UNIVERSIDADE FEDERAL DE ITAJUBÁ

Banco de Dados II

COM 231

Bancos de Dados Objeto-Relacionais

Vanessa Cristina Oliveira de Souza

Tipos de Bancos de Dados

Bancos de Dados Relacionais

+

Bancos de Dados Orientados a Objetos

=

Bancos de Dados Objeto-Relacionais

Tipos de Bancos de Dados

BDR x BDOO x BDOR

Critério	BDR	BDOO	BDOR
padrão	SQL-2	ODMG 3.0	SQL-3
suporte a dados complexos	não	sim	sim
performance	alta	baixa	espera-se que seja alta
maturidade	maduro	razoavelmente maduro	razoavelmente novo
uso de SQL	SQL *full*	OQL (em geral, não é *full*)	SQL estendido para objetos
vantagem	eficiência de acesso	modelo de dados rico	modelo rico + eficiência de acesso
uso comercial	larga escala	pequena escala	tendência: alcançar larga escala

Banco de Dados Objeto-Relacional

- Os BDOR modelam **objetos** armazenados em **tabelas**

- Utilizam as tabelas do modelo relacional, mas nelas são armazenados objetos, com seus **atributos e comportamentos**, unindo assim os paradigmas.

- Os dados são armazenados em relações, mas pode-se armazenar **dados complexos** *abstraindo seu comportamento* da mesma forma como é feito na orientação a objetos.

Banco de Dados Objeto-Relacional

- Utiliza os conceitos de supertabelas, supertipos, herança, reutilização de código, encapsulamento, controle de identidade de objetos (OID), referência a objetos, consultas avançadas e alta proteção dos dados.

- A área de atuação dos SGBDs Objeto-Relacionais tenta suprir a dificuldade dos sistemas relacionais convencionais, que é o de representar e manipular **dados complexos**.

Banco de Dados Objeto-Relacional

- Na prática, cria-se um tipo *X* (que pode ser visto como objeto nesse contexto) que tem *n* atributos (ou colunas); e depois ao se criar uma *tabela A*, uma das suas colunas será do tipo X, e nessa coluna, ao invés de se armazenar um tipo comum de dados (int, varchar, decimal, etc.) se armazena o tipo X que por sua vez poderá ter várias colunas de tipos comuns ou mesmo outros tipos.

UNIVERSIDADE FEDERAL DE ITAJUBÁ

SGBDOR POSTGRESQL

SGBDOR PostgreSQL

- No *Postgres*, tabelas, relacionamentos, restrições e *triggers* são considerados objetos.

- Mas não são os “objetos” tais quais os das linguagens de programação.
  - É possível utilizar os conceitos de **herança**, **polimorfismo** e **object ids**.
  - No caso do Postgres, o “objeto” de “objeto-relacional” está ligado a todos os aspectos citados anteriormente, mais a organização dos “recursos”, como na utilização de *schemas*, e na criação de tipos e manipulação de **dados complexos**.

Tipo Composto

- Os dados complexos são implementados no PostgreSQL por meio do chamado tipo composto;
- O *tipo composto* descreve a estrutura de uma linha ou registro;
- Essencialmente, é apenas uma lista de nomes de campos com seus tipos de dado.

```
CREATE TYPE catalogo AS (
    nome            text,
    id_fornecedor   integer,
    preco           numeric
);
```

```
CREATE TABLE estoque (
    item      catalogo,
    contador integer
);
```

Tipo Composto

- Manipulação de tipos compostos

```
INSERT INTO estoque VALUES (ROW('dados de pano', 42, 1.99), 1000);
```

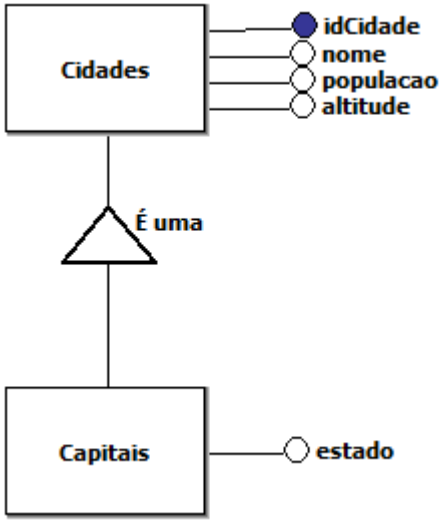
```
SELECT (item).nome FROM estoque WHERE (item).preco > 9.99;
```

```
UPDATE estoque SET item.nome = 'Catálogo 1' WHERE (item).id_fornecedor = 1;
```

