Banco de Dados OO e Objeto-Relacionais

Regina Maria Maciel Braga
regina@cos.ufrj.br

Organização

- Introdução
- SGBDOO
- SGBDOR
- Datawarehouse
- Banco de dados na WEB
- Tópicos avançados
- Programação WEB (asp)

Motivação

- Mundo computacional OO
  - linguagens de programação
    - C++, Java
  - Engenharia de Software
    - Análise, projeto, reuso
  - Redes de comunicação
    - OMG - CORBA, Microsoft – DCOM
  - Porque não pensar em Banco de Dados OO?

Motivação

- Linguagens de programação OO têm aceitação na prática
  - Problema: programação no paradigma OO sobre SGBD relacional requer mapeamentos
    - Materialização dos objetos a partir do BD
    - Dematerialização para o BD

Motivação

- Aplicações avançadas:
  - Aplicações cujos requisitos não são bem atendidos por SGBD relacional
  - Ex:
    - Projeto e produção de peças e componentes
    - Ferramentas CASE
    - Textos, hipertextos, gráfico e automação de escritório
    - Química, genética e geoprocessamento
    - Aplicações de negócios atuais

Motivação

- Tipo de dados para aplicações avançadas
  - Dados multimídia: fotos, textos não estruturados, vídeo, aúdio. (*rich content*)
  - Novos tipos de dados: séries temporais, geoespaciais, etc.
  - Novos tipos que o usuário deseja definir:
    - Dados aninhados: ex: carro com todos os seus componentes

Persistência de aplicações

- Sistema de arquivo do SO
- SGBD convencional
- inclusão de persistência sobre linguagens de programação
- novas arquiteturas de banco de dados
  - extensão de SGBD relacional
  - orientados a objetos
- novas tendências:
  - Arquiteturas de integração

Aplicações convencionais

- Requisitos de BD:
  - Uniformidade de instâncias
    - Grande número de itens de dados, com estrutura semelhante e com tamanho idêntico (coleções de registros)
  - Orientação a registros
    - Tipo básico de dados é o registro de estrutura e tamanhos fixos
  - Pequenos itens de dados
    - Campos são de pequenos tamanhos
  - Campos atômicos
    - Não há estrutura dentro dos campos
    - Registros estão na primeira forma normal

Aplicações avançadas (não convencionais)

- Requisitos de BD
  - Identificação de objetos
    - Nem sempre é aceitável de que cada objeto deve ter um atributo que o identifica. Ex: figuras geométricas
  - Objetos complexos e compostos
    - Composição de documentos. Composição de peças
  - Armazenamento de procedimentos e encapsulamento
  - Ordenação de conjuntos
  - Esquema de dados flexíveis e evolucionários
  - Integração com a LPOO
  - Consultas complexas

Aplicações avançadas (não convencionais)

- Grande diversidade na forma como a informação será
  - estruturada (listas, hierarquia, composição)
  - manipulada (operações específicas)
  - apresentada (multimídia)

Classificação de SGBD (Stonebrake)

	Dados simples	Dados Complexos
Com Consulta	2	4
Sem Consulta	1	3

1 – Dados simples sem consulta

- Exemplo
  - Processadores de textos (?)
- Não há consultas
  - Operações de persistência são
    - Ler arquivo (getFile())
    - Escrever arquivo (putFile())
- Estrutura de dados é simples (do ponto de vista de persistência)
  - Unidade de tratamento é o arquivo
- SGBD que atende a aplicação
  - Sistema de arquivos do sistema operacional
- Boa performance

Classificação de SGBD (Stonebrake)

	Dados simples	Dados Complexos
Com Consulta	2	4
Sem Consulta	Gerenciadores de arquivos	3

2 – Dados simples com consulta

- Exemplo
  - Sistema de informações administrativas: controle de pessoal
- Estrutura de dados simples
  - Aquela atendida pelo SGBD relacional
    - Tabelas “flat”
    - Tipos de dados de SQL
- Aplicação acessa a base de dados através de consultas possivelmente envolvendo várias tabelas
- Gerenciamento de transações consistente
- Segurança
- Ferramentas de interface
- SGBD que atende a aplicação
  - SGBD Relacionais

2 – Dados simples com consulta

- Exemplo de estrutura de dados (tabela) com atributos simples que podem ser descritos usando o padrão SQL92

```
create table emp(
nome varchar(30),
idade int,
salario float,
dept varchar(20));
```

- Exemplo de consulta:

```
select nome
from emp
where idade < 40 and salario > 40000;
```


2 – Dados simples com consulta

- **Porque não usar um sistema de arquivos**
  - Não existência de consultas
  - Otimizações, etc..
- **Porque não usar um SGBDOO**
  - Maioria possui SQL limitado (exceção O2);
  - Não otimizado para grande quantidade de dados

2 – Dados simples com consulta

- **Porque usar um Banco de Dados Relacional**
  - Tira partido da capacidade de multiprocessamento
  - Rodar em qualquer plataforma de hardware
  - Prover gateways para outro vendedores de banco de dados
  - Prover execução paralela de consultas
  - Prover boa performance

Classificação de SGBD (Stonebrake)

	Dados simples	Dados Complexos
Com Consulta	SGBD Relacional	4
Sem Consulta	Gerenciadores de arquivos	3

3 – Dados complexos sem consulta

- Exemplo
  - Banco de dados para engenharia de produtos
  - Banco de dados para Ferramentas CASE
- Estrutura de dados é mais complexa do que a usada em aplicações comerciais
  - Representação de um diagrama de classes

3 – Dados complexos sem consulta

- Processamento geralmente não envolve consultas mas carga e descarga em bloco de diagramas de projeto
- Aplicação trata a estrutura complexa na memória
  - Garbage collection
  - Navegação entre elementos de dados
- SGBD que atende a aplicação:
  - SGBDOO simples (sem OQL)

3 – Dados complexos sem consulta

- Comandos para criação do esquema

```
create class diagrama(
        nome varchar(30),
 componentes set-of(comp_sintatico)
                );

create class comp_sintatico (
                posicao point,
                semantico comp_semantico;
ligacoes set-of(comp_sintatico)
                );
```


3 – Dados complexos sem consulta

- Leitura de todo o diagrama de uma só vez (todo o diagrama é lido para aplicação do cliente).
- Escrita de todos os componentes do diagrama e novos tb.

