Conteúdo

- Introdução
- Orientação a Objetos e Bancos de Dados
- O Modelo Orientado a Objetos
  - O Padrão ODMG
- O Modelo Relacional Objeto
  - A linguagem SQL:1999
  - O SBBDRO Oracle
- Considerações Finais

Referências Bibliográficas

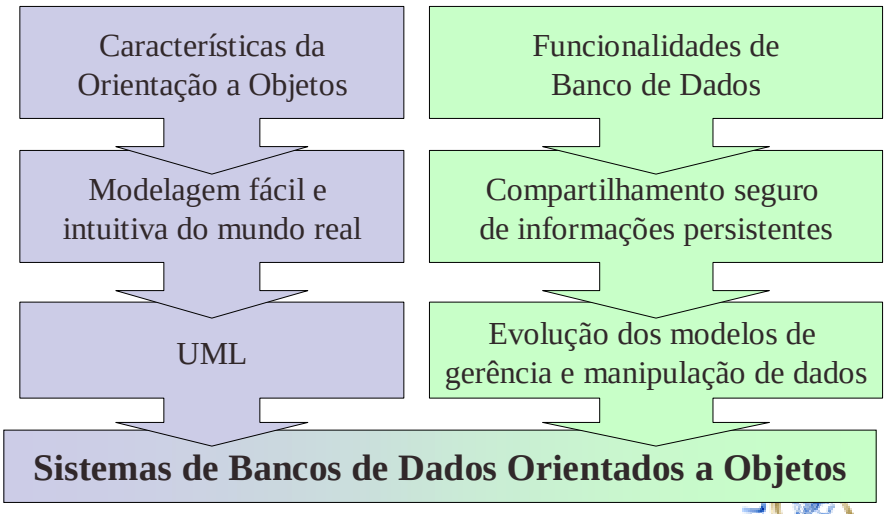
- "The Object Database Standard: ODMG 3.0"
  R. G. Cattell e D. K. Barry (editores)
  Morgan Kaufmann Publishers, 2000
- "Object Data Management"
  R. G. Cattell
  Addison-Wesley, 1994
- "UML Distilled: Applying the Standard Object Modeling Language"
  M. Fowler e K. Scott
  Addison Wesley, 2000, 2a edição
- "Database System Concepts"
  A. Silberschatz, H. Korth, e S. Sudarshan
  Mc-Graw-Hill, 2002, 4a edição

Referências Bibliográficas

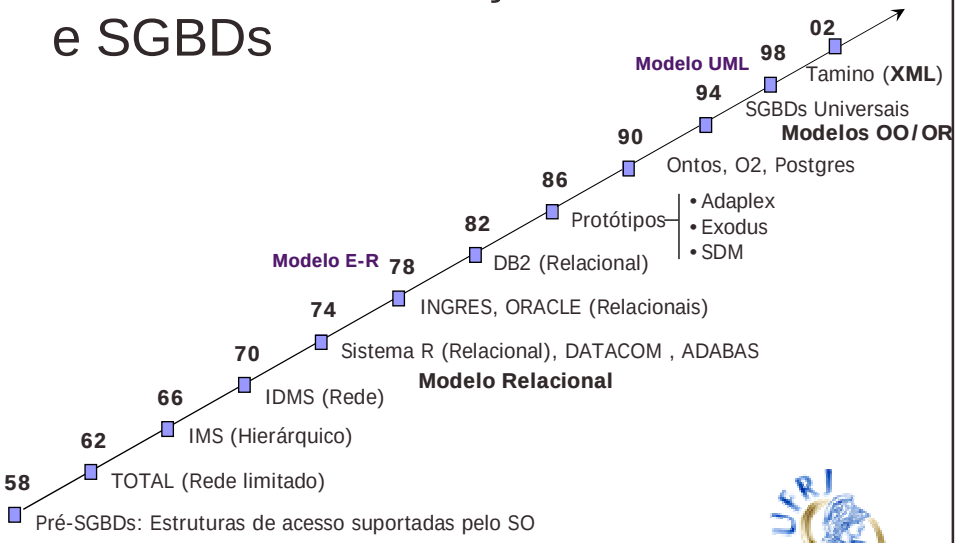
- "Object Databases: An ODMG Approach"
  R. Cooper
  International Thomson Computer Press (edição eletrônica), 1997
- "Object-Relational DBMSs: The Next Great Wave"
  M. Stonebraker, D. Moore
  Morgan Kaufmann, 1996
- "The BUCKY Object-Relational Benchmark"
  M.Carey, D. DeWitt, J.Naughton et al.
  Relatório Tecnico- U.Wisconsin (http://www.cs.wisc.edu/~naughton/bucky.html)
- "From UML to ODMG: Modeling and Implementing Object Oriented Database Applications Based on Standards",
  R.C. Mauro, M.L.Q. Mattoso
  Tutorial XIV SBBD (http://www.cos.ufrj.br/~marta)
- "Processamento de Consultas Orientadas a Objetos "
  Mattoso, M.L.Q. Ruberg, G. Victor, A. Baião, F.
  Relatório Tecnico- COPPE ES-547/01 (http://www.cos.ufrj.br)

Orientação a Objetos e Bancos de Dados

Motivação



Histórico de Evolução dos Modelos e SGBDs



Porque adotar o modelo lógico de dados OO?

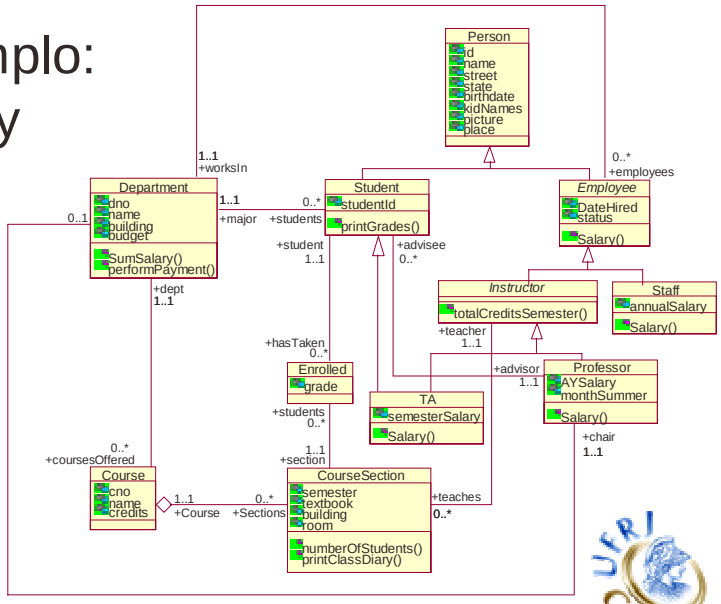
- Modelagem e programação OO cada vez mais utilizadas na prática
  - Padrão UML
- Naturalidade do modelo OO para persistência
- Requisitos de novas aplicações
  - Restrições complexas
  - Estruturas de dados complexas
  - Identidade de objetos e referências diretas
  - Código de aplicação interno ao banco de dados

O Modelo Orientado a Objetos

Conceitos do modelo de dados OO

- Objetos
  - Encapsulamento
- Classes
  - Atributos
  - Métodos
- Relacionamentos
  - Herança
  - Associação
  - Agregação
- Identidade de Objetos

Exemplo: Bucky



Objetos

- Encapsulamento de dados e de código
  - conjunto de variáveis que armazenam o estado do objeto
  - conjunto de mensagens às quais o objeto responde
  - conjunto de métodos contendo código de programa que implementa uma mensagem
- Objetos se comunicam via mensagens

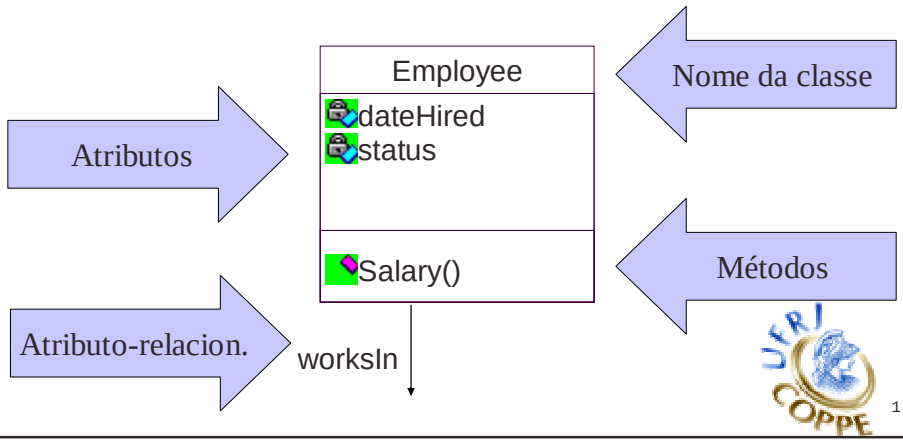
Classes

- Agrupa objetos de um mesmo tipo
  - instâncias da classe
- Define conjunto de atributos e de métodos

```
class employee {
        date            dateHired;
        string          status;
        Department      worksIn;

        int Salary();
};
```

