Basi di dati a oggetti

Sommario

- Motivazioni
- OODB
- ORDB
- Tecnologia delle basi di dati a oggetti

Tecnologia delle basi di dati relazionali

- I sistemi di gestione di basi di dati relazionali (RDBMS) hanno permesso la realizzazione efficace ed efficiente di applicazioni di tipo gestionale, caratterizzate da
  - persistenza, condivisione, affidabilità
  - dati a struttura semplice, con dati di tipo numerico/simbolico
  - transazioni concorrenti di breve durata (OLTP)
  - interrogazioni complesse, espresse mediante linguaggi dichiarativi e con accesso di tipo "associativo"
- La rapida evoluzioni tecnologica (miglioramento di prestazioni, capacità, e costi dell'hardware) ha fatto emergere nuove esigenze applicative — per le quali la tecnologia relazionale è inadeguata

Alcune aree applicative emergenti

- Progettazione assistita da calcolatore
  - CASE (Computer-Aided Software Engineering)
  - CAD (Computer-Aided Design)
  - CAM (Computer-Aided Manufacturing)
- Gestione di documenti
  - testi e automazione d'ufficio
  - dati ipertestuali
  - dati multimediali
- Altro
  - scienza e medicina
  - sistemi esperti, per la rappresentazione di conoscenza

Caratteristiche delle nuove aree applicative

- Oltre alle caratteristiche consuete di persistenza, condivisione e affidabilità, possiamo individuare
  - dati a struttura complessa
    - dati non-numerici — immagini, dati spaziali, sequenze temporali, ...
    - tipi pre-definiti e tipi definiti dall'utente (e riutilizzati)
    - relazioni esplicite ("semantiche") tra i dati (riferimenti), aggregazioni complesse
  - operazioni complesse
    - specifiche per i diversi tipi di dato — es. multimedia
    - associate anche ai tipi definiti dall'utente
    - transazioni di lunga durata

Basi di dati a oggetti

- Alcune delle precedenti caratteristiche suggeriscono l'introduzione di nozioni dal paradigma orientato agli oggetti nel mondo delle basi di dati
- A partire dalla metà degli anni '80, sono stati realizzati numerosi sistemi di gestione di basi di dati a oggetti (ODBMS) — di tipo prototipale o commerciale
- I sistemi sono stati sviluppati indipendentemente, senza nessuna standardizzazione circa il modello dei dati o i linguaggi da utilizzare
- Dopo un periodo di "evoluzione," inizia ad esserci una convergenza su modello e linguaggio, per far sì che sistemi realizzati da produttori diversi possano almeno interoperare

Tecnologia degli ODBMS

- La prima generazione di ODBMS è composta dai linguaggi di programmazione a oggetti *persistenti*, che realizzano solo alcune caratteristiche delle basi di dati, senza supporto per l'interrogazione, in modo incompatibile con gli RDBMS
- Gli ODBMS della seconda generazione realizzano un maggior numero di caratteristiche delle basi di dati, e generalmente forniscono un supporto all'interrogazione
- Due tecnologie di ODBMS
  - OODBMS (*Object-Oriented*): una tecnologia *rivoluzionaria* rispetto a quella degli RDBMS
  - ORDBMS (*Object-Relational*): una tecnologia *evoluzionaria* rispetto a quella degli RDBMS

Un modello dei dati a oggetti

- Una base di dati a oggetti è una collezione di oggetti
- Ciascun oggetto ha un identificatore, uno stato, e un comportamento
  - l'identificatore (OID) garantisce l'individuazione in modo univoco dell'oggetto, e permette di realizzare riferimenti tra oggetti
  - lo stato è l'insieme dei valori assunti dalle proprietà dell'oggetto — è in generale un valore a struttura complessa
  - il comportamento è descritto dall'insieme dei metodi che possono essere applicati all'oggetto

