

Bancos de Dados Orientados a Objetos

Banco de Dados II

Prof. Guilherme Tavares de Assis

Universidade Federal de Ouro Preto – UFOP
Instituto de Ciências Exatas e Biológicas – ICEB
Departamento de Computação – DECOM

Introdução

- Alguns fatores que contribuem para a utilização de Bancos de Dados Orientados a Objetos (BDOOs) são:
 - BDOOs surgiram, inicialmente, para atender às necessidades de aplicações mais complexas, envolvendo tipos de dados para armazenar imagens ou grandes textos.
 - BDOOs permitem ao projetista especificar a estrutura de objetos complexos e operações que podem ser aplicadas aos mesmos.
 - BDOOs podem ser facilmente integrados a linguagens de programação orientadas a objetos que, atualmente, são bem utilizadas no desenvolvimento de produtos de *software*.

Introdução

- Exemplos de BDOOs (protótipos experimentais):
 - ORION (*Microelectronics Corporation*), OPENOODB (*Texas Instruments*), Sistema IRIS (HP), ODE (*Lucent Technologies*), Projeto ENCORE/ObServer (*Brown University*).
- Exemplos de sistemas de gerência de bancos de dados orientados a objetos (SGBDOO):
 - GEMSTONE/OPAL, ONTOS, Objectivity, Versant, ObjectStore (O2), ARDENT, POET, Caché.
- O *Object Data Management Group* (ODMG) é uma organização de fabricantes e usuários de SGBDOOs que vem criando padrões para tais sistemas.

Conceitos de Orientação a Objetos

- A origem do termo "orientado a objetos" está nas linguagens de programação orientadas a objetos.
- O objeto possui dois componentes, a saber:
 - estrutura de dados: define o estado do objeto;
 - operações: definem o comportamento do objeto.

Conceitos de Orientação a Objetos

- Encapsulamento
 - A estrutura interna de um objeto é escondida, sendo o objeto acessível por meio de operações predefinidas.
 - As operações podem ser utilizadas para atualizar o estado do objeto (atualização) e recuperar partes do estado do objeto (consulta).

Conceitos de Orientação a Objetos

- Herança
 - Novos tipos de objetos podem herdar parte de estruturas e operações de tipos de objetos previamente definidos.
 - Facilita desenvolver novos tipos de objetos por meio da reutilização de definições de tipos de objetos existentes.
- Polimorfismo
 - O polimorfismo está ligado à capacidade de uma mesma operação ser aplicada a diferentes tipos de objetos.
 - Assim, uma mesma operação pode referir-se a distintas implementações, dependendo do tipo de objeto ao qual é aplicada.

Identidade de Objeto

- SGBDOOs oferecem um identificador único (OID) para cada objeto armazenado no banco de dados.
 - O OID não é visível ao usuário externo, sendo apenas utilizado internamente pelo sistema para identificar exclusivamente cada objeto.

Relacionamentos entre Objetos

- Relacionamentos entre objetos são representados, explicitamente, por meio de um par de referências inversas (para relacionamentos binários).
 - Colocam-se os OIDs de objetos relacionados nos próprios objetos, mantendo-se a integridade referencial.

Construtores de Tipo

- Objetos podem ser de tipos complexos.
 - Um tipo complexo pode ser construído por meio do aninhamento de construtores de tipo.
- Os construtores de tipos básicos são:
 - Átomo (*atom*): representa valores atômicos básicos como, por exemplo, números, cadeias de caracteres e booleanos.
 - Tupla (*struct*): representa um tipo estruturado composto pelos nomes dos atributos e seus respectivos valores ou OIDs. O formato do tipo é $\langle a_1 : v_1, a_2 : v_2, \dots, a_n : v_n \rangle$, onde, para $1 \leq k \leq n$, a_k é um nome de atributo e v_k é um valor ou um OID do atributo em questão.