Representação de Classes

Classe = Atributos + Métodos



Relacionamentos

- A UML permite a representação explícita de três tipos de relacionamentos entre classes
  - Herança
  - Associação
  - Composição / Agregação

Relacionamentos

- A UML permite a representação explícita de três tipos de relacionamentos entre classes
  - Herança
  - Associação
  - Composição / Agregação

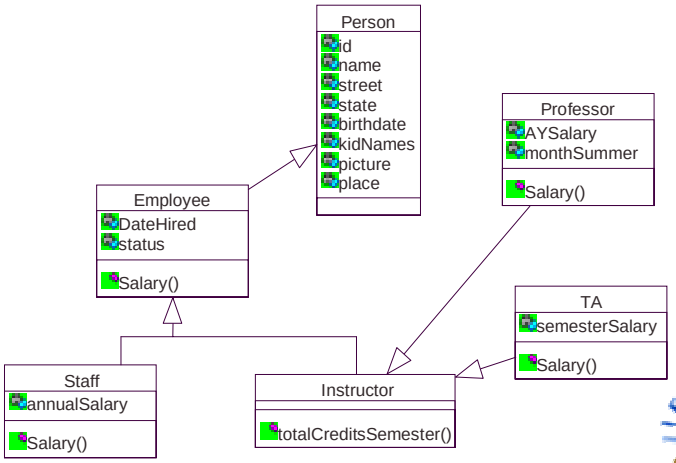
Herança

- Conceito
- Representação da Herança
- Classes Abstratas
- Polimorfismo
- Propriedade da Substituição
- Coleções Polimórficas

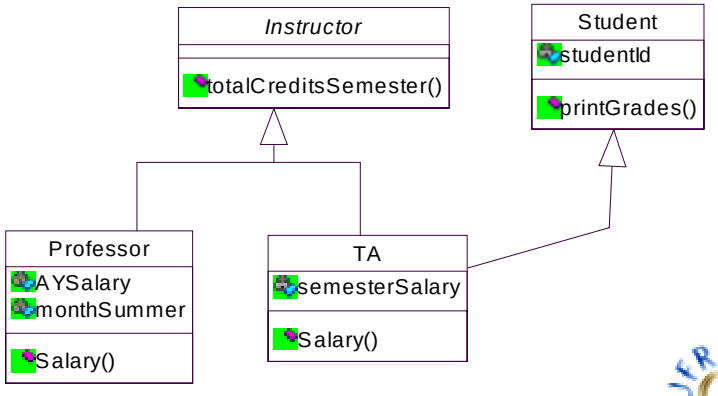
Conceito de Herança

- Representação única da estrutura em comum
- Classes são dispostas de forma hierárquica
  - relacionamento “IS-A”
  - superclasse x subclasse
- Classes mais especializadas (subclasses) “herdam” as propriedades (atributos e métodos) das suas super-classes
- Métodos de uma classe podem ser chamados para objetos de qualquer uma das suas subclasses

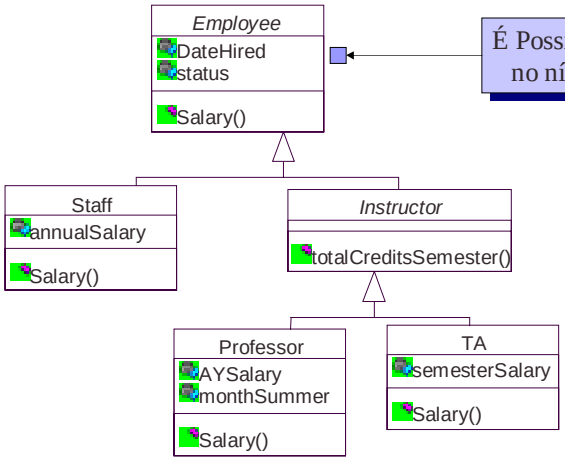
Representação de Herança



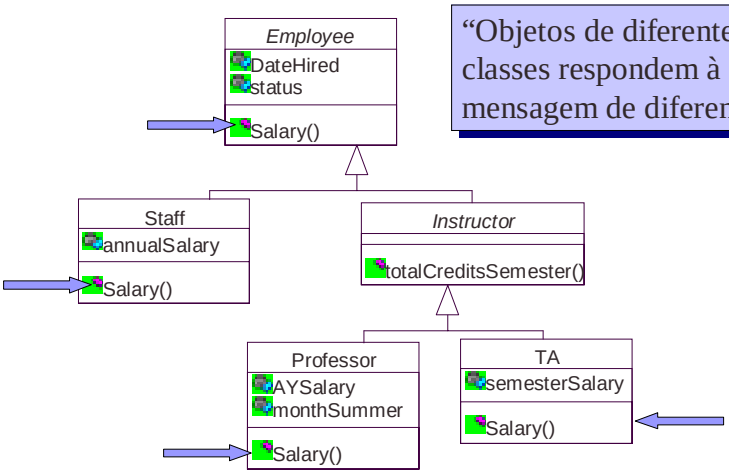
Herança Múltipla



Classe Abstrata

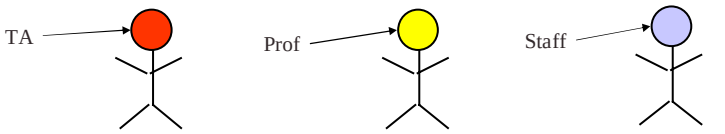


Polimorfismo



Propriedade da Substituição

```
Employee e; TA ta; Prof pr; Staff st;
... // instanciação das variáveis
e = ta;
e.Salary();
e = pr;
e.Salary();
e = st;
e. Salary();
```

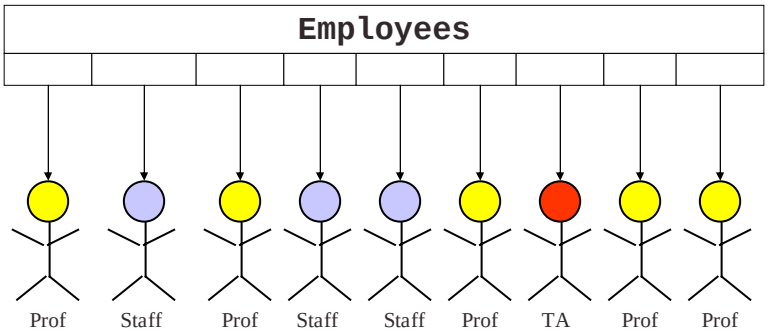


double Salary( )
{**return**
apptFrac*2*semesterSalary)/9.0;
};

double Salary( )
{**return**
AYSalary*(9+monthSummer)/9.0;
};

double Salary( )
{**return**
annualSalary;
};

Coleções Polimórficas

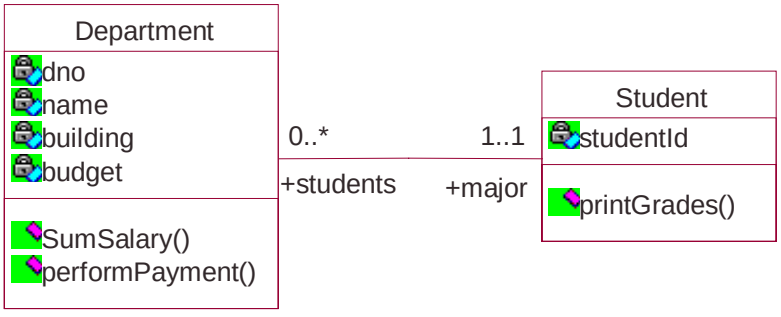



```
for( int i = 0; i < Employees.length; i++ ){
  e = Employees[i];
  System.out.println( e.name );
  System.out.println( e.Salary() );
}
```

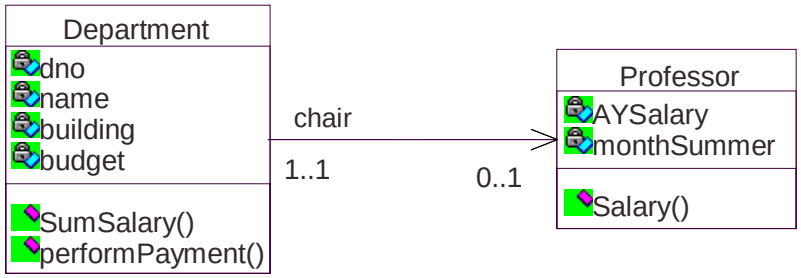
Relacionamentos

- A UML permite a representação explícita de três tipos de relacionamentos entre classes
  - Herança
  - Associação
  - Composição / Agregação

Associação Simples



Associação Unidirecional

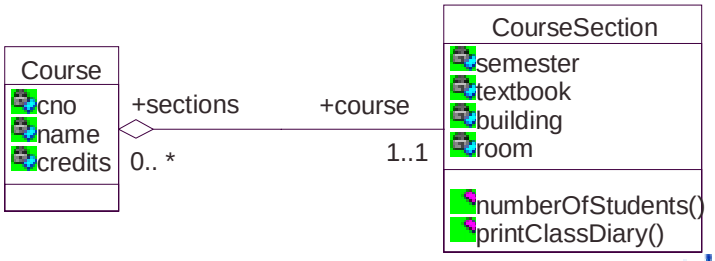


Relacionamentos

- A UML permite a representação explícita de três tipos de relacionamentos entre classes
  - Herança
  - Associação
  - Composição / Agregação

Agregação

- Objetos complexos ou compostos
  - relacionamento “IS-PART-OF”
  - diversos níveis de granularidade



Objetos compostos

- Objetos podem ser agregados para formar objetos compostos
  - Ex: Capítulos podem ser agrupados para formar um livro
- Agrupamento pode ocorrer em diversos níveis
  - Ex: Parágrafos formam uma seção, seções formam um capítulo ...
- Possibilidade de prover facilidades para cópia, remoção, armazenamento contíguo ...

