DENTRO LA SCATOLA

Rubrica a cura di

Fabio A. Schreiber

Dopo aver affrontato negli scorsi anni due argomenti fondanti dell'Informatica – il modo di codificare l'informazione digitale e la concreta possibilità di risolvere problemi mediante gli elaboratori elettronici – con questa terza serie andiamo ad esplorare "*Come parlano i calcolatori*". La teoria dei linguaggi e la creazione di linguaggi di programmazione hanno accompagnato di pari passo l'evolversi delle architetture di calcolo e di gestione dei dati, permettendo lo sviluppo di applicazioni sempre più complesse, svincolando il programmatore dall'architettura dei sistemi e consentendogli quindi di concentrarsi sull'essenza del problema da risolvere.
Lo sviluppo dell'Informatica distribuita ha comportato la nascita, accanto ai linguaggi per l'interazione tra programmatore e calcolatore, anche di linguaggi per far parlare i calcolatori tra di loro – i protocolli di comunicazione. Inoltre, la necessità di garantire la sicurezza e la privatezza delle comunicazioni, ha spinto allo sviluppo di tecniche per "non farsi capire" da terzi, di quì l'applicazione diffusa della crittografia.
Di questo e di altro parleranno le monografie quest'anno, come sempre affidate alla penna (dovrei dire tastiera!) di autori che uniscono una grande autorevolezza scientifica e professionale ad una notevole capacità divulgativa.

Linguaggi per Basi di Dati

Letizia Tanca

1. Introduzione

L'informatica può essere definita come la scienza che si occupa della *rappresentazione* e della *gestione* dell'informazione per mezzo di macchine digitali. Il problema della rappresentazione dell'informazione riguarda lo studio di metodi di descrizione appropriati all'elaborazione da parte di una macchina digitale; consiste perciò nell'*astrarre* i concetti importanti per la realtà applicativa di interesse da quelli trascurabili. Il problema della gestione riguarda invece l'uso e la trasformazione dell'informazione in maniera funzionale al problema applicativo da risolvere.
Ancora ai tempi di Turing, non era chiaro se fosse più opportuno utilizzare un modello di calcolo che distinguesse in maniera netta questi due aspetti; infatti, nella computazione basata sulla macchina di Von Neumann, che è diventata il modello standard di elaboratore, vi è una netta distinzione tra il ruolo dei programmi (preposti alla *gestione* dell'informazione) e quello dei dati (il cui ruolo è *rappresentarla*).
La macchina concreta proposta da Turing, in alternativa, prevedeva che i programmi potessero agire anche su parti del programma stesso, e così, eventualmente, automodificarsi; questo approccio è presente nella disciplina dell'Intelligenza Artificiale, i cui tipici linguaggi di programmazione hanno un paradigma che permette al programma di modificare se stesso. A parte questa disciplina, però, nell'informatica l'approccio "a-la-Von Neumann" è rimasto quello più comune, e ha dato luogo anche alla tradizione di mantenere in memoria centrale (o virtuale) il(i) programma(i) in elaborazione, mentre i dati, che di solito costituiscono un insieme piuttosto voluminoso, restano in memoria di massa, da "richiamare" all'occorrenza.
Anche per questo motivo le comunità degli esperti di rappresentazione dell'informazione e di gestione della stessa sono in qualche modo distinte: i primi si sono concentrati sulla ottimizzazione della organizzazione e dell'accesso a memoria di massa, per le caratteristiche di persistenza e di maggior quantità di spazio disponibile di quest'ultima.
L'articolo è organizzato come segue: dopo una breve carrellata sulle nozioni preliminari a pro-

posito delle basi di dati, introdurremo i modelli dei dati, e in particolare il modello relazionale; questa trattazione preliminare è necessaria, perché per parlare di linguaggi per basi di dati non si può prescindere dalle definizioni più importanti di questa disciplina. Ci concentreremo quindi sui linguaggi di interrogazione per le basi di dati relazionali, che costituiscono oggi il 90% dei sistemi in uso, e infine faremo anche un breve riferimento ai sistemi e ai linguaggi di interrogazione orientati agli oggetti.
Lo sforzo dei ricercatori e dei progettisti di linguaggi di interrogazione continua comunque a concentrarsi anche sullo studio di proprietà matematiche (logiche o algebriche) dei linguaggi, che richiede l'uso di formalismi appropriati. Per questo motivo sono stati da lungo tempo introdotti i cosiddetti linguaggi *formali* di interrogazione, cui sarà specificamente dedicato un altro articolo di questa serie.

