
        try { if (this.stmt != null) this.stmt.close(); } catch (Exception ignore) {}
        this.conn.close(); } }
```



Camada de Persistência - Materialização

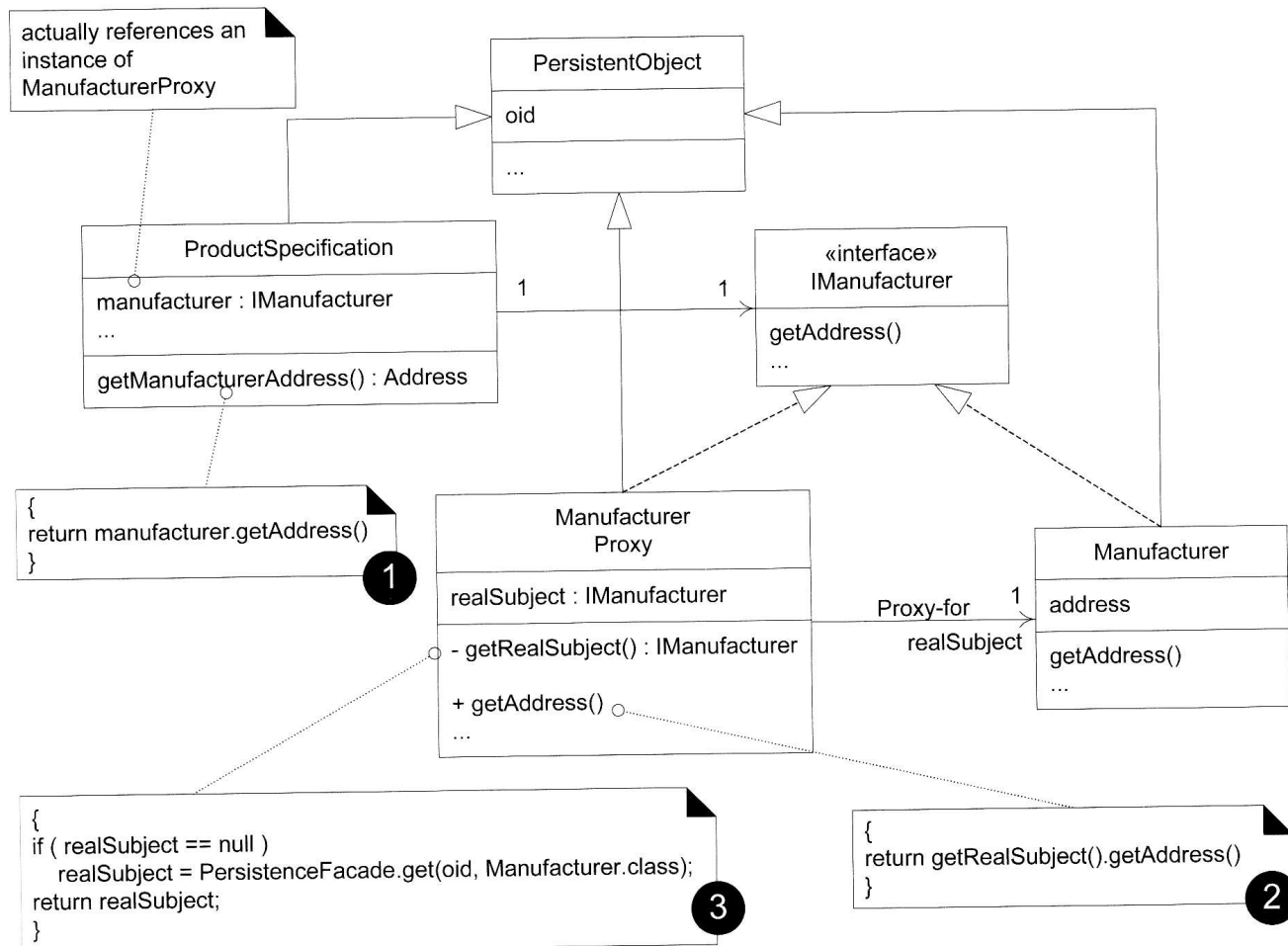
Lazy Materialization (Materialização Por Demanda):

Adia a materialização dos objetos referenciados.

Pode ser implementada usando o padrão de projeto Proxy.

Camada de Persistência - Materialização

Lazy Materialization (Materialização Por Demanda):






Camada de Persistência - Cache

4. Cache:

Para aumentar a performance, os objetos materializados são mantidos pelos mapeadores das classes em um cache.

↳ quando um objeto é carregado ou inserido na base de dados na primeira vez, ele é incluído no cache.



Mapeamento OO - Relacional

Mapeamento Estrutural

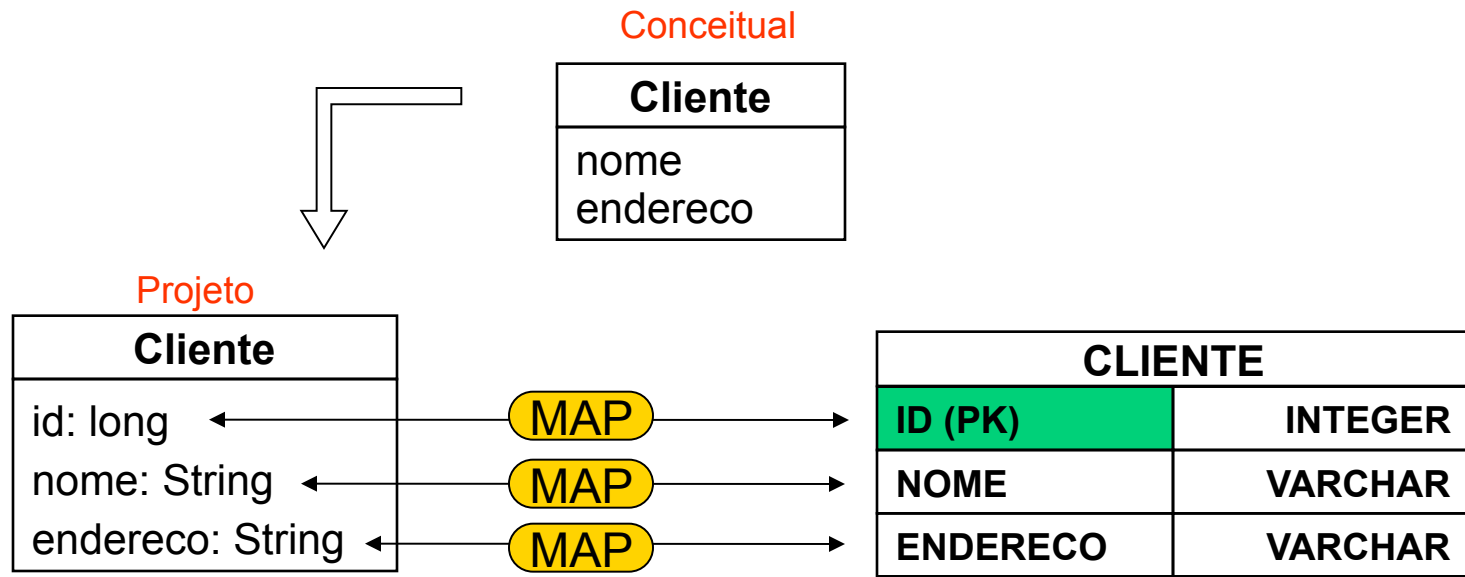
- Classes / Atributos
- Associações
- Herança

Mapeamento de Operações na Base de Dados

- Recuperação ou Instanciação
- Inserção
- Atualização
- Remoção

Classes - Mapeamento Estrutural

- ➡ Para cada atributo mapeado deve existir uma coluna na tabela correspondente cujo domínio é compatível com o tipo do atributo.
- ➡ O atributo identificador da classe deve estar mapeado para a coluna correspondente à chave primária. Caso não exista um atributo identificador na classe original, é necessário criá-lo.

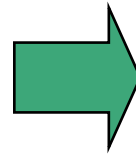


Exemplo


➡ Mapeamento da Classe Avaliador para a Tabela

Classe de Projeto

```
public class Avaliador extends Pessoa {  
    private String login;  
    private String senha;  
    private String email;  
    private String origem;  
    private List<Topico> topicos;  
    private List<Revisao> revisoes;
```



```
CREATE TABLE AVALIADOR (  
    NOME      VARCHAR(50),  
    LOGIN     VARCHAR(15),  
    SENHA     VARCHAR(8),  
    EMAIL     VARCHAR(30),  
    ORIGEM    VARCHAR(20),  
    PRIMARY KEY (NOME) )
```



Exercício

- Crie as tabelas do exercício da conferência no banco de dados.

Classes - Mapeamento Operações

Recuperação:

cli := new Cliente()

cli.id,cli.nome,cli.endereco:= SELECT ID, NOME, ENDERECO FROM CLIENTE WHERE ID = oid

Inserção:

INSERT INTO CLIENTE(ID,NOME,ENDERECO) VALUES (cli.id, cli.nome, cli.endereco)

Atualização:

UPDATE CLIENTE SET NOME = cli.nome, ENDERECO = cli.endereco WHERE ID = cli.id

Remoção:

DELETE FROM CLIENTE WHERE ID = cli.id

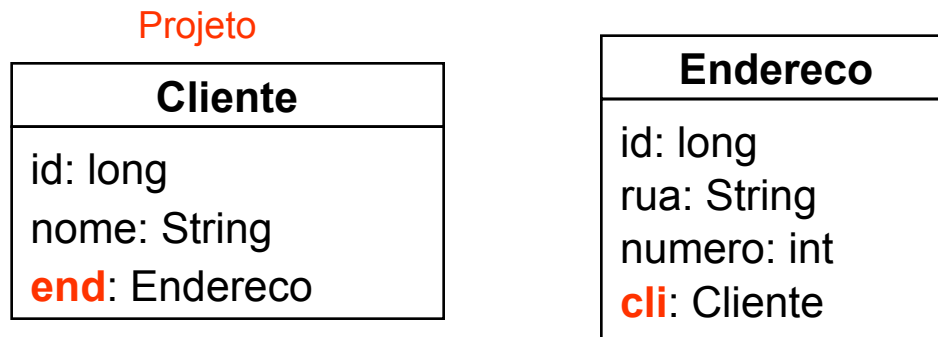
Exercício 1

- Complete o método put do mapeador MapeadorAutor (INSERT E UPDATE).
- Complete o método get do mapeador MapeadorAutor (SELECT).

Obs: - Utilize como base o MapeadorChefeComite.

Associações 1:1 - Mapeamento Estrutural

- As classes associadas podem ser mapeadas para uma mesma tabela ou tabelas separadas.

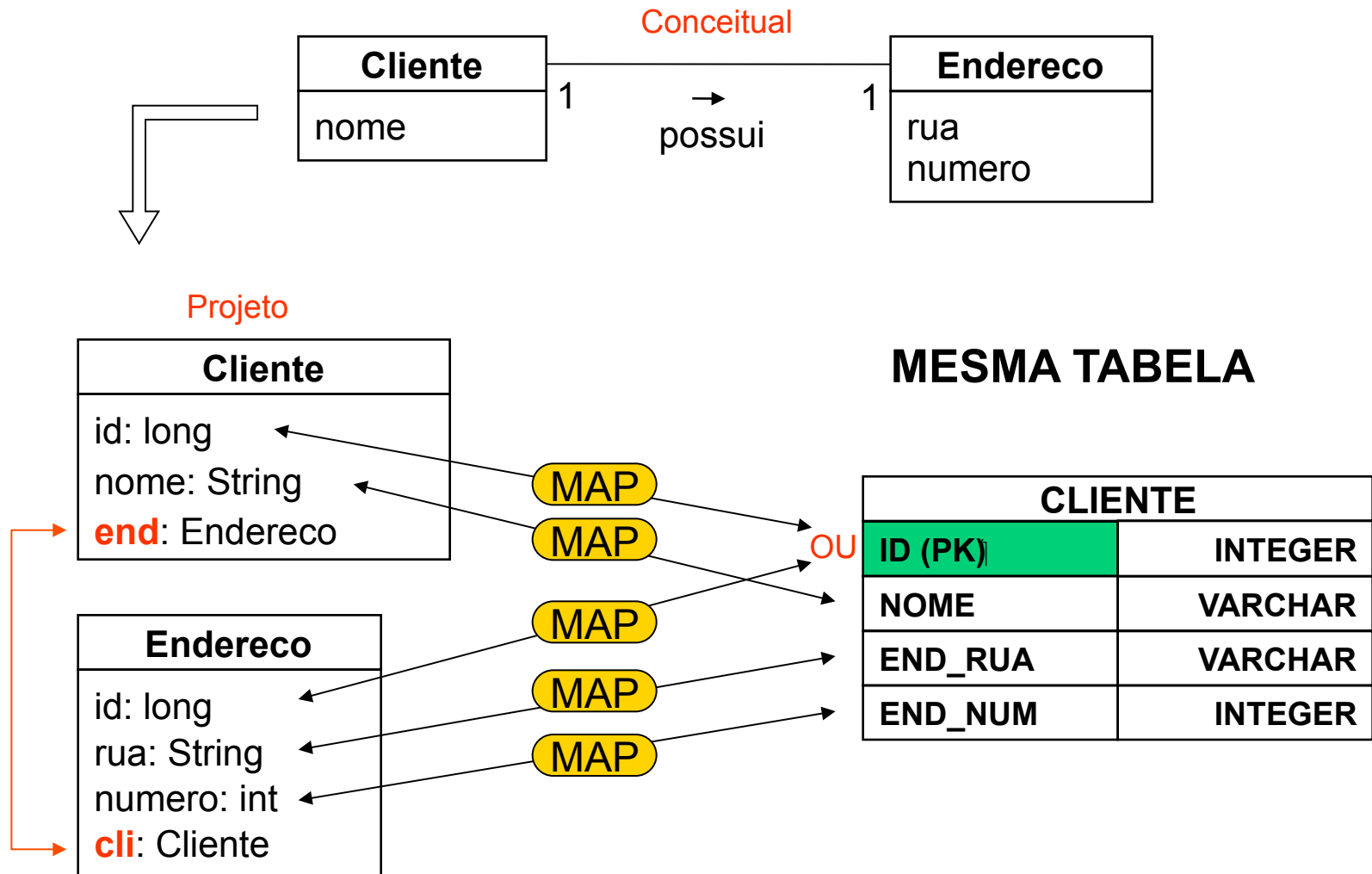


- Se as classes são mapeadas para uma mesma tabela, o mapeamento é direto.

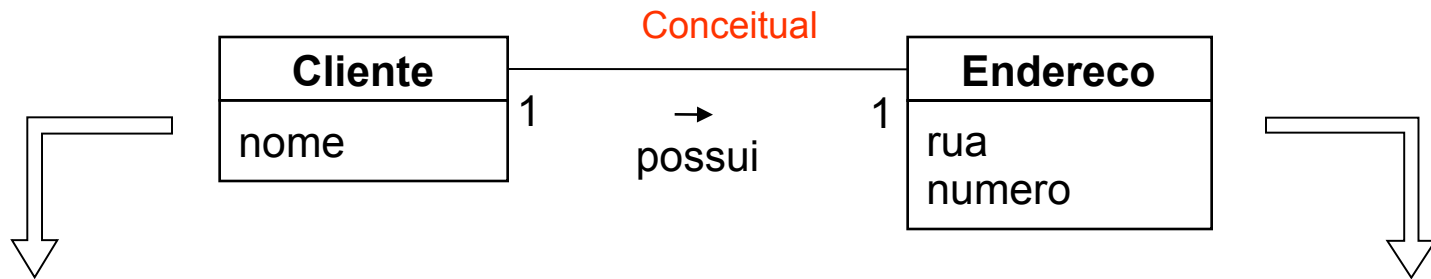
- Se as classes mapeiam para tabelas separadas, uma chave estrangeira precisa estar definida em uma das tabelas.

Obs: É conveniente que as classes associadas refiram-se mutuamente através de referências inversas.

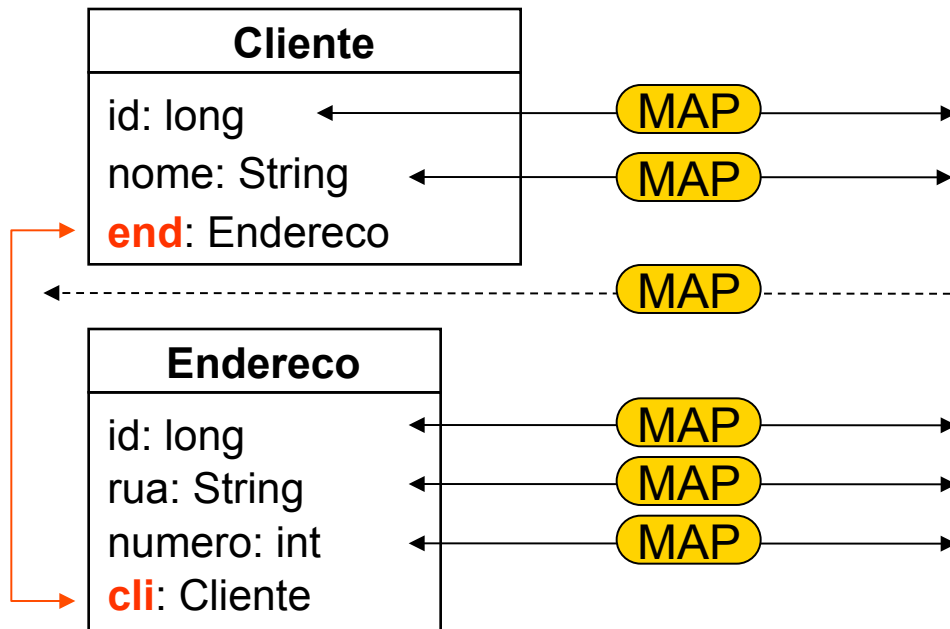
Associações 1:1 - Mapeamento Estrutural



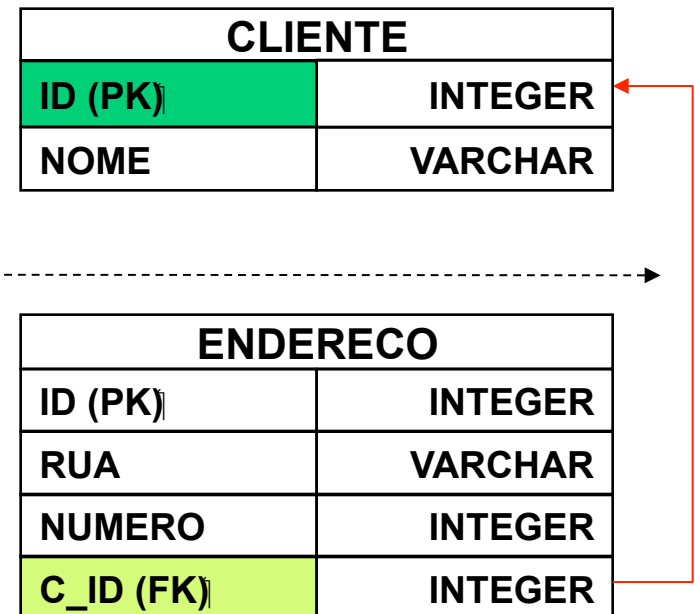
Associações 1:1 - Mapeamento Estrutural



Projeto



TABELAS SEPARADAS



Associações 1:1 - Mapeamento Operações

Caso 1: Cliente(1) -> Endereco(1)

- Recuperação: a partir do **oid** de um cliente