Identidade de Objetos

- Cada objeto possui uma identidade independente do seu estado
- O estado pode ser modificado sem mudar a identidade
- Idênticos e Iguais são dois conceitos diferentes (profundidade)
- Conceito de chave deve ser preservado

O Padrão ODMG

ODMG - Object Database Management Group

- Grupo formado pelos principais fabricantes de banco de dados OO além de um grande número de empresas interessadas num padrão para SGBDOO.
- ODMG 3.0
  - "The Object Database Standard: ODMG 3.0", Cattell et al. (ed), Morgan Kaufmann Publishers, 2000
- Java Binding
  - Base para a especificação do Java Data Objects (JDO)
- Sintonia com outros padrões
  - OMG, SQL:99
  - XML: linguagem de especificação de objetos ODMG baseada em XML (OIFML)
- http://www.odmg.org

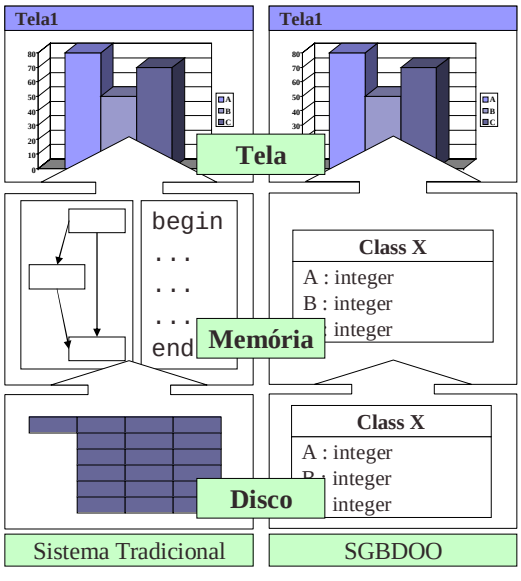
Banco de Dados OO

- GOA (COPPE/UFRJ)
- Caché
- db4o
- Javera
- Jasmine - CA
- JDBCStore
- Jodad
- Jevan - W3Apps
- JYD
- Objectivity
- ObjectStore - eXcelon
- UniObjects - Ardent
- Poet
- Versant
- Vortex
- O2

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
  - Aplicações portáveis

ODMG - Arquitetura



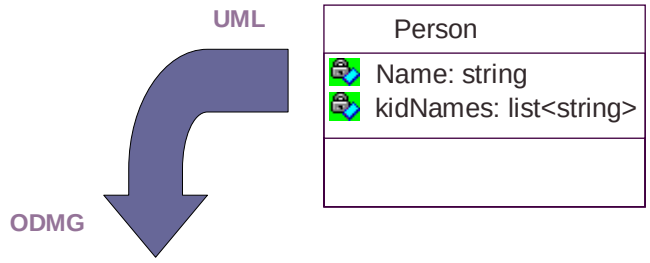
ODMG – define padrões:

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

Modelo de Objetos - ODL

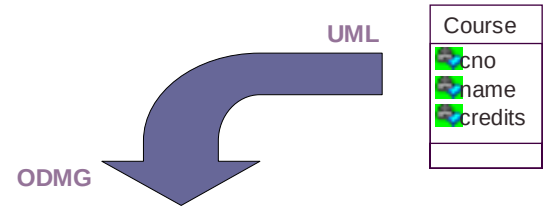
- Uma interface (type) pode ter várias implementações (classes)
  - opcionais: extensão de classe, chaves
- **Objetos** (classes) x **Literais** (valores)
- Atributos
  - Simples (Atômico)
  - Estruturado (set, bag, list, array, struct)
- Relacionamentos
  - herança múltipla (ISA, EXTENDS)

Mapeamento de interfaces



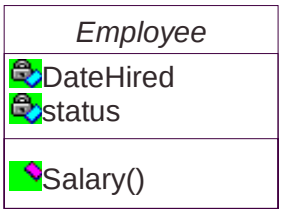

```
interface Person
{
  attribute string name;
  attribute list<string> kidNames;
};
```


Mapeamento de classes




```
class Course
(extent courses, key cno)
{
   attribute short cno;
   attribute string name;
   attribute short credits;
};
```

Mapeamento de classe abstrata




```
interface Employee
{
  attribute date DateHired;
  attribute short status;
  double Salary ();
};
```

ODL - Definição de Classes (estrutura)

```
class Course
(  extent courses, key cno )
{
  attribute    string     name;
  attribute    short      cno;
  attribute    short      credits;

  relationship Department dep
       inverse Department::coursesOfferred;

  relationship list<CourseSection> section
       inverse CourseSection::course;
}
```

Atributos

- Simples
- Chave
- Complexos
  - Referência
  - Coleção
  - Derivados

```
class Course
      name: string,
      cno: integer
      dept : Department
      sections: list[CourseSection],
      ... ?
```

Atributos simples

- Tipos básicos pré-existentes
  - integer, string...
- Tipos definidos pelo usuário através da especificação da representação e operações comuns

Dependendo da implementação, os tipos básicos podem ser tratados sintaticamente e semanticamente como objetos. Esta abordagem porém não é vantajosa quanto a eficiência.

Atributos complexos

- Referências (relacionamento)
  - Não podem ser corrompidas. A referência é invalidada automaticamente quando o objeto referenciado é apagado.
  - Independente dos valores do objeto referenciado. (Identidade)

Atributos complexos

- Coleções
  - listas
  - conjuntos
  - vetores
- Primeira forma normal é violada
- Possibilidade de estabelecer uma ordem entre os elementos

Atributos complexos

- Atributos derivados (Procedimento)

  Preço total = Quantidade * Preço Unitário

  - Atributos virtuais
    - Sintaxe de acesso a atributo e a procedimento devem ser iguais
      - POSTGRES, O_2
  - Atualização de atributos derivados
    - procedimentos *get* e *set*

Relacionamentos

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

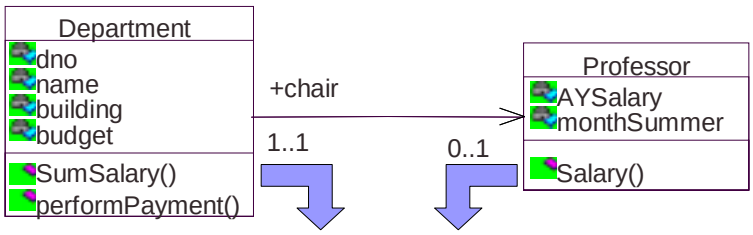
Relacionamentos

Associação (UML) → Relacionamento (ODMG)

- Binária Unidirecional
  - atributo de referência
- Binária bi-direcional
  - relacionamento
- Binária com atributo, N-ária, Classe
  - classe relacionamento

Relacionamentos

Associação Unidirecional

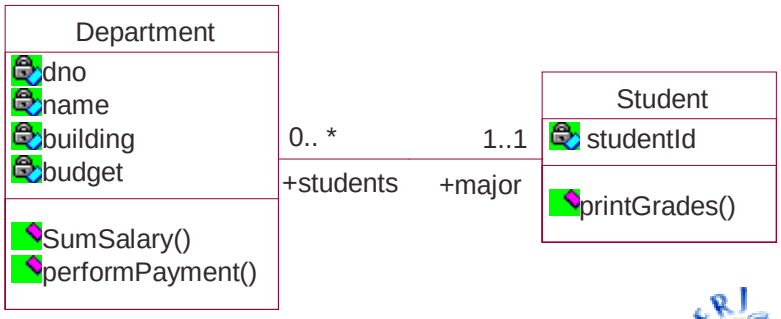



```
class Department
(extent departments, key dno) {
  attribute short dno;
  attribute string name;
  attribute Professor chair;
…  };
```

```
class Professor
(extent professors){
  attribute short AYSalary;
  attribute short monthSummer;
  double Salary()
};
```