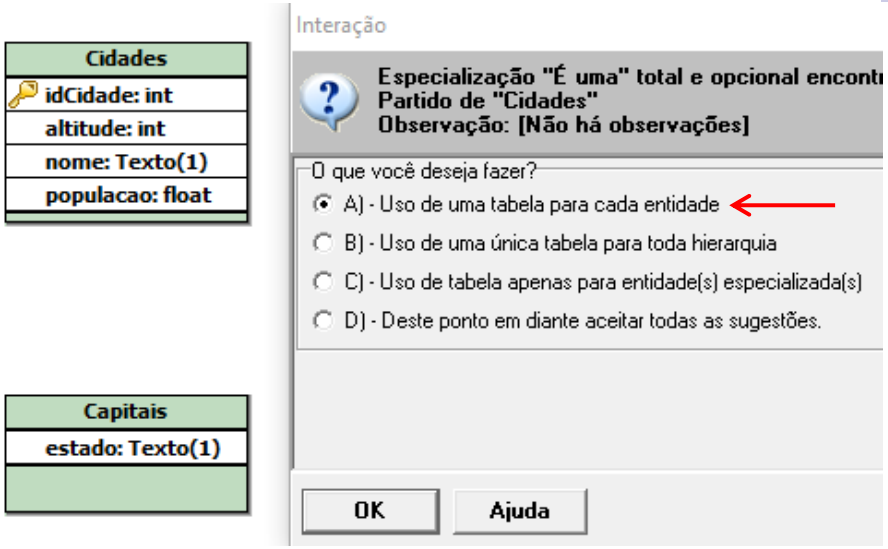
```
CREATE TYPE catalogo AS (
    nome            text,
    id_fornecedor   integer,
    preco           numeric
);
```

```
CREATE TABLE estoque (
    item      catalogo,
    contador integer
);
```