```
cli := new Cliente()
```

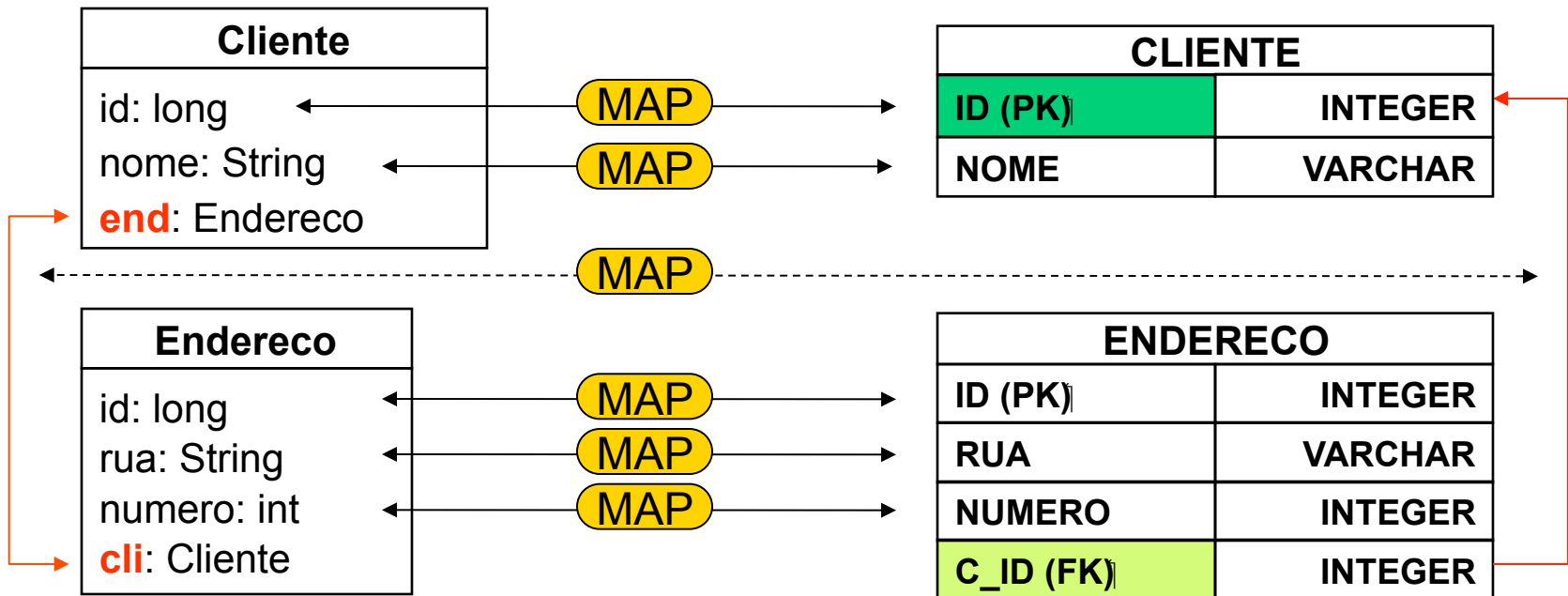
```
cli.id, cli.nome := SELECT ID, NOME FROM CLIENTE WHERE ID = oid
```

```
end := new Endereco()
```

```
end.id, end.rua, end.numero := SELECT ID, RUA, NUMERO FROM ENDERECO  
WHERE C_ID = cli.id
```

```
cli.end := end
```

```
end.cli := cli
```



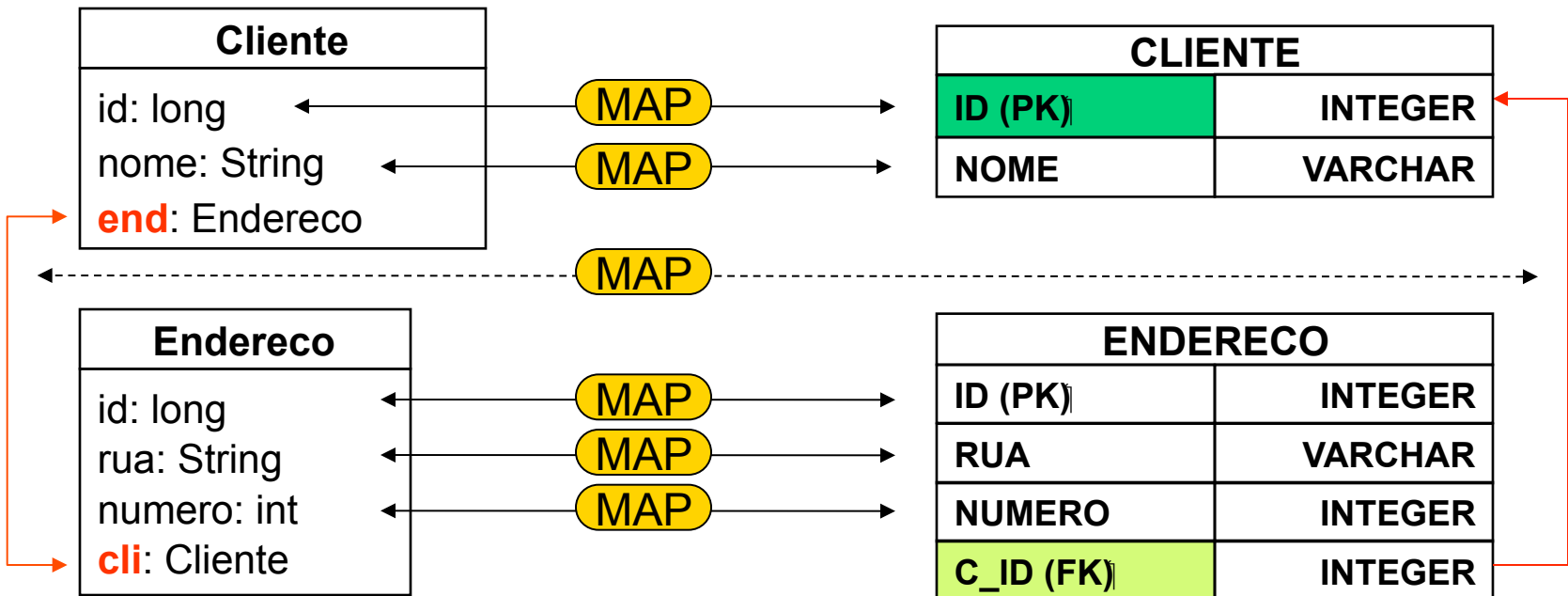
Associações 1:1 - Mapeamento Operações

Caso 1: Cliente(1) -> Endereco(1)

- Inserção: a partir de um cliente