Construtores de Tipo

- Os construtores de tipos não-básicos, denominados tipos de coleção, são:
 - *set*: representa um conjunto de elementos distintos $\{v_1, v_2, \dots, v_n\}$ do mesmo tipo como, por exemplo, um conjunto de OIDs.
 - *bag*: representa um conjunto de elementos não necessariamente distintos $\{v_1, v_2, \dots, v_n\}$ do mesmo tipo.
 - *list*: representa uma lista ordenada de elementos $[v_1, v_2, \dots, v_n]$ do mesmo tipo como, por exemplo, uma lista ordenada de OIDs.
 - *array*: representa um vetor unidimensional de elementos $[v_1, v_2, \dots, v_n]$ do mesmo tipo, sendo semelhante a uma lista, porém apresentando um tamanho máximo.
- Estes construtores representam, então, coleções de objetos que podem ser não-ordenadas (*set* ou *bag*) ou ordenadas (*list* ou *array*).

Construtores de Tipo

```

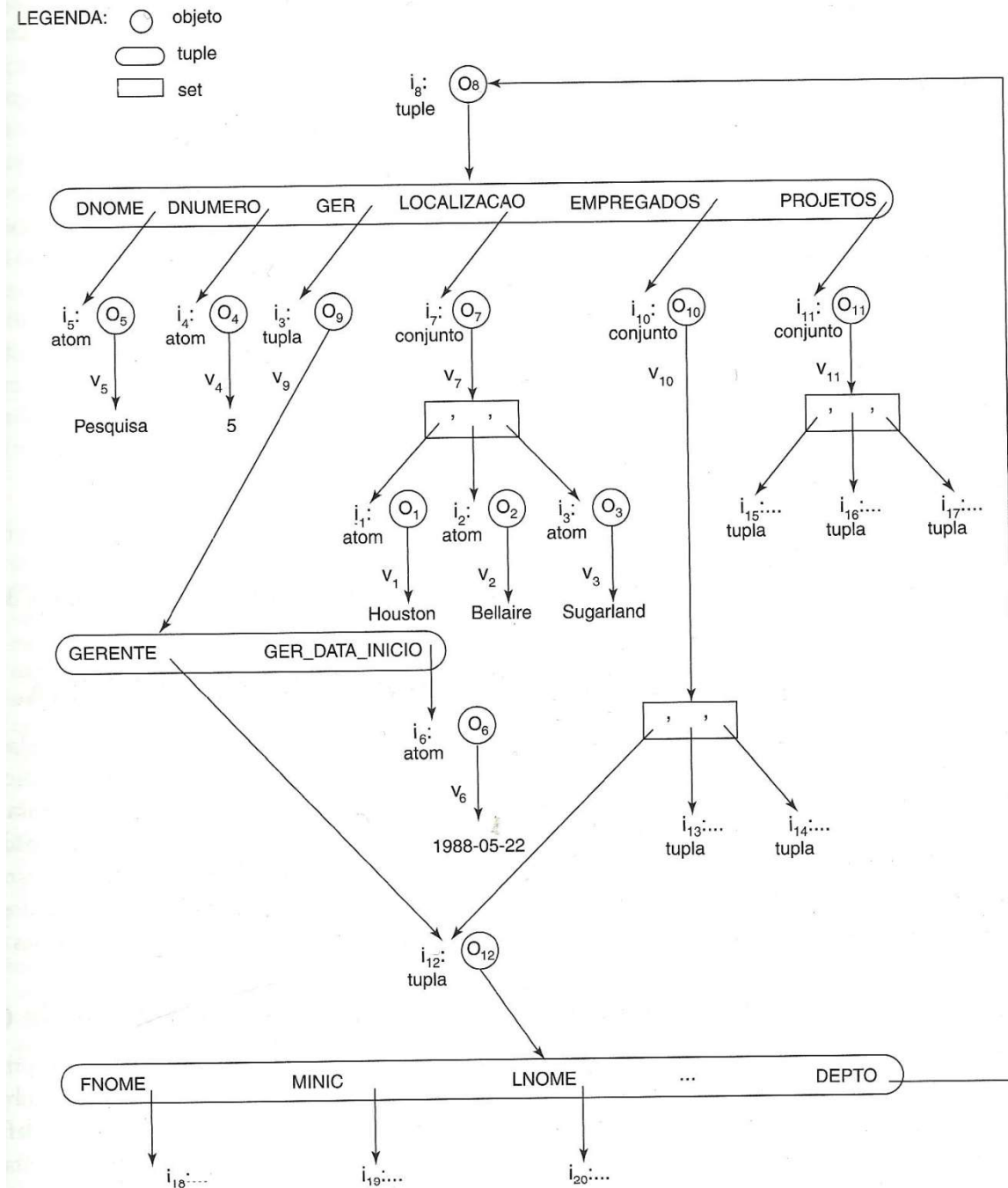
define class FUNCIONARIO
tuple(
    Pnome:                string;
    Minicial:              char;
    Unome:                 string;
    Cpf:                   string;
    Data_nascimento:      DATA;
    Endereco:              string;
    Sexo:                  char;
    Salario:               float;
    Supervisor:            FUNCIONARIO;
    Dep:                   DEPARTAMENTO;

define class DATA
tuple(
    Ano:                   integer;
    Mes:                   integer;
    Dia:                   integer;

define class DEPARTAMENTO
tuple(
    Dnome:                 string;
    Dnumero:               integer;
    Ger:                   tuple(
        Gerente: FUNCIONARIO;
        Data_inicio: DATA);
    Localizacoes
    Funcionarios:          set(FUNCIONARIO);
    Projetos:              set(PROJETO);