Un modello dei dati a oggetti

- Gli oggetti sono associati ad un tipo (intensione) e ad una classe (implementazione)
  - un tipo è una astrazione che permette di descrivere (1) lo stato e (2) il comportamento di un oggetto
  - una classe descrive l'implementazione di un tipo — struttura dei dati e implementazione di metodi tramite programmi
- Gli oggetti vengono raggruppati in collezioni (estensioni)
- Faremo le seguenti ipotesi semplificative
  - il concetto di *classe* descrive sia l'implementazione sia l'estensione di un tipo
  - ogni tipo è associato ad una sola classe

Tipi — parte statica

- Un *tipo* descrive le *proprietà* di un oggetto (la parte statica) e l'interfaccia dei suoi *metodi* (la parte dinamica)
- Relativamente alla parte statica, i tipi vengono costruiti a partire da
  - un insieme di *tipi atomici* (numeri, stringhe, ...)
  - un insieme di *costruttori di tipo*, tra loro ortogonali
    - *record-of*(A_1:T_1, ..., A_n:T_n)
    - *set-of*(T), *bag-of*(T), *list-of*(T)
  - un *riferimento* ad altro tipo definito nello schema è considerato un tipo, utilizzato per rappresentare relazioni tra oggetti (associazioni)

Un esempio di tipo complesso

```
automobile: record-of(
   targa: string, modello: string,
   costruttore: record-of(
       nome: string,
       presidente: string,
       stabilimenti: set-of(
             record-of(
                   nome: string, citta: string,
                   addetti: integer))),
   colore: string, prezzo: integer,
   partiMeccaniche: record-of(
       motore: string,
       ammortizzatore: string))
```

Un esempio di valore complesso

- E' possibile definire dei valori complessi *compatibili* con un tipo complesso

```
V1: [targa: "MI67T891", modello: "uno",
   costruttore: [
       nome: "FIAT", presidente: "Agnelli",
       stabilimenti: {
              [nome: "Mirafiori", citta: "Torino",
               addetti: 10000],
              [nome: "Trattori", citta: "Modena",
               addetti: 1000]}],
   colore: "blu", prezzo: 15.5M,
   partiMeccaniche: [
       motore: "1100CV", ammortizzatore: "Monroe"]]
```

Oggetti e valori

- L'uso di tipi e valori complessi permette di associare ad un singolo oggetto una struttura qualunque
- Viceversa, nel modello relazionale alcuni concetti devono essere rappresentati tramite più relazioni
- Tuttavia, la rappresentazione proposta per `automobile` non è "normalizzata": vediamo come decomporla utilizzando dei *riferimenti tra oggetti*
- Un *oggetto* è una coppia (*OID, Valore*), dove *OID* (*object identifier*) è un valore atomico definito dal sistema e trasparente all'utente, e *Valore* è un valore complesso
- Il valore assunto da una proprietà di un oggetto può essere l'OID di un altro oggetto (realizzando così un riferimento)

Riferimenti

```
automobile: record-of(
        targa: string, modello: string,
        costruttore: *costruttore,
        colore: string, prezzo: integer,
        partiMeccaniche: record-of(
                motore: string,
                ammortizzatore: string))
costruttore: record-of(
        nome: string, presidente: string,
        stabilimenti: set-of(*stabilimento))
stabilimento: record-of(
        nome: string, citta: string,
        addetti: integer)
```

