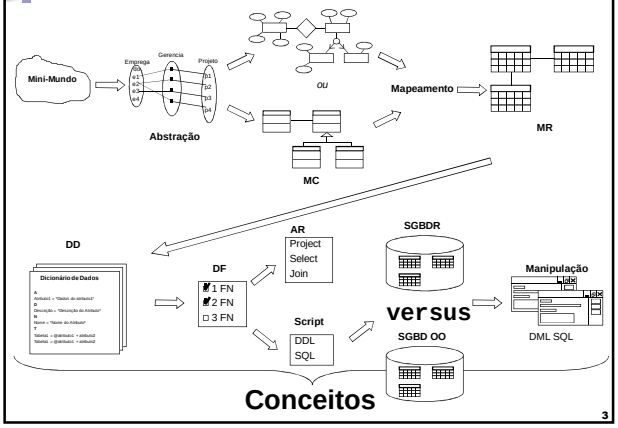
Banco de Dados II
SGBD Orientado a Objetos

Osmar de Oliveira Braz Junior

Objetivos

- Conhecer os principais conceitos de SGBDOO.
- Analisar as vantagens e desvantagens de se utilizar um SGBDOO.
- Especificar DDL para SGBDOO.
- Realizar consultas em um SGBDOO.



BD com tecnologia OO

- São divididos em dois grupos os Puramente Orientado a Objeto (**Pure Object Oriented DBMS -ODBMS**), que baseia-se somente no modelo de dados Orientado a Objetos.
- Usam declarações de classes muito semelhantes das linguagens orientadas a objetos. Ex. BD Jasmini .

BD com tecnologia OO

- E os Objeto Relacional (**Object Relacional DBMS -ORDBMS**) , são bancos relacionais que possibilitam o armazenamento de objetos, permitem a relação de objetos, herança , identificadores de objeto.
- Identificadores de Objeto é um identificador interno do banco para cada objeto, são atribuídos somente pelo DBMS e não pelos usuários. Não tem padrão único de implementação como os BD relacionais.
- Ex.: Oracle, Postgres, Informix, BD2, Titanium.

Banco de Dados Objeto-Relacional

- **SGBDs Objeto-Relacional** combinam os benefícios do modelo Relacional com a capacidade de modelagem do modelo OO.
- Fornecem suporte para consultas complexas sobre dados complexos.
- Atendem aos requisitos das novas aplicações e da nova geração de aplicações de negócios.

Vantagens do OO

- Em relação a bancos de dados relacionais temos como vantagens **qualidade do software, reutilização, portabilidade, facilidade de manutenção e escalabilidade**.
- No relacional diversas vezes a linguagem de programação é completamente diferente a utilizada na **RDBMS**.

7

Vantagens do OO

- Os dados dos bancos relacionais utilizam a **linguagem SQL** no qual permite que os sistemas relacionais desenvolvidos por muitos fornecedores possam se comunicar entre si e acessar banco de dados comuns.
- Em contra partida, os bancos de dados orientados a objetos **não possuem uma linguagem padrão** dificultando a interoperacionalidade entre os bancos de dados de diferentes fornecedores.

8

Características de SGBDOO

- Em um **SGBDOO** os objetos da base de dados são tratados como objetos da linguagem de programação, possuem características de e princípios do paradigma de orientação a objetos.

9

Características de SGBDOO

- **Persistência**, os dados de um processo ou transação persistem após o término da execução do mesmo. A **persistência** é requisito evidente para bases de dados, a persistência deve permitir que qualquer objeto, independente de seu tipo, torne-se persistente.

10

Características de SGBDOO

- **Objeto complexos**, suporte a objetos grande em tamanhos e a objetos estruturados, como tuplas, vetores e listas. Tendo a necessidade de suporte as operações que manipulam estes objetos.
- **Identidade de objeto**, cada objeto da base possui um identificador único e imutável, gerado automaticamente pelo sistema.

11

Características de SGBDOO

- **Encapsulamento**, o objeto da base de dados encapsula dados que definem sua estrutura interna e operações que definem sue comportamento, a estrutura interna e a definição do comportamento de um objeto ficam escondidas, e o objeto é acessado através das operações pré definidas para seu tipo.
- **Tipos, Classes e Herança**, suporte a hierarquias de tipos ou hierarquias de classes através do conceito de herança , o que permitem que novos tipos sejam definidos a partir de tipos de classes pré definidos. Os subtipos de subclasses herdam os atributos e as rotinas das superclasses, podendo no entanto possuir atributos e rotinas próprias.