```
INSERT INTO CLIENTE(ID,NOME) VALUES (cli.id,cli.nome)
```

```
INSERT INTO ENDERECO(ID,RUA,NUMERO,C_ID) VALUES (cli.end.id, cli.end.rua,  
cli.end.numero, cli.id)
```



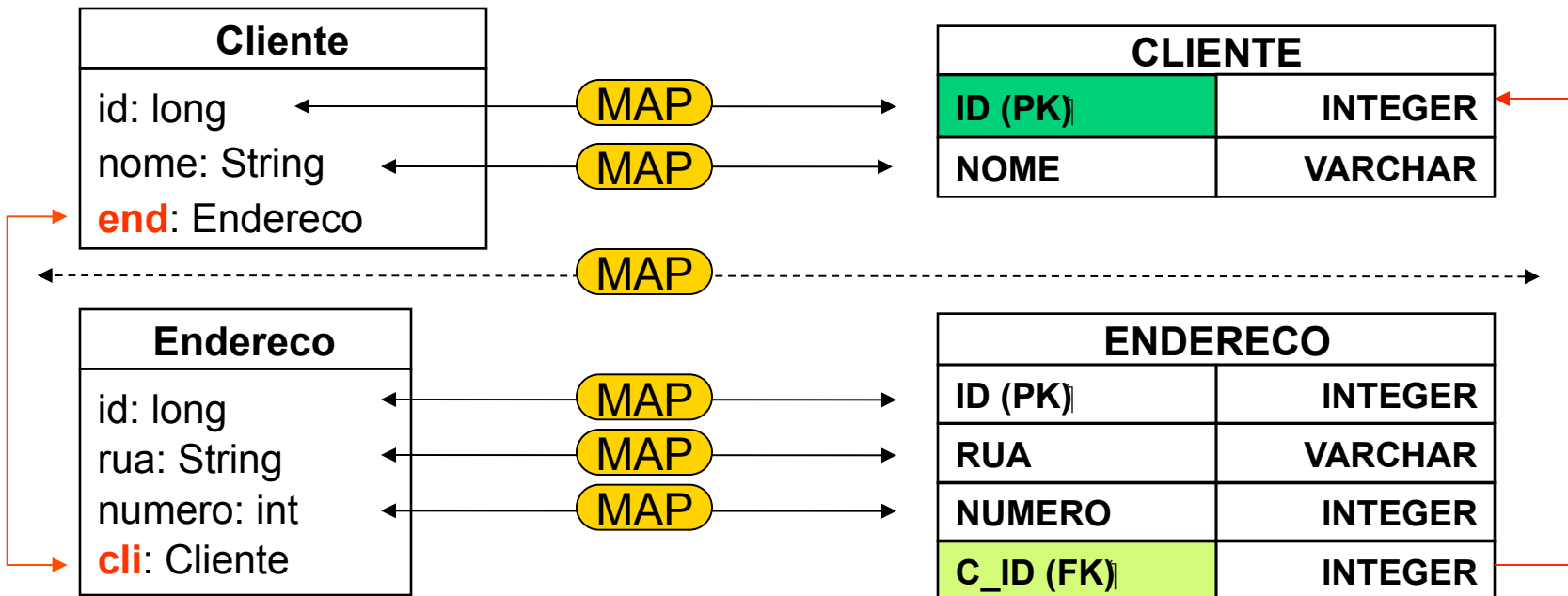
Associações 1:1 - Mapeamento Operações

Caso 1: Cliente(1) -> Endereco(1)

- Atualização: a partir de um cliente

UPDATE CLIENTE SET NOME = cli.nome WHERE ID = cli.id

UPDATE ENDERECO SET RUA = cli.end.rua, NUMERO = cli.end.numero
WHERE ID = cli.end.id

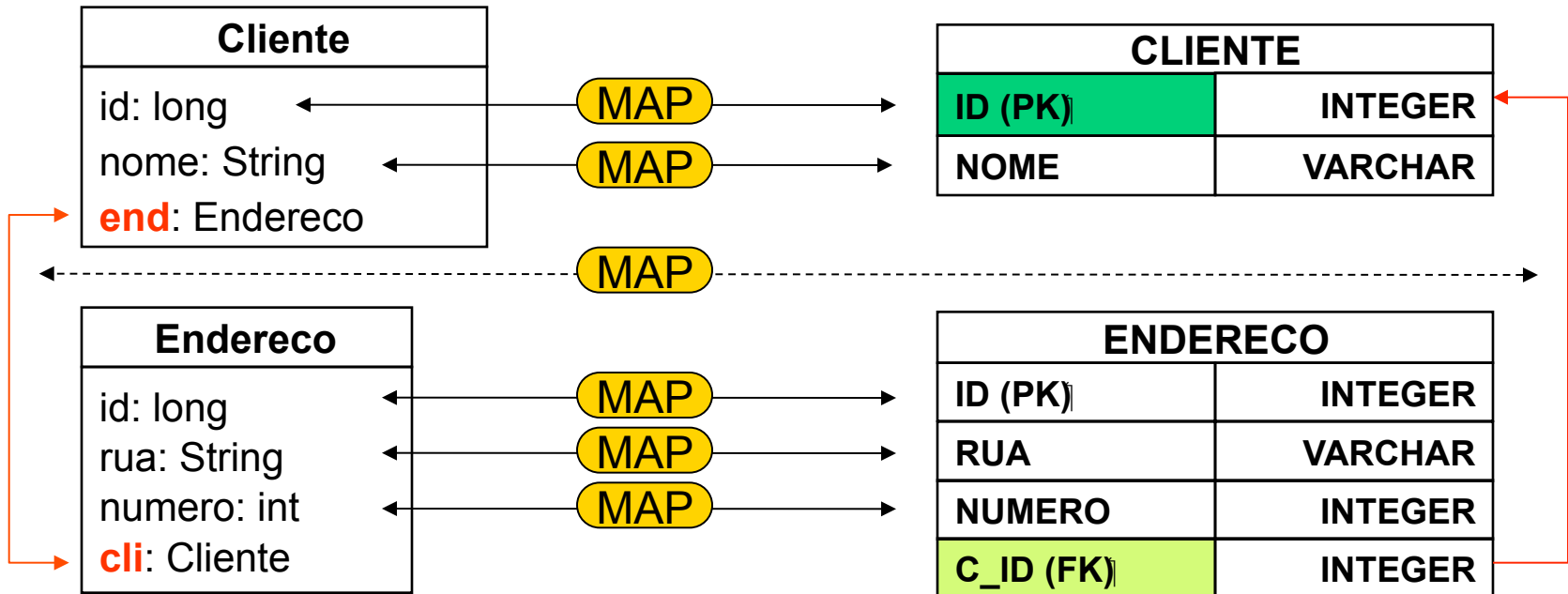


Associações 1:1 - Mapeamento Operações

Caso 1: Cliente(1) -> Endereco(1)

- Remoção: a partir de um cliente

```
DELETE FROM ENDERECO WHERE C_ID = cli.id * OU
UPDATE ENDERECO SET C_ID = NULL WHERE C_ID = :cli.id **
DELETE FROM CLIENTE WHERE ID = cli.id
```



*cardinalidade mínima de Cliente em Endereço é 1 (Endereço tem que ter um Cliente)

** cardinalidade mínima de Cliente em Endereco é 0 (Endereço pode não ter um Cliente)

Associações 1:1 - Mapeamento Operações

Caso 2: Endereco(1) → Cliente(1)

- Recuperação: a partir do **oid** de um endereço

```
end := new Endereco()
```

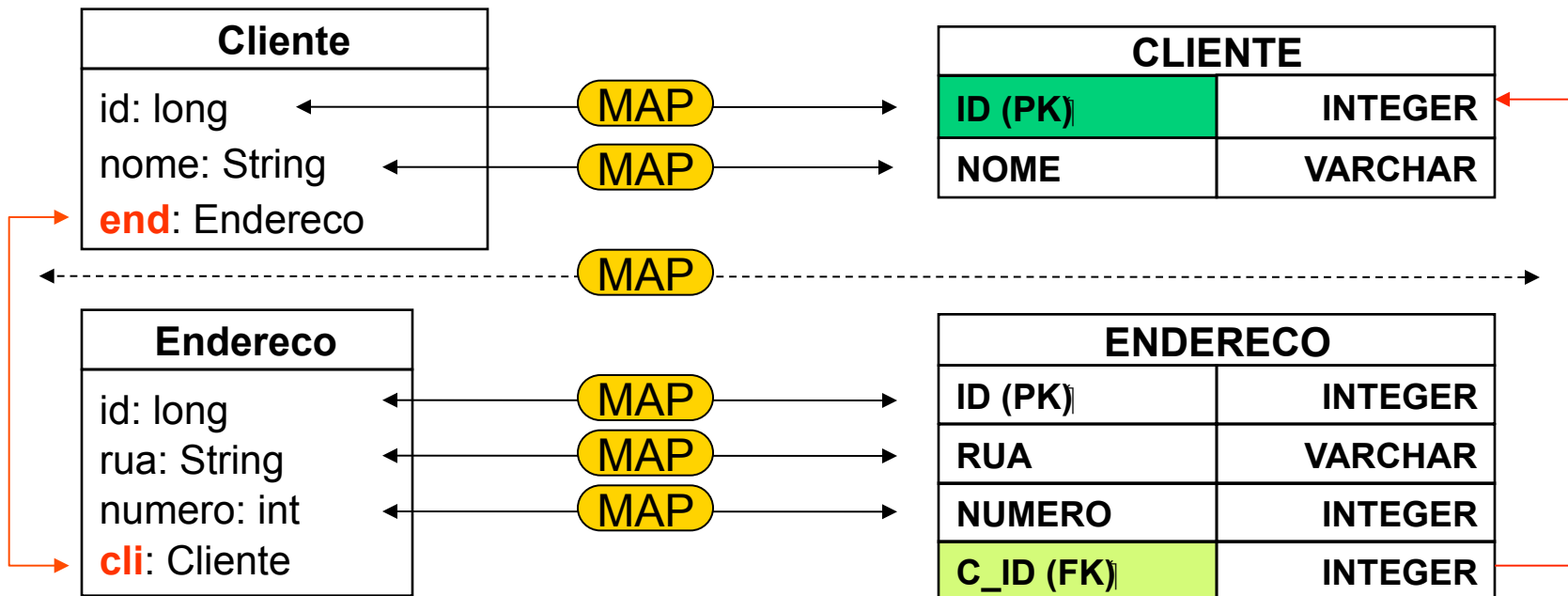
```
end.id, end.rua, end.numero, var_cid := SELECT ID, RUA, NUMERO, C_ID  
FROM ENDERECO WHERE ID = oid
```