3 – Dados complexos sem consulta

- **Porque não usar um sistema de arquivos tradicional?**
  - Aplicação tem que ler manualmente as informações do diagrama e de seus componentes
  - Conversão do formato do disco para o formato em memória (problema de identificação)

3 – Dados complexos sem consulta

- **Porque não usar um banco de dados relacional**
  - Não existem tipos de dados adequados para este tipo de aplicação em SGBDs puramente relacionais
  - Necessidade de mapeamentos muitas vezes complexos

Classificação de SGBD (Stonebrake)

Com Consulta	SGBD Relacional	4
Sem Consulta	Gerenciadores de arquivos	SGBDOO ? Serialização de Java
	Dados simples	Dados Complexos

4 – Dados complexos com consulta

- Exemplo de aplicação
  - Sistema para CESAMA
    - Armazena informação sobre dutos, bombas estações, nível de reservatórios, etc.
    - Necessita consultar dados sobre estas entidades, inclusive imagens

4 – Dados complexos com consulta

- Processamento envolve consultas ao BD para procurar imagens baseado em atributos da imagem e informações em descritores de imagem
  - Linguagem de consulta estendida à objetos complexos (SQL3 ou OQL)
- SGBD que atende a aplicação:
  - SGBD Objeto-Relacionais
  - SGBDOO Completo (ex. O2).

4 – Dados complexos com consulta

- Criação do esquema

```
create  table slides(
id int,
data date,
caption document,
imagem photo_cd_picture);

create table marcacao(
nome varchar(30);
localizacao point);
```

4 – Dados complexos com consulta

- Tipos de consultas:

select id

from slides P, marcacao L S

where **sunset(P.imagem)** and **contains(P.caption, L.nome)** and L.localizacao <= S.Localizacao+20 and S.nome = “Barreira do Triunfo”;

4 – Dados complexos com consulta

- **Porque não usar um SGBDR?**
  - Porque não suporta tipos complexos e nem consultas sobre métodos
  - O processamento de sunset teria que ser feito no cliente (imagens teriam que ser transmitidas para o espaço do cliente)
- **Porque não usar um SGBDOO e sistemas de arquivos**
  - Falta de SQL (alguns possuem OQL)

Classificação de SGBD (Stonebrake)

	Dados simples	Dados Complexos
Com Consulta	SGBD Relacional	SGBD Objeto-Relacional
Sem Consulta	Gerenciadores de arquivos	SGBDOO

Porque Objeto-Relacional é a solução? (segundo Stonebrake..)

- Duas forças de mercado
  - Automação de novas aplicações multimídia, especialmente utilizando a web:
    - Consultas ad-hoc
    - Manipulação de objetos multimídia
  - Aplicações comerciais vão migrar para o quadrante 4
    - Sistemas de suporte a decisão com objetos complexos
    - Diminuição do custo do hardware.

Porque SGBDOO é a solução (segundo IDC..)

- SGBDOO foram totalmente construídos para suporte a aplicações complexas
  - Otimizações são todas baseadas no modelo OO e não no relacional, como é o caso dos SGBDOR
  - Suporte direto a todos os conceitos da OO: encapsulamento, herança, etc.
  - Banco de dados flexível e com todos os tipos de dados necessários para mapeamento de aplicações OO
- **No entanto, o IDC atesta que o mercado nos próximos anos é Objeto-Relacional**

SGBDOO

Tecnologia de banco de dados

+

Conceitos de sistemas orientados a objetos

- A maioria dos SGBDOOs vieram da aplicação de persistência em linguagens de programação OO
- **Padrão para SGBDOO : ODMG**

SGBDOO

- **ODMG: Object Database Management Group**
- **Fundação : setembro de 1991**
- **Objetivo: definir um padrão para garantir a portabilidade das aplicações escritas para bases de objetos.**
- **Presidente: R.G.G. Cattell**
- **Web: http://www.odmg.org**
- **Companhias com voto:**
  - **Praticamente toda a indústria de ODBMS/ODM**

✓ Poet ✓Object Design ✓GemStone
✓ Servio ✓ Objectivity ✓ Versant
✓ UniSQL ✓ Ontos, CA, Sybase, Microsoft, ...

SGBDOO

- Importância do Padrão
- **SQL**
  - Independência do SGBD:
    - portabilidade e interoperabilidade entre SGBDs
- **ODMG**
  - Independência do SGBD +
  - Harmonia entre o modelo da LP e da LMD
    - Engloba os dados e operações da aplicação

SGBDOO

- Características básicas
  - Base de Dados: coleção de objetos que são instâncias de classes (definidas no esquema da base de dados)
  - encapsulamento
  - identificador do objeto

SGBDOO

- Características Básicas
  - valores complexos, estruturas de objetos
  - objetos compostos
  - hierarquia de classes (herança) com acoplamento tardio

SGBDOO

- ODMG
  - Modelo de Objetos
  - Linguagem de Definição de Objetos - ODL
  - Linguagem de Consulta - OQL
  - Ligações com LPOO
  - Metadados
  - Controle de Concorrência
  - Modelo de Transações

SGBDOO

- Modelo de objetos: objetos e valores.
  - cada objeto possui um IDO.
  - Objeto: conjunto de valores
  - O valor pode ser simples ou estruturado e pode incluir referências a (IDOs) de outros objetos.

SGBDOO

- Objetos são categorizados em tipos.
  - Um tipo define o estado e o comportamento de suas instâncias.
  - O comportamento de um objeto: conjunto de operações (métodos).
  - Tipos são organizados em um grafo de subtipos e supertipos (herança).
  - O conjunto de todas as instâncias de um tipo é denominado **extensão** do tipo.