12

Características de SGBDOO

- **Polimorfismo** também chamado de sobrecarga, permite que um mesmo nome de operação seja utilizado para implementações diferentes, dependendo do tipo de objeto ao qual a operação é aplicada.
- **Ligação Tardia (Late Binding)** realiza a tradução de nomes das operações em endereços de programas em tempo de execução. O binding realizado em tempo de compilação, ao contrário , é chamado de binding estático, o binding atrasado, embora seja lento e dificulte a checagem de tipos é necessário para viabilizar a utilização de sobrecarga de operações.

13

Características de SGBDOO

- **Extensibilidade** é o conjunto de tipos pré definidos do sistema que deve ser extensível, permitindo que o programadores definam novos tipos.
- No entanto o tratamento de tipos definidos pelo usuário é diferente dos que são definidos pelo sistema.
- Estas diferenças devem ser imperceptíveis para o programadores e para a aplicação.

14

Banco de Dados Oracle

- O banco de dados Oracle com sua opção de objetos, permite a manipulação e criação de objetos assim podemos classificá-lo como um banco **Objeto Relacional**.
- As entidades do mundo real são definidas pelos usuários, podendo possuir operações implementadas no servidor, todos os objetos criados terão um identificados único que nunca terá o valor alterado durante todo o ciclo de vida do objeto.
- As referências são os ponteiros no banco de dado e definem relacionamentos entre objetos que torna o acesso navegacional mais rápido do que o acesso padrão.

15

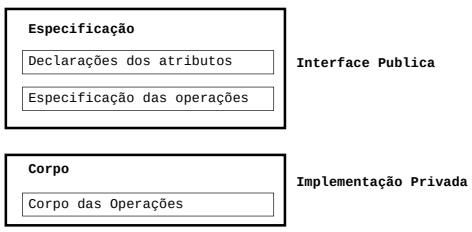
Banco de Dados Oracle

- As visões de objetos, permitem criar abstrações de objetos sobre os banco de dados relacionais. Em adição ao mecanismo tradicional de visões.

16

Tipos de Objetos

- Um tipo de objeto em Oracle possui a seguinte estrutura:



17

Tipos de Objetos

- Exemplo de especificação da interface pública de um objeto
- **Sintaxe resumida:**

```
CREATE [OR REPLACE] TYPE nomeTipo AS OBJECT (
  [lista de atributos]
  [lista de operações]
);
```

18

Exemplo Tipos

```
create or replace type TTelefone as object(
  telefoneId number,
  numero varchar(10),
  tipo varchar(30),
  ddd varchar2(3),
  Member function getFone return varchar
);

create or replace type body TTelefone as
  Member function getFone return varchar Is
  Begin
    return '(' || ddd || ')' || '-' || numero;
  end;
end;
```

19

Operações

- São **funções** ou **procedimentos** que são declarados na definição de um tipo de objeto.
- Exigem o uso de parênteses (mesmo sem parâmetros). O uso de **( )** é para diferenciar o operação de um procedimento ou função comum.
- Podem ser
  - MEMBER ou STATIC
  - MAP ou ORDER (para ordenação)
  - Construtor

20

Operações

- Member
  - São as operações mais comuns.
  - Implementam as operações das instâncias do tipo.
  - São invocados através da qualificação de objeto **`objeto.operacao().`**

21

Exemplo Herança de Tipos

```
create or replace type TPessoa as object(
  pessoaId number,
  nome varchar(50),
  email varchar(100),
  telefone TTelefone
) not final;

create or replace type TPessoaJuridica under TPessoa(
   cnpj varchar(16)
);

create or replace type TPessoaFisica under TPessoa(
  cpf varchar(11),
  conjugue REF TPessoaFisica
);
```

22

Herança de Tipos

- Suporta **herança simples**
- Permite criar uma hierarquia de subtipos especializados
- Os tipos derivados (subtipos) herdam os atributos e operações dos tipos ancestrais (supertipos)
- Os sub-tipos podem acrescentar novos atributos ou operações e/ou redefinir as operações dos supertipos