```
cli := new Cliente()
```

```
cli.id, cli.nome := SELECT ID, NOME FROM CLIENTE WHERE ID = var_cid
```

```
end.cli := cli
```

```
cli.end := end
```



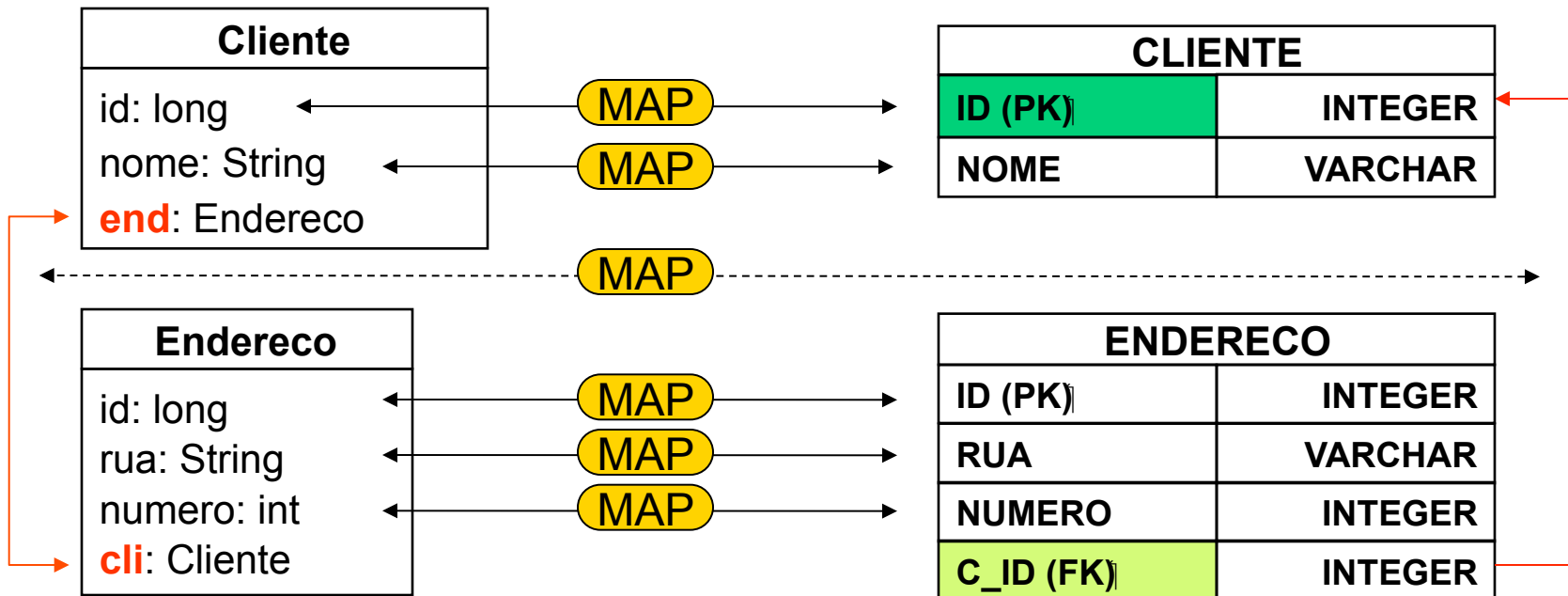
Associações 1:1 - Mapeamento Operações

Caso 2: Endereco(1) -> Cliente(1)

- Inserção: a partir de um endereço

INSERT INTO CLIENTE(ID,NOME) VALUES (end.cli.id, end.cli.nome)

INSERT INTO ENDERECO(ID,RUA,NUMERO,C_ID) VALUES (end.id, end.rua,
end.numero, end.cli.id)



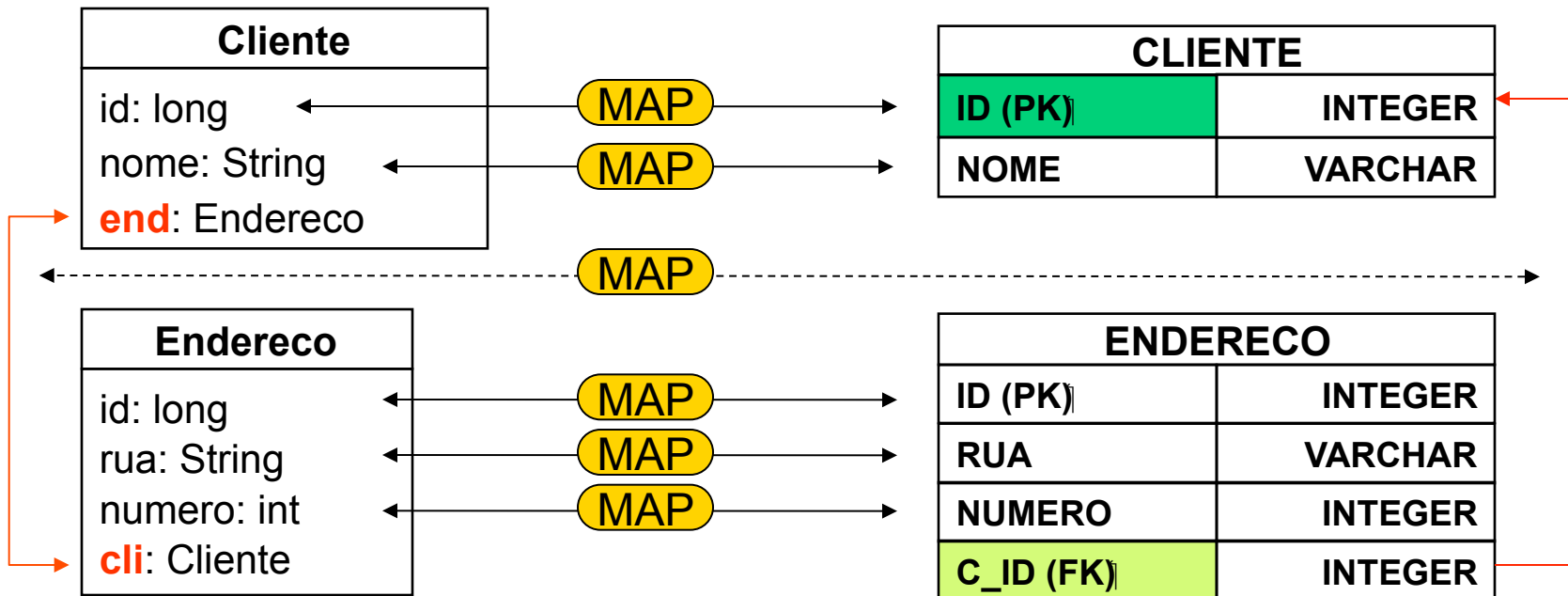
Associações 1:1 - Mapeamento Operações

Caso 2: Endereco(1) -> Cliente(1)

- Atualização: a partir de um endereço

UPDATE ENDERECO SET RUA = end.rua, NUMERO = end.numero WHERE ID = end.id

UPDATE CLIENTE SET NOME = end.cli.nome WHERE ID = end.cli.id



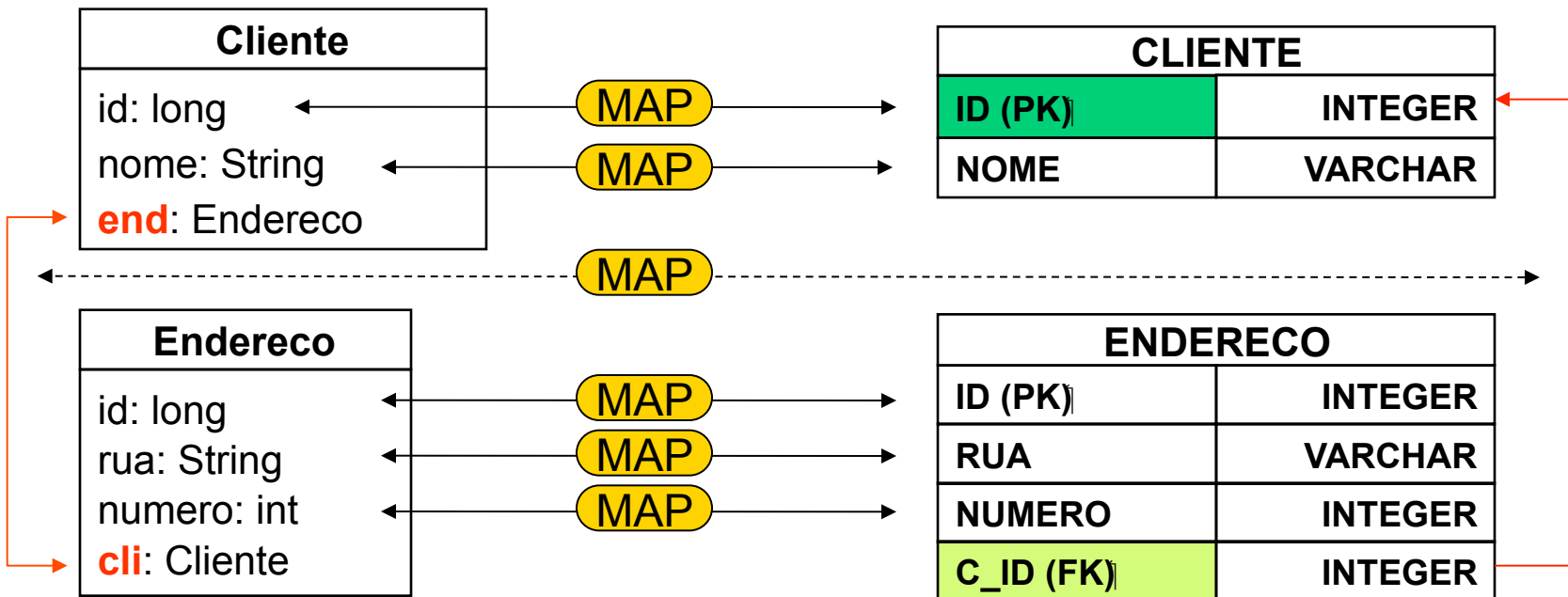
Associações 1:1 - Mapeamento Operações

Caso 2: Endereco(1) → Cliente(1)

- Remoção: a partir de um endereço

DELETE FROM ENDERECO WHERE ID = end.id

DELETE FROM CLIENTE WHERE ID = end.cli.id *



*cardinalidade mínima de Endereço em Cliente é 1 (Cliente tem que ter um Endereço). Caso contrário, não faz nada.

Associações 1:N - Mapeamento Estrutural

- ➡ As classes associadas devem estar mapeadas em tabelas separadas
- ➡ A tabela correspondente à classe do lado N da associação deve conter uma chave estrangeira para a tabela correspondente à classe do lado 1

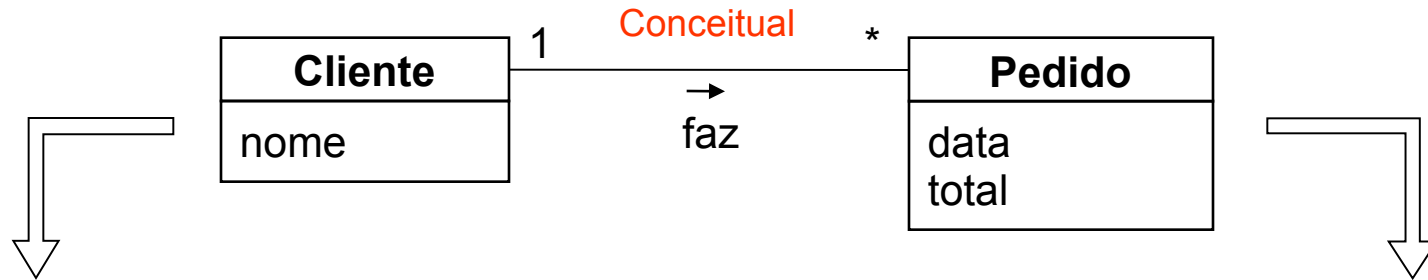
Projeto

Cliente
id: long nome: String pedidos : Collection

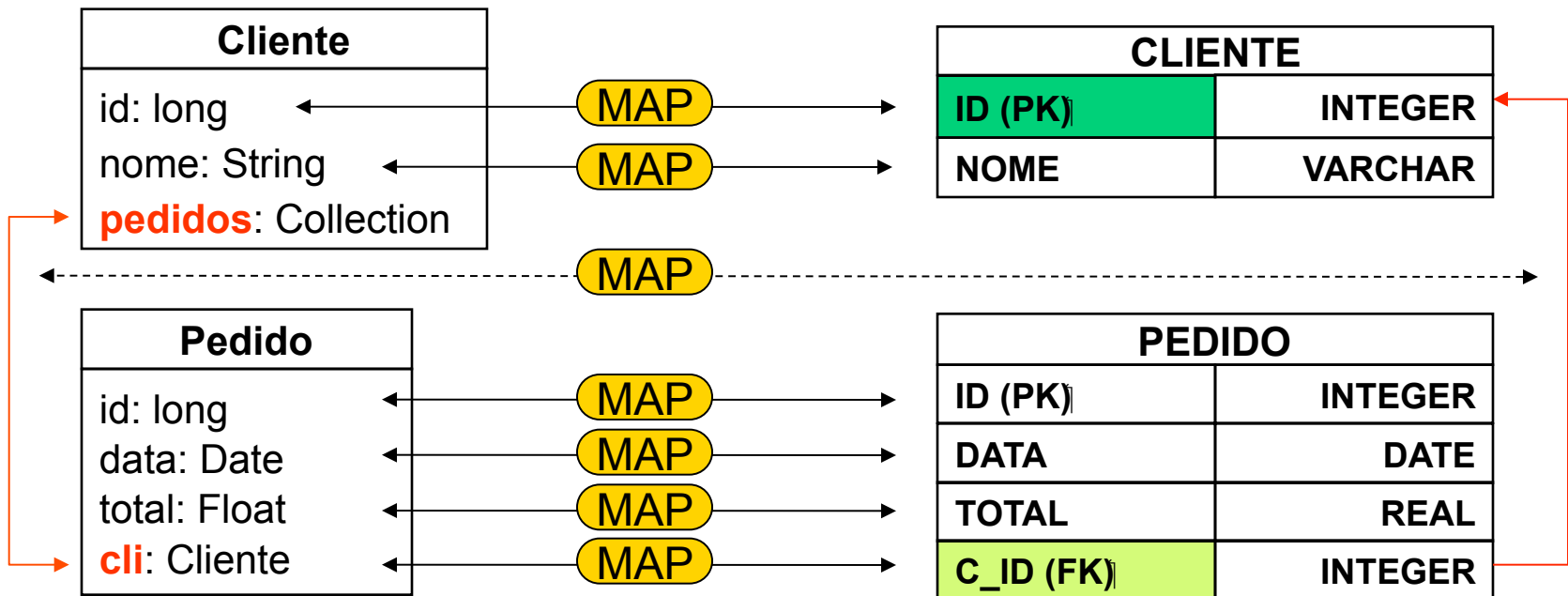
Pedido
id: long data: Date total: Float cli : Cliente

Obs: É conveniente que as classes associadas refiram-se mutuamente através de referências inversas

Associações 1:N - Mapeamento Estrutural



Projeto



Exemplo

➡ Mapeamento da Associação Avaliador-Revisão para a Tabela

Classe de Projeto

```
public class Avaliador extends Pessoa {  
    private String login;  
    private String senha;  
    private String email;  
    private String origem;  
    private List<Topico> topicos;  
    private List<Revisao> revisoes;  
  
    public class Revisao {  
        private double clareza;  
        ...  
        private Artigo artigo;  
        private Avaliador avaliador;  
    }  
}
```