2. Nozioni preliminari

Nei sistemi informatici (e non solo in essi), le *informazioni* vengono rappresentate attraverso i *dati*. Le *informazioni* sono le notizie, gli elementi che consentono di avere conoscenza più o meno esatta di fatti, situazioni, modi di essere. I dati sono ciò che è immediatamente presente alla conoscenza, prima di ogni elaborazione; in pratica, nell'informatica sono costituiti da simboli che debbono essere elaborati. Per esempio, il numero 30 è un dato, ma costituisce informazione solo se inquadrato in un contesto di riferimento: "Mario ha 30 anni" oppure "Giovanni ha ottenuto 30 all'esame di Informatica".
Una *base di dati* è una collezione di dati, utilizzati per rappresentare le informazioni di interesse per una o più applicazioni di una organizzazione. Una tale collezione di dati è però particolarmente efficace se se ne mette in evidenza la capacità informativa, organizzandola secondo una struttura, detta *modello dei dati*, mantenuta e gestita da un apposito sistema. Un modello dei dati è un insieme di costrutti utilizzati per organizzare i dati di interesse e descriverne la dinamica; i componenti fondamentali di un modello dei dati sono perciò i suoi meccanismi di strutturazione (o costruttori di tipo). Per esempio, il modello relazionale prevede il costruttore *relazione* (tabella), che permette di strutturare la base di dati in insiemi di record che hanno tutti la stessa forma. Tale struttura realizza, di fatto, una organizzazione *logica* dei dati, che permette di *trasformare i dati stessi in informazioni*. Con riferimento all'esempio citato sopra, il dato "30", ritrovato nel record relativo all'esame di Informatica dello studente Giovanni, nel campo VOTO, fornisce l'informazione che Giovanni ha ottenuto 30 all'esame.
Un Sistema di Gestione di Basi di Dati (*DataBase Management System*, DBMS) ha il compito di:
- ❑ mantenere i dati strutturati secondo un modello dei dati;
- ❑ condividere i dati tra più applicazioni: poiché le stesse informazioni, eventualmente diversamente organizzate, possono essere utilizzate da applicazioni diverse, è utile mettere i relativi dati a fattor comune, per evitare ridondanze;
- ❑ controllare la concorrenza di vari utenti sugli stessi dati: la base di dati è una risorsa condivisa, ma per motivi di efficienza occorre elevare il più possibile il livello di concorrenza; ciò vuol dire che il controllo di concorrenza deve scendere a un livello di granularità molto basso, eventualmente del singolo record;
- ❑ mettere a disposizione meccanismi di definizione della privatezza dei dati e di limitazioni all'accesso (autorizzazioni);
- ❑ garantire l'affidabilità sia delle operazioni che vengono svolte sui dati, sia del dispositivo di memorizzazione;
- ❑ ultimo compito, ma tra i più importanti: permettere l'accesso ai dati in lettura e scrittura. A tale scopo, il primo passo consiste nel ritrovare i dati mediante opportuni linguaggi, detti di *interrogazione*. Come vedremo, interrogare una base di dati mediante questi linguaggi permette di accedere in modo diretto ai dati *basandosi sulle loro proprietà*. Su questo aspetto ci concentreremo in questo articolo.