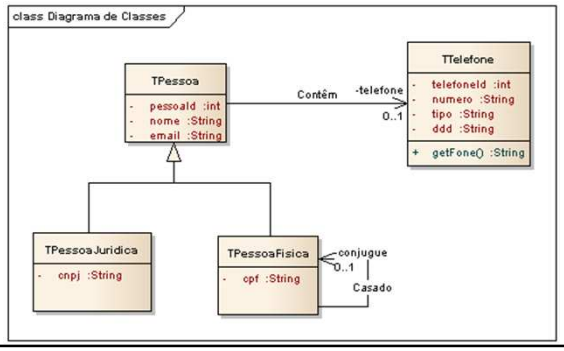
23

FINAL e NOT FINAL

- Tipos e operações podem ser **FINAL** e **NOT FINAL**
- A definição do tipo do objeto determina se um subtipo pode ser derivado.
- Por **padrão** os tipos de objeto são do tipo **FINAL**. Por isso, para permitir subtipos, deve ser obrigatoriamente adicionada à expressão NOT FINAL na declaração do tipo do objeto.
- Por exemplo, o slide anterior apresenta uma expressão onde é declarado que o tipo **TipoPessoa** pode ser derivado, permitindo que sejam criados subtipos a partir dele.

24

Modelo



25

Alterando Tipos

- As alterações de tipos é muito semelhante a alterações de atributos para as tabelas.
- Uso do comando **ALTER TYPE**, permite:
  - modificar ou evoluir um tipo objeto,
  - adicionar e remover atributos,
  - adicionar e remover operações,
  - modificar um atributo numérico para aumentar o length, precision, ou scale,
  - modicar um atributo string para aumentar seu length.

26

Alterando Tipos

- Para adicionar o **atributo celular** do tipo telefone ao tipo pessoa podemos utilizar o comando:

```
alter type Tpessoa add attribute celular TTelefone
cascade;
```

- Para remover o atributo celular do tipo pessoa utilize o comando:

```
alter type Tpessoa drop attribute celular cascade;
```

- A opção **CASCADE** propaga a mudança para todos os tipos dependentes, com isto as alterações no tipo refletem em todas as estruturas que a utilizam.

27

Alterando Tipos

- Não se pode remover um subtipo antes de remover suas respectivas instâncias na tabela que armazena as tuplas daquele subtipo (substitutability).

```
DROP TYPE TPessoaJuridica VALIDATE;
```

- Primeiro deve ser apagado as instancias em tabelas do seu tipo:

```
DELETE FROM PessoaJuridica WHERE p IS OF
(TPessoaJuridica);

DROP TYPE TPessoaJuridica VALIDATE;
```

28

Tabelas

- Uma tabela de objetos difere de uma tabela relacional em vários **aspectos**.
- Cada linha de uma tabela de objetos possui um identificador de objeto o **OID** definido pelo oracle quando a linha é inserida na tabela.
- Um **OID** é um ponteiro para um objeto "**linha**" um "**Row Object**, ele permite que as linhas de "**Row Objects**" de uma tabela de objetos possam ser referenciadas em atributos de outros objetos ou em colunas de tabelas relacionais.

29

Exemplo Tabelas

```
create Table PessoaJuridica of TPessoaJuridica;

create Table PessoaFisica of TPessoaFisica;

create Table Cliente (
  pessoa TPessoaFisica,
  detalhes varchar(4000)
);
```

30

Tipo REF

- É um ponteiro lógico para um "**Row Object**".
- É definido a partir do **OID** do objeto. Oferece acesso rápido/direto a objetos relacionados. Não garante integridade referencial, tem que usar **referential constraint**, neste caso **REF** pode apontar para qualquer objeto do tipo apontado.
- Os **REFs** e coleções de **REFs** são utilizados na modelagem de associações entre objetos. Por ex. o relacionamento entre um Pedido e um cliente.