```
CREATE TABLE REVISAO (  
    NOME_ARTIGO    VARCHAR(50),  
    NOME_AVALIADOR VARCHAR(50),  
    CLAREZA DOUBLE,  
    PRIMARY KEY (NOME_ARTIGO,NOME_AVALIADOR),  
    FOREIGN KEY (NOME_ARTIGO) REFERENCES  
        ARTIGO(NOME),  
    FOREIGN KEY (NOME_AVALIADOR) REFERENCES  
        AVALIADOR(NOME) )
```

Associações 1:N - Mapeamento Operações

Caso 1: Cliente(1) → Pedido(N)

- Recuperação: a partir do **oid** do cliente

cli := new Cliente()

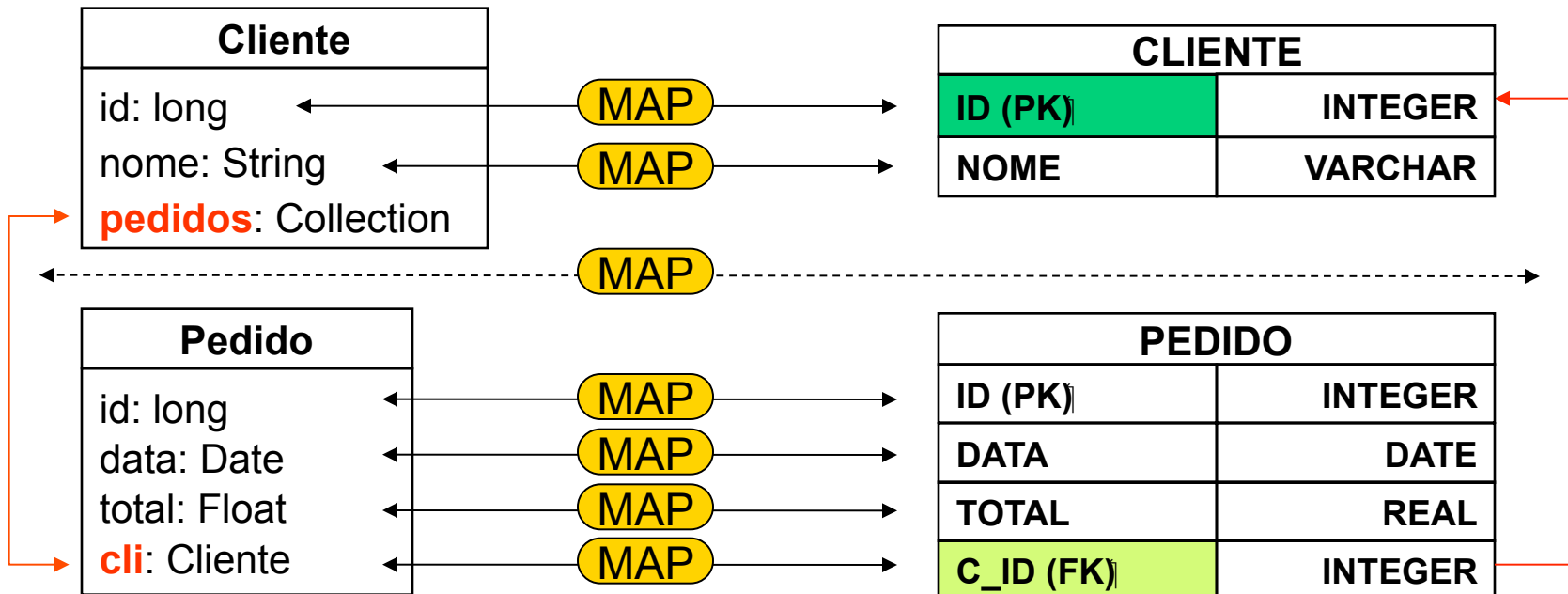
cli.id, cli.nome := SELECT ID, NOME FROM CLIENTE WHERE ID = **oid**

pedidos(id,data,total)* := SELECT ID, DATA, TOTAL FROM PEDIDO
WHERE **C_ID** = cli.id (instancia coleção)

cli.pedidos := pedidos

p/ cada :ped em cli.pedidos

ped.cli := cli



Associações 1:N - Mapeamento Operações

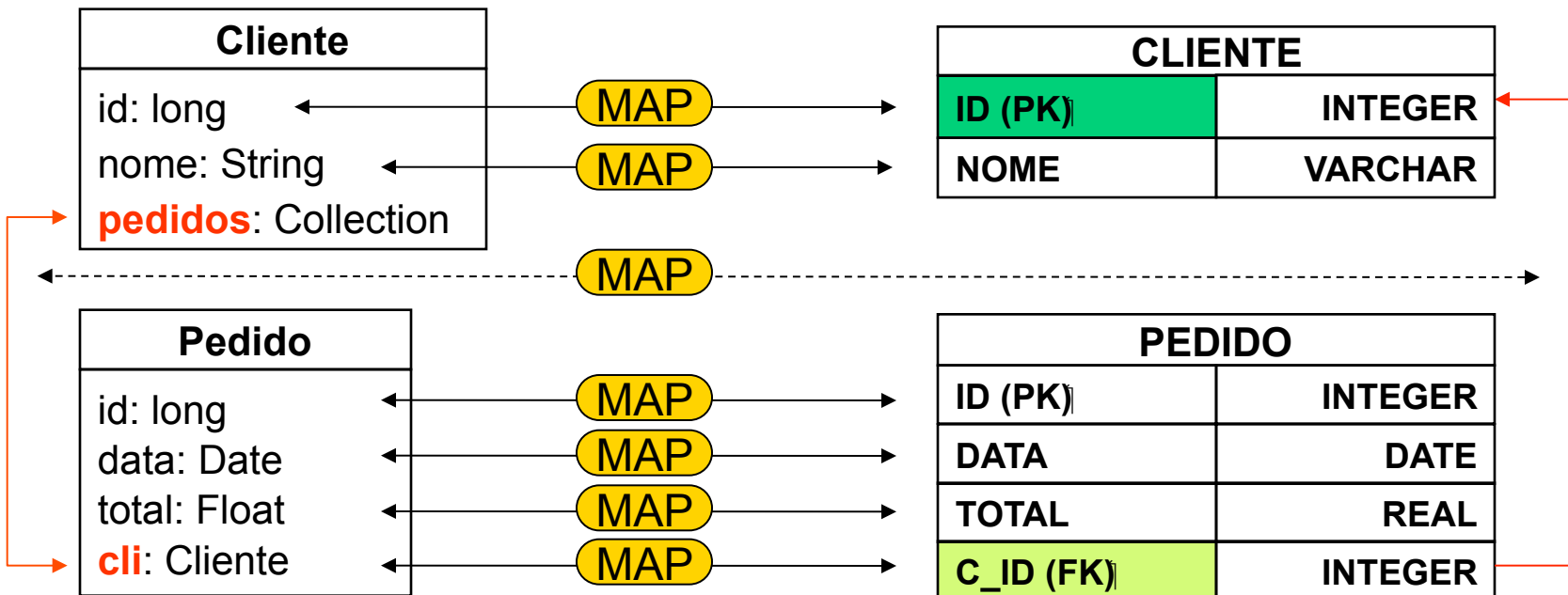
Caso 1: Cliente(1) → Pedido(N)

- Inserção: a partir de um cliente

INSERT INTO CLIENTE(ID,NOME) VALUES (cli.id, cli.nome)

p/ cada pedido :ped em cli.pedidos

INSERT INTO PEDIDO(ID,DATA,TOTAL,C_ID) VALUES (ped.id, ped.data, ped.total,
cli.id)



Associações 1:N - Mapeamento Operações

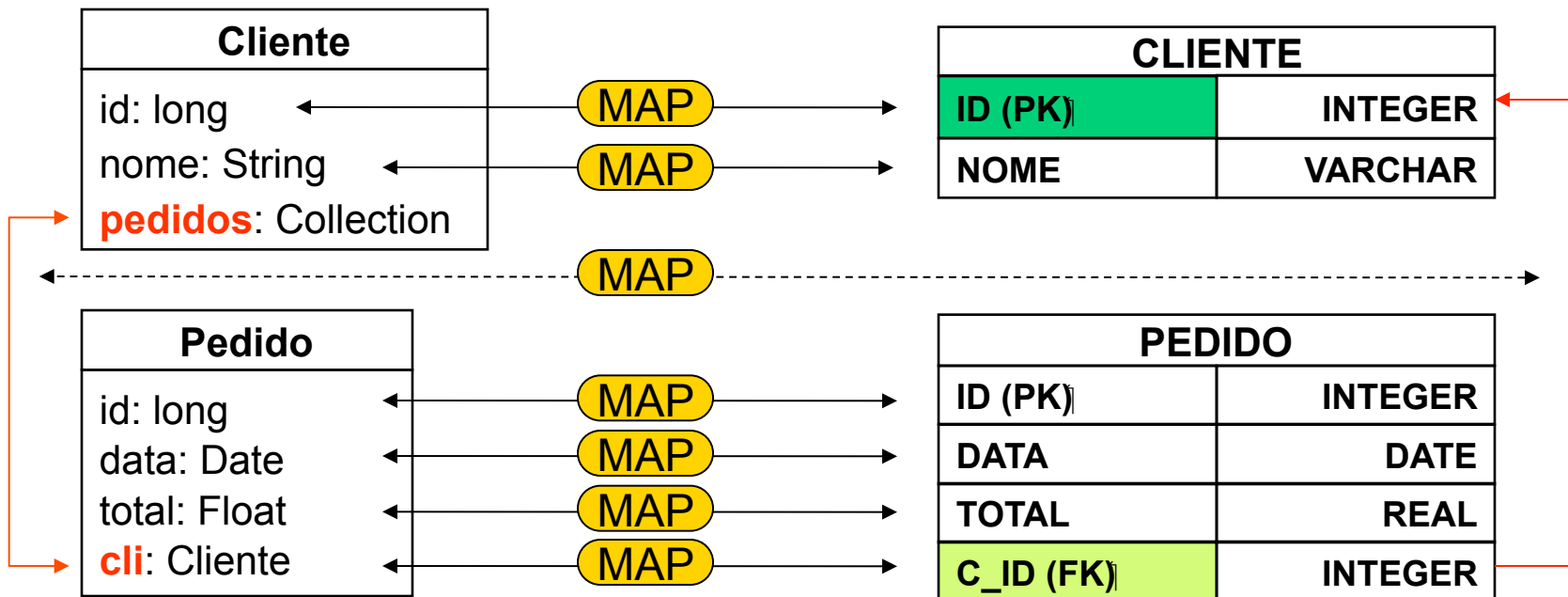
Caso 1: Cliente(1) → Pedido(N)

- Atualização: a partir de um cliente

UPDATE CLIENTE SET NOME = cli.nome WHERE ID = cli.id

p/ cada pedido :ped em cli.pedidos

UPDATE PEDIDO SET DATA = ped.data, TOTAL = ped.total WHERE ID = ped.id

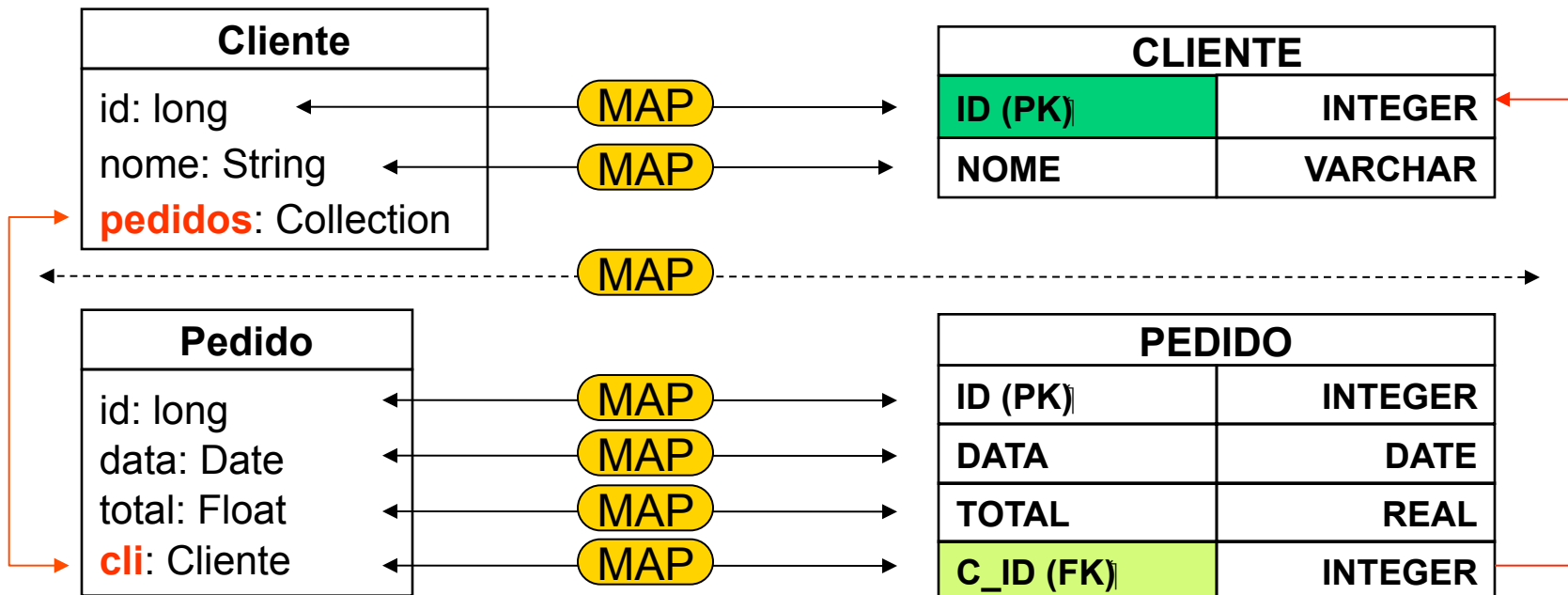


Associações 1:N - Mapeamento Operações

Caso 1: Cliente(1) → Pedido(N)

- Remoção: a partir de um cliente

DELETE FROM PEDIDO WHERE C_ID = cli.id * OU
UPDATE PEDIDO SET C_ID = NULL WHERE C_ID = cli.id **
DELETE FROM CLIENTE WHERE ID = cli.id



*cardinalidade mínima de Cliente em Pedido é 1 (Pedido tem que ter um Cliente)

** cardinalidade mínima de Cliente em Pedido é 0 (Pedido pode não ter um Cliente)

Exercício 2

- Complete o método getRevisoes do mapeador MapeadorAvaliador (SELECT), que é chamado pelo método get.

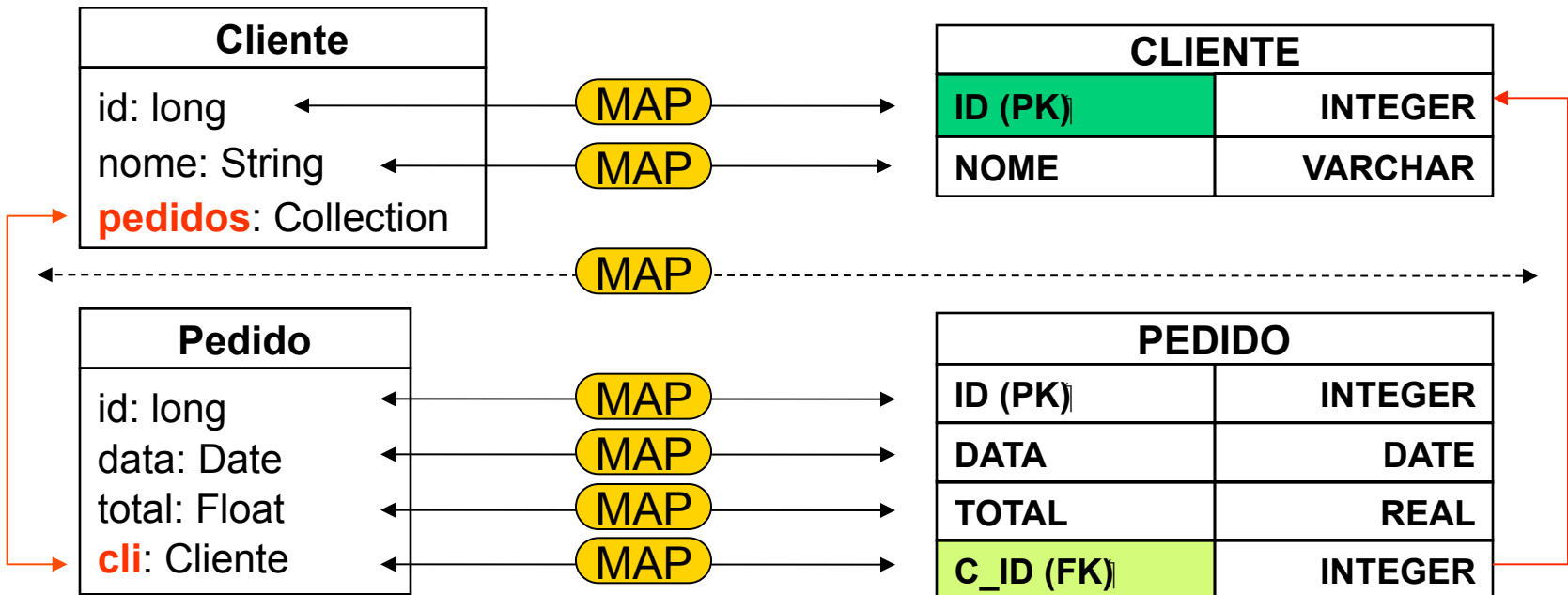
Obs: - Utilize como base o método getTopicos do MapeadorAvaliador.

Associações 1:N - Mapeamento Operações1

Caso 2: Pedido(N) → Cliente(1)

- Recuperação: a partir do **oid** de um pedido

```
ped := new Pedido()  
ped.id, ped.data, ped.total, var_cid := SELECT ID, DATA, TOTAL, C_ID  
FROM PEDIDO WHERE ID = oid  
  
cli := new Cliente()  
cli.id, cli.nome := SELECT ID, NOME FROM CLIENTE WHERE ID = var_cid  
  