Come già detto, un *modello dei dati* è una collezione di concetti che possono essere usati per rappresentare la realtà: oltre all'esempio delle tabelle, possiamo a questo scopo pensare ad alberi, grafi ecc.. I modelli costituiscono quindi una strutturazione semplificata della realtà, che ne accoglie aspetti specifici e aiuta a comprenderla meglio.
Lo *schema dei dati*, invece, serve per rappresentare una specifica parte della realtà di interesse, mediante un modello dei dati. Per esempio, un modello ad albero si presta in modo naturale a rappresentare l'albero genealogico o il

tracciato di un fiume; un modello a grafo ben si presta a rappresentare il sistema fluviale o il sistema stradale di un Paese, mediante il modello relazionale organizzo in tabelle gli studenti con i relativi esami e voti.
A questo punto, avendo deciso qual è il modello dei dati più opportuno, ed avendo deciso come modellare la realtà di interesse progettando uno schema, i dati veri e propri, cioè il contenuto informativo, costituiscono l'*istanza* della base di dati, cioè una collezione di valori che *rispetta la struttura dettata dallo schema*. Per esempio, avendo deciso di rappresentare mediante un albero la genealogia delle famiglie, la famiglia Tanca e la famiglia Rossi saranno istanze di alberi genealogici. La rete viaria della Germania come è nella cartina edita il giorno 22 febbraio 2006 sarà un'istanza dello schema di rete viaria basato sul modello a grafo, mentre i dati sugli studenti e gli esami del Politecnico di Milano, in questo preciso istante, sono un'istanza della base di dati relazionale rappresentante studenti ed esami di cui sopra. Possiamo anche dire che, mentre lo schema descrive i dati a livello *intensionale*, fornendone, cioè, le *proprietà strutturali*, l'istanza ne esibisce l'aspetto *estensionale*, e cioè *costituisce l'informazione stessa*.
In un DBMS i dati si possono vedere, o meglio descrivere, a livelli diversi. Questa possibilità permette di liberare da una parte il progettista della base di dati, dall'altra il programmatore delle applicazioni, dalla preoccupazione di come i dati sono fisicamente organizzati in memoria di massa, concentrandosi sul loro significato piuttosto che sulla loro organizzazione fisica.
Il più alto livello di astrazione è il livello *esterno*, o *delle applicazioni*. A questo livello, i programmi applicativi vedono i dati organizzati secondo il modello dei dati del DBMS prescelto (o anche, eventualmente, secondo modelli diversi), ma con una visione personalizzata, detta *vista*, che permette di trascurare dati non di pertinenza di quell'applicazione, riorganizzando i dati di pertinenza in maniera appropriata e allo stesso tempo realizzando il controllo della privatezza delle informazioni. Il livello *logico*, immediatamente sottostante, descrive i dati, sempre in accordo col modello dei dati del DBMS, ma secondo una visione comune, che prevede l'organizzazione dei dati stessi in uno schema unico. Infine, il livello *fisico* descrive il modo in cui i dati sono organizzati nel supporto fisico di memorizzazione, tipicamente il disco magnetico, che però è oggigiorno affiancato anche da altre tecnologie, come i dischi ottici o le memorie flash.