Especialização no MR

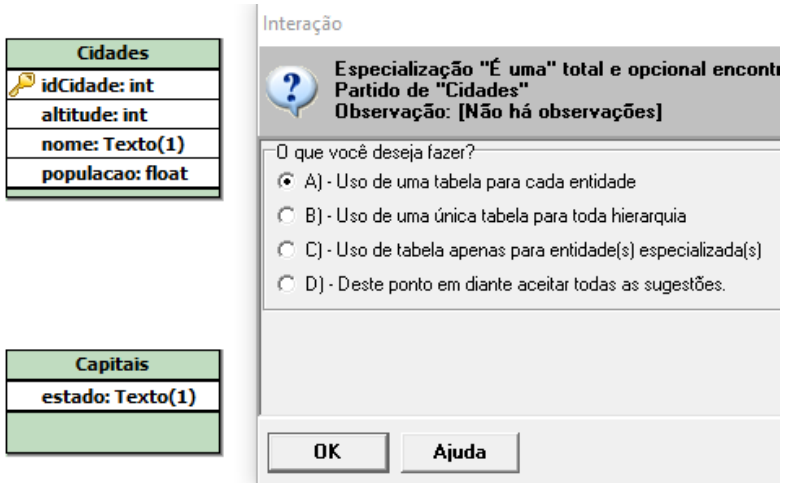


Modelo Conceitual

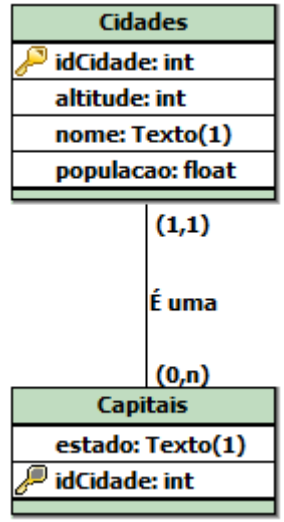


Migração para o modelo lógico

Especialização no MR

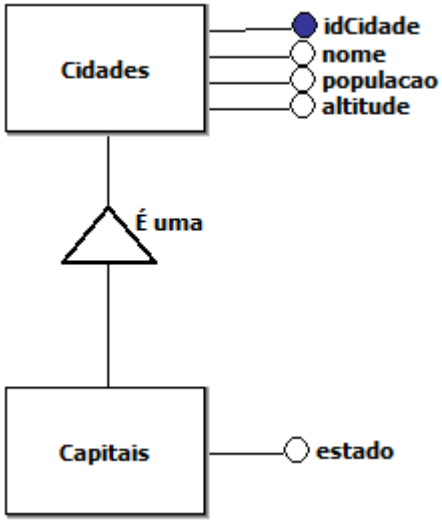


Migração para o modelo lógico

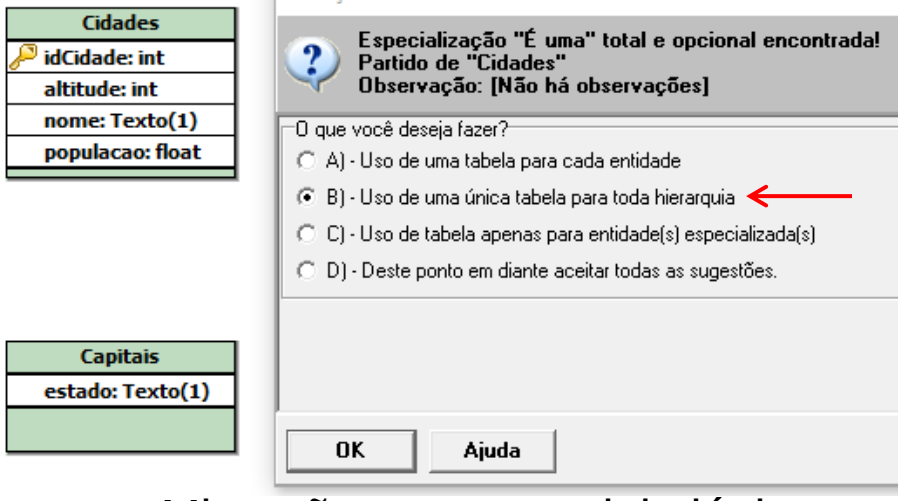


Modelo lógico

Especialização no MR

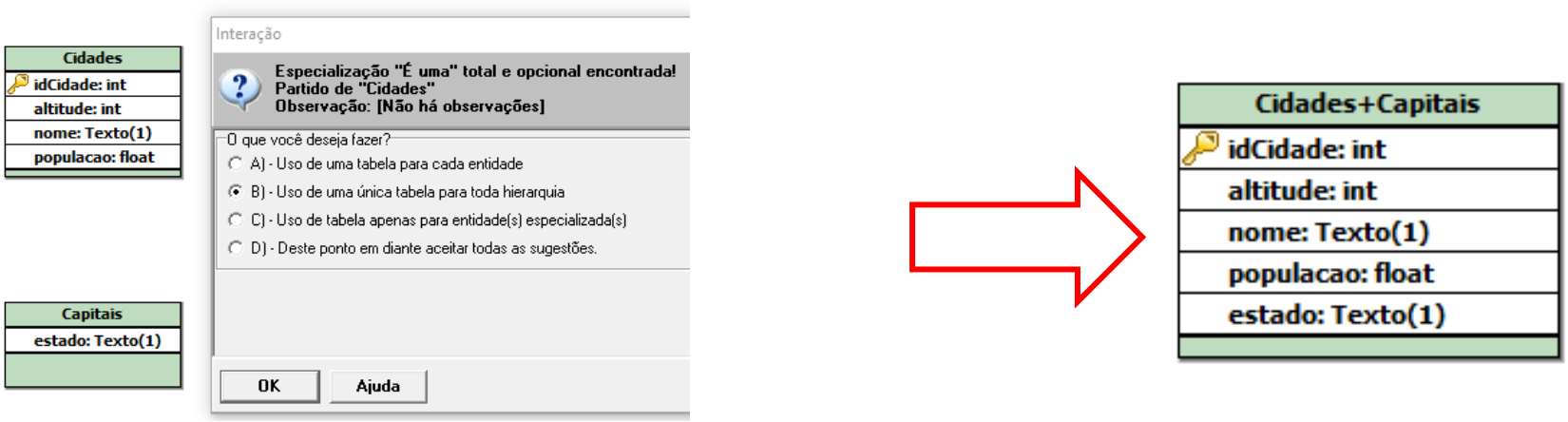


Modelo Conceitual



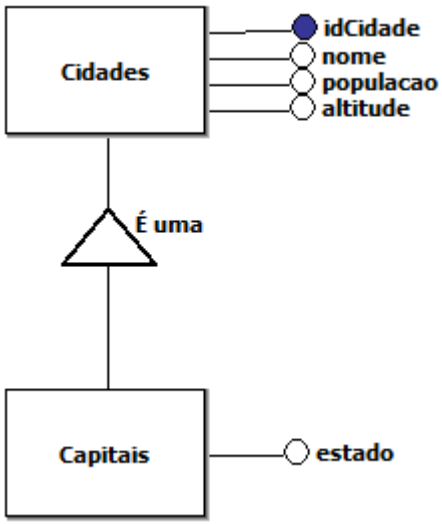
Migração para o modelo lógico

Especialização no MR

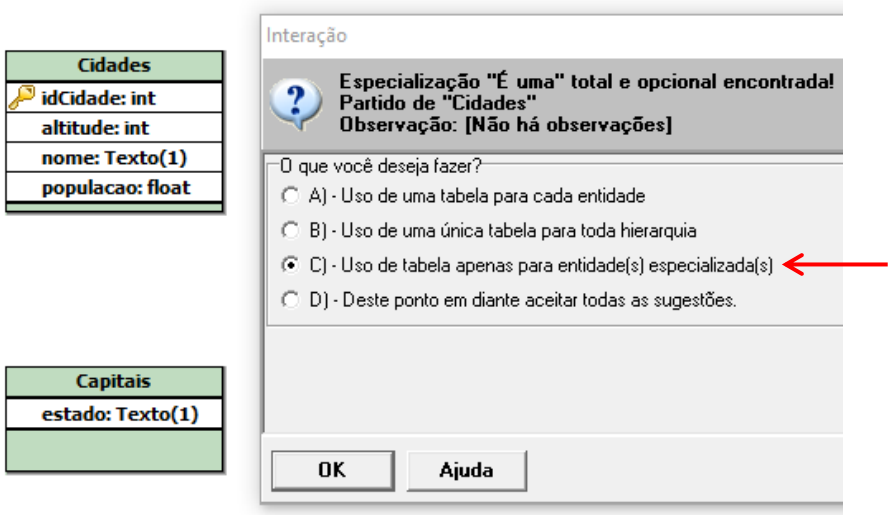


Migração para o modelo lógico