SGBDOO

- ODMG
  - Distinção entre generalização e realização
    - Herança de comportamento (realização): interfaces.
    - Herança de estado (generalização): classes
  - A maioria dos BDs não fazem esta distinção.
    - Geralmente utilizam apenas uma herança (extends) para representar os dois tipos.

SGBDOO

- Identificador de Objetos
  - Cada objeto possui uma identidade independente de seu valor
  - O valor pode ser modificado sem mudar a identidade
  - Idênticos e Iguais são dois conceitos diferentes
  - Usuários não tem acesso aos identificadores
  - O conceito de chave deve ser preservado

SGBDOO

- Atributos
  - Simples: valores literais, incluindo os atributos BLOBs
    - name: String
  - Coleção: são usados para representar conjuntos, ordenados ou não
    - sections: list [Section]
  - Referência: representar relacionamentos entre os objetos.
    - dept : Department

SGBDOO

- Relacionamentos
  - Implementada através atributos de referência.
    - Maior poder semântico do SGBD,
    - é mais natural para o usuário
    - facilita a criação de tipos complexos.
    - **Essa representação é um dos maiores diferenciais entre o modelo relacional e o modelo orientado a objetos**.

SGBDOO

- Relacionamentos
  - Nome
  - Grau (binário, n-ário)
  - Cardinalidade
    - 1 x 1
    - 1 x n
    - n x m
  - Direção
    - uni, bi-direcional

Course:
name: string,
cno: integer,
credits: integer,
dept : Department ⇔ coursesOffered in Department,
sections: list[Section];

Department:
name: string,
dno: string,
coursesOffered: set(Course) ⇔dept in Course,
chair:Professor;

Course:
name: string,
cno: integer,
credits: integer,
dept : Department ⇔ coursesOffered in Department,
sections: list[Section];

Department:
name: string,
dno: string,
coursesOffered: set(Course) ⇔dept in Course,
chair:Professor;

Profa Regina Braga

SGBDOO

- **Relacionamento Binário**
  - inclusão de um atributo de referência em uma das classes (ou ambas) envolvidas no relacionamento
  - Ex:atributo dept da classe Course.
  - Ao contrário do modelo relacional, na orientação a objetos o relacionamento entre dois objetos não é **obrigatoriamente** simétrico e possui direção.

47

Course:
name: string,
cno: integer,
credits: integer,
dept : Department ⇔ coursesOffered in Department,
sections: list[Section];

Department:
name: string,
dno: string,
coursesOffered: set(Course) ⇔dept in Course,
chair:Professor;

Profa Regina Braga

SGBDOO

- Relacionamento Binário
  - O relacionamento curso e seu departamento pode ser representado apenas na classe Course.
  - Neste caso, para se saber o curso (ou cursos) de um determinado departamento é necessário percorrer todos os objetos da classe Course. (RUIM!!!!!)

48

Course:
name: string,
cno: integer,
credits: integer,
dept : Department ⇔
coursesOffered in
Department,
sections: list[Section];

Department:
name: string,
dno: string,
coursesOffered:
set(Course) ⇔dept in
Course,
chair:Professor;

Profa Regina Braga

SGBDOO

- **Relacionamento binário**
  - Alguns relacionamentos são semanticamente uni-direcionais.
    - Ex: chefe do departamento (atributo chair da classe Course).
    - não faz muito sentido registrar no objeto da classe Professor o departamento que eventualmente ele chefia.

Course:
name: string,
cno: integer,
credits: integer,
dept : Department ⇔
coursesOffered in
Department,
sections: list[Section];

Department:
name: string,
dno: string,
coursesOffered:
set(Course) ⇔dept in
Course,
chair:Professor;

Profa Regina Braga

SGBDOO

- Outra vantagem da representação orientada a objetos consiste nos relacionamentos n x m.
  - criar atributo do tipo coleção em uma das classes ou em ambas.
    - Ex: Course e Department.

Course:
name: string,
cno: integer,
credits: integer,
dept : Department ⇔ coursesOffered in Department,
sections: list[Section];

Department:
name: string,
dno: string,
coursesOffered: set(Course) ⇔dept in Course,
chair:Professor;

SGBDOO

- **Relacionamento Inverso**
  - **Controle de integridade entre os relacionamentos**
  - uma representação semântica para que fique explícito a ocorrência de dois atributos *inversos*.
    - passa a realizar automaticamente o controle da consistência das informações.
    - Ex:

      dept : Department ⇔
      coursesOffered in
      Department

SGBDOO

- **Relacionamento Não Binário**
  - **Criação de uma classe relacionamento para que sua estrutura possa representar o relacionamento entre essas classes.**
- **Binário com Atributo**
  - **Criação de uma classe relacionamento**
- **Nesse caso, a solução possui as mesmas desvantagens de representação do modelo relacional.**

SGBDOO

- **Relacionamentos**
  - **Vantagens OO sobre REL**
    - **uso de IDOs ao invés de chaves para representar**
    - **todos os relacionamentos podem ser "vistos" pela navegação**
    - **Possibilidade de uso de listas ordenadas no relacionamento**
    - **a representação de relacionamentos via objetos é menos freqüente (ex. n x m)**
  - **Vantagens REL sobre OO**
    - **integridade referencial**

SGBDOO

Voo:
nome: string,
tarifa: float,
assentos disponíveis: list[Pessoa],
voos_trechos: list(Voo)
End;

- **Objetos Compostos**
  - **Agregação**
    - **Atributos** agregados formam objetos
    - um vôo possui nome, tarifa, assentos_ disponiveis.
  - **Objetos** agregados formam *objetos compostos*
  - Um vôo é composto de outros vôos.

SGBDOO

- **Semântica de Agregações**
  - Construtores pré- definidos na modelagem para : cópia, remoção e agrupamento físico.
  - Ex: quando da remoção do todo, remover todas as partes automaticamente?
- **Relacionamento de agregação**
  - Tornar explícito no relacionamento "é-parte- de" que um determinado objeto é componente de outro.