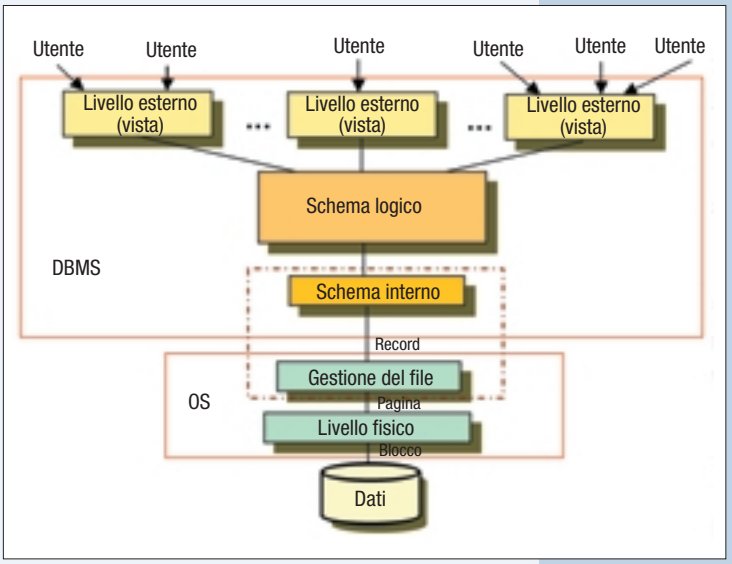

FIGURA 1
Architettura ANSI/SPARC delle moderne basi di dati

La figura 1 mostra la cosiddetta Architettura ANSI/SPARC delle moderne basi di dati. Il sistema di gestione della base di dati si fa carico di mantenere lo schema logico dei dati indipendente dalle strutture fisiche di memorizzazione, realizzando dunque una proprietà fondamentale delle moderne basi di dati, la cosiddetta *indipendenza dei dati*. L'accesso avviene infatti solo tramite il livello esterno (le cosiddette *viste d'utente*)che può, nel caso di un sistema informativo semplice, con esigenze poco diversificate, coincidere con il livello logico. Il DBMS garantisce il disaccoppiamento tra il livello esterno e quello logico (*indipendenza logica*), facendo in modo che aggiunte o modifiche alle viste non richiedano modifiche al livello logico e viceversa, e tra il livello logico e quello fisico (*indipendenza fisica*), facendo in modo che le modifiche allo schema logico lascino inalterata l'organizzazione fisica dei dati e viceversa. Coerentemente con quanto detto, i DBMS prevedono due tipi principali di modelli dei dati: i *modelli fisici*, che servono per organizzare i dati in memoria di massa tenendo conto della struttura a blocchi dei dischi, e i *modelli logici*, che vengono usati dai programmi e dagli umani per riferirsi ai dati descrivendoli mediante uno schema logico, sia al livello logico che al livello esterno (o *delle viste d'utente*).

I modelli fisici sono utilizzati nei DBMS esistenti, ma ad essi fa riferimento solo il DBMS per realizzare le strutture fisiche di memorizzazione (*schema interno*); esempi di tali modelli sono strutture indicizzate come alberi bilanciati (B-tree, B+-tree), file sequenziali, file ordinati, file organizzati con strutture hash ecc.. I modelli logici sono pure utilizzati nei DBMS esistenti, ma ad essi fanno riferimento i programmi, i progettisti e gli utenti finali per vedere i dati organizzati in accordo con il loro significato.