Relacionamentos

Associação Bidirecional



Relacionamentos

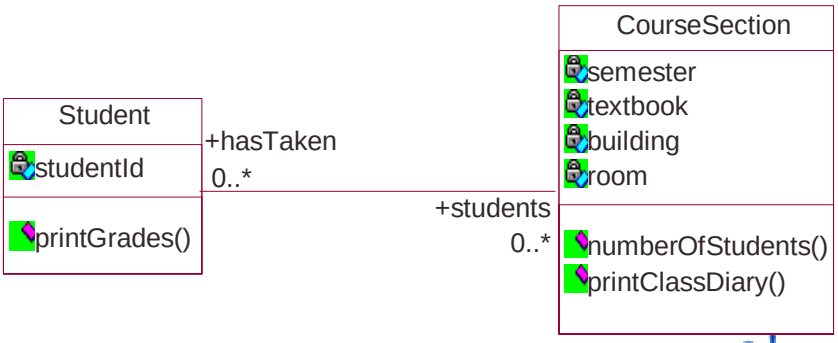
Associação Bidirecional → Atributo Inverso

```
class Department
(extent departments,
 key dno)
{
attribute short dno;
attribute string name;
attribute string building;
attribute string budget;

relationship set
<Student>students
  inverse Student::major;

double sumSalary();
void performPayment(); };
```

```
class Student
(extent students,
 key studentId)
{
attribute short studentId;

relationship
<Department> major
  inverse Department
      ::students;

void printGrades();
};
```

Relacionamentos

Associação Binária N x M



Relacionamentos

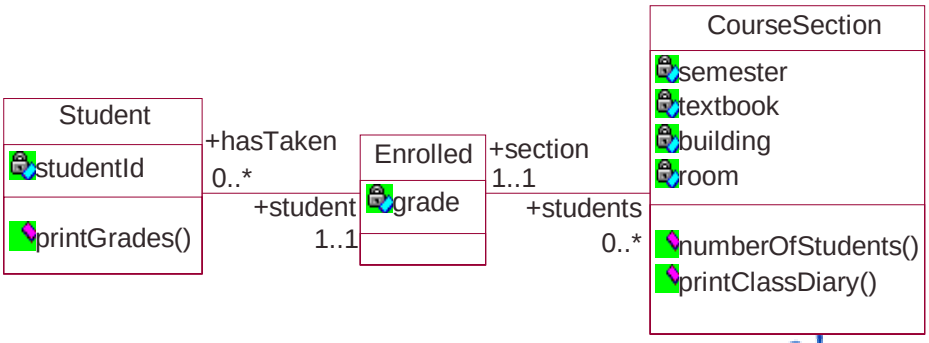
Associação Binária N x M → Atributo Inverso

```
class Student
(extent students,
 key studentId)
{
attribute short studentId;
relationship set
  <CourseSection > hasTaken
  inverse
      CourseSection::students;
...};
```

```
class CourseSection
(extent coursesections)
{
attribute short semester;
attribute string textbook;
attribute string building;
attribute short roomNo;
attribute short noStudents;
relationship set
  <Student> students
  inverse
       Student::hasTaken;
 ...};
```

Relacionamentos

Associação Binária N x M



Relacionamentos

Associação Binária com Atributo ...

```
class Student
(extent students,
 key studentId)
{
attribute short studentId;
relationship set
  <Enrolled> hasTaken
  inverse
    Enrolled::students;
void printGrades();
};
```

```
class CourseSection
(extent coursesections)
{
attribute short semester;
attribute string textbook;
attribute string building;
attribute short roomNo;
attribute short noStudents;
relationship set
  <Enrolled> students
  inverse
    Enrolled::section;
 … };
```

Relacionamentos

Associação Binária com Atributo (cont.)

```
class Enrolled
(extent enrolleds)
{attribute real grade;
relationship <Student> student
  inverse Student::hasTaken;
relationship <CourseSection> section
  inverse CourseSection ::students;
```

Relacionamentos

Associação Binária com Atributo (ex. O2 ¬ODMG)

```
class CourseSection
(extent coursesections)
{
attribute short semester;
attribute string textbook;
attribute string building;
attribute short roomNo;
attribute short noStudents;
relationship set
  < Student > students
  inverse
    Student::section;
 … };
```

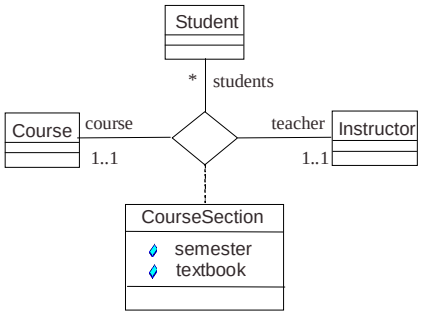
```
class Student
(extent students,
 key studentId)
{
attribute short studentId;
relationship set struct
( <CourseSection > hasTaken,
  real grade
) section
  inverse
    CourseSection::students;
void printGrades();
};
```

Relacionamentos

- Relacionamento não binário
  - Há necessidade de criação de uma classe específica para expressar o relacionamento
  - A notação "." reduz os inconvenientes da nova classe

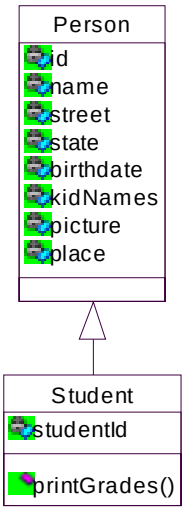
Relacionamentos

Associação Ternária



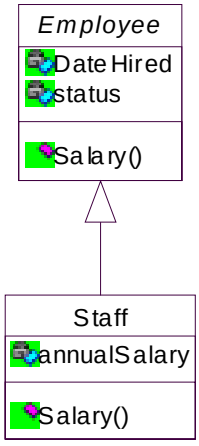

```
class Course { ... } ;
class Student { ... } ;
class Instructor { ... }

class CourseSection
(extent courseSections
 key(course,students,teacher))
{
relationship Course course;
relationship set<Student>students;
relationship Instructor teacher;
attribute integer semester;
attribute string textbook;
... };
```

Herança Simples




```
class Person
( extent people, key id )
{
  attribute int id;
  attribute string name;
...
}

class Student extends Person
( extent students, key studentId )
{
  attribute int studentId;
}
```

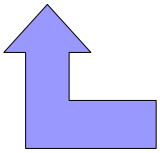
Herança simples com Interface

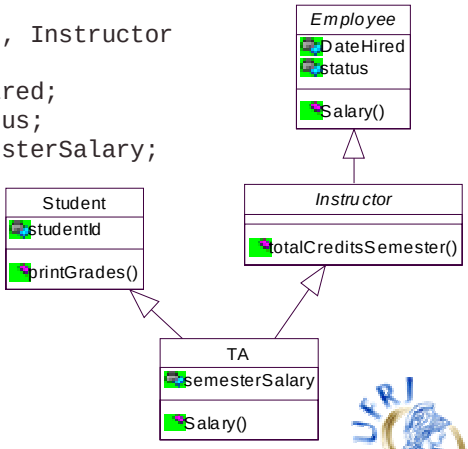



```
interface Employee
{
  attribute date DateHired;
  attribute short status;
  double Salary ();
};

class Staff : Employee
{
  attribute date DateHired;
  attribute short status;
  attribute double annualSalary;
  double Salary ();
}
```

Herança Múltipla

```
class TA extends Student, Instructor
{
   attribute date DateHired;
   attribute string status;
   attribute double semesterSalary;

   relationship ...

   double Salary();
}
```





ODL - Definição de Classes (operações)

```
class Professor extends Instructor
(extent professors)
{
  attribute short AYSalary;
  attribute short monthSummer;

  double Salary()
  {
  return AYSalary*(9+monthSummer)/9.0;
  }
} ;
```

OQL, O-R SQL : Definição de Consultas

- acesso associativo
- expressões de caminho
- herança
- métodos
- polimorfismo
- pertinência de conjuntos

select **resultado**
from **operando**
[where **predicado**]

OQL , O-R SQL : Sintaxe

select **c** ← Resultado
from **courses c** ← Operando
where **c.dept.chair.state = "AM"**; ← Predicado

- **Resultado**: objetos, literais
- **Operando**: coleções (extent)
- **Predicado**: expressões de caminho