SGBDOO

- **Persistência**
  - **por tipo (Extensão (Extent) da classe)**
    - O sistema mantém o conjunto de instâncias de cada classe. O nome do conjunto é o mesmo da classe.
  - **Explícita (Objetos com nome)**
    - Cada objeto de cada classe deve ser nomeado: coleções de objetos, objetos individuais, etc.
  - **por alcance**
    - Todos os objetos apontados por objetos persistentes também são persistentes.

SGBDOO

- Um BD é acessado de duas formas
  - Navegacional
  - Associativa (linguagem de consulta)
- Requisitos de uma linguagem de consulta OO
  - **Integrada com as linguagens de programação**
    - mesmo sistema de tipos
  - Consulta sobre objetos persistentes, temporários, distribuídos ou objetos com versões

SGBDOO

- OQL
  - Linguagem associativa de consultas da ODMG
  - Inspirada em SQL
  - Originária do O2
  - Construção básica:
    - SELECT...FROM...WHERE
- **Trabalhar com o modelo de dados OO**
  - IDO, encapsulamento, herança, relacionamentos complexos, polimorfismo, ...

SGBDOO

- Extents (extensão) são conjuntos de todos elementos de uma classe persistente.
- Em SQL, as tabelas são consultadas,
- Em OQL, Extents ou outros conjuntos é que são consultados.
- Em Poet:
  - Quando formular uma consulta, utilizar o nome da classe seguindo da palavra Extent.
  - Ex: Se a classe se chama Cartoon seria "CartoonExtent"

SELECT *

FROM CartoonExtent AS cartoon

- Retorno de todos os objetos cartoon.

SGBDOO

- Variável-conjunto
  - Quando utilizamos um Extent em um SELECT, uma variável conjunto deve ser associada com o extent
  - A variável conjunto permite que percorramos cada membro do conjunto e possamos filtrar os elementos na cláusula WHERE e no conjunto-resultado.
  - Ex:

SELECT *

FROM CartoonExtent AS **cartoon**

WHERE cartoon.title_ = "Baseball Bugs"

SGBDOO

- Exemplos de consultas em Poet

SELECT *

FROM FilmExtent AS film (forma de especificar a variável de extensão film)

SELECT film

FROM film IN FilmExtent (outra forma da variável film)

SGBDOO Poet

SELECT film.title_

FROM FilmExtent AS film (seleção de um conjunto de strings)

SELECT film.title_, film.year_

FROM FilmExtent AS film (não é permitido em Poet, apesar de ser permitido em ODMG (struct(film.title, film.year))

SGBDOO

SELECT film.directors_
FROM FilmExtent as film
WHERE film.title_ = "Porky The Giant Killer“ (seleção de atributo complexo (conjunto))

SELECT film
FROM FilmExtent AS film, film.directors_ AS director
WHERE director.indexName_ = "Avery, Tex“
(duas variáveis, sendo uma referente a outra)

SGBDOO

SELECT * FROM PersonExtent AS person
WHERE person.name LIKE "M*“ (uso do LIKE)

SELECT film
FROM FilmExtent AS film
WHERE film.year_ >= 1961 AND film.year_ <= 1965
ORDER BY film.year_ (uso de operadores lógicos)

SGBDOO

- =, == (igualdade)
- !=, <> (diferente)
- >, >=, <, <=
- Qualificadores para “case sensitive’
  - WHERE film.title_ =<caseinsensitive> "star wars"
  - WHERE film.title_ =<ci> "star wars"
  - WHERE film.title_ =<ignorecase> "star wars“

SGBDOO

- Teste da pertinência de um determinado valor em um conjunto

```
SELECT *
    FROM FilmExtent as film
     WHERE film.studio_ IN
  StudioExtent
```

SGBDOO

- Order by

```
SELECT * FROM FilmExtent AS film
  ORDER BY film.year_ ASC, film.title_ DESC
```

- Count

```
SELECT COUNT(*)  FROM FilmExtent AS film
    WHERE film.year_ = 1961

COUNT ( SELECT * FROM FilmExtent AS film
      WHERE film.title_ LIKE "Duck*" )

SELECT film  FROM FilmExtent AS film
 WHERE ( SELECT COUNT(*) FROM film.directors_ ) > 1
```


SGBDOO

- Exists

```
SELECT * FROM FilmExtent AS f
    WHERE ( EXISTS director IN f.directors_ :
  director.indexName_ LIKE "Jones*" )

SELECT f FROM FilmExtent AS f WHERE ( EXISTS d
  IN f.directors_ : d.indexName_ LIKE "Jones*") AND
  ( d.year_ > 1960 )
```

- For ALL

```
FOR ALL film IN FilmExtent : film.year_ > 1929
```

SGBDOO

- Define: auxilia a reduzir a complexidade de uma consulta

```
DEFINE sixties AS
    SELECT *
     FROM FilmExtent AS film
      WHERE film.year_ > 1959 AND film.year_ <
1970;
SELECT *
 FROM s IN sixties
  WHERE s.title_ LIKE "Bunny*"
```


SGBDOO

- Se a consulta retorna somente um item, pode-se converter o resultado de um conjunto para um objeto

```
ELEMENT (
 SELECT film
  FROM FilmExtent AS film
   WHERE film.title_ LIKE "Duck
Dodgers*" )
```

Esta consulta só retorna um objeto da classe Film

SGBDOO

- Casting

SELECT *

FROM FilmExtent as film

WHERE ((Cartoon)film).toons_.name_ = "Bugs Bunny“

- Distinct
  - **Poet não suporta a cláusula distinct, apesar da mesma fazer parte da especificação do ODMG**

SGBDOO

- “embedded” em LP ou utilizada stand-alone
  - Mesmo sistemas de tipos da LP
- Consultas são especificadas como tipo String nas LP

SGBDOO

```
String queryString = “select cartoon from CartoonExtent
   as cartoon where cartton.year = 1953”;

OQLQuery query = new OQLQuery(queryString);
Object result = query.execute();

If ( result instanceof CollectionofObject)
{.....
   Iterator iter = ((CollectionOfObject) result).select();
 while (iter.hasNext()) {.....}}
```


SGBDOO

- Expressões de caminho
  - Pode-se utilizar tanto a notação “.” quanto “->”
    - Exemplos

```
SELECT film
FROM FilmExtent AS film
WHERE film.studio_.name_= "Merrie Melodies“
Ou
SELECT film
FROM FilmExtent AS film
WHERE film.studio_->name_= "Merrie Melodies“
```

SGBDOO

- **Escopo: o que pode ser consultado**
  - **A extensão de uma classe mantida pelo sistema**
  - A extensão de uma classe engloba a extensão de todas as subclasses ?
    - Varia de BD para BD (como será em Poet?)

