Il modello orientato agli oggetti

Un **database ad oggetti** è un sistema per la gestione dei database che memorizza le informazioni sotto forma di oggetti. La modalità di rappresentazione dei dati è la stessa di quella usata nella **programmazione ad oggetti**.
Quando si parla di *database ad oggetti* si fa riferimento a due differenti famiglie di sistemi, caratterizzate da differenti approcci usati per la memorizzazione dei dati:

- **database orientati agli oggetti**, indicati anche come **Object Oriented Database Management System** (*OODBMS*), sviluppati a partire dai principi della programmazione ad oggetti;
- **database relazionali ad oggetti**, indicati come **Object-Relational Database Management System** (*ORDBMS*), nei quali si è cercato di integrare i due approcci. I sistemi relazionali ad oggetti gestiscono i dati memorizzandoli in tabelle, secondo l'approccio dei database relazionali, ma dispongono di un sistema di frontiera che trasforma gli oggetti nei record di opportune tabelle e viceversa. In questo modo un'applicazione che manipola oggetti interagisce con il DBMS inviando ad esso oggetti e ricevendone oggetti: in pratica, dialoga con il DBMS come se esso memorizzasse oggetti.

Un *OODBMS* è il risultato della combinazione dei principi della programmazione ad oggetti con quelli dei sistemi per la gestione dei database. In un database ad oggetti coesistono quindi sia i concetti della programmazione ad oggetti, quali: **incapsulamento**, **polimorfismo** ed **ereditarietà** così come le caratteristiche tipiche dei sistemi di basi di dati, includendo il supporto alle transazioni, i sistemi per garantire l'integrità e la persistenza dei dati e la presenza di linguaggi di interrogazione ad hoc.

Vale la pena di fare alcune precisazioni in merito ai tradizionali sistemi per la gestione dei database relazionali, o **Relational Database Management System** (*RDMBS*). Come si vedrà nel Capitolo 4, dedicato al modello relazionale dei dati, i tradizionali *RDBMS* memorizzano i dati in molte tabelle logicamente correlate e opportunamente *normalizzate* per evitare la duplicazione dei dati. Le tabelle sono organizzate in righe (o *record*) e colonne, ognuna delle quali contiene i dati di uno specifico *campo* o *attributo*. All'incrocio di una riga con una colonna sono memorizzati dati di tipo atomico, quali: un numero intero o reale, una stringa di caratteri, una data, un valore di verità. In un campo di una qualsiasi colonna non è quindi possibile memorizzare strutture dati complesse quali: array, record, valori ripetuti, insiemi di valori.

A differenza di quanto avviene con gli *RDBMS*, i database orientati agli oggetti (*OODBMS*) memorizzano oggetti. Gli oggetti trattati sono gli stessi di quelli considerati dai linguaggi orientati agli oggetti, come Java e C++, e sono di conseguenza composti sostanzialmente da due parti:

- i *dati* che definiscono le caratteristiche di un oggetto e che possono essere sia semplici dati di tipo atomico, sia strutture dati arbitrariamente complesse oppure riferimenti ad altri oggetti;
- i *metodi* che definiscono il comportamento dinamico dell'oggetto e specificano, mediante funzioni, come creare e modificare lo stato di un oggetto.

Questo significa, in pratica, che gli oggetti contengono sia dati che codice eseguibile, e che gli *OODBMS* memorizzano sia lo stato di un oggetto, che le modalità con il quale esso deve essere usato.

Questo spiega la ragione per cui gli *OODBMS* sono stati pensati per soddisfare le esigenze applicative in settori dove si opera con realtà fatte di oggetti complessi, difficilmente modellabili con il paradigma del modello relazionale o, meglio, dove il modello relazionale dei dati risulta inadeguato o inefficiente. Questo avviene, nelle applicazioni **CAD/CAM** (*Computer Aided Design/Computer Aided Manifacturing*), **CASE** (*Computer Aided Software Engineering*), oppure nella catalogazione e recupero di documenti con dati multimediali, quali immagini, suoni e dati geografici.