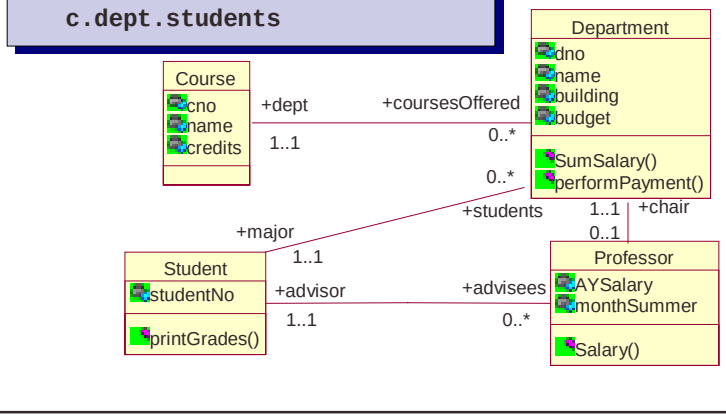
OQL, O-R SQL : Consultas

select **resultado**
from **operando**
[where **predicado**]

- **resultado** representa uma das variáveis (ou combinação) presentes em **operando** e identifica a origem da lista de objetos/valores que será fornecida como resultado da consulta
- **operando** da consulta consiste de expressões do tipo **coleção v** ,onde **v** representa os objetos em **coleção**
- **predicado** é o conjunto de cláusulas que representam as condições que devem ser satisfeitas pelas variáveis de lista de variáveis

Expressões de Caminho

```
courses c
   c.dept.chair.name
   c.dept.chair.advisees
   c.dept.students
```



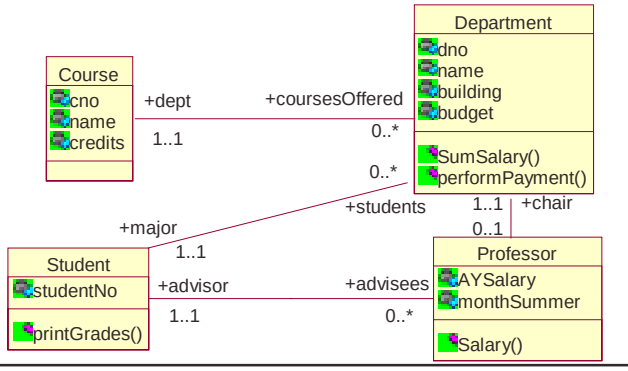
Expressões de Caminho

OQL

```
select c.name
  from courses c
 where c.dept.chair.state = "AM";
```

SQL

```
select c.name
  from Course c, Department d,
       Professor p
 where c.dept = d.dno
   and d.chair = p.id
   and p.state = "AM";
```



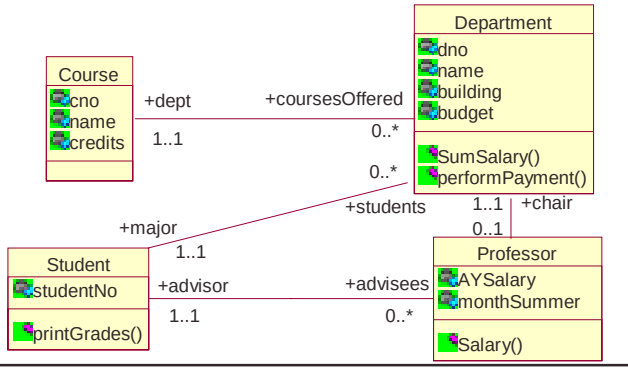
Expressões de Caminho

OQL

```
select c
  from courses c,
       c.dept.students s
 where s.city = "Manaus";
```

SQL

```
select c.*
  from Course c,
       Department d, Student s
 where c.dept = d.dno
   and d.dno = s.majors
   and s.city = "Manaus";
```



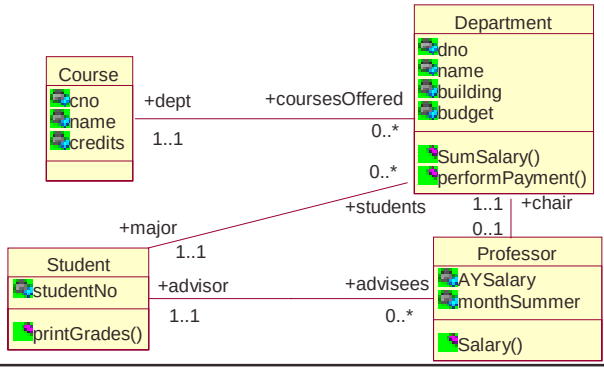
Expressões de Caminho

OQL

```
select struct (dept:d, std:s.name)
  from departments d,
       d.chair.advisees s
 where d.chair.name = "Altigran";
```

SQL

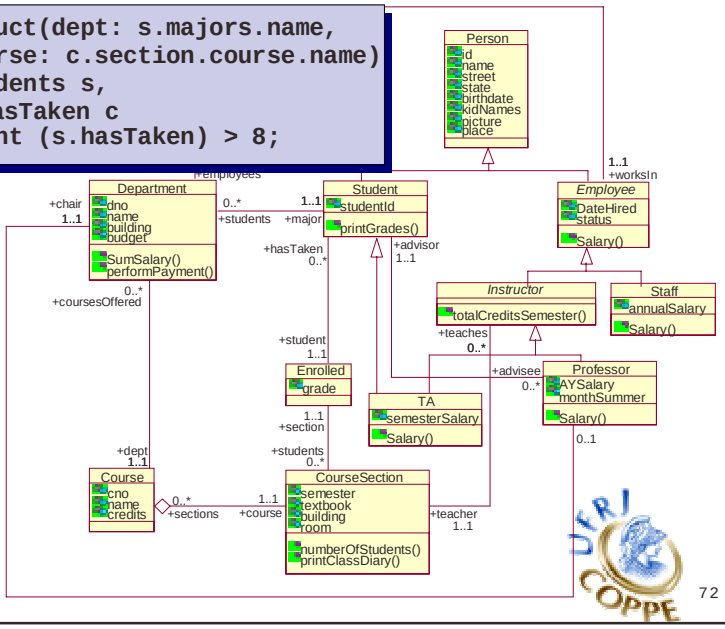
```
select d.*, s.name
  from Department d,
       Professor p, Student s
 where d.chair = p.id
   and s.advisor = p.id
   and p.name = "Altigran";
```



Expressão de Caminho no resultado

OQL

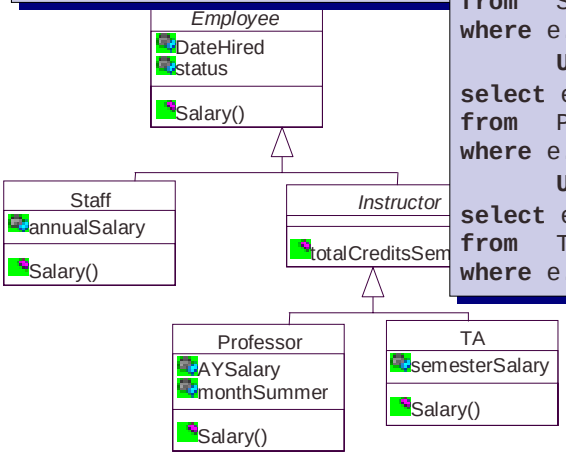
```
select struct(dept: s.majors.name,
       course: c.section.course.name)
  from students s,
       s.hasTaken c
 where count (s.hasTaken) > 8;
```



Herança

OQL

```
select e.name, e.street, e.zip
from   employees e
where e.DateHired > 2000;
```



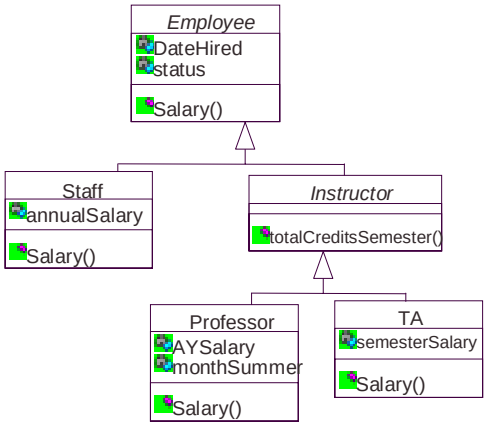
SQL

```
select e.name, e.street, e.zip
from   Staff e,
where e.DateHired > 2000;
       UNION ALL
select e.name, e.street, e.zip
from   Professors e
where e.DateHired > 2000;
       UNION ALL
select e.name, e.street, e.zip
from   TA e
where e.DateHired > 2000;
```


Herança e associação

OQL

```
select e.name, e.street, e.zip
  from employees e
 where e.DateHired > 2000
   and e.worksin.budget > 10k;
```



SQL

```
select e.name, e.street, e.zip
  from Staff e, Department d
 where e.DateHired > 2000
   and d. budget > 10k
   and e.worksin = d.deptNo;
       UNION ALL
select e.name, e.street, e.zip
  from Staff e, Department d
 where e.DateHired > 2000
   and d. budget > 10k
   and e.worksin = d.deptNo;
       UNION ALL
select e.name, e.street, e.zip
  from Staff e, Department d
 where e.DateHired > 2000
   and d. budget > 10k
   and e.worksin = d.deptNo;
```