Especialização no MR

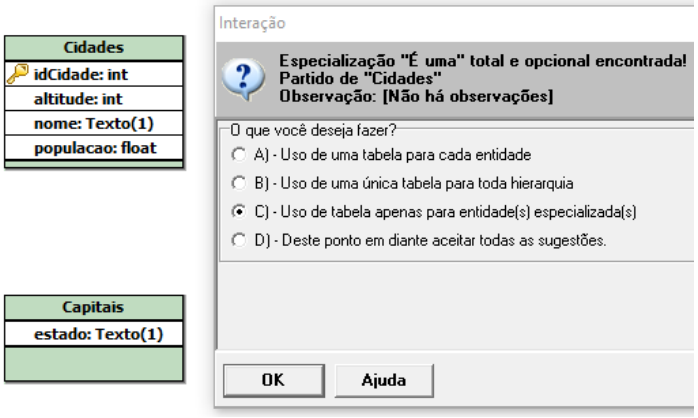


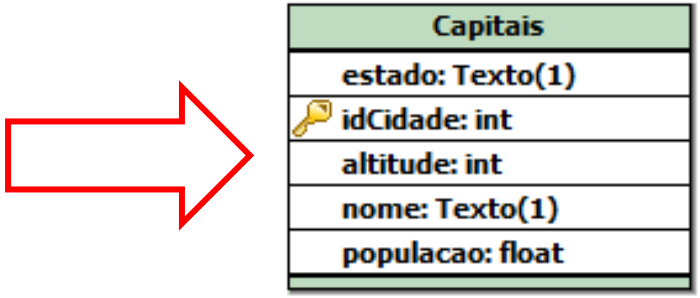
Modelo Conceitual



Migração para o modelo lógico

Especialização no MR





Migração para o modelo lógico

Herança

- Herança é um conceito da orientação à objetos onde, uma determinada classe herda características (no caso das classes - propriedades ou atributos e métodos ou funções) de uma outra "super" classe.

- A relação de herança pode ser lida da seguinte forma: "a classe que herda É UM TIPO DA super classe (classe herdada)".