Consideriamo l'esempio di un oggetto pensato per modellare lo studente che frequenta i corsi di una scuola superiore. Per descrivere lo stato dello studente consideriamo: i suoi dati anagrafici, la classe alla quale è iscritto, i voti ottenuti nelle diverse materie, le assenze alle lezioni e, infine, i voti finali e il giudizio complessivo di fine anno. Un oggetto di tipo *Studente* può essere allora definito, usando una notazione informale, con le dichiarazioni sotto riportate.

```
Studente: record ( Matricola:   integer,
                   Nome:        string,
                   Cognome:     string,
                   Indirizzo:   record(Via:     string,
                                       Numero:  string,
                                       Città:   string ),
                   Telefono:    record(Casa:    string,
                                       Mobile:  string ),
                   Nascita:     date,
                   CodFisc:     string,
                   Iscrizione:  date,
                   IscrittoA:   record(Classe:  string,
                                       Sezione: string ),
                   Voti:        set(
                                record(Materia: string,
                                       Data:    date,
                                       Causale: string,
                                       Voto:    float )),
                   Assenze:     set(
                                record(Data:    date,
                                       Motivo:  string,
                                       Note:    string )),
                   VotoFinale:  set(
                                record(Materia: string,
                                       Voto:    string )),
                   Giudizio:    record(Promosso:boolean,
                                       Note:    string ) )
```

Il tipo *Studente* è un dato di tipo *record* contenente sia un insieme di informazioni anagrafiche rappresentate con dati di tipo atomico: *Matricola*, *Nome*, *Cognome*, *Nascita*, *Codice Fiscale* e *data di iscrizione* alla scuola, sia informazioni di tipo complesso quali: *Indirizzo*, *Telefono* e *IscrittoA* e *Giudizio* che sono a loro volta dei record, oltre a *Voti*, *Assenze* e *VotoFinale* che sono *insiemi* (*set*) composti da record. *Studente* è una struttura dati complessa; un oggetto *S1* di tale tipo, può essere rappresentato dai valori:

```
S1:[34507, "Giovanni","Bianchi",["Via Garibaldi","10","Roma"],["06-12341234",
"335-12341234"],*25-02-1998*,"BNCGVN98B25F205W",*20-05-2014*,
["Terza","B"],{},{},{},[null,null]]
```

dove si ipotizza che le date debbano essere qualificate inserendole in una coppia di asterischi, che i campi di un record siano elencati tra una coppia di parentesi quadre, e che un insieme di elementi sia rappresentato elencando il valore degli elementi che lo compongono tra una coppia di parentesi graffe.
S1 descrive uno studente di nome *Giovanni Bianchi*, con matricola *34507* che abita a *Roma*, in *Via Garibaldi* al numero civico *10*, è nato il *25 febbraio 1998*, ha codice fiscale *BNCGVN98B25F205W* e si è iscritto il *20 maggio 2014* alla classe *Terza* sezione *B*. Lo studente descritto non ha per ora alcun voto attribuito, non ha fatto assenze e non sono registrati i voti di fine anno e il giudizio finale.

Si può inoltre dire che, in relazione ai recapiti telefonici, valgono le seguenti uguaglianze:

```
S1.Telefono.Casa == "06-12341234"
S1.Telefono.Mobile == "335-12341234"
```

Per interagire con un oggetto di tipo *Studente* bisogna utilizzare appositi metodi che compongono la definizione del tipo stesso. Nella terminologia della programmazione ad oggetti, questo tipo di dato prende il nome di **classe**. Di seguito sono elencati a titolo di esempio, alcuni tra i metodi previsti per manipolare un oggetto di tipo *Studente*:

```
void creaStudente( ValoriAllaCreazione )
void modificaStudente( NuoviDati )
void eliminaStudente()
. . .
void inserisciVoto( Materia,Voto,Data,Causale )
. . .
set votiFineAnno()
record giudizioFinale()
```

I metodi si concretizzano in procedure o funzioni, il cui codice è inserito nella definizione del tipo, mediante i quali il programmatore agisce su un oggetto. Per esempio, per attribuire un voto di profitto in un'interrogazione di *Matematica* allo studente identificato con l'oggetto *S1* bisogna usare il metodo *insersciVoto* nel seguente modo:

```
S1.inserisciVoto("Matematica",7.5,*25-11-2014*,"Interrogazione")
```

Per rappresentare la stessa entità nel modello relazionale si devono usare, per rispettare le regole di normalizzazione, le cinque tabelle elencate di seguito. Le righe di ogni tabella sono composte dai campi (o attributi) elencati tra parentesi dopo il nome della tabella. In ogni tabella compare, sottolineato, l'attributo che costituisce la chiave primaria della tabella e, in corsivo, l'attributo che permette di stabilire collegamenti tra i dati in diverse tabelle. Per esempio il campo *Matricola* nella tabella *Voti* permette di sapere a quale studente è stato attribuito un voto.

```
Studenti (Matricola, Nome, Cognome, Via, Numero, Città, TelCasa, TelMobile,
          Iscrizione, DataNascita, CodFisc, DataIscrizione, Classe, Sezione)
Voti (ID, Materia, Data, Causale, Voto, Matricola)
Assenze (ID, Data, Motivo, Note, Matricola)
VotiFinali (ID, Materia, Voto, Matricola)
Giudizio (ID, Promosso, Note, Matricola)
```

Il confronto fra i due modelli sviluppati permette di affermare che:
- il modello relazionale dei dati è meno aderente alla realtà del precedente oggetto;
- per ricostruire tutte le informazioni che compongono uno studente è necessario collegare tra di loro (con diverse operazioni di *join*) dati dispersi in molte tabelle;
- il modello relazionale dei dati non dice nulla sulle operazioni che si devono eseguire sui dati che descrivono uno studente.