```

Tal grafo corresponde à representação de um objeto complexo Departamento



Objetos Complexos Não-estruturados

- Objetos complexos não-estruturados são tipos de dados que requerem um grande volume de armazenamento, tais como tipos de dados que representam imagens ou grandes objetos de texto.
 - São conhecidos como *binary large objects* (BLOBs).
 - São considerados não-estruturados porque um SGBD não conhece a sua estrutura; logo, um SGBD não possui a capacidade de processar diretamente condições de seleção e outras operações baseadas em valores desses objetos.
 - Em SGBDOOs, pode-se definir um novo tipo de dado complexo para os objetos não-estruturados, implementando, por exemplo, métodos para seleção desses objetos.

Modelo de Objetos ODMG

- O *Object Data Management Group* (ODMG) propôs um modelo de dados padrão para bancos de dados orientados a objetos: modelo de objetos no qual se baseiam a *Object Definition Language* (ODL) e a *Object Query Language* (OQL).
 - O modelo provê uma terminologia padrão para os conceitos de bancos de dados de objetos.

Objetos e Literais

- Um objeto é descrito por meio de 4 características, a saber:
 - identificador: é o identificador único do objeto (OID);
 - nome: consiste no nome único do objeto, dentro de um banco de dados;
 - tempo de vida: especifica se o objeto é persistente (objeto armazenado no banco de dados que persiste após o término da aplicação) ou transiente (objeto que desaparece quando a aplicação termina);
 - estrutura: especifica se o objeto é atômico (incluindo o construtor de tipo tupla) ou de coleção.

Objetos e Literais

- Um literal é um valor que não tem identificador de objeto.
- Existem três tipos de literais, a saber:
 - Atômicos: correspondem aos valores de tipos de dados básicos (inteiro, real, lógico, caractere, cadeia de caracteres).
 - Estruturados: correspondem aos valores do construtor de tipo tupla (*struct* na ODL).
 - De coleção: especificam valores que são coleções de objetos ou valores, e não possuem OID (*set*<*t*>, *bag*<*t*>, *list*<*t*>, *array*<*t*> e *dictionary*<*k,v*>).
 - *Dictionary*<*k,v*> corresponde a uma associação de valores <*k,v*>, na qual *k* é uma chave de pesquisa associada ao valor *v*; pode ser útil para criar um índice sobre uma coleção de valores.

Objetos Atômicos

- Na ODL, objetos atômicos são especificados por meio da palavra-chave *class*.
 - Objetos atômicos são criados como classes, especificando-se suas propriedades e operações.
- As propriedades definem o estado do objeto, sendo diferenciadas entre atributos e relacionamentos.
- As operações definem o comportamento do objeto, representando os métodos que podem ser aplicados ao objeto.

Objetos Atômicos

- Um atributo é uma propriedade que descreve algum aspecto do objeto em questão.
 - Os valores de atributos são literais (com estrutura simples ou complexa) ou OIDs de outros objetos.
- Um relacionamento é uma propriedade que especifica que dois objetos em um banco de dados estão mutuamente relacionados.
 - Somente relacionamentos binários são explicitamente representados.
 - Cada relacionamento binário é representado por um par de referências inversas, ou em apenas uma direção.