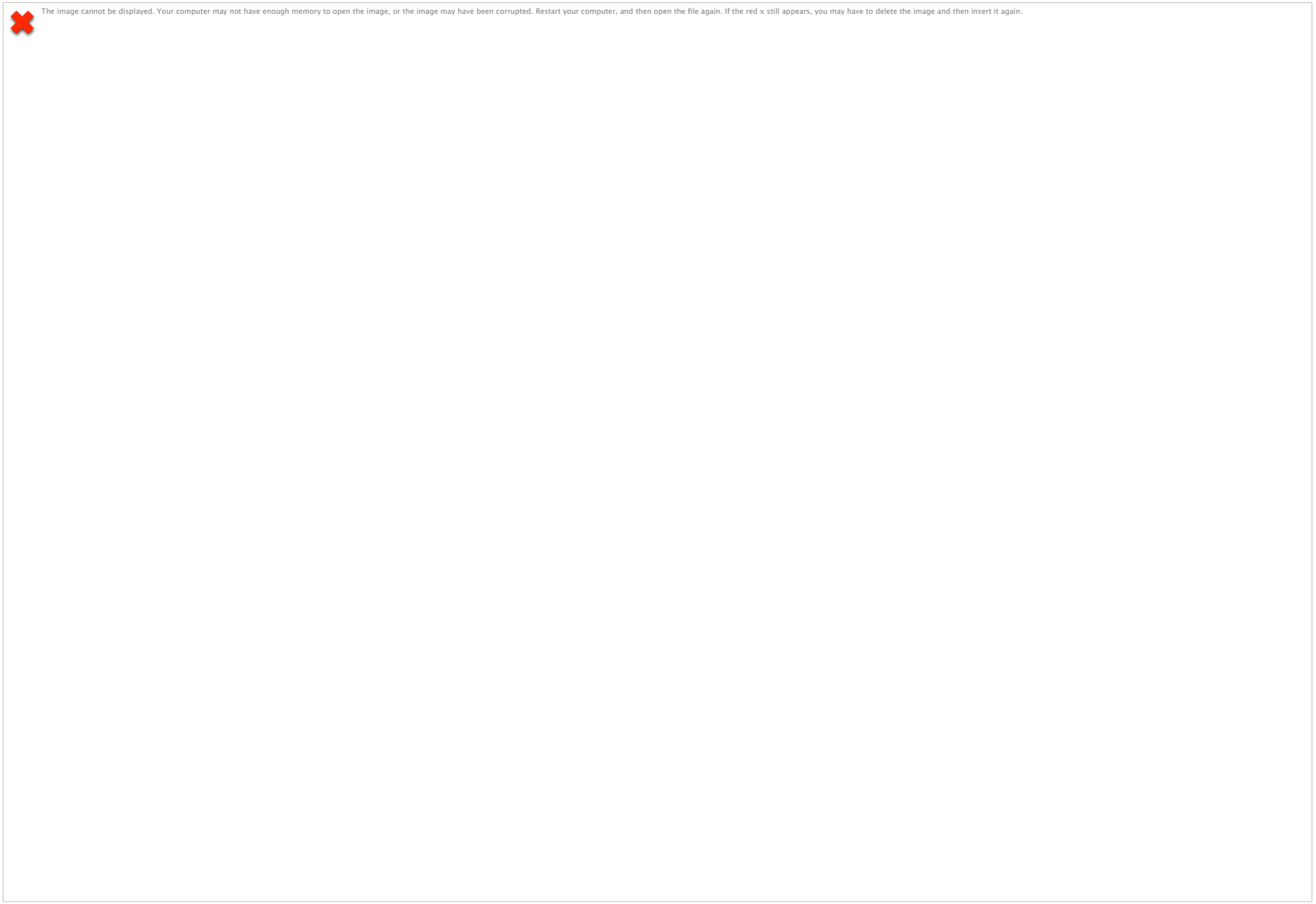
Riferimenti

- Un insieme di oggetti compatibili con lo schema

```
O1: <OID1, [targa: “MI67T891”, modello: “uno”,
                costruttore: OID2, colore: “blu”,
                prezzo: 15.5M, motore: “1100CV”,
                ammortizzatore: “Monroe”]>
O2: <OID2, [nome: “FIAT”, presidente: “Agnelli”,
                stabilimenti: {OID3,OID4}]>
O3: <OID3, [nome: “Mirafiori”, citta: “Torino”,
                addetti: 10000]>
O4: <OID4, [nome: “Trattori”, citta: “Modena”,
                addetti: 1000]>
```

Identità e uguaglianza

- Tra gli oggetti sono definite le seguenti relazioni
  - *identità* (O1=O2) — richiede che gli oggetti abbiano lo stesso identificatore
  - *uguaglianza superficiale* (O1==O2) — richiede che gli oggetti abbiano lo stesso stato, cioè stesso valore per proprietà omologhe
  - *uguaglianza profonda* (O1===O2) — richiede che le proprietà che si ottengono seguendo i riferimenti abbiano gli stessi valori (non richiede l'uguaglianza dello stato)


The image cannot be displayed. Your computer may not have enough memory to open the image, or the image may have been corrupted. Restart your computer, and then open the file again. If the red x still appears, you may have to delete the image and then insert it again.