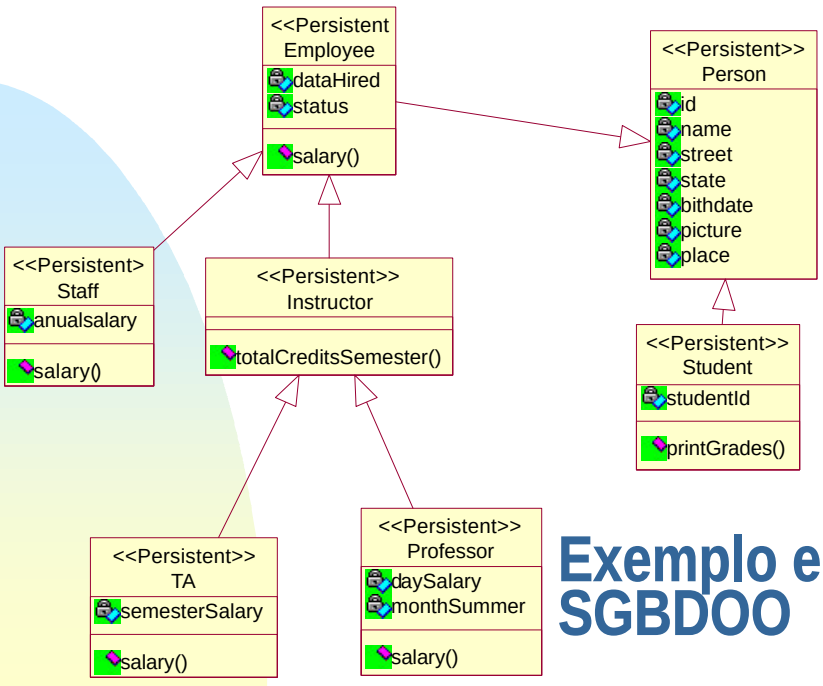
SGBDOO

- **Coleções mantidas pelo usuário diferentes das extensões**
- **Os predicados podem conter métodos**
  - **Varia de BD para BD (como será em Poet e O2?)**

SGBDOO

- **O2 (melhor SGBDOO que existiu)**
  - **Exemplo de programa O2C e OQL**
  - **Imprimir o nome dos clientes de Salvador**

```
o2 set (Cliente) cliSSA;
oql ( cliSSA ,
"select x from x in $1 where \
x. cidade = 'Salvador' ",
Cliente );
for x in cliSSA display (x. nome);
ou
cliSSA. display;
```

Exemplo em SGBDOO

Exemplo em SGBDOO

- Consultar todos dos empregados que ganham mais que R$ 96.000 por ano
- **OO**

select x from employees x **where** x.salary >= 96000;

- **Rel**

select x. **from** staffs x **where** x.salary >= 96000;
union all
select x **from** professors x **where** x.salary >= 96000;
union all
select x **from** tas x **where** x.salary >= 96000;

SGBDOO

- Exemplo Polimorfismo
- **OO**

select x from employees x **where** x.salary >= 96000;

- **Rel**

select x **from** staffs x **where** x.annualSalary >= 96000
union all
select x **from** professors x **where**
(x.salary*(9+x.monthSummer)/9.0) >= 96000
union all
select x **from** tas x
where (apptFraction*(2*x.salary)) >= 96000

SGBDOO

- **Classe**
  - Fábrica de objetos
- Um SGBDOO verdadeiro possui o conceito de classe
- Ex:

```
class Vôo
 type tuple (
 public nome: string ,
 read tarifa: real ,
 read assentos_ reservados: list (Pessoa))
 ...
 method
 public solicita_ reserva ( nb : integer): boolean,
 public muda_ tarifa ( tarifa_ nv : real) ...
 end;
```


SGBDOO

- **Método**
  - **Um método está sempre associado a uma classe.**

ex:

```
method body solicita_ reserva (rn:
  integer) : boolean in class Vôo
    {
    if ( self-> assentos_ disp >= rn)
    return true;
    else
    return false;
    }
```

SGBDOO

- **Criando objetos**

```
run body {
o2 Voo v;
v = new Vôo;
v-> nome = "Alitalia 128";
printf(" O vôo %s foi criado.\ n", v-> nome);
v-> muda_ tarifa( 1500.00);
v-> muda_ assentos (250);
v-> display;}
```

- **Esta seria a implementação de um procedimento em O2 (stored procedure)**

SGBDOO

- **Criando objetos e coleções**

```
class Pessoa
public type tuple (
nome: string ,
ano_ nasc: integer)
run body
{
o2 Pessoa p1, p2, p3;
p1 = new Pessoa; p1-> nome = "Vivaldi"; p1-> ano_ nasc = 1678;
p2 = new Pessoa; p2-> nome = "Bach"; p2-> ano_ nasc = 1681;
p3 = new Pessoa; p3-> nome = "Handel"; p3-> ano_ nasc = 1685;
}
name Barrocos : set (Pessoa);
Barrocos += {p1, p2, p3};
```

SGBDOO

- Navegação

```
class Vôo type tuple (
public nome: string ,
read tarifa: real ,
public assentos_ reservados: set (Pessoa) )
method public solicita_ reserva ( nb : integer):
   boolean,
end;

name Vôos : set ( Vôo );
```