Polimorfismo

OQL

```
select x.name, x.salary
  from employees x
 where x.salary >= 96000;
```

SQL

```
select x.name, x.salary
  from Staff x
 where x.annualSalary >= 96000
       union all
select x.name, x.salary
  from Professor x
 where (x.salary*(9+x.monthSummer)/9.0) >= 96000
       union all
select x.name, x.salary
  from TA x
 where (apptFraction*(2*x.salary)) >= 96000
```

Pertinência de conjuntos

OQL

```
select x.name, x.salary
  from staffs x
 where “Maria” IN x.kidNames;
```

SQL

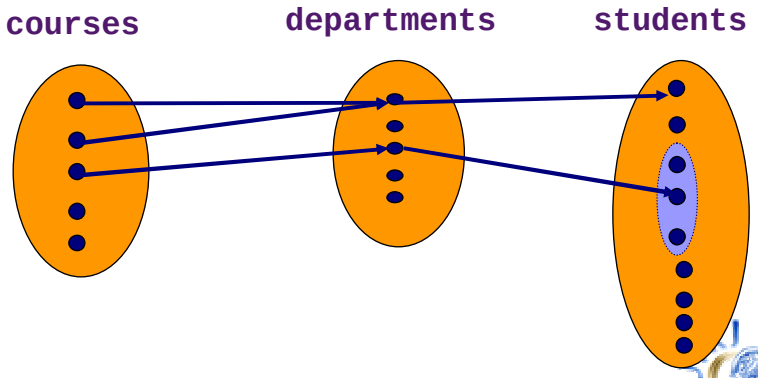
```
select x.name, x.salary
  from Staff x, Kids k
 where x.id = k.id
   and k.kidName = “Maria”
```

Estratégias de Processamento

- Modelo de objetos possibilita novas estratégias
  - Direção
    - descendente x ascendente (atributos inversos)
  - Operador
    - junção x referência (ponteiros)
- Grande aumento de desempenho

Estratégias de Processamento

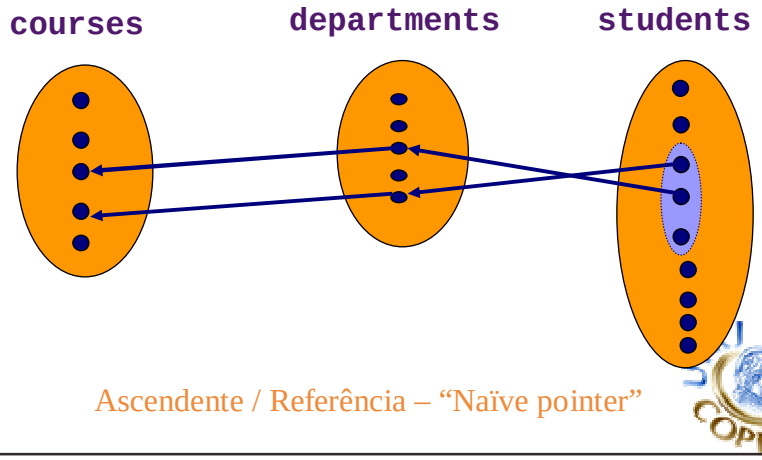
```
select c
  from courses c, c.dept.students s
 where s.city = "Manaus";
```



Descendente / Referência – "Naïve pointer"

Estratégias de Processamento

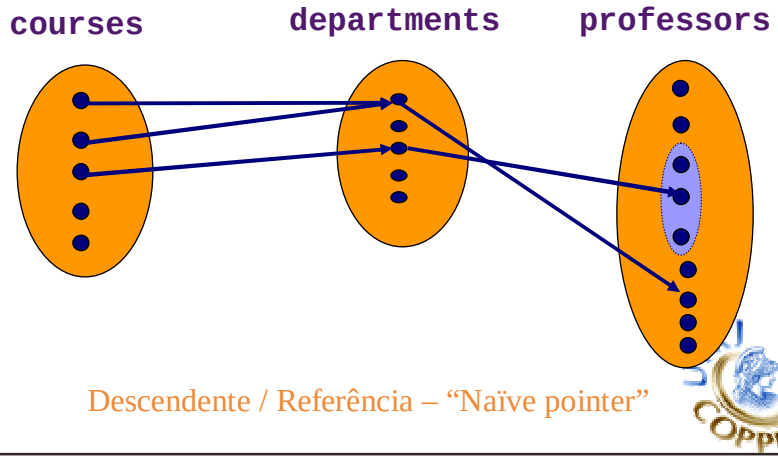
```
select c
  from courses c, c.dept.students s
 where s.city = "Manaus";
```



Ascendente / Referência – "Naïve pointer"

Estratégias de Processamento

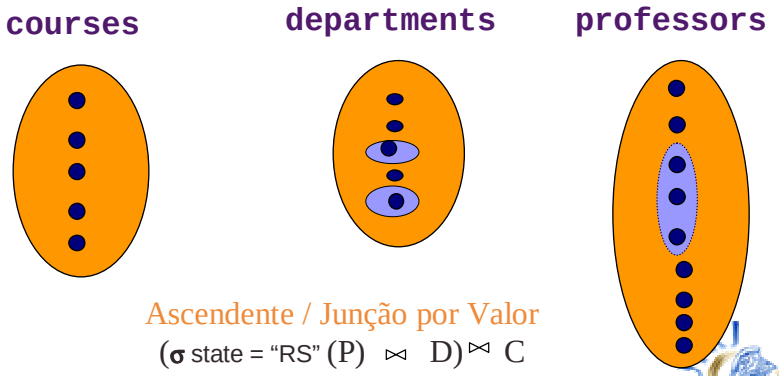
```
select c.name
  from courses c
 where c.dept.chair.state = "AM";
```



Descendente / Referência – "Naïve pointer"

Estratégias de Processamento

```
select c.name
  from courses c
 where c.dept.chair.state = "AM";
```



Ascendente / Junção por Valor

(σ state = "RS" (P) $\bowtie$ D) $\bowtie$ C

- Gerenciador de Objetos baseado no padrão ODMG
  - Processadores OQL / ODL
- OQL estendida com primitivas para mineração de dados em bases de objetos
- Processamento Paralelo de Consultas
- Interface com a linguagem Java
- Armazenamento de documentos XML

www.cos.ufrj.br/~goa

O Modelo de dados Relacional Objeto

Modelo Relacional Objeto

Relacional + Objeto

- Extensão do modelo relacional tradicional
  - Utilização da tecnologia e otimizações existentes
    - Suporte à SQL, gerência de transações, processamento e otimização de consultas, etc...
    - Migração gradual e transparente de sistemas legados
  - Sistema de tipos mais rico - Tipos de dados complexos
  - Manipulação de objetos pelo usuário
  - Extensão da linguagem SQL
    - SQL:1999, SQL:200n...

Modelo Relacional Objeto

- Resposta dos Bancos de Dados Relacionais à Orientação a Objetos
- Migração transparente
- Incorpora novas funcionalidades e capacidade de modelagem para tratar dados complexos (objetos) sobre estruturas físicas relacionais (tabelas)
  - Representações distintas em memória e no disco
  - "Gap semântico"

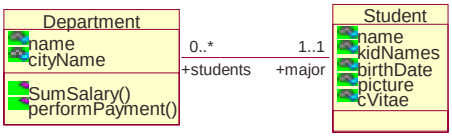
Elementos do Modelo Relacional Objeto

- Relações Aninhadas
- Tipos Complexos
  - Coleções e Objetos longos (Large Objects – LOBs)
    - Documentos XML
  - Tipos Estruturados
- Herança
  - Tipos, Tabelas
- Tipo Referência
- Consultas
  - Expressões de caminho
- Funções e Procedimentos

Relações Aninhadas

- Atributos atômicos (tradicional) ou relações
  - Modelagem mais natural das aplicações
    - Coleções, tipos estruturados
  - Modelagem mais fácil de entender
  - Relações dentro de relações
  - Fisicamente, ainda são tabelas distintas

Relações Aninhadas

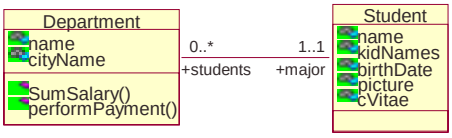


Atributos multivalorados (coleções)

Departments

name	cityName	students	...
Computer Science	New York	{David Dewitt, Eddie Smith}	
Biology	San Diego	{Susan Smith}	
Mathematics	New York	{Jonh Walsh, George Gold}	

Relações Aninhadas



Atributos multivalorados (coleções)

Tipos Estruturados

Students

name	kidNames	major	...
David Dewitt	{David, Mary, John}	(Computer Science, New York)	
Susan Smith	{Carol, Steve}	(Biology, San Diego)	
Jonh Walsh	{Emily, Mary}	(Mathematics, New York)	