Semantica dei riferimenti

- Il concetto di riferimento presenta analogie con quello di *puntatore* nei linguaggi di programmazione, e con quello di *chiave esterna* in un sistema relazionale. Tuttavia
  - i puntatori possono essere corrotti (dangling, appesi); i riferimenti a oggetti (in un buon ODBMS) vengono invalidati automaticamente in caso di cancellazione di un oggetto referenziato
  - le chiavi esterne sono visibili, in quanto realizzate tramite valori; gli identificatori d'oggetto non sono associati a valori visibili dall'utente
  - modificando gli attributi di una chiave esterna, è possibile perdere riferimenti; modificando il valore di un oggetto referenziato, il riferimento continua ad esistere

Relationships (ODMG-93)

- Riferimenti e collezioni di riferimenti permettono di rappresentare *relazioni* (*binarie*) tra due tipi A e B
  - una relazione uno a uno può essere rappresentata con attributo "riferimento" in ciascun tipo
  - una relazione uno a molti, aggiungendo un riferimento da A a B, e un insieme di riferimenti da B ad A
  - una relazione molti a molti, aggiungendo un insieme di riferimenti a ciascun tipo
- In questo modo vengono definite due proprietà, una in ciascuna classe, che non sono indipendenti: ognuna è l'inversa dell'altra
- Alcuni modelli definiscono una gestione automatica delle relazioni

Relazioni

```
Documento: record-of(
        titolo: string, revisione: date,
        capitoli: list-of(*Capitolo<->doc),
        autori: list-of(*Persona<->pubs))

Capitolo: record-of(
        titolo: string, numero: integer, testo: text,
        doc: *Documento<->capitoli)

Persona: record-of(
        nome: string,
        pubs: list-of(*Documento<->autori))
```

Vincoli di integrità referenziale

- E’ possibile definire diverse semantiche per le relazioni, in riferimento alla possibilità di cancellare oggetti
  - *nessuna verifica* — quello che avviene nei linguaggi di programmazione (nota: un indirizzo di memoria può casualmente venire riassegnato)
  - *validazione dei riferimenti* — è possibile scaricare automaticamente i riferimenti agli oggetti cancellati
  - *integrità delle relazioni* — è possibile scaricare automaticamente i riferimenti inversi agli oggetti non più referenziati
  - *personalizzata* — ad esempio, è possibile fare delle cancellazioni di oggetti in cascata, o prevenirle

Metodi — parte dinamica

- Il paradigma OO deriva dal concetto di *tipo di dato astratto*
- Un *metodo* è una procedura utilizzata per incapsulare lo stato di un oggetto, ed è caratterizzata da una *interfaccia* (o segnatura) e una *implementazione*
  - l'interfaccia comprende tutte le informazioni che permettono di invocare un metodo (il tipo dei parametri)
  - l'implementazione contiene il codice del metodo
- Il *tipo* di un oggetto comprende, oltre alle proprietà, anche le interfacce dei metodi applicabili a oggetti di quel tipo
- Ipotizziamo che i metodi siano assimilabili a *funzioni*, ovvero possono avere più parametri di ingresso ma un solo parametro di uscita

Metodi

- In prima approssimazione, i metodi possono essere dei seguenti tipi
  - *costruttori* — per costruire oggetti a partire da parametri di ingresso (restituendo l'OID dell'oggetto costruito)
  - *distruttori* — per cancellare gli oggetti, ed eventuali altri oggetti ad essi collegati
  - *accessori* — funzioni che restituiscono informazioni sul contenuto degli oggetti (proprietà derivate)
  - *trasformatori* — procedure che modificano lo stato degli oggetti, e di eventuali altri oggetti ad essi collegati
- Un metodo può essere *pubblico* o *privato*