31

Tipo REF

- Constituem um mecanismo simples para navegar entre objetos. Pode-se utilizar a notação estendida de "**pontos**" para seguir os ponteiros sem a necessidade de junções explícitas.
- Na especificação do tipo **TPessoaFisica** o atributo conjugue é uma referência ao próprio tipo.

```
create or replace type TPessoaFisica under TPessoa(
  cpf varchar(11),
  conjugue REF TPessoaFisica
);
```

32

Construtor

- Criado implicitamente ao criar um tipo de objeto
- Deve ser exatamente igual ao nome do tipo
- Pode haver mais de um construtor

33

Exemplo Inserção

- **Inserindo dados na tabela criada a partir do tipo TPessoaJuridica:**

```
insert into PessoaJuridica
  values (1,'ACME SA','acme@gmail.com',
  TTelefone(1,'111','22','333'),'123456789000012');
```

```
create Table PessoaJuridica of TPessoaJuridica;
```

```
create or replace type TPessoaJuridica
        under TPessoa(
  cnpj varchar(16)
);
```

```
create or replace type TPessoa
                as object(
  pessoaId number,
  nome varchar(50),
  email varchar(100),
  telefone TTelefone
) not final;
```

- Como a tabela for criada inteiramente a partir do tipo, não tem necessidade do construtor para TPessoaJuridica **somente** para **TTelefone**.

34

Exemplo Inserção

```
create or replace type TPessoa as object(
  pessoaId number,
  nome varchar(50),
  email varchar(100),
  telefone TTelefone
) not final;
```

- **Tabela criada com um atributo de tipo:**

```
insert into cliente(pessoa, detalhes)
  values (
  TPessoaFisica(1,'Joao da Silva','joao@gmail.com',
  TTelefone(1,'123','12','123'),'12345678912',NULL),
  'cliente desde 1999');
```

```
create Table cliente (
  pessoa TPessoaFisica,
  detalhes varchar(4000)
);
```

```
create or replace type TPessoaFisica
        under TipoPessoa(
  cpf varchar(14),
  conjunte ref TPessoaFisica
);
```

```
create or replace type TTelefone as object(
  telefoneId number,
  numero varchar(10),
  tipo varchar(30),
  ddd varchar2(3),
  Member function getFone return varchar
);
```

35

Exemplo Inserção com NULL

- **Se algum campo não for preenchido utilize `NULL`:**
  - Não foi inserido telefone e conjugue para o cliente.

```
insert into cliente(pessoa, detalhes)
  values (
  TPessoaFisica(2,'Pedro da Silva','pedro@gmail.com',
  NULL,'12345678912',NULL),
  'cliente desde 1999');
```

36

Exemplo Inserção com REF

- Para criar uma **referência** vamos primeiro inserir uma pessoa física:

```
insert into PessoaFisica
values (1,'Joao da Silva','joao@gmail.com',
      TTelefone(2,'134','12','123'),'123456789', NULL);
```

- Usamos REF para recuperar a referência(conjugue):

```
insert into PessoaFisica
      select TPessoaFisica (2,'Maria da Silva',
            'maria@gmail.com',
            NULL,'123456789',REF(pf))
      from PessoaFisica pf
      where pf.pessoaId = 1;
```

37

Atualização

- A atualização ocorre de forma muito semelhante ao modelo relacional.

```
update PessoaJuridica pj
set pj.telefone = TTelefone(3,'123','123','321')
where pj.pessoaId = 1;
```

38

Exclusão

- Para excluir um registro utilize o comando:

```
delete from PessoaJuridica pj
where pj.pessoaId = 1;
```

39

Consulta

- Existem diferenças significativas no modo de utilização de uma tabela de objetos.
- Cada linha dentro de uma tabela de objetos possuirá um **OID**, e essas linhas poderão ser referenciadas como objetos.

40

Exemplo Consulta

- **Consulta**

```
select * from cliente;
```

- **Resultado**

	PESSOA	DETALHES
1	[PROFBD2.TPESSOAFISICA]	cliente desde 1999
2	[PROFBD2.TPESSOAFISICA]	cliente desde 1999

ou

PESSOA	DETALHES
PROFBD2.TPESSOAFISICA(1,'Joao da Silva', 'joao@gmail.com',PROFBD2.TTELEFONE(1,'123','12' ,'123'),'12345678912',NULL)	cliente desde 1999
PROFBD2.TPESSOAFISICA(2,'Pedro da Silva', 'pedro@gmail.com',NULL,'12345678912',NULL)	cliente desde 1999

41

Exemplo Consulta

```
select c.pessoa.nome
from cliente c;

select c.pessoa.nome, c.pessoa.telefone.numero
from cliente c;

select c.pessoa.nome
from cliente c
where c.pessoa.pessoaId = 1;
```

42

Exemplo Consulta com Operação

```
select c.pessoa.nome,
  c.pessoa.telefone.numero,
  c.pessoa.telefone.getFone()
from cliente c;
```