3. Il modello relazionale dei dati

I tre modelli logici dei dati tradizionali sono il modello *gerarchico*, quello *reticolare* e quello *relazionale*. I modelli gerarchico e reticolare sono basati, rispettivamente, su strutture ad albero e su strutture a grafo. Sono modelli in realtà abbastanza vicini alle strutture fisiche di memorizzazione, poiché in essi i record contengono al loro interno dei puntatori che fanno da riferimento ad altri record ad essi collegati. Per tornare all'esempio degli studenti, dei corsi e degli esami, in una base di dati gerarchica o reticolare ogni record relativo a un esame conterrebbe dei puntatori, rispettivamente, al record del corso e a quello dello studente. Si realizza perciò una struttura logica "sovrapposta" a quella fisica, con una evidente perdita di indipendenza dalla struttura di memorizzazione.
Il modello relazionale è più astratto e meglio si presta a realizzare tale proprietà, in quanto in tale modello i record non possono contenere puntatori ad altri record, ma i riferimenti fra dati in record appartenenti a strutture (relazioni) diverse sono rappresentati per mezzo dei valori dei dati stessi. Per esempio, nel caso relazionale il record dell'esame contiene, oltre al voto e alla data, anche il codice del corso e la matricola dello studente, che implementano perciò un riferimento basato su valori.
Negli ultimi 15 anni, sulla scia dei linguaggi di programmazione orientati agli oggetti, è stato studiato anche il modello a oggetti. Il modello a oggetti è un ibrido tra queste due concezioni, in quanto, da una parte, si colloca a un alto livello di astrazione: il concetto di classe e quello di oggetto fanno infatti riferimento a quello di tipo di dato astratto, un'astrazione, appunto, che permette di riunire in sé la definizione della struttura di dati e quella dei sottoprogrammi che implementano le operazioni. D'altra parte, l'oggetto ha un suo identificatore che permette ai programmi e agli altri oggetti di far riferimento ad esso, e che in un certo senso svolge il ruolo del puntatore esplicito dei modelli reticolare e gerarchico. In questo articolo ci concentreremo sui *linguaggi per basi di dati relazionali*, che costituiscono ormai l'80% dei sistemi di gestione di basi di dati esistenti. A tale scopo, introdurremo brevemente il modello relazionale dei dati.
Il modello relazionale fu proposto nel 1970 da Edgar F. Codd, in un articolo fondamentale[1], per favorire l'indipendenza dei dati, ma reso disponibile come modello logico in DBMS reali nel 1981; non è facile infatti implementare l'indipendenza mantenendo i necessari livelli di efficienza e affidabilità. Esso si basa sul concetto matematico di *relazione*.
In matematica, si dice relazione un qualsiasi sottoinsieme del prodotto cartesiano di uno o più insiemi, detti *domini*. Il numero di tali insiemi è detto *grado*, o *arità*, della relazione, mentre il numero dei suoi elementi è la sua *cardinalità*. Consideriamo per esempio tre insiemi: l'insieme M di tutte le possibili matricole di studenti, composto da stringhe alfanumeriche di lunghezza 6, i cui primi due caratteri sono alfabetici e i restanti quattro sono numerici; l'insieme C di tutti i possibili codici di corsi, composto da stringhe numeriche di lunghezza 5, e l'insieme V dei voti possibili, cioè i numeri compresi tra 18 e 31 (che rappresenterà il 30 e lode). Una relazione matematica su $M \times C \times V$ sarà un qualsiasi sottoinsieme di questo prodotto cartesiano, e avrà grado 3. I suoi elementi saranno le terne *ordinate* che al primo posto hanno una matricola, al secondo un codice corso e al terzo un valore compreso tra 18 e 31. Un esempio di relazione è presente in tabella 1.

AB3145	56747	28
AC1455	44567	24
BA3768	98709	30
AG5676	88076	19

TABELLA 1

1 E. F. Codd: A Relational Model of Data for Large Shared Data Banks. *Commun. ACM*, Vol. 13, n. 6, 1970, p. 377-387.

Nel modello relazionale si apportano alcune variazioni a questo concetto fondamentale. Innanzitutto, una relazione *avrà sempre un numero finito di elementi*. Inoltre, mentre le n-uple di una relazione matematica sono ordinate, cioè l'*i*-esimo valore di ciascuna proviene dall'*i*-esimo dominio, nel modello relazionale dei dati si preferisce identificare i domini mediante dei *nomi*, detti *attributi*, che permettono di riferirsi ai valori presenti in una *n*-pla senza dover conoscere la loro posizione, che diventa irrilevante. Nel modello relazionale, le *n*-ple così ottenute si chiamano *tuple*.
A questo punto, una relazione è un insieme di tuple, e può essere appropriatamente visualizzata come una tabella dove:

1. i valori di ciascuna colonna sono fra loro omogenei (cioè appartengono allo stesso dominio),
2. le righe (tuple) sono diverse fra loro (ricordiamo che si tratta di un insieme),
3. le intestazioni delle colonne sono diverse tra loro, e perciò i campi sono distinguibili mediante il loro nome e non la loro posizione. Sempre pensando al nostro esempio, le due relazioni presenti in tabella 2 e in tabella 3 sono perfettamente equivalenti.