Metodi

```
automobile: record-of(
        targa: string, modello: string,
        costruttore: *costruttore,
        colore: string, prezzo: integer,
        partiMeccaniche: record-of(
                motore: string,
                ammortizzatore: string),
        public Init(                        // costruttore
                targa_par: string,
                modello_par: string, colore_par: string,
                prezzo_par: integer): automobile,
        public Prezzo(): integer,  // accessore
        public Aumento(             // trasformatore
                ammontare: integer))        // void
```

Classi — estensioni dei tipi

- Abbiamo ipotizzato che una classe raccoglie tutti gli oggetti di uno stesso tipo
- Una classe è un contenitore di oggetti, che possono essere dinamicamente aggiunti o tolti alla classe (tramite costruttori e distruttori)
- Ad una classe è associato un solo tipo, ovvero gli oggetti in una classi sono tra loro omogenei (hanno le stesse proprietà e rispondono agli stessi metodi)
- Una classe definisce anche l'implementazione dei metodi, che va specificata in qualche linguaggio di programmazione

Implementazione e invocazione di metodi

```
body Init(targa_par: string, modello_par: string,
   colore_par: string, prezzo_par: integer):
   automobile  in class automobile
co2{
   self->targa = targa_par;
   self->modello = modello_par;
   self->colore = colore_par;
   self->prezzo = prezzo_par;
   return(self);
}$

execute co2{         // in un programma
   o2 automobile X; // dichiarazione di variabile O2
   X = new(automobile);
   [X Init("Mi56T778", "Panda", "blu", 12M)]; }$
```

Disaccoppiamento (o disadattamento) di impedenza

- In un RDBMS, esiste un disaccoppiamento di impedenza tra i linguaggi con i quali vengono scritte le applicazioni (che manipolano variabili scalari) e l'SQL, che estrae insiemi di ennuple. Si usano allo scopo i *cursori*
- Si dice che gli ODBMS risolvono questo problema, in quanto gli oggetti persistenti possono essere manipolati direttamente tramite le istruzioni del linguaggio di programmazione (procedurale)
- Il PL di un ODBMS deve permettere l'accesso alle componenti di un valore complesso — ad esempio, i record con l'operatore dot ".", i riferimenti con l'operatore "->", le collezioni con opportuni iteratori

Classi ed estensioni

- I concetti di classe e di estensione non sono identici
- Una *classe* è una implementazione di un tipo — significa che uno stesso tipo può avere implementazioni diverse. Questo è particolarmente importante non per assegnare semantiche diverse a oggetti di uno stesso tipo (sigh!), ma ad esempio per implementare una (unica) semantica di un tipo in riferimento a piattaforme architetturali diverse (nel caso di una base di dati a oggetti distribuita)
- Una *estensione* è una collezione di oggetti aventi lo stesso tipo; un oggetto può appartenere a più collezioni, essere rimosso da una collezione, ecc.

Altre caratteristiche del paradigma a oggetti

- Tra i tipi (e le classi) di una base di dati a oggetti, è possibile definire una gerarchia di ereditarietà, con le usuali relazioni di sottotipo, l'ereditarietà dei metodi, la possibilità di overloading, overriding, late binding, ereditarietà multipla, ...
- Tutto funziona come nei linguaggi di programmazione a oggetti
- Esiste tuttavia una importante differenza: gli oggetti di un programma sono oggetti di breve durata, gli oggetti di una base di dati sono oggetti di lunga durata, con conseguenze non banali...