Tipos Complexos

- Coleções
  - Conjuntos (sets), vetores (arrays) e multiconjuntos (multisets)
    - Apenas vetores no padrão SQL:1999
  - Representação direta de atributos multi-valorados presentes na modelagem da aplicação

```
create table Students(
...
   kidNames varchar(20) array[10]
...
)
```

Tipos Complexos

- Objetos Longos
  - Fotografias, imagens médicas de alta resolução, vídeos
  - Representação direta de objetos da aplicação, armazenados na base de dados
    - não em arquivos soltos no disco
  - tipos de dados para objetos longos no padrão SQL:1999
    - Clob (caracteres), blob (binários)
    - Armazenamento/publicação de dados XML

```
create table Students(
...
   cVitae clob(10KB)
   picture blob(10MB)
...
)
```

Tipos Complexos

- Tipos Estruturados
  - Atributos atômicos
  - Atributos compostos

```
create type Department as(
   name varchar(20),
   cityName varchar(20))

create type Student as(
   name varchar(20),
   kidNames varchar(20) array[10]
   birthDate date,
   major Department)

create table Students of Student
```

Tipos Complexos

- Tipos Estruturados
  - Métodos
    - corpo definido separadamente

```
create type Professor as(
   name varchar(20),
   AYsalary integer )
method giveraise(percent integer)

create method giveraise(percent integer) for Professor
begin
   set self.AYsalary = self.AYsalary +
                       (self.AYsalary * percent)/100;
end
```

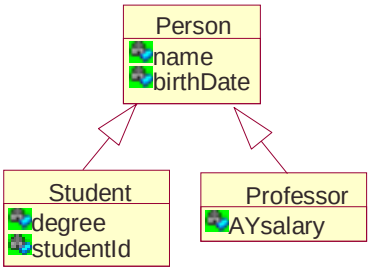

Tipos Complexos

- Tipos Estruturados
  - Valores de tipos estruturados são criados através de funções construtoras
    - ≠ métodos construtores da OO, que criam objetos

```
create function Department( n varchar(20), b varchar(20))
   returns Department
begin
   set name = n;
   set cityName = b;
end

insert into Students values
('Sarah', array['Dave','Linda'], '17-oct-1970',
   Department('computer Science', 'San Diego'))
```

Herança - de Tipos

- Relacionamento supertipo/subtipo
- Atributos e métodos herdados dos supertipos
  - Polimorfismo
- Apenas herança simples (múltipla não é suportada)




```
create type Person(
   name varchar(20),
   birthDate date)

create type Student under Person(
   degree varchar(20),
   studentId varchar(20))

create type Professor under Person(
   AYsalary integer)
```

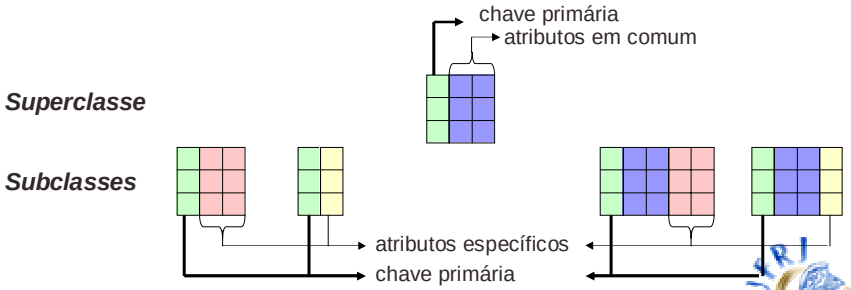

Herança - de Tabelas

- Especialização/generalização do modelo E-R
- Tipos das tabelas filhas devem ser sub-tipos da tabela pai
- Todas as tuplas das tabelas filhas estão implicitamente presentes na tabela pai
  - Consultas à tabela *People* (do tipo *Person*) retornam tuplas das tabelas *People*, *Students* e *Professors*
  - "**only** People" permite consultas apenas à tabela People

```
create table People of Person

create table Students of Student under People

create table Professors of Professors under Person
```

Herança - de Tabelas

- Diferentes alternativas de armazenamento das subtabelas, para aumentar eficiência
  - Atributos em comum apenas na tabela pai
  - Atributos em comum replicados nas tabelas filhas



Tipo Referência

- É um ponteiro lógico para um objeto de um tipo
- Modelam relacionamentos de associação entre objetos evitando o uso de chaves estrangeiras

```
create type Department(
   name varchar(20),
   cityName varchar(20,
   chair ref(Professor) scope Professors)
create table Departments of Department
insert into Departments values ('Geology', 'San Diego', null)
update Departments
   set chair = ( select ref(p) from Professors as p
                 where name = 'John')
   where name = 'Geology'
```

Consultas

- Expressões de caminho (EC)
  - “desreferenciando” atributos do tipo referência
  - Provêm um fácil e intuitivo mecanismo de navegação entre os objetos
  - simplificam consultas
    - “escondem” do usuário as operações de junção
  - Algoritmos de processamento de EC, em geral, permanecem os mesmos do modelo relacional
    - Junções baseadas em valor

```
select chair->name from Departments

select c from courses c
where c>dept>chair.state = “SC”;
```

Funções, Procedimentos e Métodos

- Padrão SQL:1999 permite manipulação de código de programa em 3 tipos:
  - Funções
  - Procedimentos
  - Métodos
    - Funções associadas a tipos, variável **self**
- Linguagem de programação
  - “Bindings” para Java, C, C++
  - PL/SQL (Oracle), TransactSQL (MS SQL Server)

Funções, Procedimentos e Métodos

```
create function deptsCountInCity(dname varchar(20))
   returns integer
begin
   declare num integer;
   select count(d) into num from Departments where d.name =
   dname;
   return num;
end

create method giveraise(percent integer) for Professor
begin
    set self.AYsalary = self.AYsalary +
              (self.AYsalary * percent)/100;
end
```

O SBBDRO Oracle

ORACLE 9i

- SGBD Relacional Objeto compatível com padrão SQL:1999
- API para C++ seguindo especificação padrão da ODMG
- Recursos
  - Tipo Objeto
  - Tipo REF
  - Visão de Objetos
  - Coleções
  - ... (herança, métodos, ...)
  - Armazenamento dados XML (CLOB)

Tipo Objeto

- Implementação do tipo estruturado
  - abstrações de entidades do mundo real

```
CREATE TYPE T_PESSOA AS OBJECT
(
  NOME VARCHAR2(30),
  TELEFONE VARCHAR2(20)
);
CREATE TABLE TAB_PESSOA OF T_PESSOA;
INSERT INTO TAB_PESSOA VALUES (
  "John Smith",
  "1-800-555-1212" );
SELECT VALUE(P) FROM TAB_PESSOA P
WHERE P.NOME = "John Smith";
```

Tabelas de Objetos

- Duas formas distintas de acesso
  - tabela de uma única coluna contendo objetos do tipo definido
    - operações de orientação a objetos
  - tabela com cada coluna representando um atributo do tipo definido
    - operações relacionais

Tipo REF

- É um ponteiro lógico para um objeto
- Tipos REF e coleções de REFs modelam associações entre os objetos evitando o uso de chaves estrangeiras
- Provêm um fácil e intuitivo mecanismo de navegação entre os objetos, notação ponto '.'
- Segundo a própria Oracle, as operações de junção são evitadas sempre que possível

Tipo REF

- Implementação Oracle para o tipo Referência
- Referências podem se tornar inválidas ("**is dangling**") por causa da remoção do objeto

```
CREATE TYPE T_PESSOA AS OBJECT (
  NOME VARCHAR2(30),
  TELEFONE VARCHAR2(20),
  DATA_NASCIMENTO DATE,
  PAI REF T_PESSOA SCOPE IS TAB_PESSOA,
  MEMBER FUNCTION GET_NOME RETURN VARCHAR,
  ORDER FUNCTION MATCH( P T_PESSOA ) RETURN INTEGER);

DECLARE REF_PESSOA REF TO T_PESSOA;
SELECT REF(P) INTO REF_PESSOA
FROM TAB_PESSOA P
WHERE P.NOME = 'MARTA MATTOSO';
```


Desreferenciando REFs

- Acessar o objeto referenciado por um REF significa desreferenciar um REF
- O Oracle provê o operador DEREF para desreferenciar um REF
- Desreferenciar um Dangling REF retorna um ponteiro NULL

Obtendo REFs

- Pode-se obter o REF de um objeto utilizando-se o operador REF em uma consulta
- A consulta só pode retornar um único objeto

```
DECLARE REF_PESSOA REF TO T_PESSOA;

SELECT REF(P) INTO REF_PESSOA
FROM TAB_PESSOA P
WHERE P.NOME = 'FERNANDA';
```

