LO STANDARD ODMG

OMG (Object Management Group)

associazione privata nata nel 1989 con lo scopo di promuovere l'uso di standard nell'area o-o

Data General, HP, Sun, Canon, American Airlines, Unisys, Philips, Prime, Gold Hill, Soft-Switch, 3 Com $+_{1991}$ AT&T, Digital, NCR, Bull, IBM, Olivetti

ODMG (Object Database Management Group)

è uno dei working group di OMG, che consiste dei maggiori produttori di OODBMS (circa il 90% del mercato)

voting members: Object Design, Objectivity, Ontos, O2 Technology, Versant Object Technology

non-voting members: HP, Intellectic, Servio, Itasca, Texas Instruments

ODMG 2.0: JavaSoft, Lucent, POET Software, UniSQL, IBEX, GemStone Systems

ODMG

- scopo: sviluppare una serie di standard per favorire portabilità, riusabilità e interoperabilità degli OODBMS commerciali

- successo dei RDBMS legato all'esistenza di standard, differenze tra i modelli dei vari OODBMS sono un ostacolo alla loro diffusione

- ODMG nel contesto o-o stesso ruolo di SQL in quello relazionale

- 1993: ODMG-93 standard [R. Cattell, The Object Database Standard: ODMG-93, Morgan-Kaufmann, 1993]

- 1997: ODMG 2.0 standard [R. Cattell et al., The Object Database Standard: ODMG 2.0, Morgan-Kaufmann, 1997]

ODMG

- *Object Model* (modello dei dati ad oggetti)

- *Object Definition Language* (ODL) - la base è l'interface definition language (IDL) di CORBA

- *Object Query Language* (OQL) - linguaggio di interrogazione dichiarativo - la base è SQL

- *Language Bindings*, per C++, Smalltalk, Java

- manca un OML comune

ODMG

componenti addizionali:

- *OMG Object Model Profile* mostra come l'Object model di ODMG sia un superset dell'Object model di OMG, e come l'Object model di ODMG rientri nella struttura componenti/profili di OMG

- *OMG ORB Binding* mostra come gli oggetti ODMG possano agire come oggetti OMG per quel che riguarda lo scambio di messaggi in ambiente distribuito

ODMG 2.0 Object Model

è un modello basato sui concetti classici del paradigma o-o:

- le nozioni di base sono quelle di *oggetto* e *letterale*

- gli oggetti e i letterali sono suddivisi in base ai loro *tipi*

  tutti gli elementi di un certo tipo hanno le stesse proprietà e lo stesso comportamento

- i tipi sono organizzati in gerarchie di *ereditarietà*

$\boxed{\text{ODMG 2.0 Object Model}}$

Oggetti e Letterali

- ogni oggetto ha un unico identificatore, un letterale non ha identificatore
- i letterali sono costanti (immutabili) mentre gli oggetti possono essere modificati
- oggetti
  - atomici
  - collection
    (Set, Bag, List, Array, Dictionary)
  - strutturati
    (Date, Interval, Time, Timestamp)

ODMG 2.0 Object Model

Oggetti e Letterali

- letterali
  - atomici

    (long, short, unsigned long, unsigned short, float, double, boolean, octet, char, string, enum)
  - collection

    (set, bag, list, array, dictionary)
  - strutturati

    (date, interval, time, timestamp

    + struct)

ODMG 2.0 Object Model

Oggetti

- identità: *object identifier* unico e immutabile – generato dall'OODBMS non dalle applicazioni

- *nome*: ad un oggetto possono essere assegnati uno o più nomi significativi per l'utente, unici nell'intero database

- *lifetime*: oggetti transienti o persistenti

- *stato*: proprietà

- *comportamento*: operazioni

ODMG 2.0 Object Model

Oggetti: Proprietà

- lo stato di un oggetto è costituito dai valori per un insieme di *proprietà*

  tali proprietà possono essere:

  - *attributi* dell'oggetto

    i cui valori possono essere oggetti o letterali

  - *associazioni* tra l'oggetto ed altri oggetti – simili ad associazioni nel modello ER

    solo associazioni binarie, cioè tra due tipi

    solo i tipi oggetto possono partecipare ad associazioni

ODMG 2.0 Object Model

Oggetti: Associazioni

- un'associazione viene dichiarata definendo una coppia di *traversal path*, uno per ogni direzione di attraversamento dell'associazione

- si possono dichiarare associazioni di tipo one-to-many e many-to-many mediante i tipi collezione set, list, bag

- l'OODBMS è responsabile del mantenimento dell'integrità referenziale per le associazioni (se si cancella un oggetto che partecipa ad un'associazione, allora anche i traversal path verso tale oggetto sono cancellati)

ODMG 2.0 Object Model

Oggetti: Comportamento

- il comportamento di un oggetto è definito da un insieme di segnature di *operazioni* che possono essere eseguite su o dall'oggetto

- ogni segnatura definisce
  - il nome di un'operazione
  - il nome e il tipo dei suoi argomenti
  - i tipi dei valori restituiti
  - i nomi delle eccezioni che l'operazione può sollevare

- nozioni usuali di overloading, overriding e late binding

ODMG 2.0 Object Model

Classi e Interfacce

- definizione di un object type: *specifica* + una o più *implementazioni*

- specifica: descrizione astratta e indipendente dall'implementazione delle operazioni, eccezioni e proprietà visibili all'utente

- due costrutti per definire la specifica:

  - *interfaccia* specifica solo il comportamento astratto

  - *classe* specifica comportamento e stato astratto

- attributi ed associazioni possono essere dichiarati in un'interfaccia, ma specificano comportamento (accessor methods)

ODMG 2.0 Object Model

Classi: Extent

- *extent* di un tipo = insieme delle sue istanze
  - nella definizione di una classe può essere specificato il nome dell'extent corrispondente
  - tutti gli oggetti istanza di una classe appartengono al suo extent
  - l'extent di una classe include gli extent delle sue sottoclassi
  - le interfacce non sono direttamente istanziabili (il loro extent è costituito dall'unione degli extent delle sottoclassi)

ODMG 2.0 Object Model

Classi: Chiavi

- se le istanze di una classe possono essere univocamernte identificate dal valore di alcune proprietà è possibile definire tali proprietà come *chiavi*

- l'unicità dei valori di chiave è assicurata all'interno di un'extent

- vincolo di integrità semantico, non utilizzato per l'identificazione degli oggetti (OID e nomi)

ODMG 2.0 Object Model

Ereditarietà

ODMG supporta due relazioni di ereditarietà:

ISA e *EXTENDS*

Gerarchia	*ISA*	*EXTENDS*
Ereditarietà di	comportamento	stato + comportamento
Ereditarietà Multipla	SI	NO
supertipo ↑ **sottotipo**	interfaccia ↑ classe o interfaccia	classe ↑ classe

conflitti per ereditarietà multipla del comportamento: si impedisce name overloading

ODMG 2.0: ODL

header
  {*Lista Attributi*
   Lista Associazioni
   Lista Metodi}

dove

- *header* può essere
  - di interfaccia, della forma

    `interface` *Nome* [:*Lista Supertipi*]
  - di classe, della forma

    `class` *Nome* [:*Lista Supertipi*]
                [`EXTENDS` *Lista Superclassi*]
    [(`extent` *Nome Extent*
     `key[s]` *Lista Attributi*)]

ODMG 2.0: ODL

- ogni attributo nella lista è dichiarato come:

  `attribute` *Dominio* *Nome*

- ogni associazione nella lista è dichiarata come:

  `relationship` *Dominio* *Nome*
  `inverse` *Classe* *Nome Inv*

  dove *Dominio* può essere o *Classe*, o una collezione di elementi di *Classe*

- ogni metodo nella lista è dichiarato come:

  Tipo *Nome*(*Lista Par*)
  [`raises` *Lista Eccezioni*]

  dove *Lista Par* è una lista di parametri specificati come

  `in` | `out` | `inout` *Tipo Nome*

ODMG 2.0: ODL

Esempio

```
class Impiegato
(   extent impiegati   key nome)
{   attribute string nome;
    attribute unsigned short stipendio;
    attribute unsigned short telefono[4];
    attribute Impiegato superiore;

    relationship Progetto progetto
       inverse Progetto::assegnati;
    relationship Progetto dirige
       inverse Progetto::capo;
    relationship Set<Task> tasks
       inverse Task::partecipanti;

    int premio(); }

class Documento
(   extent documenti   key titolo)
{   attribute string titolo;
    attribute List<Impiegato> autori;
    attribute string stato;
    attribute string contenuto; }

class Articolo EXTENDS Documento
(   extent articoli)
{   attribute string rivista;
    attribute date data_pubbli; }
```

ODMG 2.0: ODL

Esempio (segue)

```
class Progetto
(   extent progetti   key nome)
{   attribute string nome;
    attribute Set<Documento> documenti;
    attribute Set<Task> tasks;

    relationship Set<Impiegato> assegnati
       inverse Impiegato::progetto;
    relationship Impiegato capo
       inverse Impiegato::dirige; }

class Task
(   extent tasks)
{   attribute unsigned short mesi_uomo;
    attribute date data_in;
    attribute date data_fine;
    attribute Impiegato responsabile;

    relationship Set<Impiegato> partecipanti
       inverse Impiegato::tasks; }
```

ODMG 2.0: ODL

Esempio - 2

```
class Movie (extent Movies key (title, year)) {
  attribute string title;
  attribute short year;
  attribute short length;
  attribute enum Film {color,black&White} type;
  relationship Set<Star> stars
       inverse Star::starredIn;
  relationship Studio ownedBy
       inverse Studio::owns;
  relationship MovieExec producedBy
       inverse MovieExec::produces;
  void changeTypeOrYear();
  };
class Star (extent Stars) {
  attribute string name;
  attribute struct {string street, string city} address;
  attribute enum Gend {male, female} gender;
  relationship Set<Movie> starredIn
       inverse Movie::stars;
  relationship Set<Cartoon> voiceIn
       inverse Cartoon::voices;
  void confirmDel();
  };
```

ODMG 2.0: ODL

Esempio - 2 (segue)

```
class Studio (extent Studios keys address, name) {
  attribute string name;
  attribute string address;
  relationship MovieExec president
       inverse MovieExec::manages;
  relationship Set<Movie> owns
       inverse Movie::ownedBy;
  };
class MovieExec (extent MovieExecs) {
  attribute string name;
  attribute struct {string street, string city} address;
  attribute short netWorth;
  relationship Set<Movie> produces
       inverse Movie::producedBy;
  relationship Studio manages
       inverse Studio::president;
  };
class Cartoon extends Movie (extent Cartoons) {
  relationship Set<Star> voices
       inverse Star::voiceIn;
  };
```

ODMG 2.0: OQL

- linguaggio di interrogazione SQL-like

  (select-from-where)

- linguaggio funzionale in cui gli operatori possono essere liberamente composti

- i risultati di ogni interrogazione hanno un tipo che appartiene al sistema dei tipi di ODMG

- le interrogazioni possono essere annidate

- permette di interrogare oggetti denotabili attraverso i loro nomi

ODMG 2.0: OQL

Path Expressions

- accesso navigazionale agli oggetti
- dot notation
- si applica a: attributi, associazioni, metodi (con late-binding)
- `myMovie` nome di oggetto di tipo `Movie`

  `myMovie.title`, `myMovie.stars`,
  `myMovie.changeTitleOrYear`
- possono avere lunghezza arbitraria