Object Database Management Group e ODMG-93

- ODMG è composto da partecipanti di tipo accademico e industriale, ed in particolare i principali produttori di OODBMS. Citiamo: SunSoft, O2, HP, TI, AT&T, Versant, ONTOS, Objectivity, Itasca, ...
- ODMG-93 è una proposta di standard per OODBMS, che i partecipanti intendono realizzare o supportare
  - un modello a oggetti
  - ODL, un linguaggio di definizione basato su IDL (OMG)
  - OQL, un linguaggio di interrogazione, "basato" su SQL3
  - Language bindings per C++ e Smalltalk
- ODMG-93 *non è* uno standard formale nè una specifica di OODBMS

Il modello ODMG-93

- Le *proprietà* e le *operazioni* di un tipo sono chiamate *caratteristiche*
- E' possibile definire una gerarchia di tipi, alcuni dei quali sono *astratti* (non instanziabili)
- L'*estensione* di un tipo è l'insieme di tutte le sue istanze
- Un tipo ha una o più *implementazioni* (*classi*) — ad esempio, una classe può richiedere di implementare una proprietà di tipo **Set** come **Set_as_Btree**
- Alcuni oggetti possono avere un *nome*, per riferirsi ad essi esplicitamente nei programmi — altri possono essere identificati mediante interrogazioni

Persistenza

- Gli oggetti possono essere **temporanei** (come nei programmi tradizionali) o **persistenti**.
- La persistenza può essere specificata in vari modi (non sempre tutti disponibili in uno stesso sistema):
  - inserimento in una classe persistente
  - raggiungibilità da oggetti persistenti
  - "denominazione": si può definire un nome ("handle") per un oggetto persistente

OQL — Object Query Language

- Linguaggio SQL-like per basi di dati a oggetti, inizialmente sviluppato per O2, adottato (con modifiche) da ODMG, basato sui seguenti principi
  - non è computazionalmente completo, ma può invocare metodi, e metodi possono includere interrogazioni
  - permette un accesso dichiarativo agli oggetti
  - basato sul modello ODMG
  - ha una sintassi *simile* a SQL
  - ha primitive di alto livello per le collezioni
  - non ha operatori di aggiornamento
  - può essere ottimizzato in quanto dichiarativo (???)

OQL per esempi (1)

- Seleziona il presidente

```
Presidente
```

- Seleziona i subordinati del presidente

```
Presidente.subordinati
```

- Seleziona gli stipendi degli impiegati di nome Pat

```
select distinct x.stipendio
from x in Impiegati
where x.nome = “Pat”
```

- Seleziona nome e età degli impiegati di nome Pat

```
select distinct struct(n: x.nome, e: x.eta())
from x in Impiegati
where x.nome = “Pat”
```

OQL per esempi (2)

- Attraversamento di una relazione

```
select struct
       (dip: x.nome_dip, dir: x.direttore.nome)
from x in Dipartimenti
```

- Dovrebbe essere equivalente a

```
select struct
       (dip: x.nome_dip, dir: y.nome)
from x in Dipartimenti, y in Impiegati
where x.direttore = y
```

- E' possibile definire valori a struttura complessa

```
struct(e: 36, nome: "Pat")
```

- E' possibile creare nuovi oggetti

```
Impiegato(nome: "Pat", ...)
```

OQL per esempi (3)

- Per ciascun impiegato, seleziona il nome e l'insieme dei subordinati con stipendio maggiore di 100000

```
select distinct struct(nome: x.nome,
       spp: ( select y
                    from y in x.subordinati
                    where y.stipendio > 100000 ))
from x in Impiegati
```

- La clausola **select** si può utilizzare anche entro la clausola **from**

```
select struct(n: x.nome, e: x.eta())
from x in (  select y from y in Impiegati
                    where y.stipendio > 100000 )
where x.nome = "Pat"
```

OQL per esempi (4)

- Creazione di viste — e accesso a un attributo composto

```
define Romani as
        select x
        from x in Impiegati
        where x.nascita.citta = “Roma”
```

- Appartenenza

```
select x.eta()
from x in Impiegati
where x in Romani
```

- Quantificatori (queste sono query booleane)

```
for all x in Impiegati: x.stipendio > 200000
exists x in Impiegati: x.eta() < 19
```