Le precedenti considerazioni sono utili per comprendere i motivi che hanno portato allo sviluppo degli *OODBMS*, quali problemi essi risolvono, quali vantaggi (e svantaggi) essi presentano rispetto ai tradizionali *RDBMS*, in quali tipi di applicazioni sono indicati e in quali non sono indicati.

Motivazioni per l'introduzione degli OODBMS

Attualmente ci sono molte applicazioni sviluppate con un linguaggio di programmazione orientato agli oggetti, come Java, che però usano un tradizionale *RDBMS* per memorizzare i dati. Si nota allora che, usando un database relazionale per memorizzare i dati contenuti negli oggetti dell'applicazione, bisogna scomporre ogni oggetto nelle sue parti e, solo dopo quest'operazione, memorizzarlo nelle tabelle che compongono il database. Viceversa, per trasferire i dati memorizzati nel database relazionale negli oggetti usati dall'applicazione, bisogna ricostruire ogni singolo oggetto mediante operazioni di congiunzione (*join*) tra le tabelle. Tutto questo è oneroso, sia in tempo di elaborazione, sia per la necessità di sviluppare il codice che esegue l'impaccamento e lo spacchettamento dei dati. Il fenomeno è indicato come **impedance mismatch** (o *discordanza d'impedenza*). Essa si presenta per la necessità di far dialogare un linguaggio di programmazione che manipola le variabili di un singolo oggetto con il linguaggio SQL che agisce su intere tabelle e insiemi di righe. Per ovviare al *mismatch d'impedenza* si può pensare di memorizzare i dati in modo coerente con quello usato dai linguaggi di programmazione, come avviene negli *OODBMS*.

Vantaggi degli OODBMS rispetto ai RDBMS

- Gli oggetti sono un modello più adatto, rispetto ai record del modello relazionale, per rappresentare le entità complesse del mondo reale. Si pensi, per esempio, a un sistema CAD usato per progettare motori automobilistici e alla necessità di modellare la complessità di un motore.
- In un *OODBMS* si possono memorizzare un numero qualsiasi di tipi elementari, così come altri dati arbitrariamente complessi e non solo dati elementari. Per gestire le medesime informazioni in un RDBMS occorre usare molte piccole tabelle normalizzate, collegate tra loro mediante chiavi esterne.
- Eliminazione dell'*impedance mismatch* e della conseguente necessità di assemblare e disassemblare i dati contenuti negli oggetti, risparmiando tempo di esecuzione ed evitando di dover scrivere codice.
- In un OODBMS non c'è separazione tra la descrizione dei dati e la loro manipolazione, perché nel database è memorizzato non solo lo stato del sistema modellato, ma anche l'insieme delle operazioni ammissibili sui quei dati.

Svantaggi degli OODBMS rispetto ai RDBMS

- Gli OODBMS sono meno efficienti degli RDBMS quando si considerano realtà poco complesse e si devono considerare grandi quantità di dati semplici, come avviene, per esempio, nelle tipiche applicazioni gestionali.
- Gli oggetti usati negli OODBMS sono più complessi e più difficili da trattare rispetto ai record delle tabelle del modello relazionale.
- Nel mondo degli RDBMS ci sono molti *tools* disponibili e gli RDBMS sono di uso generalizzato.
- Il mondo degli OODBMS è meno standard e diffuso: in effetti esiste un linguaggio d'interrogazione standard per l'interrogazione dei database ad oggetti, **OQL** (*Object Query Language*), sviluppato da **ODMG** (*Object Database Management Group*); gli standard per gli *RDBMS,* tuttavia, sono più stabili di quelli degli *OODBMS*.

Dalle precedenti osservazioni segue che i database ad oggetti dovrebbero essere usati quando bisogna operare con dati molto complessi, oppure quando i dati sono correlati da associazioni complesse, per esempio, in applicazioni CAD/CAM.
Non è invece conveniente usare i database ad oggetti quando non sono necessari molti *join* tra le tabelle di un database relazionale per correlare i dati e ci sono grandi volumi di dati di tipo semplice generati dall'applicazione, come avviene, per esempio, in molte applicazioni di natura gestionale.