Objetos Atômicos

```

class Empregado
( extent todos_empregados
  key ssn )
{
  attribute string nome;
  attribute string ssn;
  attribute date datanascimento;
  attribute enum genero {M,F} sexo;
  attribute short idade;
  relationship Departamento trabalha_em
    inverse Departamento::possui_empregados;
  void redesignar_empregado(string novo_dnome)
    raises (dnome_invalido);
}

class Departamento
( extent todos_departamentos
  key dnome, dnumero )
{
  attribute string dnome;
  attribute short dnumero;
  attribute struct Ger_Depto {Empregado gerente, date datainicio}
    ger;
  attribute set<string> localizacoes;
  attribute struct Projetos {string nomeprojeto, time horas_semana}
    projetos;
  relationship set<Empregado> possui_empregados inverse Empregado::trabalha_em;
  void incluir_empregado(in string novo_enome) raises(enome_invalido);
  void alterar_gerente(in string novo_nome_ger; in date datainicio);
}

```

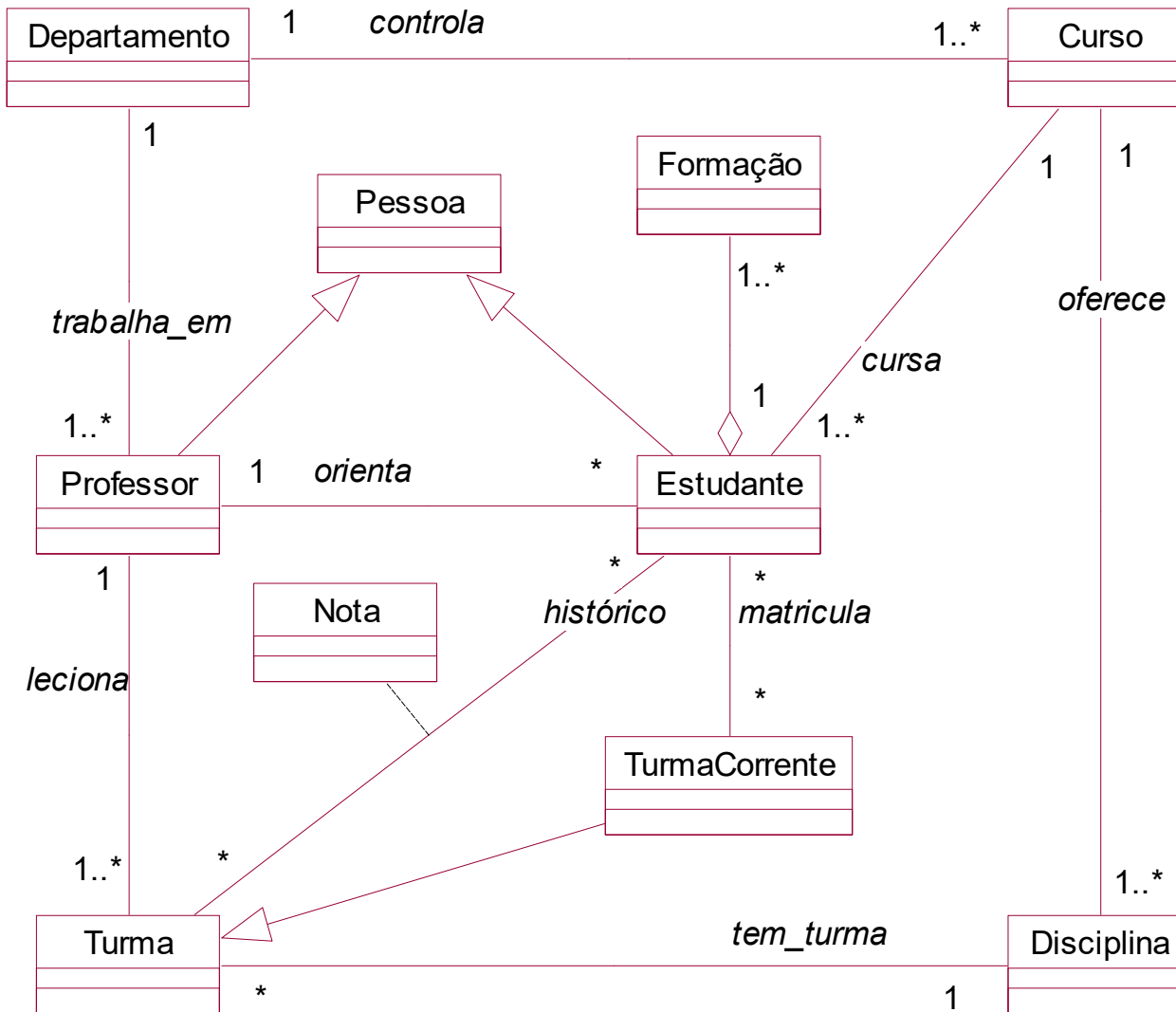
Extensões

- No modelo de dados ODMG, pode-se declarar uma extensão (palavra-chave *extent*) para qualquer tipo de objeto (*class*).
 - Um nome é atribuído à extensão, que irá conter todos os objetos persistentes da classe.
 - No exemplo apresentado, a classe Departamento possui a extensão todos_departamentos; isso é semelhante à criação de um objeto do tipo *set*<Departamento>, que mantém persistentes todos os objetos da classe.

Linguagem de Definição de Objetos (ODL)

- A *Object Definition Language* (ODL), projetada para o modelo de objetos ODMG, é utilizada para definir os tipos de objetos para uma aplicação de banco de dados.
 - É usada, então, para especificação das classes.
 - ODL corresponde à DDL de SGBDs relacionais.

ODL – Exemplo



ODL – Exemplo

```

class Pessoa
( extent pessoas
  key identidade )
{
  attribute string nome;
  attribute string identidade;
  attribute date dataNascimento;
  attribute enum Gênero {M,F} sexo;
  attribute struct Endereço {short número, string rua, string compl, string cidade,
                             string estado, string cep} endereço;

  short idade();
};

```