OQL per esempi (5)

- Subordinati di subordinati

```
select distinct struct(imp: x, subsub: y)
from x in Impiegati,
  z in x.subordinati,
        y in z.subordinati
```

- Estrazione di singoletti

```
element(select x.eta()
          from x in Impiegati
          where x.nome = "Bob")
```

- Conversione di tipologia di collezione

```
listtoset(list(1,2,3,2))
```

- Restituisce la *lista* degli impiegati, ordinata per nome

```
sort x in Impiegati by x.nome
```

OQL per esempi (6)

- Partiziona gli impiegati in base allo stipendio

```
group x in Impiegati by (
        low: x.stipendio < 1000,
        mid: x.stipendio >= 1000 and
               x.stipendio < 5000),
        high: x.stipendio >= 5000)
```

- Questa espressione ha tipo

```
set<struct(low: boolean, mid: boolean,
                         high: boolean,
                         partition: set<Impiegato>)>
```

- Una versione estesa dell'operatore **group** permette il calcolo di funzioni aggregative

The Object-Oriented Database Manifesto

(Atkinson, Bancilhon, DeWitt, Dittrich, Maier, Zdonik)

- Una lista di funzionalità per la definizione (e la valutazione) di OODBMS.
- Include:
  - Funzionalità obbligatorie (the "golden rules")
  - Funzionalità opzionali
  - Scelte aperte

THE GOLDEN RULES

- Thou shalt support complex objects
- Thou shalt support object identity
- Thou shalt encapsulate thine objects
- Thou shalt support types or classes
- Thine classes or types shalt inherit from their ancestors
- Thou shalt not bynd prematurely
- Thou shalt be computationally complete
- Thou shalt be extensible
- Thou shalt remember the data
- Thou shalt manage very large databases
- Thou shalt accept concurrent users
- Thou shalt recover from hardware and software failures
- Thou shalt have a simple way of querying data

Funzionalità obbligatorie

- Oggetti complessi
- Identità di oggetto
- Incapsulamento
- Tipi e/o classi
- Gerarchie di classi o di tipi
- Overriding, overloading e late binding
- Completezza computazionale
- Estensibilità
- Persistenza
- Gestione della memoria secondaria
- Concorrenza
- Recovery
- Linguaggio o interfaccia di interrogazione

Funzionalita' quasi (?) obbligatorie ("no consensus")

- Dati derivati e definizione di viste
- Funzionalità DBA
- Vincoli di integrità
- Funzionalità per la modifica di schemi

Funzionalità opzionali

- Ereditarietà multipla
- Verifica dei tipi ed inferenza su di essi
- Distribuzione
- "Design transactions" (transazioni lunghe e nidificate)
- Gestione delle versioni

The Third Generation Database System Manifesto

(Stonebraker, Rowe, Lindsay, Gray, Carey, Brodie, Bernstein, Beech)

- una risposta al manifesto OODMS
- "I DBMS della prossima generazione dovranno essere ottenuti come risultato dell'evoluzione dei DBMS esistenti (relazionali)"

I principi del contromanifesto

- i DBMS di terza generazione dovranno essere una generalizzazione (compatibile) con i DBMS della seconda generazione
- oltre a fornire i servizi tradizionali di gestione dei dati, dovranno permettere la definizione di oggetti complessi e regole
- dovranno essere aperti ad altri sottosistemi

Manifesto 3GDBMS: dettagli

[1.1] rich type system
[1.2] inheritance
[1.3] functions and encapsulation
[1.4] OID's only if there are no keys
[1.5] rules and triggers

[2.1] non procedural, high level access languages
[2.2] specification techniques for collections
[2.3] updatable views
[2.4] transparency of physical parameters

[3.1] multiple high level languages
[3.2] persistent x, for many x's
[3.3] SQL is a standard (even if you don't like it)
[3.4] queries and their results are the lowest levest of communication

Basi di dati “Object-Relational”