SGBDOO O2

```
o2 Vôo v1 , v2 , vtemp ;
o2 Pessoa p1, p2 ;
v1 = new Vôo ( "Air France 147", 1500.00);
v1. assentos_ reservados += set( p1 = new
   Pessoa ( "João Reis", 19));
v2 = new Vôo ( "Varig 747", 2000.00);
v2. assentos_ reservados += set( p2 = new
   Pessoa ( "José Rios", 39);
Vôos += set ( v1 , v2 ) ;

select v  from v in Vôos,
p in v. assentos_ reservados
where p. idade >30
```

SGBDOO

- **Herança**

class Transporte **type tuple (public** nome: **string** ,
read tarifa: **real** , **private** assentos_ disp **: integer)**
Method public muda_ assentos **(** assentos_ disp **:**
integer), end;

class Vôo **inherit** Transporte **type tuple (**
read assentos_ reservados: **list** (Pessoa) **)**
Method .....end;

class Trem **inherit** Transporte **...**

name Transp **: set (**Transporte) **; name** Vôos **: set(** Vôo) **;**
name Trens **: set (**Trem) **;**

SGBDOO

o2 Vôo v1, v2;
o2 Trem t1, t2;
v1 **= new** Vôo **("Air France 147", 1500.00);**
v2 **= new** Vôo **("Varig 747", 2000.00);**
Transp **+= set (** v1 **,** v2 **) ;**
Vôos **+= set (** v1 **,** v2 **) ;**
t1 **= new** Trem **("Maria Fumaça", 15.00);**
t2 **= new** Trem **("Vera Cruz", 100.00);**
Transp **+= set (** t1 **,** t2 **) ;**

select x **from** Transp x **where** x. tarifa >= 100.00 **;**
Resposta: v1, v2, t2

SGBDOO

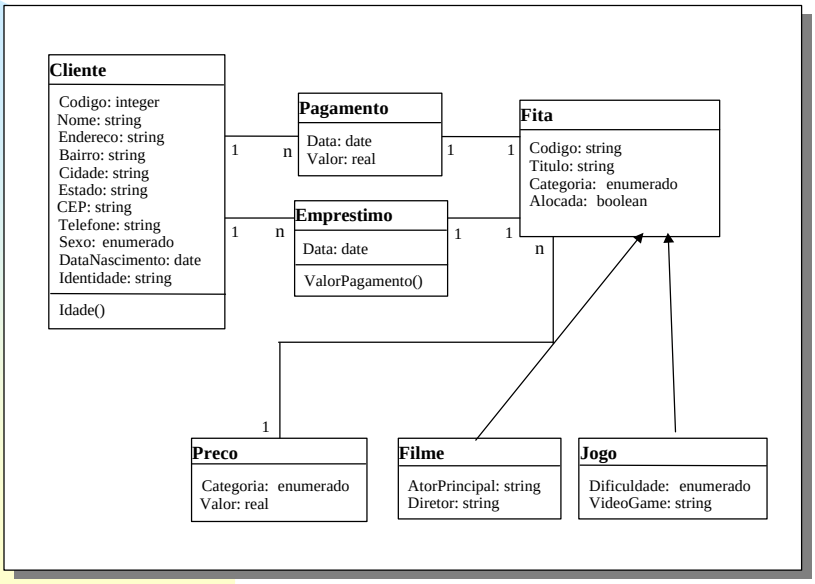
- **Exemplos: operadores p/ coleções**
- **ordenação**

select x **from** Transp x **where** x. tarifa >= 100.00
order by x. tarifa, x. nome

- **usando agregação**

max (select tarifa **from** Transp **)**
select * **from** Transp
group by baixo: tarifa < 100.00,
medio: tarifa >= 100.00 and tarifa < 1000.00,
alto: tarifa >= 1000.00

SGBDOO x SGBDR



SGBDOO: Mapeamento Direto

Class Cliente (
private Nome String,
private Identidade Char(15),
private Emprestimos:
List(Emprestimo) inverse Cliente
.....
(public solicita-end: string
public calcula- emprestimos(
Emprestimos)real}

Class Emprestimo (
private cliente Cliente,
private fita fita,
private data Date)
(public calcula multa: re

Class Fita (
Titulo String,
Categoria String,
Alocada Boolean,
Preço Real)

Class Filme inherit Fita
Ator_Principal String,
Diretor String,

Class Jogo inherit Fita
Dificuldade String,
Video_Game String

Objeto Composto

Herança

Encapsulamento

Referência pela Identidade do Objeto (ponteiro).

SGBDR

CREATE TABLE Fita (
Id Integer not null,
Codigo Integer not null,
Titulo Char(50) not null,
Categoria Char(30) not null,
Alocada Char(1) not null,
Tipo Char(10) not null,
Ator_Principal Char(50),
Diretor Char(50),
Dificuldade Char(30),
Video_Game Char(30),
Id_Preco Integer not null,
Primary Key(ID));

Programas de aplicação
solicita-end: string
calcula-emprestimo(Emprestimos): real)}

CREATE TABLE Emprestimo (
Id Integer not null,
Id_Cliente Integer not null,
Id_Fita Integer not null,
Data Date not null,
Primary Key(ID));

CREATE TABLE Cliente (
ID Integer not null,
Codigo Integer not null,
Nome Char(50) not null
Endereco Char(100),
Data_Nascimento Date,
Identidade Char(15),
Primary Key(ID));

Necessidade de **join** para recuperar emprestimo] de um dado cliente

Campos SubUtilizados (só para o caso de filme)

Campo adicional por conta de não se ter herança

SGBDR - Outra Estratégia

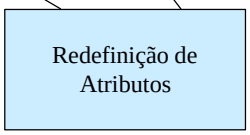
```
CREATE  TABLE  Filme (
    Id Integer  not null,
    Codigo Integer not null,,
    Categoria Char(30) not null,
    Alocada Char(1) not null
          Ator_Principal Char(50),
    Diretor Char(50),
    Id_Preco Integer not null,

    Primary Key(ID));
```