ped.cli := cli  
cli.pedidos.add(ped)
```



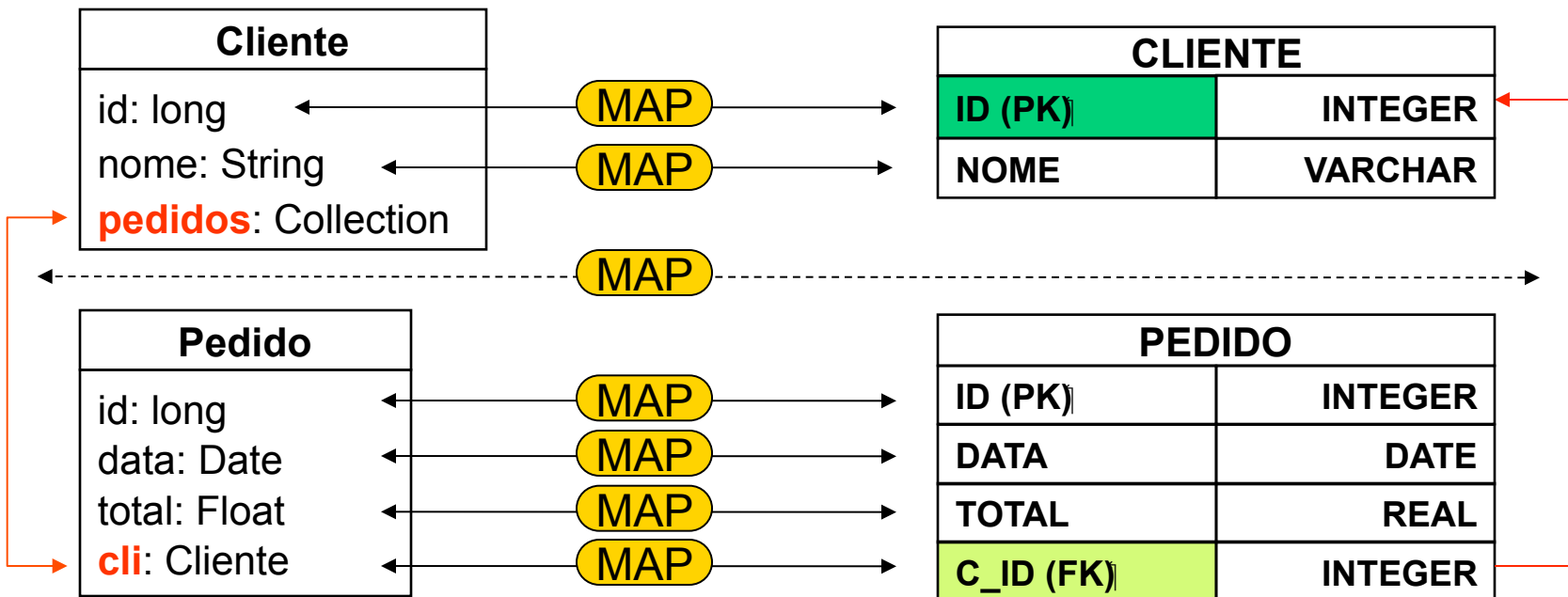
Associações 1:N - Mapeamento Operações

Caso 2: Pedido(N) -> Cliente(1)

- Inserção: a partir de um pedido

INSERT INTO CLIENTE(ID,NOME) VALUES (ped.cli.id, ped.cli.nome) // se cliente ainda não existia

INSERT INTO PEDIDO(ID,DATA,TOTAL,C_ID) VALUES (ped.id, ped.data, ped.total,
ped.cli.id)



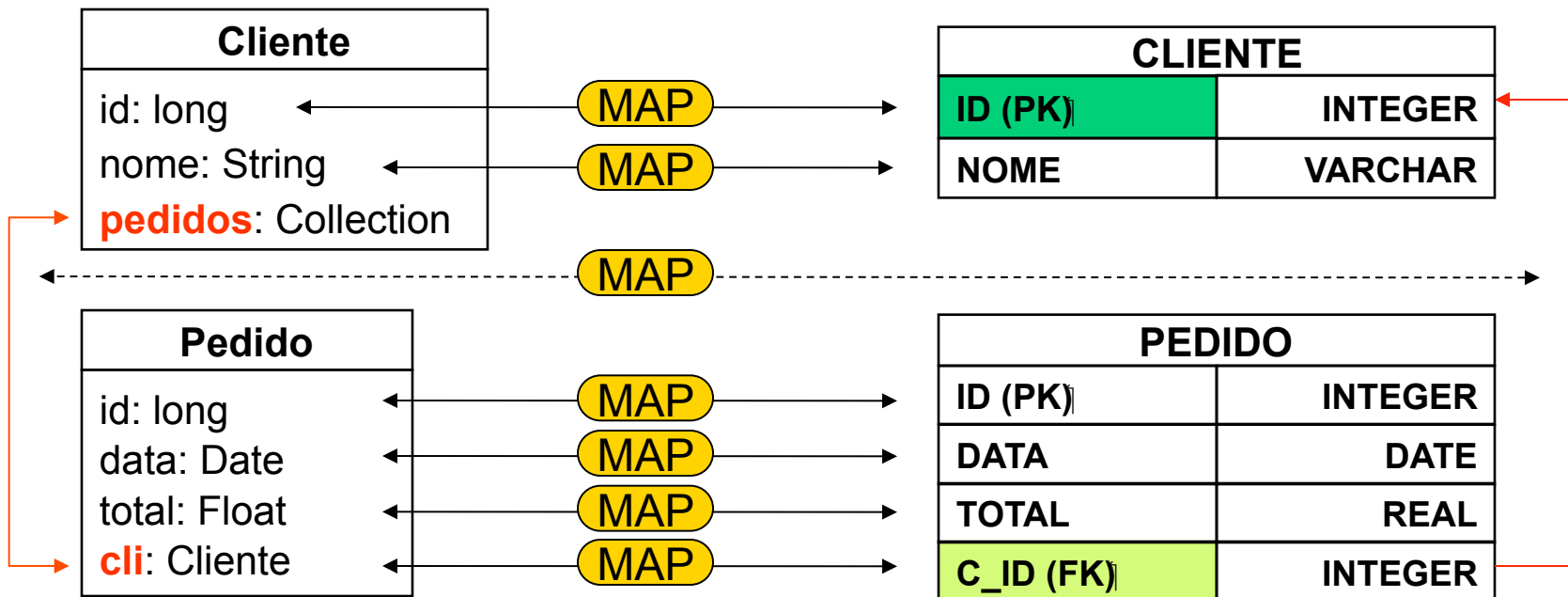
Associações 1:N - Mapeamento Operações

Caso 2: Pedido(N) → Cliente(1)

- Atualização: a partir de um pedido

UPDATE PEDIDO SET DATA = ped.data, TOTAL = ped.total WHERE ID = ped.id

UPDATE CLIENTE SET NOME = ped.cli.nome WHERE ID = ped.cli.id



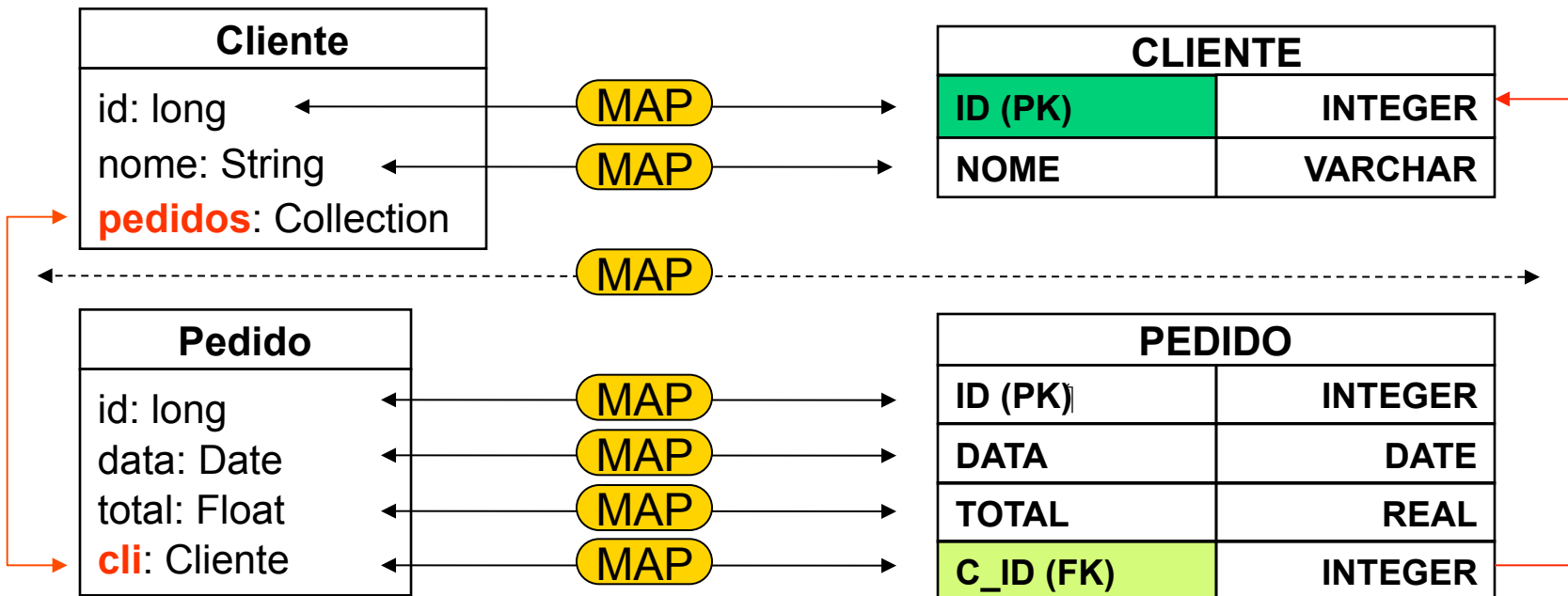
Associações 1:N - Mapeamento Operações

Caso 2: Pedido(N) → Cliente(1)

- Remoção: a partir de um pedido

DELETE FROM PEDIDO WHERE ID = ped.id

DELETE FROM CLIENTE WHERE ID = ped.cli.id *



* Se cardinalidade mínima de Pedido em Cliente é 1, remoção só deve ocorrer se `ped.cli.pedidos.size() = 1`

Associações N:N - Mapeamento Estrutural

- ➡ As classes associadas devem estar mapeadas em tabelas separadas
- ➡ Uma terceira tabela deve ser definida para representar a associação, com chaves estrangeiras que se referem às tabelas das classes associadas

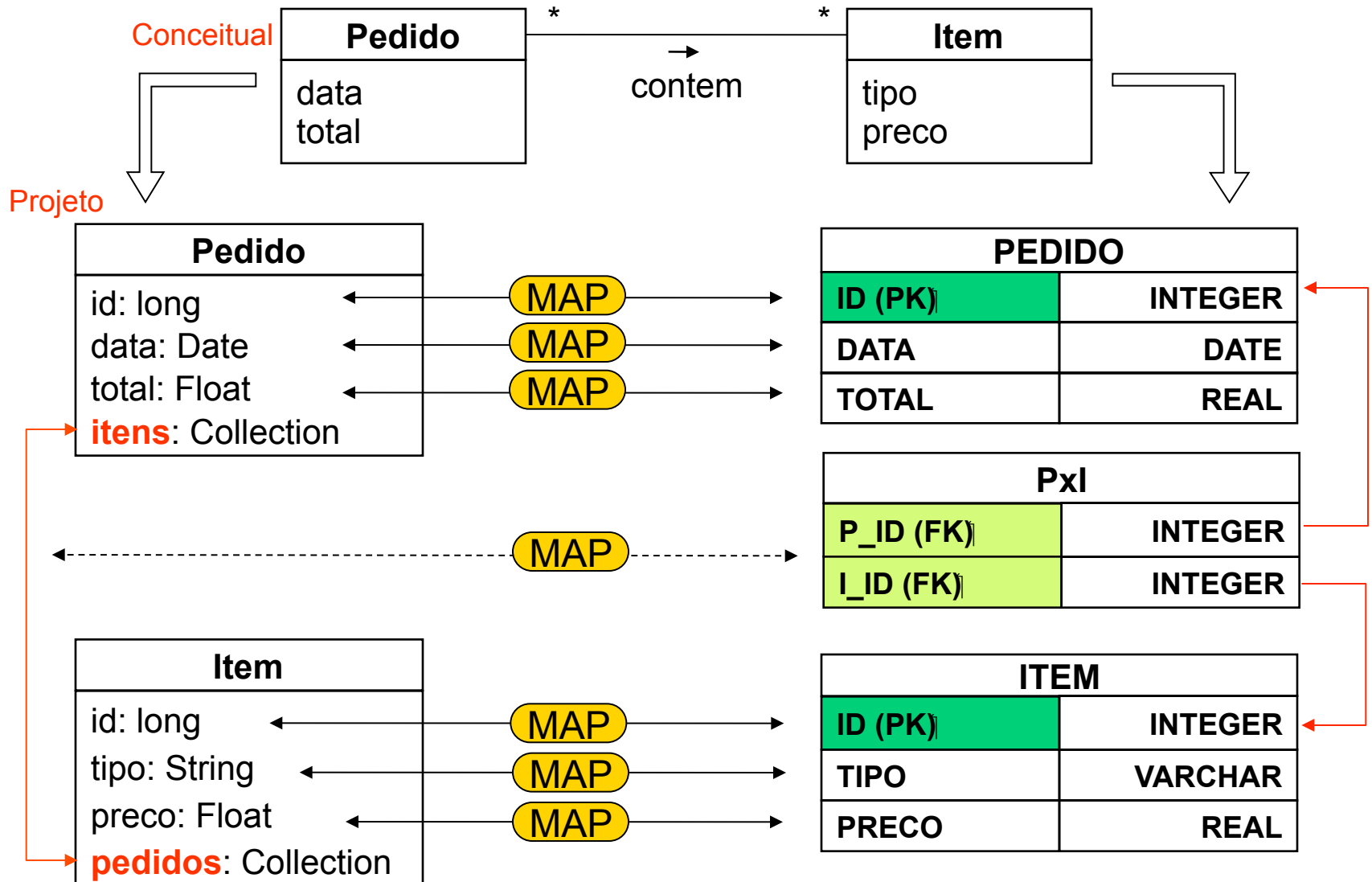
Projeto

Pedido
id: long
data: Date
total: Float
itens : Collection

Item
id: long
tipo: String
preco: Float
pedidos : Collection

Obs: É conveniente que as classes associadas refiram-se mutuamente através de referências inversas.

Associações N:N - Mapeamento Estrutural

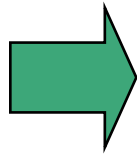


Exemplo

➡ Mapeamento da Associação Avaliador-Topico para a Tabela

Classe de Projeto

```
public class Avaliador extends Pessoa {  
    private String login;  
    private String senha;  
    private String email;  
    private String origem;  
    private List<Topico> topicos;  
    private List<Revisao> revisoes;
```



```
CREATE TABLE TOPICO_AVALIADOR (  
    NOME_AVALIADOR VARCHAR(50),  
    NOME_TOPICO VARCHAR(50),  
    PRIMARY KEY (NOME_AVALIADOR,NOME_TOPICO),  
    FOREIGN KEY (NOME_AVALIADOR) REFERENCES  
        AVALIADOR(NOME),  
    FOREIGN KEY (NOME_TOPICO) REFERENCES  
        TOPICO(NOME) )
```

Associações N:N - Mapeamento Operações

Pedido(N) -> Item(N)

- Recuperação: a partir do **oid** de um pedido