- Modello dei dati
- Linguaggio di interrogazione

- La futura versione di SQL (noto come SQL-99)
- Si tratta di una estensione “compatibile” di SQL-2 (cioè il codice SQL-2 è valido anche come SQL-3)

Modello dei dati di SQL-3

- È possibile definire tipi:
  - tipi ennupla, con struttura anche complessa e con gerarchie:
    - utilizzabili per definire tabelle con lo stesso schema
    - utilizzabili come componenti
    - utilizzabili nell'ambito di relationship
  - tipi astratti

Interrogazioni in SQL-3

- Le interrogazioni SQL-2 sono ammesse in SQL-3
- Inoltre:
  - si possono “seguire” i riferimenti
  - si possono citare gli OID (se visibili)
  - si può accedere alle strutture interne
  - si puo’ nidificare (nest) e denidificare (unnest)

Interrogazioni in SQL-3

```
select President -> Name
from Manufacturer
where Name = 'Fiat'
```

```
select Name
from Manufacturer, Industrial
where Manufacturer.Name = 'Fiat'
  and Manufacturer.President = Industrial.ManufId
```

```
select Maker -> President -> Name
from Automobile
where MechanicalParts.Motor = 'XV154'
```

Unnesting e nesting in SQL-3

```
select C.Name, S.City
from Manufacturer as C, C.Factories as S
```

```
select City, set(name)
from manufacturer
group by City
```

Tecnologia delle basi di dati a oggetti

- La realizzazione di sistemi di gestione di basi di dati a oggetti solleva un insieme di problemi tecnologici specifici
- Alcune di queste problematiche
  - rappresentazione dei dati e degli identificatori
  - indici complessi
  - architettura client-server
  - modello transazionale
  - architettura a oggetti distribuiti

Rappresentazione dei dati

- Possiamo pensare a rappresentazioni relazionali di una base di dati a oggetti — facendo riferimento a gerarchie di classi
  - approccio *orizzontale* — ogni oggetto viene rappresentato in modo "contiguo" entro la classe più specifica di appartenenza
  - approccio *verticale* — gli oggetti sono suddivisi nelle proprie componenti (proprietà), le quali sono memorizzate contiguamente
- L'approccio orizzontale favorisce l'accesso agli oggetti nel loro complesso, l'approccio verticale la ricerca di oggetti sulla base di una loro proprietà
- I BLOB sono rappresentati su file specifici

Rappresentazione degli identificatori

- Esistono diversi approcci per la rappresentazione degli OID
  - mediante *indirizzo fisico*, ovvero riferimento alla memoria di massa
  - mediante *surrogato*, cioè un valore simbolico associato univocamente ad un oggetto. L'accesso all'oggetto avviene mediante strutture di accesso
- Nel caso (frequente) di oggetti distribuiti su più sistemi, bisogna garantire l'univocità degli identificatori

Indici complessi

- I linguaggi di accesso prevedono l'uso di *path expressions*, per navigare le componenti degli oggetti ed accedere oggetti ad essi collegati
- Gli operatori "." e "->" sono utilizzati per accedere le proprietà di un tipo record, per seguire riferimenti, e talvolta per accedere le componenti di una collezione

```
select x.subordinati.subordinati.citta_nascita
from x in Impiegati
```

- Sui cammini più utilizzati, deve essere possibile definire degli indici, che consentano un accesso in avanti (determina le città di nascita dei sub-subordinati) oppure in indietro (trova i sup-superiori dei nati a Milano)
- Meccanismi di indicizzazione per tipi di dato specifici (dati multimediali, spaziali, sequenze temporali)

Modello transazionale

- Applicazioni orientate alla progettazione richiedono modelli transazionali più complessi che non quelli basati sul locking
  - transazioni di lunga durata (minuti, ore, o anche giorni)
  - transazioni complesse:
    - transazioni nidificate
    - saga: transazioni con transazioni compensatrici
- Idee utilizzate
  - check-out e check-in di oggetti
  - versioni di oggetti,
    versioni di collezioni di oggetti,
    versioni dello schema degli oggetti