"pessoas" é uma extensão da classe "Pessoa", mantendo persistentes os objetos da classe

Atributos atômicos e estruturados de "Pessoa"

ODL – Exemplo

"Professor" é subclasse de "Pessoa"

```
class Professor extends Pessoa
( extent professores )
{
```

```
  attribute string cargo;
  attribute float  salário;
  attribute string sala;
  attribute string fone;
```

"trabalha_em" é um relacionamento que associa cada instância de "Professor" a uma instância de departamento (OID), sendo o inverso do relacionamento "tem_professores" de Departamento

```
  relationship Departamento trabalha_em inverse Departamento::tem_professores;
  relationship set<Estudante> orienta inverse Estudante::orientador;
  relationship set<Turma> leciona inverse Turma::professor;
  void reajustaSalario (in float percentual);
  void promove (in string novo_cargo);
```

```
};
```

"reajustaSalario" e "promove" são operações de "Professor"

ODL – Exemplo

```

class Estudante extends Pessoa
( extent estudantes )
{
    attribute string matrícula;
    relationship set<Formação> formação;
    relationship Curso cursa inverse Curso::tem_estudantes;
    relationship set<Nota> turmas_completadas inverse Nota::estudante;
    relationship set<TurmaCorrente> matriculado_em inverse
        TurmaCorrente::estudantes_matriculados;
    relationship Professor orientador inverse Professor::orienta;
    float mediaGeral();
    void matricula (in Turma t);
    void atribuiNota (in Turma t, in double nota) raises (nota_inválida);
    void atribuiOrientador (in string nome) raises (nome_inválido);
};

```

ODL – Exemplo

```
class Formação
```

```
{
```

```
  attribute string escola;
```

```
  attribute string grau;
```

```
  attribute short ano;
```

```
};
```

```
class Disciplina
```

```
( extent disciplinas key codDisc )
```

```
{
```

```
  attribute string nomeDisc;
```

```
  attribute string codDisc;
```

```
  attribute string ementa;
```

```
  attribute short numCreditos;
```

```
  attribute short cargaHorária;
```

```
  relationship set<Turma> tem_turmas inverse Turma::disciplina;
```

```
  relationship Curso oferecida_por inverse Curso::oferece;
```

```
};
```