Si vede ora che, come già abbiamo avuto occasione di dire, il modello relazionale è *basato su valori*, cioè i riferimenti fra dati in relazioni diverse sono rappresentati per mezzo di valori dei domini che compaiono nelle tuple. In tal modo si realizza la desiderata indipendenza dalle strutture fisiche, che possono cambiare dinamicamente, si rappresenta solo ciò che è rilevante dal punto di vista dell'applicazione (dell'utente), i dati sono portabili piu' facilmente da un sistema ad un altro, e i riferimenti sono bidirezionali, contrariamente al caso dei puntatori, che sono direzionali. Ovviamente, a livello fisico le strutture di memorizzazione sfruttano puntatori per quanto necessario all'implementazione efficiente delle operazioni, in modo completamente trasparente all'utente.
Data una relazione R sui domini $D1$, ...Dn, lo *schema di R* è dato dal nome R seguito dall'insieme dei suoi attributi $A1$,..., An, e si indica con R ($A1$,..., An). Un insieme di schemi di relazione con nomi diversi:
$R = \{R1 (\_X1\_), ..., Rn (\_Xm\_)\}$ (ove ***_Xi_*** è il vettore degli attributi della relazione Ri) costituisce lo *schema di una base di dati*. Un'*istanza di relazione* su uno schema R (X) è un insieme di tuple su X, mentre un'*istanza di base di dati* su uno schema $R = \{R1 (X1), ..., Rm (Xm)\}$ è un insieme di m istanze di relazioni sugli elementi di R.

Codice corso	Matricola	Voto
56747	AB3145	28
44567	AC1455	24
98709	BA3768	30
88076	AG5676	19

TABELLA 2

Matricola	Codice corso	Voto
AB3145	56747	28
AC1455	44567	24
BA3768	98709	30
AG5676	88076	19

TABELLA 3

Un altro concetto molto importante è quello di *chiave di una relazione*: si tratta di un insieme (minimale) di attributi che identifica univocamente le tuple di quella relazione. Una piccolissima base di dati relazionale, contenente informazioni su studenti, corsi ed esami, potrebbe avere il seguente schema, dove gli attributi sottolineati costituiscono le chiavi:

CORSI (CodiceCorso, NomeCorso,Docente,Semestre,N_Crediti)
STUDENTI (MatricolaStudente, Nome, Cognome, Indirizzo, CorsoDiLaurea,Anno)
ESAMI (MatricolaStudente,CodiceCorso,Voto)

4. Linguaggi di interrogazione per basi di dati relazionali

Come si è detto in precedenza, uno dei compiti più importanti del DBMS è permettere ai suoi utenti, siano essi umani o programmi, di ritrovare i dati mediante i *linguaggi di interrogazione*, che consentono di accedere in modo diretto ai dati *basandosi sulle loro proprietà*.
Supponiamo di essere interessati a trovare i nomi e cognomi di tutti gli studenti che hanno ottenuto almeno 28 all'esame di Fondamenti di Informatica. Piuttosto che cercare nella tabella CORSI il codice di Fondamenti di Informatica, e poi scandire i blocchi del file ESAMI, alla ricerca dei voti maggiori di o uguali di 28 che corrispondono al codice trovato, e poi ancora cercare nella tabella