43

Exemplo Consulta com REF

- Com os dados relacionados as consultas podem ser realizadas mais facilmente:

```
select ref(pf)
from PessoaFisica pf;
```

- O resultado da consulta e apresentado abaixo perceba que não é necessário usar junção para trazer os dados relacionados.

```
PROFBD2.TPESSOAFISICA(1,'Joao da Silva','joao@gmail.com',
      PROFBD2.TTELEFONE(2,'134','12','123'),'123456789',NULL)
PROFBD2.TPESSOAFISICA(2,'Maria da Silva','maria@gmail.com',
      NULL,'123456789','oracle.sql.REF@3df04063')
```

44

Exemplo Consulta com REF

- Para consultar o nome da pessoa física e o nome do seu conjugue escrevemos:

```
select pf.nome, pf.conjugue.nome
from PessoaFisica pf;
```

- No resultado da consulta perceba que não é necessário usar junção para trazer os dados relacionados.

nome	conjugue.nome
Joao da Silva	NULL
Maria da Silva	Joao da Silva

45

Constraints

- As **constraints** podem ser especificadas normalmente como no modelo relacional.
- Para atributos tipo **REF** a integridade referencial é semelhante ao **FOREIGN KEY**.
  - A constraint Primary Key não pode ser especificada para uma coluna do tipo REF.

46

Exemplo Tipos com Constraints

Apague as tabelas anteriores!

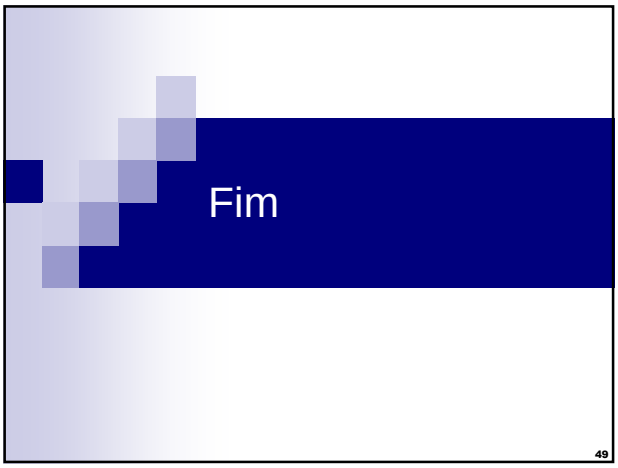
```
create Table PessoaJuridica of TPessoaJuridica(
   constraint PK_PessoaJuridica primary key(pessoaId),
   constraint NN_PessaoJuridica_nome nome not null
);

create Table PessoaFisica of TPessoaFisica(
   constraint PK_PessoaFisica primary key(pessoaId),
   constraint NN_PessaoFisica_nome nome not null,
   constraint UK_PessoaFisica_email UNIQUE (email),
   constraint CK_PessoaFisica_pessoaId CHECK (pessoaId > 0),
   constraint FK_PessoaFisica_Conjugue FOREIGN KEY (conjugue)
                                          REFERENCES pessoaFisica
);

create Table Cliente (
   pessoa TPessoaFisica,
   detalhes varchar(4000) constraint NN_Cliente_detalhes not null,
   constraint PK_Cliente primary key(pessoa.pessoaId),
   constraint CK_Cliente_nome check (pessoa.nome is not null)
);
```

47

Bibliografia

- **Principal**
  - DATE, C. J. **Introdução a Sistemas de Banco de Dados.** 7. ed. Rio de Janeiro: Campus, 2000. 803 p.
  - ELMASRI, S. N.; NAVATHE, B.S.. **Sistemas de Banco de Dados:** Fundamentos e Aplicações. 3. ed. Rio de Janeiro: Livros Técnicos e Científicos, 2002. 837 p.
  - SILBERSCHATZ, A. ; KORTH, H.F. ; SUDARSHAN, S. **Sistema de Banco de Dados. 5**. ed. Rio de Janeiro: Elsevier, 2006.
- **Complementar**
  - FREEMAN, R. **Oracle, referência para o DBA:** técnicas essenciais para o dia-a-dia do DBA. Rio de Janeiro: Elsevier, 2005.
  - RAMALHO, J. A. **Oracle:** Oracle 10g, ed. São Paulo: Pioneira Thomsom Learning, 2005.

48


Fim
49