ODL – Exemplo

```
class Turma
( extent turmas )
{
  attribute string codTurma;
  attribute short ano;
  attribute short semestre;
  relationship Disciplina disciplina inverse Disciplina::tem_turmas;
  relationship set<Nota> estudantes inverse Nota::turma;
  relationship Professor professor inverse Professor::leciona;
};
```

```
class TurmaCorrente extends Turma
( extent turmas_corrente )
{
  relationship set<Estudante> estudantes_matriculados inverse Estudante::matriculado_em;
  void matricula (in string matrícula) raises (matrícula_inválida, turma_cheia);
};
```

ODL – Exemplo

```
class Departamento
( extent departamentos key codDepto )
{
  attribute string codDepto;
  attribute string nomeDepto;
  relationship set<Professor> tem_professores inverse Professor::trabalha_em;
  relationship set<Curso> controla inverse Curso::pertence_a;
};
```

```
class Curso
( extent cursos key nomeCurso )
{
  attribute string nomeCurso;
  relationship Departamento pertence_a inverse Departamento::controla;
  relationship set<Estudante> tem_estudantes inverse Estudante::curso;
  relationship set<Disciplina> oferece inverse Disciplina::oferecida_por;
};
```

ODL – Exemplo

```
class Nota
( extent notas )
{
  attribute double nota;
  relationship Turma turma inverse Turma::estudantes;
  relationship Estudante estudante inverse Estudante::turmas_completadas;
};
```

Linguagem de Consulta a Objetos (OQL)

- A *Object Query Language* (OQL) é a linguagem proposta para o modelo de objetos ODMG.
- A sintaxe OQL é semelhante à sintaxe da SQL, com características adicionais para conceitos ODMG, como identidade de objeto, objetos complexos, relacionamentos, operações, herança e polimorfismo.

OQL – Exemplo

1. Recuperar o nome de todas as disciplinas com carga horária igual a 36 horas.

```
select d.nomeDisc  
from d in disciplinas  
where d.cargaHorária = 36;
```

- Observações:
 - Uma consulta necessita de um ponto de entrada para o banco de dados, o qual é um objeto persistente nomeado.
 - Para a maioria das consultas, o ponto de entrada é o nome de uma extensão (*extent*) da classe.
 - No exemplo acima, o objeto nomeado "disciplinas" é o ponto de entrada, cujo tipo é *set<Disciplina>*.

OQL – Exemplo

- Observações:
 - Quando uma coleção é referenciada em uma consulta, deve-se definir uma variável de iteração (*d* na consulta 1), que passa sobre cada objeto da coleção.
 - A variável de iteração do exemplo pode ser especificada de três formas:
 - *d in disciplinas*
 - *disciplinas d*
 - *disciplinas as d*
 - O tipo do resultado da consulta acima é *bag<string>*, porque o tipo de "nomeDisc" é *string*.
 - Em geral, o resultado de uma consulta será do tipo *bag* para "SELECT ... FROM" e do tipo *set* para "SELECT DISTINCT ... FROM".

OQL – Exemplo

2. Recuperar todos os cursos.

`cursos;`

- Observações:
 - No caso, basta fazer uma referência à coleção de objetos persistentes da classe "Curso".
 - O resultado é do tipo *set*<Curso>.

OQL – Exemplo

3. Criar uma visão de nome *cccurso* que recupere o objeto curso cujo nome é 'Ciência da Computação'.

```
define cccurso as  
element (select c  
          from  c in curso  
          where c.nomeCurso = 'Ciência da Computação');
```

- Observações:
 - O comando *define* cria uma visão, que se torna persistente e pode ser referenciada pelo seu nome (*cccurso*, no exemplo).
 - O operador *element* define que o resultado da consulta é um objeto atômico (elemento único) ao invés de uma coleção.
 - Se a consulta retornar mais de um elemento ou se for vazia, então o operador *element* gera uma exceção; no exemplo acima, o resultado é um único elemento pois "nomeCurso" é chave.

OQL – Exemplo

4. Usando a visão da consulta 3, recuperar o conjunto de professores que trabalham para o departamento que controla o curso de 'Ciência da Computação'.

```
cccurso.pertence_a.tem_professores;
```

- Observações:
 - O exemplo acima mostra o uso de expressão de caminho.
 - Uma expressão de caminho começa por um nome de objeto persistente ou por uma variável de iteração: o nome é seguido por zero ou mais nomes de relacionamentos, nomes de atributos ou nomes de métodos (que retornam algum valor) conectados usando a notação de ponto.