Visão de Objetos

```
CREATE TABLE EMP (
  ID NUMBER (5),
  NOME VARCHAR2 (20),
  SALARIO NUMBER (9, 2),
);

CREATE TYPE T_EMP  (
  ID NUMBER (5),
  NOME VARCHAR2 (20),
  SALARIO NUMBER (9, 2),
);

CREATE VIEW V_EMP OF T_EMP
WITH OBJECT IDENTIFIER (ID) AS
SELECT E.ID, E.NOME, E.SALARIO
FROM EMP E
WHERE SALARIO > 2000;
```

Coleções

- Tipos de dados de coleções:
  - VARRAYs
  - Tabelas Aninhadas (Nested Tables)
- Estes tipos de coleção podem ser utilizados em qualquer lugar onde os outros tipos podem ser utilizados

VARRAYs

```
CREATE TYPE T_TELEFONES AS VARRAY(3) OF VARCHAR2(20);

CREATE TYPE T_PESSOA AS OBJECT
(
  NOME VARCHAR2(30),
  TELEFONES T_TELEFONES,
  DATA_NASCIMENTO DATE,
  PAI REF T_PESSOA SCOPE IS TAB_PESSOA,
  MEMBER FUNCTION GET_NOME RETURN VARCHAR,
  ORDER FUNCTION MATCH( P T_PESSOA ) RETURN INTEGER
  ...
);
```

Nested Tables

```
CREATE TYPE T_TELEFONES AS TABLE OF T_TELEFONE;

CREATE TYPE T_PESSOA AS OBJECT
(
  NOME VARCHAR2(30),
  TELEFONES T_TELEFONES,
  DATA_NASCIMENTO DATE,
...
)
NESTED TABLE TELEFONES STORE AS TAB_TELEFONES;
```

Consultas em Coleções

```
SELECT P.NOME, P.TELEFONES
FROM TAB_PESSOA P;
```

```
NOME     TELEFONES
-------- --------
'MARY'   T_TELEFONES('1234-5678', '2222-3333')
```

```
SELECT P.NOME, TEL.*
FROM TAB_PESSOA P, TABLE(P.TELEFONES) TEL;
```

```
NOME     TELEFONE
-------- --------
'MARY'   '1234-5678'
'MARY'   '2222-3333'
```

Métodos

- Funções ou procedimentos que modelam o comportamento dos objetos
- Armazenados no banco de dados através de PL/SQL ou Java
- Podem ser classificados em
  - Membros
  - Estáticos
  - Construtores
  - Comparação

Métodos Membros

- Forma como aplicações acessam os dados dos objetos
- Possui sempre parâmetro implícito SELF, logo trabalha com os atributos de um objeto específico (“1 tupla”)
- É chamado da seguinte forma:

OBJETO.METODO()

Métodos Membros (ex.)

```
CREATE TYPE T_PESSOA AS OBJECT
(
  NOME VARCHAR2(30),
  TELEFONE VARCHAR2(20),
  MEMBER FUNCTION GET_NOME RETURN VARCHAR,
  ...
);

CREATE TYPE BODY T_PESSOA AS
MEMBER FUNCTION GET_NOME RETURN VARCHAR IS
BEGIN
  RETURN SELF.NOME;
END GET_NOME;
...
END;
```

Métodos Estáticos

- "Métodos de classe"
  - Trabalham com dados globais do tipo do objeto e não com o objeto específico
  - Não possuem o parâmetro SELF
  - É chamado da seguinte forma:

    TIPO.METODO()

Métodos Estáticos (ex.)

```
CREATE TYPE T_PESSOA AS OBJECT
(
  NOME VARCHAR2(30),
  TELEFONE VARCHAR2(20),
  DATA_NASCIMENTO DATE,
  MEMBER FUNCTION GET_NOME RETURN VARCHAR,
  STATIC FUNCTION PESSOA_MAIS_VELHA RETURN T_PESSOA,
  ...
);
```

Métodos Construtores

- Responsável por criar o objeto e instanciar seus atributos
- Definido pelo sistema
- Existente em todos os tipos de objeto

```
P = T_PESSOA('Marta Mattoso', '2562-8694', '28/01/1970')
```

Métodos de Comparação

- Para comparar dois objetos de tipos criados pelo usuário, o mesmo deve criar uma ordenação para o tipo usando métodos de mapeamento (map methods) ou métodos de ordenação (order methods)
- Recurso que possibilita a indexação de valores de tipos estruturados criados pelo usuário

Métodos de Mapeamento

- Produzem um único valor de um tipo pre-definido (DATE, NUMBER, VARCHAR) para ser utilizado como comparação
- Toda comparação do tipo >, <, =, etc. ou DISTINCT, GROUP BY, ORDER BY chama automaticamente este método de mapeamento, por isto que somente um método deste tipo (ou de ordenação) pode ser declarado por tipo de objeto

Métodos de Mapeamento (ex.)

```
CREATE TYPE T_PESSOA AS OBJECT
(
  NOME VARCHAR2(30),
  TELEFONE VARCHAR2(20),
  DATA_NASCIMENTO DATE,
  MAP MEMBER FUNCTION GET_NOME RETURN VARCHAR,
  ...
);
```

Métodos de Ordenação

- São mais gerais que os métodos de mapeamento
- É uma função com um parâmetro declarado para outro objeto do mesmo tipo e retorna:
  - <0, caso o objeto SELF seja menor que o parâmetro
  - 0, caso sejam iguais
  - >0, caso o objeto SELF seja menor que o parâmetro

Métodos de Ordenação (ex.)

```
CREATE TYPE T_PESSOA AS OBJECT
(
  NOME VARCHAR2(30),
  TELEFONE VARCHAR2(20),
  DATA_NASCIMENTO DATE,
  MEMBER FUNCTION GET_NOME RETURN VARCHAR,
  ORDER FUNCTION MATCH( P T_PESSOA ) RETURN INTEGER
  ...
);
```

Métodos de Ordenação (ex.)

```
CREATE TYPE BODY T_PESSOA AS
ORDER MEMBER FUNCTION MATCH (P T_PESSOA) RETURN INTEGER IS
BEGIN
  IF SELF.NOME < P.NOME THEN
    RETURN -1;
  ELSIF SELF.NOME > P.NOME THEN
    RETURN 1;
  ELSEIF SELF.DATA_NASCIMENTO < P.DATA_NASCIMENTO
    RETURN -1;
  ELSEIF SELF.DATA_NASCIMENTO > P.DATA_NASCIMENTO
    RETURN 1;
  ELSE
    RETURN 0;
  END IF;
END;
...
END;
```

Herança

- Apenas herança simples

```
CREATE TYPE T_EMPLOYEE UNDER T_PERSON
CREATE VIEW Employees OF T_EMPLOYEE UNDER Persons
```

- Permite adição de atributos e métodos, e redefinição de métodos
  - Polimorfismo e propriedade da substituição
  - Controle do usuário sobre a definição de tipos e métodos “herdáveis”
    - FINAL e NOT FINAL
- Tipos de objetos abstratos

```
CREATE TYPE T_PESSOA AS OBJECT(...) NOT INSTANTIABLE;
```

- Permite consulta a objetos de toda a hierarquia, ou restritos a uma tabela específica

Herança

- O Oracle implementa herança simples, ou seja, um subtipo pode ter apenas um supertipo
- Pode se especializar os atributos e métodos de um supertipo da seguinte maneira:
  - Adicionar novos atributos
  - Adicionar novos métodos
  - Modificar a implementação de alguns métodos

Tipos FINAL e NOT FINAL

- Para permitir que um tipo possa possuir subtipos este deve ser definido como NOT FINAL. Por default um tipo de objeto é FINAL.

```
CREATE TYPE T_PESSOA AS OBJECT
(
  NOME VARCHAR2(30),
  TELEFONES T_TELEFONES,
  DATA_NASCIMENTO DATE,
...
) NOT FINAL;
```

Métodos FINAL e NOT FINAL

- Para permitir que um método não possa ser sobrescrito nos subtipos este deve ser declarado como FINAL. Ao contrário de tipos de objetos, por default, um método é NOT FINAL.

```
CREATE TYPE T_PESSOA AS OBJECT
(
  NOME VARCHAR2(30),
  TELEFONES T_TELEFONES,
  FINAL MEMBER FUNCTION GET_NOME RETURN VARCHAR,
...
) NOT FINAL;
```

Criando Subtipos

```
CREATE TYPE T_ALUNO UNDER T_PESSOA
(
  DRE VARCHAR2(15),
  ...
);
```

Tipos de Objetos Abstratos

- Não há construtor
- Não se pode instanciar estes objetos

```
CREATE TYPE T_PESSOA AS OBJECT(...)
NOT INSTANTIABLE NOT FINAL;

CREATE TYPE T_ALUNO UNDER T_PESSOA(...);
```

Tipos de Objetos Abstratos

- Um método também pode ser declarado NON INSTANTIABLE para criar um método em um tipo de objeto sem implementação (esta irá se encontrar nos subtipos)
- Somente em tipos de objetos NON INSTANTIABLE

Material do curso

http://www.cos.ufrj.br/~baiao