STUDENTI i nomi e cognomi corrispondenti alle matricole rintracciate, un utente certamente preferisce semplicemente enunciare, in un opportuno linguaggio: "voglio trovare tutti gli studenti che hanno ottenuto almeno 28 all'esame di Fondamenti di Informatica", così *dichiarando* quale proprietà i dati richiesti debbano soddisfare.
Oltre che per interrogare, si ha bisogno di interagire con la base di dati per altre due ragioni fondamentali. Da una parte, dobbiamo essere in grado di creare, modificare ed eventualmente cancellare *componenti dello schema*, dall'altra, invece, dobbiamo essere in grado di inserire, modificare e cancellare *tuple*, cioè *l'istanza* della base di dati. Qualunque sia il modello dei dati sottostante, i linguaggi per basi di dati sono dunque tradizionalmente composti da due ingredienti fondamentali: il *Linguaggio di Definizione dei Dati* (*Data Definition Language*, DDL) e il *Linguaggio di Manipolazione dei Dati* (*Data Manipulation Language*, DML), operanti, rispettivamente, sullo schema e sull'istanza della base di dati. Il linguaggio di interrogazione, che anch'esso opera sull'istanza, è parte del DML.
Il linguaggio di interrogazione costituisce però la parte più importante di questo complesso, poiché l'attività di interrogare è di gran lunga la più frequente, e richiede di realizzare i meccanismi per *accedere* ai dati, necessari anche a tutte le operazioni di modifica. Questa è quindi la parte che richiede un maggiore sforzo da parte dei progettisti del linguaggio, sia dal punto di vista dell'ottimizzazione, sia per concepire linguaggi la cui semantica sia ben chiara all'utente.
Un linguaggio di interrogazione *dipende dal modello logico dei dati*: si intuisce infatti che, per quanto dichiarativa, la struttura della interrogazione debba far riferimento alla percezione che l'utente ha della struttura dei dati. Per i modelli logici basati su riferimenti, come quello gerarchico, il reticolare, o il modello a oggetti, l'utente dovrà comunque, nello scrivere una interrogazione, far riferimento ai legami tra gli oggetti; viceversa, in un modello basato su valori, come quello relazionale, poiché tali legami sono espressi attraverso i dati stessi, essi vengono introdotti come condizioni sui dati, analogamente alle proprietà che i dati debbono soddisfare. Per esempio, la specifica dell'interrogazione "trovare tutti gli studenti che hanno ottenuto almeno 28 all'esame di Fondamenti di Informatica" conterrà anche l'indicazione che il codice dell'esame cercato nella tabella ESAMI dovrà essere uguale al codice del corso di Fondamenti di Informatica trovato nella tabella CORSI.
Lo sforzo dei ricercatori e dei progettisti di linguaggi di interrogazione si è da lungo tempo concentrato sul concepire linguaggi *dichiarativi*, linguaggi cioè che mascherano i dettagli dei meccanismi di esecuzione permettendo all'utente di trascurare l'aspetto implementativo e concentrarsi sulla natura dei dati ricercati. Per questo motivo, al lavoro dei progettisti di DBMS si è associato quello di generazioni di ricercatori, che hanno escogitato linguaggi *formali* di interrogazione, che, pur mantenendo l'equivalenza con i costrutti dei linguaggi adottati dai DBMS commerciali, si lasciassero studiare in termini di proprietà matematiche (logiche o algebriche), prestandosi a ragionamenti non ambigui. A questi linguaggi sarà specificamente dedicato un altro articolo di questa serie.
Si noti che *il risultato di qualsiasi interrogazione a una base di dati relazionale è sempre una relazione*, cioè un insieme di tuple che soddisfa le proprietà specificate.
Tra i linguaggi commerciali per basi di dati, ne sono stati concepiti di due categorie fondamentali. La prima comprende i *linguaggi testuali*, che specificano l'interrogazione in termini di stringhe di testo, analogamente alla maggior parte dei linguaggi di programmazione. Esempio principe di questa categoria è il linguaggio relazionale SQL (*Structured Query Language*), definito presso i laboratori IBM di San Jose nella sua prima versione negli anni '70 per il primo prototipo di DBMS relazionale, *System R*.
Un esempio di specifica dell'interrogazione di prima in SQL è il seguente:

```
SELECT Nome, Cognome
FROM STUDENTI,ESAMI,CORSI
WHERE STUDENTI.Matricola=ESAMI.Matricola AND
ESAMI.CodiceCorso=CORSI.CodiceCorso AND
CORSI.NomeCorso="Fondamenti di Informatica"
and ESAMI.Voto>=28.
```

SQL è un linguaggio molto potente, in grado di esprimere interrogazioni molto sofisticate ed anche alcune forme elementari di aggregazione, come la media, la somma, e funzioni semplici come il minimo e il massimo di un insieme. Bisogna però stare attenti a non confondere il potere espressivo dei linguaggi di interrogazione per basi di dati con quello dei linguaggi di program-