OQL – Exemplo

- Observações:
 - No exemplo, "cccurso" é o nome de um objeto persistente do tipo *Curso*; assim, em seguida pode vir qualquer nome de atributo, relacionamento ou método da classe "Curso".
 - Foi então usado o relacionamento "pertence_a", que é do tipo *Departamento*; assim, em seguida pode vir qualquer nome de atributo, relacionamento ou método da classe "Departamento".
 - Foi então usado o relacionamento "tem_professores", que é do tipo *set<Professor>*; logo, o resultado final é do tipo *set<Professor>* e consiste no conjunto de professores acessíveis a partir do objeto "cccurso" via "pertence_a" e "tem_professores".

OQL – Exemplo

5. Usando a visão da consulta 3, recuperar o nome dos professores que trabalham para o departamento que controla o curso de 'Ciência da Computação'.

```
select p.nome  
from p in cccurso.pertence_a.tem_professores;
```

- Observações:
 - Ocorre a herança de atributos, já que o atributo "nome" está definido na classe "Pessoa" e não na classe "Professor".
 - A consulta não poderia ser escrita como
cccurso.pertence_a.tem_professores.nome;
porque existe uma ambigüidade: não está claro se o resultado é do tipo set<string> ou bag<string>, já que "nome" não é chave.

OQL – Exemplo

6. Criar uma visão de nome *curso_view* que recupere o objeto curso cujo nome é fornecido como parâmetro.

```
define curso_view (nome_curso) as  
element (select c  
           from c in curso  
           where c.nomeCurso = nome_curso);
```

- Observação:
 - A visão acima pode ser usada da seguinte forma para recuperar o curso de nome 'Matemática':
curso_view (Matemática);

OQL – Exemplo

7. Recuperar o nome e a média geral de todos os estudantes orientados pelo professor 'João da Silva'.

- Usando o relacionamento "orientador" de "Estudante":

```
select e.nome, e.mediaGeral()  
from e in estudantes  
where e.orientador.nome = 'João da Silva';
```

"mediaGeral" retorna o valor da média geral das notas do estudante.

- Usando o relacionamento "orienta" de "Professor":

```
select e.nome, e.mediaGeral()  
from e in (select p.orienta  
             from p in professores  
             where p.nome = 'João da Silva');
```

OQL – Exemplo

8. Recuperar o nome, o endereço (apenas cidade e estado) e a formação de todos os estudantes do curso de 'Ciência da Computação'.

```
select struct (e.nome,  
                endereço: struct (cidade: e.endereço.cidade,  
                                   estado: e.endereço.estado),  
                formação: (select struct (grau: f.grau,  
                                           ano: f.ano,  
                                           esc: f.escola)  
                                   from f in e.formação))  
from e in estudantes  
where e.cursa.nomeCurso = 'Ciência da Computação';
```

OQL – Exemplo

9. Recuperar o nome das disciplinas cursadas pela estudante 'Maria Ferreira', e a nota obtida em cada uma, no segundo semestre de 2012, ordenado de forma decrescente pelas notas.

```
select n.turma.disciplina.nomeDisc, n.nota  
from n in notas  
where n.turma.ano = 2012 and n.turma.semestre = 2 and  
       n.estudante.nome = 'Maria Ferreira'  
order by n.nota desc;
```

- Observação:
 - A coleção retornada por uma consulta com a cláusula *order by* é do tipo *list*, pois o resultado é uma coleção ordenada.

OQL – Exemplo

10. Recuperar a maior nota obtida pela estudante 'Maria Ferreira' no segundo semestre de 2002.

```
max (select n.nota  
      from n in notas  
      where n.turma.ano = 2012 and n.turma.semestre = 2 and  
            n.estudante.nome = 'Maria Ferreira' );
```

- Observação:
 - As seguintes funções agregadas podem operar sobre uma coleção: min, max, count, sum e avg.

OQL – Exemplo

11. Recuperar o nome dos professores que orientam mais de 5 estudantes.

```
select p.nome  
from p in professores  
where count (p.orienta) > 5;
```

- Observação:
 - As funções agregadas podem ser aplicadas a qualquer coleção de tipos e podem ser usadas em qualquer parte da consulta.