  `myMovie.ownedBy.president.address.city`

ODMG 2.0: OQL

SELECT-FROM-WHERE

- `select` seguito da una lista di espressioni

- `from` seguito da una lista di dichiarazioni di variabili

  una variabile è dichiarata specificando

  - un'espressione il cui valore ha tipo collezione (set o bag)
  - opzionalmente la keyword `as`
  - il nome della variabile

  tipicamente l'espressione è l'extent di una classe

- `where` seguito da un'espressione a valori booleani, in cui possono comparire solo costanti e variabili dichiarate nel `from`

ODMG 2.0: OQL

SELECT-FROM-WHERE: Esempio

- determinare l'anno di *Via col vento*

```
select m.year
from Movies m
where m.title = "via col vento"
```

- determinare i nomi degli attori di *Casablanca*

```
select s.name
from Movies m, m.stars s
where m.title = "casablanca"
```

ODMG 2.0: OQL

Eliminazione dei duplicati

- le query viste producono un bag non un set come risultato

- possibilità di eliminare i duplicati con `select distinct`

- esempio determinare i nomi di tutte le star di film Disney

```
select distinct s.name
from Movies m, m.stars s
where m.ownedBy.name = "disney"
```

ODMG 2.0: OQL

Risultati di tipo strutturato

- le espressioni nella clausola `where` non devono necessariamente essere semplici variabili

- in particolare, possono contenere costruttori di tipo (es. `struct`)

- esempio determinare le coppie di star che vivono allo stesso indirizzo

```
select distinct
  struct(star1: s1, star2: s2)
from Stars s1, Stars s2
where s1.address = s2.address
      and s1.name <> s2.name
```

ODMG 2.0: OQL

Sottointerrogazioni

- si può utilizzare un'espressione select-from-where (interrogazione) ovunque può apparire una collezione

- in particolare, le sottointerrogazioni possono apparire anche nella clausola `from`

- esempio determinare i nomi delle star di film Disney (alternativa)

```
select distinct s.name
from (select m
      from Movies m
      where m.ownedBy.name = "disney") d,
      d.stars s
```

ODMG 2.0: OQL

Ordinamento del risultato

- per rendere il risultato dell'interrogazione una lista, invece che un set o un bag, si può usare la clausola `order by`

- tale clausola, del tutto analoga a quella SQL, è seguita da una lista di espressioni

  (`asc` e `desc` come in SQL)

- esempio determinare i film Disney, ordinati in base alla lunghezza e (a parità di lunghezza) in ordine alfabetico

```
select m
from Movies m
where m.ownedBy.name = "disney"
order by m.length, m.title
```

ODMG 2.0: OQL

Quantificatori

- le espressioni per testare se tutti, o almeno uno, i membri di una collezione soddisfano una condizione sono, rispettivamente:

  `for all` x `in` S `:` $C(x)$

  `exists` x `in` S `:` $C(x)$

- esempio determinare i nomi delle star di film Disney (alternativa 3)

```
select s
from Stars s
where exists m in s.starredIn:
          m.ownedBy.name = "disney"
```

  nomi delle star che appaiono solo in film Disney `exists` $\rightarrow$ `forall`

ODMG 2.0: OQL

Espressioni Aggregate

- OQL usa le stesse 5 funzioni aggregate di SQL: `avg`, `count`, `sum`, `min`, `max`

- si applicano a tutte le collezioni i cui membri siano di tipo appropriato

- esempio determinare la lunghezza media dei film

```
avg(select m.length from Movies m)
```

ODMG 2.0: OQL

Group-by

- la forma della clausola `group by` in OQL è
  - la keyword `group by`
  - una lista di *partition attributes*, separati da virgole - ognuno di questi ha la forma

    nome di campo : espressione

  la forma è cioè $\mathtt{group\ by\ f_1 : e_1, \ldots, f_n : e_n}$

- il valore ottenuto dal raggruppamento è un insieme di strutture della forma

  $$\mathtt{struct(f_1 : v_1, \ldots, f_n : v_n, partition : P)}$$

  dove P denota il bag di oggetti dell'interrogazione che appartengono al gruppo

ODMG 2.0: OQL

Group-by

- la clausola `select` di un'interrogazione che contiene `group by` può contenere solo i campi $\mathtt{f}_1, \ldots, \mathtt{f}_n$ e `partition`