Herança

```
CREATE TABLE cidades (
    idCidade    int,
    nome        text,
    populacao   float,
    altitude    int,
    primary key (idCidade)
);

CREATE TABLE capitais (
    estado      char(2)
) INHERITS (cidades);
```

```
CREATE TABLE public.capitais
(
    idcidade ,
    nome ,
    populacao ,
    altitude ,
    estado character(2) COLLATE pg_catalog."default"
)
    INHERITS (public.cidades)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;
```

- As linhas da tabela capitais *herdam* todos os atributos (nome, população e altitude) de sua tabela ancestral, cidades.

- No PostgreSQL uma tabela pode herdar de zero ou mais tabelas.

Herança

```
CREATE TABLE cidades (
    idCidade    int,
    nome        text,
    populacao   float,
    altitude    int,
    primary key (idCidade)
);

CREATE TABLE capitais (
    estado      char(2)
) INHERITS (cidades);
```

```
CREATE TABLE public.capitais
(
    idcidade ,
    nome ,
    populacao ,
    altitude ,
    estado character(2) COLLATE pg_catalog."default"
)
    INHERITS (public.cidades)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;
```

- Sempre que houver uma atualização na tabela **capitais** (insert, update e delete), o mesmo registro é atualizado também na tabela **cidades**.
  - O contrário não é verdadeiro

Herança

```
16  INSERT INTO cidades VALUES (50, 'Itajuba', 97000, 856);
17  SELECT * FROM cidades;
18
```

Data Output | Explain | Messages | Query History

	idcidade integer	nome text	populacao double precision	altitude integer
1	50	Itajuba	97000	856

- Sempre que houver uma atualização na tabela capitais (insert, update e delete), o mesmo registro é atualizado também na tabela cidades.
  - O contrário não é verdadeiro

Herança

```
19  INSERT INTO capitais VALUES (60, 'Belo Horizonte', 2501576, 852, 'MG');
20  SELECT * FROM cidades;
21
```

Data Output | Explain | Messages | Query History

	idcidade integer	nome text	populacao double precision	altitude integer
1	50	Itajuba	97000	856
2	60	Belo Horizonte	2501576	852

- Sempre que houver uma atualização na tabela capitais (insert, update e delete), o mesmo registro é atualizado também na tabela cidades.
  - O contrário não é verdadeiro

Herança

```
26  SELECT * FROM ONLY cidades;
```

Data Output | Explain | Messages | Query History

	idcidade integer	nome text	populacao double precision	altitude integer
1	50	Itajuba	97000	856

- O termo "ONLY" antes de cidades indica que a consulta deve ser executada apenas na tabela cidades, sem incluir as tabelas descendentes de cidades na hierarquia de herança
  - SELECT, UPDATE, DELETE

Herança

- Limitações
  - A PRIMARY KEY não impede que a tabela **capitais** tenha linhas com identificadores idênticos aos da tabela **cidades** e, por padrão, estas linhas duplicadas aparecem nas consultas à tabela cidades.

```
28  INSERT INTO capitais VALUES (50, 'São Paulo', 12176866, 760, 'SP');
```

Data Output | Explain | Messages | Query History

	idcidade integer	nome text	populacao double precision	altitude integer
1	50	Itajuba	97000	856
2	60	Belo Horizonte	2501576	852
3	50	São Paulo	12176866	760

Herança

- Limitações
  - Uma FOREIGN KEY definida na tabela cidades não se propagará automaticamente para a tabela capitais.
    - Nem vice-versa
  - Especificar para uma coluna de outra tabela REFERENCES cidades(idCidade) permite à outra tabela conter os nomes das cidades, mas não os nomes das capitais. Não existe um maneira boa para contornar este problema.

Herança

```
16  CREATE TABLE aluno (
17      idAluno     int,
18      nomeAluno   text,
19      cidade     int,
20      primary key (idAluno),
21      foreign key (cidade) REFERENCES cidades(idCidade)
22          on update cascade on delete restrict
23  );
24
25  INSERT INTO aluno VALUES (10, 'João da Silva', 60);
```

Data Output | Explain | Messages | Query History

```
ERROR:  insert or update on table "aluno" violates foreign key constraint "aluno_cidade_fkey"
DETAIL:  Key (cidade)=(60) is not present in table "cidades".
SQL state: 23503
```

Diferentes Modelagens

- Modelo Relacional *versus* Modelo Objeto/Relacional

Análise de Performance

- Modelo Relacional *versus* Modelo Objeto/Relacional