OQL – Exemplo

12. Recuperar o número de professores cadastrados no sistema.

```
count (p in professores);
```

OQL – Exemplo

13. Recuperar o nome dos estudantes que já cursaram a disciplina 'Banco de Dados II'.

```
select e.nome  
from e in estudantes  
where 'Banco de Dados II' in (select t.turma.disciplina.nomeDisc  
                                from t in e.turmas_completadas);
```

OQL – Exemplo

- Observações:
 - As expressões de pertinência e quantificação retornam um valor lógico.
 - Seja v uma variável, c uma expressão de coleção, b uma expressão do tipo lógico e e um elemento do tipo dos elementos da coleção c . Tem-se que:
 - $(e \text{ in } c)$ retorna verdadeiro se e é um membro de c .
 - $(\text{for all } v \text{ in } c: b)$ retorna verdadeiro se todos elementos de c satisfazem b .
 - $(\text{exists } v \text{ in } c: b)$ retorna verdadeiro se pelo menos um elemento em c satisfaz b .

OQL – Exemplo

14. Responda: todos os estudantes do curso de 'Ciência da Computação' são orientados por professores do departamento ao qual pertence tal curso?

```
for all a in
  (select e
   from e in estudantes
   where e.cursa.nomeCurso = 'Ciência da Computação')
: a.orientador in ccurso.pertence_a.tem_professores;
```

- Observação:
 - Essa consulta retorna verdadeiro (*true*) ou falso (*false*).

OQL – Exemplo

15. Responda: algum estudante do curso de 'Ciência da Computação' tem média geral maior ou igual a 9?

```
exists a in
  (select e
   from e in estudantes
   where e.cursa.nomeCurso = 'Ciência da Computação')
: a.médiaGeral() >= 90;
```

- Observação:
 - Essa consulta retorna verdadeiro (*true*) ou falso (*false*).

OQL – Exemplo

16. Recuperar o nome do primeiro professor, considerando a ordem alfabética dos nomes dos professores.

```
first (select p.nome  
       from  p in professores  
       order by p.nome asc);
```

- Observações:
 - Em coleções listas (*list*) e vetores (*arrays*), existem operações para recuperar o i-ésimo, o primeiro e o último elemento da coleção.
 - No exemplo acima, o resultado de um SELECT usando ORDER BY é sempre uma lista; assim, o operador *first* pode ser usado para recuperar o primeiro de tal lista.

OQL – Exemplo

17. Recuperar o nome dos três primeiros professores, considerando a ordem alfabética dos nomes dos professores.

```
(select p.nome  
  from p in professores  
  order by p.nome asc) [0:2];
```

- Observação:
 - O primeiro elemento de uma lista encontra-se posição de índice 0; assim, a expressão [0:2] retorna uma lista contendo o primeiro, o segundo e o terceiro elementos do resultado de um SELECT.

OQL – Exemplo

18. Para cada curso, recuperar seu nome e seu número de estudantes.

```
select struct (nomeDoCurso, númeroDeEstudantes: count (partition))  
from e in estudantes  
group by nomeDoCurso: e.cursa.nomeCurso;
```

- Observações:
 - O resultado da consulta acima é da forma *set<struct (nomeDoCurso: string, númeroDeEstudantes: integer)>*.
 - No caso, *partition* é a palavra-chave usada para referenciar cada partição (grupo).

OQL – Exemplo

19. Para cada curso com mais de 700 estudantes, recuperar seu nome e a média geral de seus estudantes.

```
select nomeDoCurso, médiaGeraEst: avg (select p.e.médiaGera()  
                                         from p in partition)  
from e in estudantes  
group by nomeDoCurso: e.cursa.nomeCurso  
having count (partition) > 700;
```

OQL – Exemplo

- Observações:
 - A cláusula *having* pode ser usada para filtrar as partições, ou seja, para selecionar os grupos que satisfaçam a condição especificada em tal cláusula.
 - A expressão "select p.e.médiaGeral() from p in partition" retorna um *bag* de médias gerais para a partição.
 - A cláusula *from* declara uma variável de iteração *p* sobre a coleção da partição, que é do tipo *bag<struct (e: Estudante)>*.
 - A expressão *p.e.médiaGeral()* é usada para acessar a média geral de cada estudante presente na partição.