Programas de aplicação

```
solicita-end: string
calcula-emprestimo(
    Emprestimos): real)}
```

```
CREATE  TABLE  Jogo (
    Id Integer  not null,
    Codigo Integer not null,
    Categoria Char(30) not null,
    Alocada Char(1) not null,
    Id_Preco Integer not null
    Dificuldade Char(30),
    Video_Game Char(30),,
    Primary Key(ID));
```



SGBDOO x SGBDR

- Para calcular o valor dos empréstimos de um dado cliente
  - SGBDOO O2

    select p.calculaEmprestimo() from p in Clientes
    where p.Nome = “Joao”;

  - SGBDR

    select e.idEmprestimo from Cliente c
    Emprestimo e
    where c.nome = joao and e.Id_cliente = c.Id;
    (**join!!!)**

    e depois : calculaemprestimo(e.idEmprestimo);

SGBDOO - Poet

- Banco de dados OO que segue o padrão ODMG 3.0
- Direito de voto no comitê do ODMG
- De acordo com as diretivas da ODMG para acoplamento com Java (ODMG Java Binding)
- Permite o armazenamento facilitado de objetos Java
- Permite a persistência por alcance
- Versão *Trial* e portanto algumas funcionalidades não estão habilitadas

SGBDOO - Poet

- Gerente de objetos Poet: capaz de armazenar e recuperar objetos Java da base.
  - As classes que devem ser persistentes em uma aplicação Java, não precisam de ser definidas de maneira especial para serem persistentes em Poet. Tudo continua da mesma forma.
  - O Poet disponibiliza um conjunto de pacotes java , que implementam os serviços necessários. (Necessidade de alguns imports)
- Mecanismos para controle de transação, locks, acesso remoto, etc.

SGBDOO - Poet

- Banco de dados
  - Armazenamento dos objetos propriamente ditos
- Dicionário
  - Armazena os esquema do BD, a partir das classes Java que necessitam ser persistentes
- PTJ
  - Programa utilizado pelo Poet para ler e processar as classes Java (.class).
  - Extrai as informações dos arquivos .class e usa esta informação para registrar as classes persistentes no dicionário.
  - Cria arquivos adicionais de configuração _pt_meta.class

SGBDOO Poet

- Exercício – Uma aplicação simples Java com dados armazenados no Poet
- Criar uma classe Java que será a classe a qual faremos a persistência
- Detalhes da implementação
  - Faremos o projeto no Jbuilder mas temos que importar duas bibliotecas do Poet para compilarmos os programas
  - Os arquivos em bycode Java devem estar no diretório a partir do qual serão criadas as bases.
  - Para executarmos as aplicações: usar JDK e não Jbuilder

SGBDOO Poet – Classe Pessoa

```
import java.util.*;
import com.poet.odmg.*;
import com.poet.odmg.util.*;

/**
 * Representa uma pessoa do sistema de agenda.
 */
class Pessoa {
    private String nome;
    private Date nascimento;
    private String endereco;
    private ListOfObject telefones;
    transient  private Vector teste; // este atributo não será
   armazenado
```



```
    public Pessoa(String nome, Date nascimento, String endereco){
          setNome(nome);
          setNascimento(nascimento);
          setEndereco(endereco);
             telefones = new ListOfObject();}

    public void setNome(String newNome)
      { nome = newNome;}

      public String getNome()
      {return nome;}

    public void setNascimento(Date newNascimento)
      {nascimento = newNascimento;}

      public void setEndereco(String newEndereco)
      { endereco = newEndereco; }
    public String toString()
    {return(getNome()); }}
```

SGBDOO Poet

- Observações importantes
  - A classe Pessoa não contém nenhuma informação adicional dizendo que a mesma é persistente
  - A informação que a classe Pessoa é persistente fica armazenada em um arquivo de configuração
  - Ex:

    [classes\Pessoa]

    Persistent = true
  - O nome padrão do arquivo de configuração é "ptj.opt"e este arquivo deve estar no mesmo diretório dos arquivos fonte.

SGBDOO – Poet

- Não conseguimos armazenar coleções nativas de java, tipo Vector, HashTable, etc. Temos que usar os tipos do Poet (ListOfObject, SetofObject, BagOfObject, etc..)
- Outras informações importantes do arquivo ptj.opt
  - Nomear o banco de dados e o dicionário de dados

[schemata\my_dict]

onefile = false

[databases\my_base]

onefile = false

SGBDOO - Poet

- Arquivo de configuração completo

[schemata\agenda_dict]
oneFile = false

[databases\agenda_base]
oneFile = false

[classes\Pessoa]
persistent = true

- Chamar o programa ptj para criar a base de dados, criar classes Java especiais que tratarão de persistir a classe Pessoa.

  ptj – enhance – inplace –create

SGBDOO - Poet

- Programa para armazenar objetos Pessoa no Poet (Armazena)

```
import org.odmg.ObjectNameNotUniqueException;
import org.odmg.ODMGException;
import com.poet.odmg.*;
import java.util.*;

public class Armazena
{
```

```
Armazena(Database db, String name) throws
    ODMGException
    { Transaction txn = new Transaction();
      txn.begin();
      try
      { Pessoa pes = new Pessoa("Julia Villela", new
    Date(), "rua renato dias 108/501");
        db.bind(pes, name);
      }
     catch (ObjectNameNotUniqueException exc)
      { txn.abort();   throw exc;   }
     catch (POETRuntimeException exc)
      { txn.abort();   throw exc;    }
      txn.commit();
    }
```



```
public static void main(String[] args) throws ODMGException
   {Database db = new Database();
      db.open("poet://LOCAL/agenda_base“,
   Database.OPEN_READ_WRITE);
   try {
         new Armazena(db, "pessoaName");
      }
      finally {
           db.close();
        }
   }
}
```

SGBDOO - Poet

- Para rodar o programa, basta executar como qualquer programa Java
- Detalhes importantes do programa
  - db.open("poet://local/agenda_base",Database.OPEN_READ_WRITE);
    - Abre o banco de dados Agenda e o abre como acesso para leitura e escrita
  - Transaction tx = new Transaction();
  - tx.begin();
    - Uma transação sempre deve ser criada e inicializada para que seja possível a realização de qualquer operação no BD.