```
ped := new Pedido()
```

```
ped.id, ped.data, ped.total := SELECT ID, DATA, TOTAL FROM PEDIDO WHERE ID = oid
```

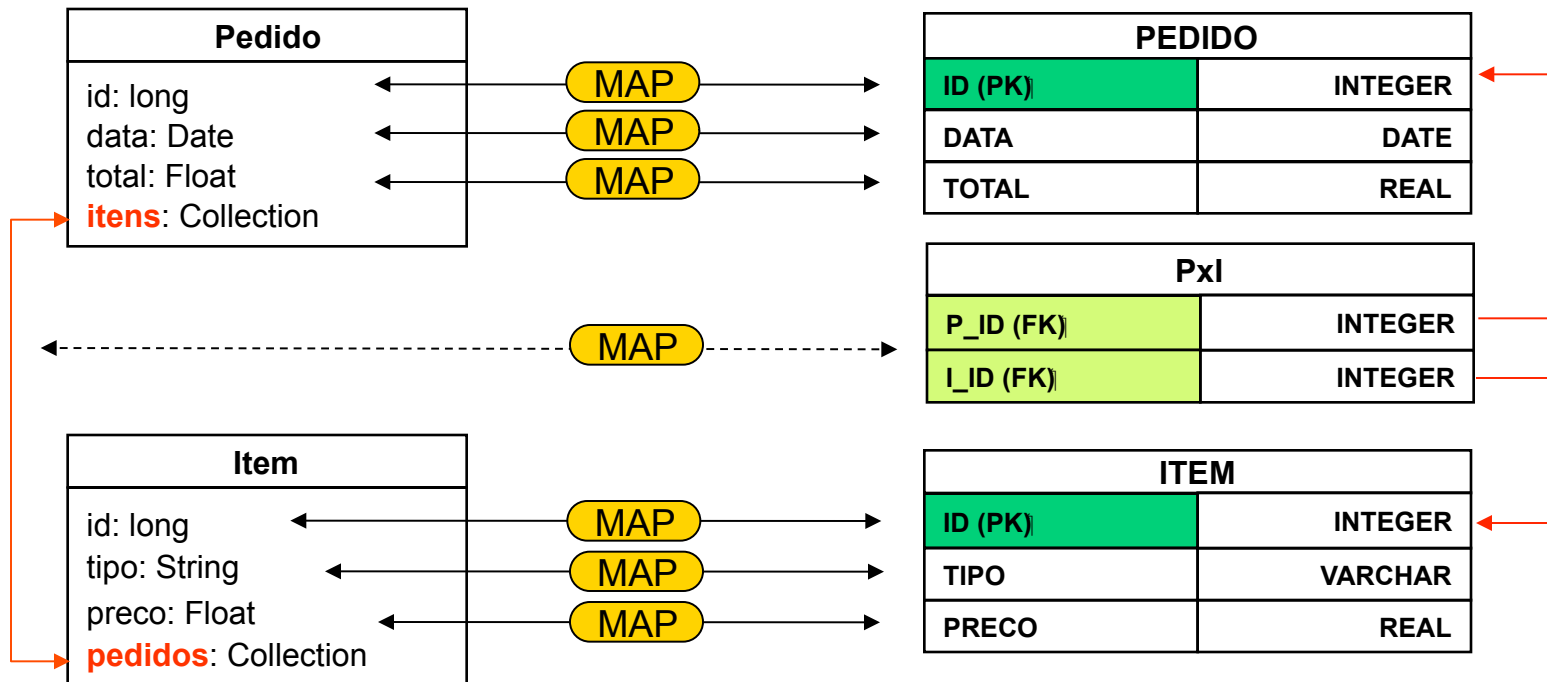
```
itens(id, tipo, preco)* := SELECT ID, TIPO, PRECO FROM ITEM, Pxl
```

```
WHERE Pxl.I_ID = ITEM.ID AND Pxl.P_ID = ped.id (instancia coleção)
```

```
ped.itens := itens
```

p/ cada :item em ped.itens

```
item.pedidos.add(ped)
```



Associações N:N - Mapeamento Operações

Pedido(N) -> Item(N)

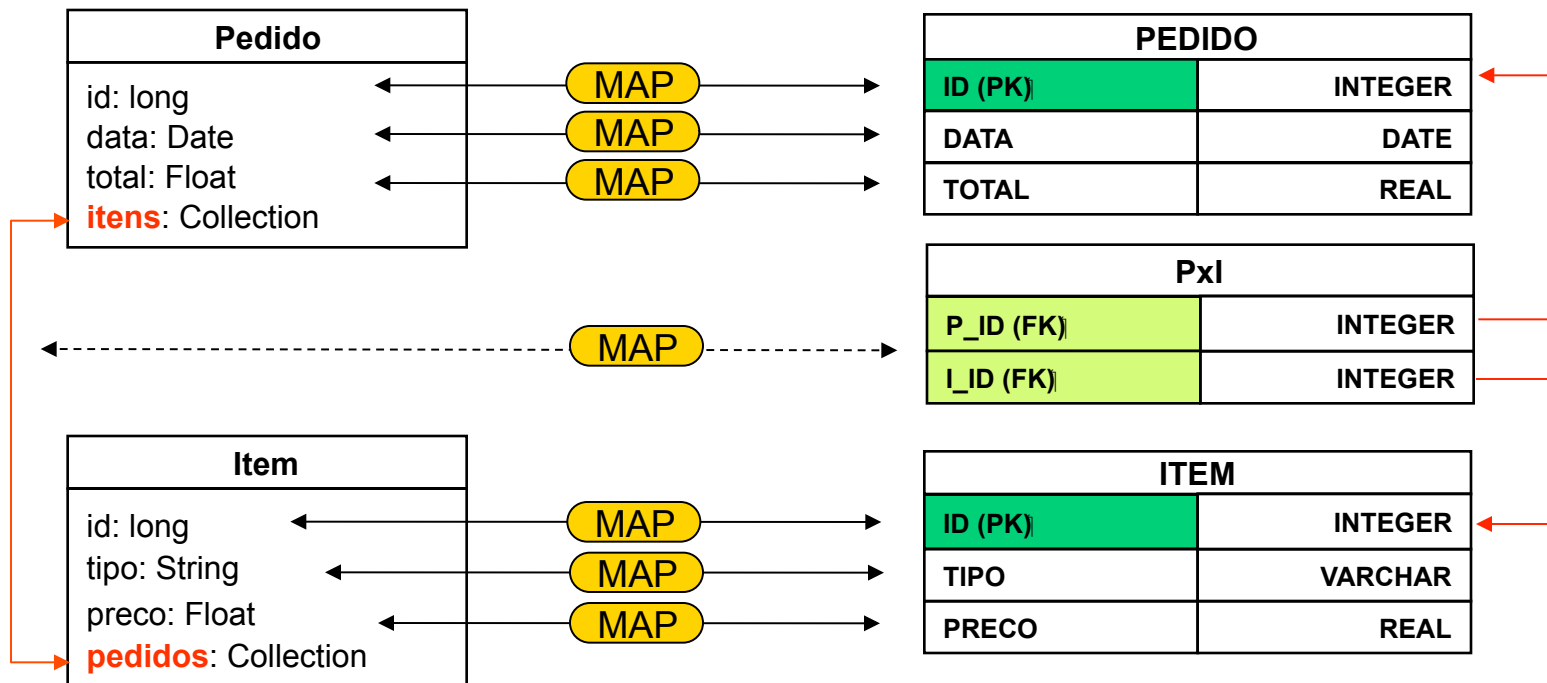
- Inserção: a partir de um pedido

INSERT INTO PEDIDO(ID,DATA,TOTAL) VALUES (ped.id, ped.data, ped.total)

p/ cada novo item :item em ped.itens

INSERT INTO ITEM(ID,TIPO,PRECO) VALUES (item.id, item.tipo, item.preco)

INSERT INTO **PxI**(P_ID,I_ID) VALUES (ped.id, item.id)



Associações N:N - Mapeamento Operações

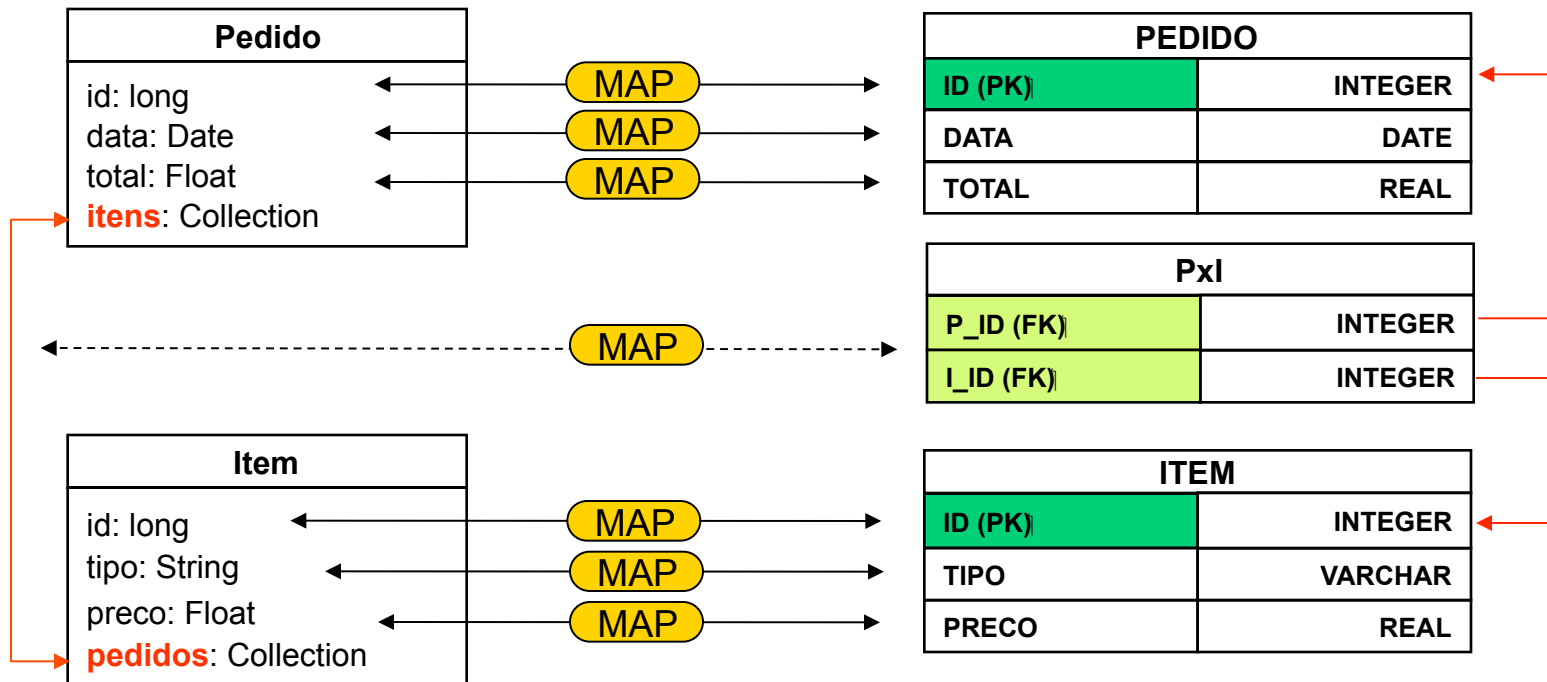
Pedido(N) -> Item(N)

- Atualização: a partir de um pedido

UPDATE PEDIDO SET DATA = ped.data, TOTAL = ped.total WHERE ID = ped.id

p/ cada item modificado :item em ped.itens:

UPDATE ITEM SET TIPO = item.tipo, PRECO = item.preco WHERE ID = item.id



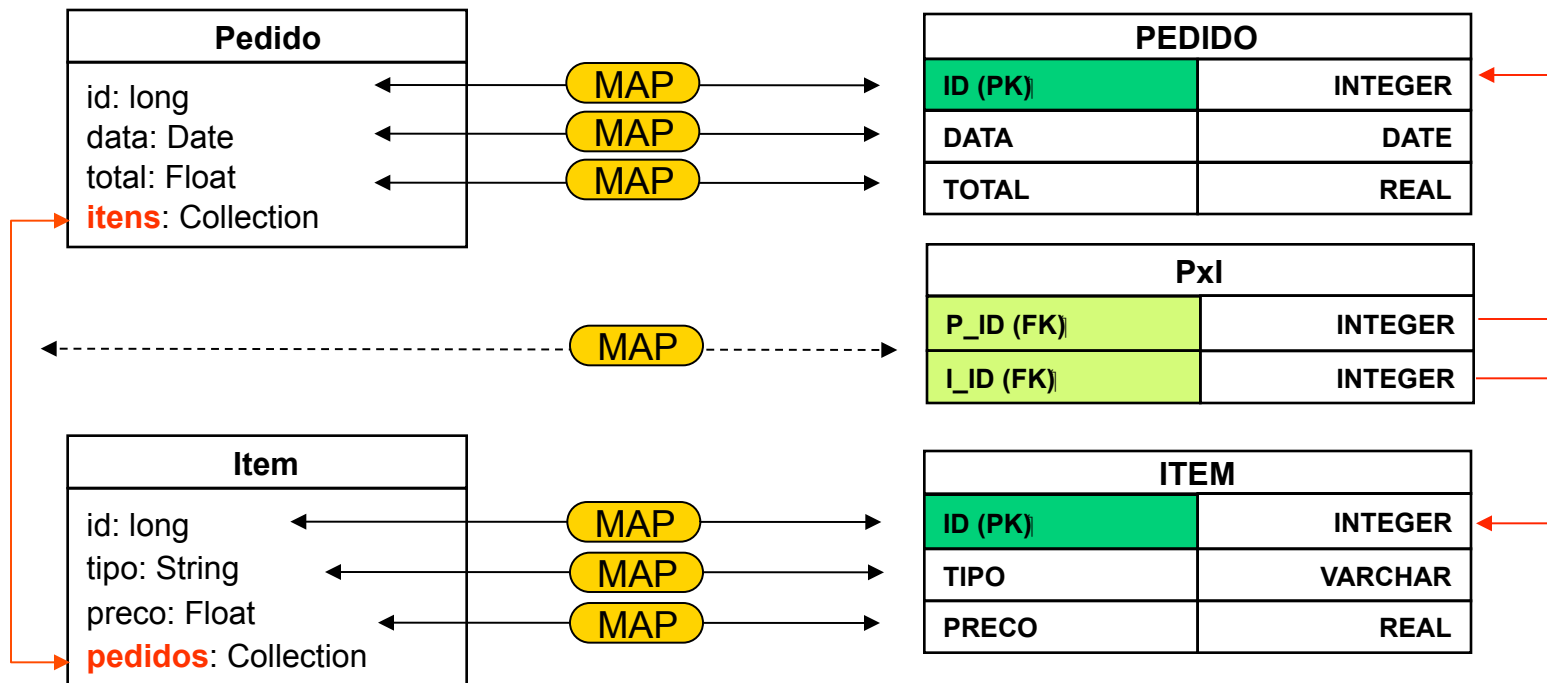
Associações N:N - Mapeamento Operações

Pedido(N) -> Item(N)

- Remoção: a partir de um pedido

DELETE FROM Pxl WHERE P_ID = ped.id

DELETE FROM PEDIDO WHERE ID = ped.id *



* Se o Item tem que ter no mínimo 1 Pedido, a remoção só deve ocorrer se p/ cada :item em ped:itens, item.pedidos.size() > 1 ou remove o Item

Exercício 3

- Complete o método getAutores do mapeador MapeadorArtigo (SELECT).

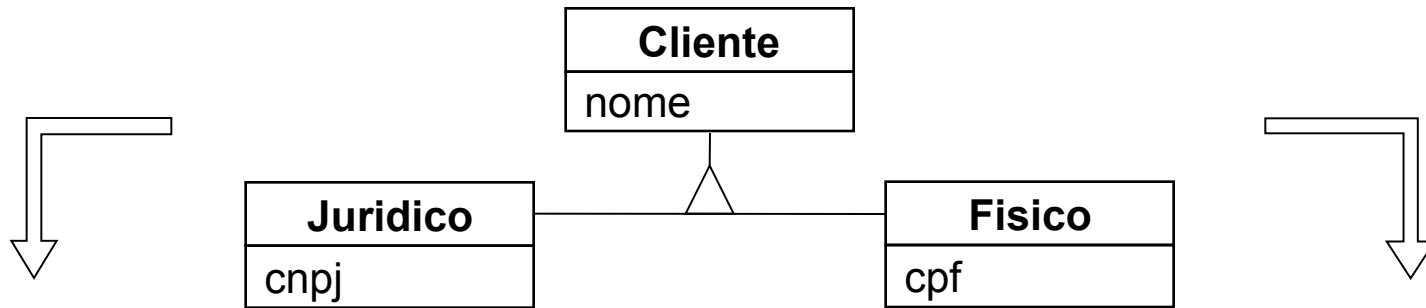
Obs: - Utilize como base o método getAvaliadoresConflitantes do MapeadorArtigo.

Herança - Mapeamento Estrutural

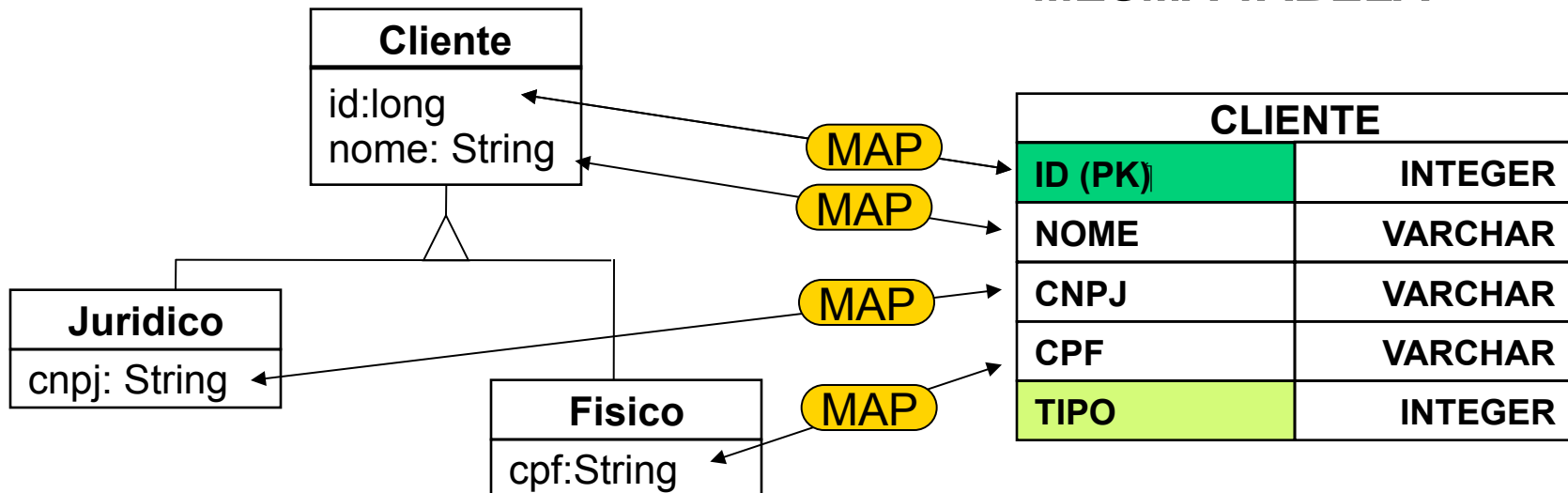
- ➡ As classes da hierarquia podem estar mapeadas para uma mesma tabela, tabelas separadas, ou cada classe em uma tabela diferente*.
- ➡ Mesma tabela: se as classes são mapeadas para uma mesma tabela, esta tabela deve conter colunas para todos os atributos da superclasse e respectivas subclasses, além de um atributo discriminador que indica o tipo do objeto.

* Usado em sistemas legados.

Herança - Mapeamento Estrutural



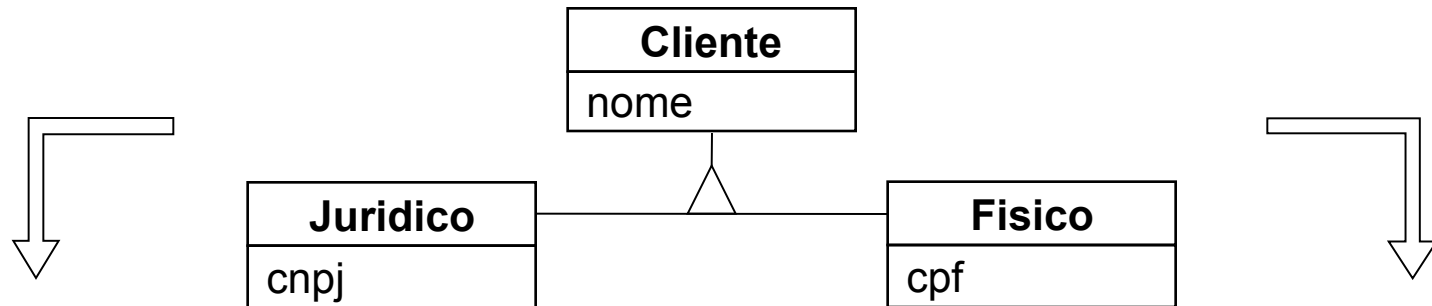
MESMA TABELA



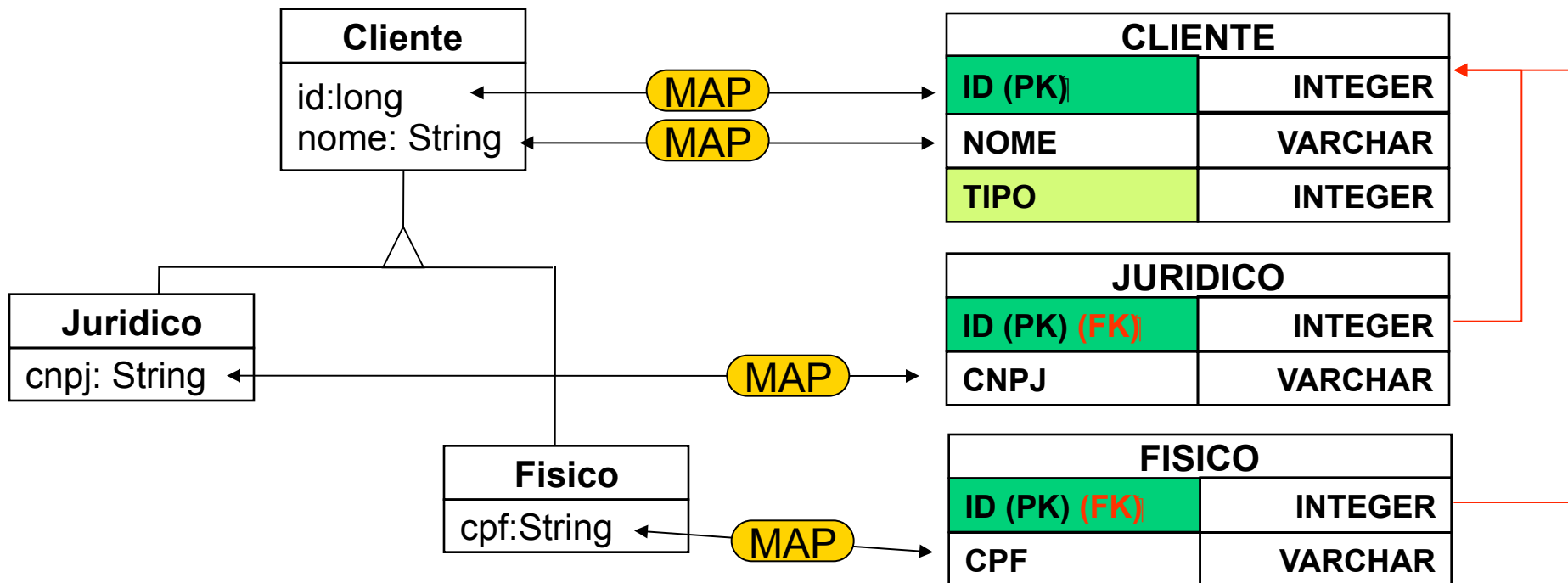
Herança - Mapeamento Estrutural

- ➡ As classes da hierarquia podem estar mapeadas para uma mesma tabela ou tabelas separadas
 - ➡ Tabelas separadas: se as classes são mapeadas para tabelas separadas, cada tabela deve conter apenas colunas para os atributos da classe correspondente. Além disso, as tabelas das subclasses devem se referir à tabela da superclasse através de uma chave estrangeira. Neste caso, o atributo discriminador fica na tabela da superclasse raiz.

Herança - Mapeamento Estrutural



TABELAS SEPARADAS



Herança - Mapeamento Operações

- Recuperação a partir da seleção na superclasse a partir de um **oid** (usando vários select's):

cid, cnome, **tipo** := SELECT ID, NOME, TIPO FROM CLIENTE WHERE ID = **oid**

Se **tipo** = "Fisico"