- esempio determinare la lunghezza totale dei film prodotti da ogni studio in ogni anno

```
select st, yr,
       totLength: sum(select p.m.length
                      from partition p)
from Movies m
group by std: m.studio, yr: m.year
```

ODMG 2.0: OQL

Clausola Having

- come in SQL, per eliminare dei gruppi ottenuti con il raggruppamento, si può usare la clausola `having` seguita da una condizione, che si applica ai campi del gruppo

- esempio determinare la lunghezza totale dei film prodotti da ogni studio in ogni anno, per le coppie studo/anno tale che lo studio ha prodotto almeno un film più lungo di 120 in quell'anno

```
select st, yr,
       totLength: sum(select p.m.length
                      from partition p)
from Movies m
group by std: m.studio, yr: m.year
having max(select p.m.length
           from partition p) > 120
```

ODMG 2.0: OQL

Operatori insiemistici

- è possibile applicare gli operatori insiemistici `union`, `intersect`, `except` a coppie di espressioni di tipo collezione

- esempio determinare i film con Harrison Ford che non sono stati prodotti da Disney

```
(select distinct m
 from Movies m, m.stars s
 where s.name = "Harrison Ford")
except
(select distinct m
 from Movies m
 where m.ownedBy.name = "Disney")
```

ODMG 2.0: OQL

Esempi

- determinare i task con almeno 20 mesi uomo il cui responsabile guadagna almeno 2000

```
select t from Tasks t
where t.mesi_uomo > 20 and
      t.responsabile.stipendio > 2000
```

- determinare la data di inizio dei task con almeno 20 mesi uomo

```
select distinct t.data_in
from Tasks t
where t.mesi_uomo > 20
```

il risultato è un letterale di tipo $\texttt{Set} < \texttt{date} >$

ODMG 2.0: OQL

Esempi

- determinare la data di inizio e la data di fine dei task con almeno 20 mesi uomo

```
select distinct
  struct(di: t.data_in, df: t.data_fine)
from Tasks t
where t.mesi_uomo > 20
```

$\mathrm{Set} < \mathrm{struct}(\mathrm{di} : \mathrm{date}, \mathrm{df} : \mathrm{date}) >$

- determinare la data di inizio e il responsabile dei task con almeno 20 mesi uomo

```
select distinct
  struct(di: t.data_in, r: responsabile)
from Tasks t
where t.mesi_uomo > 20
```

$\mathrm{Set} < \mathrm{struct}(\mathrm{di} : \mathrm{date}, \mathrm{r} : \mathrm{Impiegato}) >$

ODMG 2.0: OQL

Esempi

- determinare la data di inizio, i nomi del responsabile e dei partecipanti dei task con almeno 20 mesi uomo

```
select distinct
  struct(di: t.data_in,
         nr: responsabile.nome,
         np: (select p.nome
               from t.partecipanti as p))
from Tasks t
where t.mesi_uomo > 20
```

$$\mathrm{Set} < \mathrm{struct}(\mathrm{di} : \mathrm{date}, \mathrm{nr} : \mathrm{string}, \mathrm{np} : \mathrm{bag} < \mathrm{string} >) >$$

ODMG 2.0: OQL

Esempi

- determinare i rapporti tecnici che hanno lo stesso titolo di un articolo

```
select tr
from Rapporti_Tecnici tr, Articoli a
where tr.titolo = a titolo
```

- determinare il nome ed il premio degli impiegati con stipendio superiore a 2000 e premio superiore a 500

```
select distinct
  struct(di: i.nome, p: i.premio)
from Impiegati i
where i.stipendio > 2000
      and i.premio > 500
```

ODMG 2.0: OQL

Esempi

- determinare lo stipendio massimo dei responsabili dei task del progetto CAD

```
select max(select i.stipendio
           from p.tasks.responsabile i)
from Progetti p
where p.nome = ´CAD´
```