SGBDOO - Poet

- Detalhes importantes do programa
  - Pessoa pes = new Pessoa(....);
  - db.bind(pes, name);
    - Cria um objeto da classe pessoa e o associa com um name (raiz de persistência que está na base)
  - tx.commit();
    - Faz com que a operação seja realmente efetivada no BD.
  - Se decidíssemos que a operação não deveria ser efetivada no BD:
    - tx.abort();
  - db.close();
    - Fecha o banco de dados.

SGBDOO - Poet

- Programa para recuperar dados do BD

```
import org.odmg.ObjectNameNotFoundException;
import org.odmg.ODMGException;
import com.poet.odmg.*;

public class Recupera
{
```



```
Recupera(Database db, String name) throws ODMGException
  {  Transaction txn = new Transaction();
     txn.begin();
     try   {
        Pessoa pes = (Pessoa)db.lookup(name);
        System.out.println(pes); }
     catch (ObjectNameNotFoundException exc)
     {  txn.abort();   throw exc;   }
     catch (POETRuntimeException exc)
     { txn.abort();  throw exc;    }
     txn.commit();
  }
```

```
public static void main(String[] args) throws
    ODMGException
    {  Database db = new Database();
       db.open("poet://LOCAL/agenda_base",
    Database.OPEN_READ_WRITE);
       try {
          new Recupera(db, "pessoaName");
       }
       finally {
          db.close();       }
    }
}
```


SGBDOO - Poet

- Detalhes importantes do programa
  - Pessoa pes = (Pessoa)db.lookup(name);
    - Qualquer que seja a classe Java, objetos nomeados sempre são retornados como instâncias da classe Object e por isso temos que fazer o Cast.
    - Observem que este é um name que armazena somente uma pessoa. É a persistência de um único objeto.

SGBDOO - Poet

- Programa para Apagar objetos do BD

```
import
   org.odmg.ObjectNameNotFoundException;
import org.odmg.ODMGException;
import com.poet.odmg.*;

public class Apaga
{
```



```
Apaga(Database db, String name) throws ODMGException
   { Transaction txn = new Transaction();
      txn.begin();
      try      {
         ObjectServices.current().delete(db.lookup(name));
         db.unbind(name);      }

   catch (ObjectNameNotFoundException exc)
      {   txn.abort();
         throw exc;}

   catch (POETRuntimeException exc)
      { txn.abort();
         throw exc;      }
      txn.commit();
   }
```

```
public static void main(String[] args) throws
 ODMGException
 {
    Database db = new Database();
    db.open("poet://LOCAL/agenda_base",
 Database.OPEN_READ_WRITE);
    try {
       new Apaga(db, "pessoaName");
       }
    finally {
       db.close();
    }   }
}
```


SGBDOO - Poet

- Detalhes importantes do programa
  - db.unbind(name);
    - Remove este name da lista de names disponíveis no BD. (remoção lógica)
    - Na verdade, o objeto não é removido do banco de dados. Ele pode ser acessado através do Extent da classe Pessoa
  - ObjectServices.current().delete(db.lookup(name));
    - Este método é o que na verdade remove realmente o objeto do banco de dados.

- Outras operações no Poet
  - Armazenar um objeto usando diretamente seu EXTENT

```
……
Transaction tx= new Transaction(db);
  tx.begin();
  try {
    Pessoa pes = new Pessoa("Regina", new
Date("01/06/2001"), "rua x");
    db.bind(pes, null); //não está associando
pessoa com nenhum name mas somente com seu extent
    pes = new Pessoa("Marcelo", new
Date("02/06/2001"), "rua z");
    db.bind(pes, null);
……
```


- Recuperar objetos armazenados em um Extent

```
……
 Transaction tx= new Transaction(db);
   tx.begin();
   try
  { Extent Pessoas = new Extent(db, "Pessoa");
    while (Pessoas.hasNext())  {
       System.out.println(Pessoas.next()); } }
 catch (ObjectNameNotUniqueException exc)
 {  tx.abort();   throw exc;}
 catch (ODMGRuntimeException exc)
 { tx.abort();   throw exc;}
 tx.commit(); }
```

SGBDOO – Poet (uso de índices

- Arquivo de configuração ptj.opt

```
[schemata\agenda_dict]
oneFile = false
name = agenda_dict

[databases\agenda_base]
oneFile = false
name  = agenda_base

[classes\Pessoa]
persistent = true
useIndexes = NomeIdx

[classes\Pessoa\members\nome]
unicode = true

[indexes\NomeIdx]
class = Pessoa
significance = 50
lexicalOrder = true
members = nome
```


SGBDOO - Poet

- Achar uma pessoa usando o índice e a função findKey.

```
Extent Pessoas = new Extent(“Pessoa");
Pessoas.setIndex("NameIdx");

if (Pessoas.findKey(“Regina”))
{
    Pessoa p = (Pessoa)Pessoas.next();
     System.out.println(“Achou: "+p);
 }
```

SGBDOO - Poet

- Consulta embutida em código Java

```
……..
OQLQuery query = new OQLQuery(
        " element (                                  "+
        "     select x from x in PessoaExtent  "+
        "     where x.nome = $1                    "+
        " )                                          ");
   String nome = “Regina”;
  query.bind(nome);// nome assume o lugar de
  $1
   Pessoa p = (Person)query.execute();
……..
```


SGBDOO - Poet

- Outro exemplo de consulta embutida em código Java

```
Database db = new Database();
db.open(...);
Transaction t1 = new Transaction( db );
t1.begin();
Extent toonExtent = new Extent( db, "Toon" );
String predicate = "WHERE this.name_ LIKE
   \"*Bunny*\"";
toonExtent.setFilter( predicate );
while (toonExtent.hasNext() )
{System.out.println( toonExtent.next() );}
```