```
cli := new Fisico()  
cli.id := cid  
cli.nome := cnome  
cli.cpf := SELECT CPF FROM FISICO WHERE ID = cli.id
```

Se **tipo** = "Juridico"

```
cli := new Juridico()  
cli.id := cid  
cli.nome := cnome  
cli.cnpj := SELECT CNPJ FROM JURIDICO WHERE ID = cli.id
```

Se **tipo** = "Cliente" **

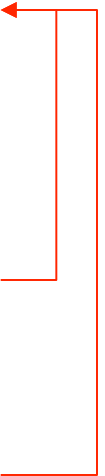
```
cli := new Cliente()  
cli.id := cid  
cli.nome := cnome
```

** a menos que cliente seja uma classe abstrata

CLIENTE	
ID (PK)	INTEGER
NOME	VARCHAR
TIPO	INTEGER

JURIDICO	
ID (PK) (FK)	INTEGER
CNPJ	VARCHAR

FISICO	
ID (PK) (FK)	INTEGER
CPF	VARCHAR



Herança - Mapeamento Operações

- Recuperação a partir da seleção na superclasse (usando outer join):

```
cid, cnome, ccnpj, ccpf, tipo := SELECT CLIENTE.ID, NOME, CNPJ, CPF, TIPO FROM  
CLIENTE LEFT OUTER JOIN JURIDICO ON CLIENTE.ID = JURIDICO.ID  
LEFT OUTER JOIN FISICO ON CLIENTE.ID = FISICO.ID  
WHERE CLIENTE.ID = oid
```

Se tipo = "Físico"

```
cli := new Fisico()  
cli.id := cid  
cli.nome := cnome  
cli.cpf := ccpf
```

Se tipo = "Jurídico"

```
cli := new Juridico()  
cli.id := cid  
cli.nome := cnome  
cli.cnpj := ccnpj
```

Se tipo = "Cliente" **

```
cli := new Cliente()  
cli.id := cid  
cli.nome := cnome
```

** a menos que cliente seja uma classe abstrata

CLIENTE	
ID (PK)	INTEGER
NOME	VARCHAR
TIPO	INTEGER

JURIDICO	
ID (PK) (FK)	INTEGER
CNPJ	VARCHAR

FISICO	
ID (PK) (FK)	INTEGER
CPF	VARCHAR



Herança - Mapeamento Operações

- Recuperação a partir da seleção na subclasse a partir do **oid** de um físico (usando vários select's)

```
fis := new Fisico()
```

```
fis.id, fis.cpf := SELECT ID, CPF FROM FISICO WHERE ID = oid
```

```
fis.nome = SELECT :nome FROM CLIENTE WHERE ID = fis.id
```

- Recuperação a partir da seleção na subclasse a partir do **oid** de um físico (usando junção)

```
fis := new Fisico()
```

```
fis.id, fis.nome, fis.cpf := SELECT CLIENTE.ID, NOME, CPF  
FROM CLIENTE JOIN FISICO  
ON CLIENTE.ID = FISICO.ID  
WHERE CLIENTE.ID = oid
```

CLIENTE	
ID (PK)	INTEGER
NOME	VARCHAR
TIPO	INTEGER

JURIDICO	
ID (PK) (FK)	INTEGER
CNPJ	VARCHAR

FISICO	
ID (PK) (FK)	INTEGER
CPF	VARCHAR



Herança - Mapeamento Operações

- Inserção de uma instância na superclasse: a partir de um cliente

INSERT INTO CLIENTE(ID,NOME, TIPO) VALUES (cli.id,cli.nome,"Cliente")

- Atualização de uma instância na superclasse:
a partir de um cliente

UPDATE CLIENTE SET NOME = cli.nome WHERE ID = cli.id

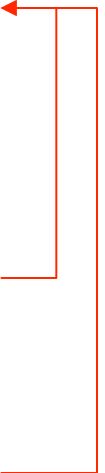
- Remoção de uma instância na superclasse:
a partir de um cliente

DELETE FROM CLIENTE WHERE ID = cli.id

CLIENTE	
ID (PK)	INTEGER
NOME	VARCHAR
TIPO	INTEGER

JURIDICO	
ID (PK) (FK)	INTEGER
CNPJ	VARCHAR

FISICO	
ID (PK) (FK)	INTEGER
CPF	VARCHAR



Herança - Mapeamento Operações

- Inserção na subclasse: a partir de um físico

```
INSERT INTO CLIENTE(ID,NOME, TIPO) VALUES (fis.id, fis.nome, "Físico")
```

```
INSERT INTO FISICO(ID,CPF) VALUES (fis.id, fis.cpf)
```

- Atualização na subclasse: a partir de um físico

```
UPDATE CLIENTE NOME = fis.nome WHERE ID = fis.id
```

```
UPDATE FISICO CPF = fis.cpf WHERE ID = fis.id
```

- Remoção na subclasse: a partir de um físico

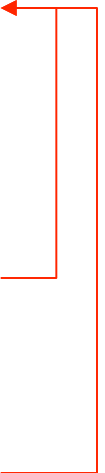
```
DELETE FROM FISICO WHERE ID = fis.id
```


```
DELETE FROM CLIENTE WHERE ID = fis.id
```

CLIENTE	
ID (PK)	INTEGER
NOME	VARCHAR
TIPO	INTEGER

JURIDICO	
ID (PK) (FK)	INTEGER
CNPJ	VARCHAR

FISICO	
ID (PK) (FK)	INTEGER
CPF	VARCHAR





Exercício

- Analise o mapeamento estrutural e comportamental da hierarquia no projeto da conferência.