mazione. Da questi ultimi, infatti, si richiede un potere espressivo pari a quello della macchina di Turing, poiché mediante un tale linguaggio occorre poter esprimere tutte le funzioni computabili. Da un linguaggio di interrogazione, invece, si richiede di essere in grado di esprimere *tutte le possibili condizioni* sui dati stessi, e perciò il confronto di espressività si fa di solito con la *logica del primo ordine*. Senza scendere in dettagli formali, per i quali si rinvia il lettore interessato ai libri di testo di basi di dati elencati in bibliografia, basti sapere che, nel caso dei linguaggi classici per basi di dati relazionali, ci si accontenta di un potere espressivo inferiore (per esempio non è possibile esprimere cicli), e per le computazioni si ricorre all'*immersione* delle interrogazioni in programmi tradizionali. Nasce così il concetto di linguaggio "embedded", espressione che indica il fatto che l'interrogazione viene inserita all'interno di un programma scritto in un linguaggio di programmazione "ospite", come ad esempio C o Java. Ciò accade quando l'interrogazione non viene effettuata da un utente in modo interattivo, ma è inserita in un programma applicativo che interagisce con la base di dati.

Una categoria di linguaggi commerciali interessante, ma utile solo in determinati casi, è quella dei linguaggi *grafici o visuali*, che forniscono interfacce "amichevoli", basate su una visualizzazione della forma che la tabella risultato dell'interrogazione dovrebbe avere. Un esempio di tali linguaggi è *Query By Example* (QBE), la cui più famosa realizzazione commerciale è Microsoft Access©. Come si può intuire, oltre al difetto di non poter essere immersi in linguaggi di programmazione, questi linguaggi hanno il problema di essere, sì, molto facili da usare nel caso di interrogazioni relativamente semplici, ma di avere una semantica poco intuitiva nel caso di interrogazioni complesse, e pertanto di essere in generale proni ad errori.

Nonostante la potenza dei linguaggi di interrogazione relativamente alle operazione di specifica di dati da cercare, molti ricercatori non sono comunque soddisfatti del loro potere espressivo. Il motivo principale di ciò è un'anomalia che insorge nell'uso di questi linguaggi nei programmi applicativi, quando vengono immersi nei linguaggi di programmazione. Si è detto che la risposta a qualsiasi interrogazione relazionale è una relazione, il che si esprime dicendo che i linguaggi di interrogazione relazionali sono *orientati agli insiemi*; i linguaggi di programmazione tradizionali gestiscono però i record uno alla volta (sono, come si dice, *orientati alle tuple*), cioè un programma interagente con la base di dati non "si aspetta" che la risposta alla interrogazione sia un insieme, ma un semplice valore, o un record.

Supponiamo ad esempio di voler inserire la nostra interrogazione in un programma applicativo che stampi l'elenco degli "Studenti Bravi", definendo come bravi coloro che hanno ottenuto almeno 28 all'esame di Fondamenti di informatica. Il programma non è in grado di ricevere e stampare direttamente l'intero set di studenti che soddisfano la proprietà richiesta, ma dovrebbe contenere un ciclo che li stampa uno per volta; questo problema viene chiamato *disaccoppiamento di impedenza* (*impedance mismatch*).

La soluzione comunemente adottata è l'uso dei *cursori*, un concetto che fa parte da tempo dell'SQL standard. Un cursore accede al risultato di una interrogazione in modo "set-oriented", e restituisce le tuple al programma una per volta. Il frammento di programma seguente svolge l'operazione richiesta.

```
...
exec sql begin declare section;
char Nome[20], Cognome[20];
exec sql end declare section;
exec sql declare StudentiBravi cursor for

SELECT Nome, Cognome
FROM STUDENTI,ESAMI,CORSI
WHERE STUDENTI.Matricola=ESAMI.Matricola AND
ESAMI.CodiceCorso=CORSI.CodiceCorso AND
CORSI.NomeCorso="Fondamenti di Informatica" AND
ESAMI.Voto>=28

exec sql open StudentiBravi;
exec sql fetch StudentiBravi into :Nome,:Cognome;
printf("StudentiBravi");
while (sqlca.sqlcode == 0)
{
printf("Nome Studente: %s %s ",Nome,Cognome);
exec sql fetch StudentiBravi into :Nome,:Cognome;
}
exec sql close StudentiBravi;
}
...
```

Il disaccoppiamento di impedenza è un problema interessante, la cui soluzione pratica non

soddisfa molti ricercatori, e che ha spinto la comunità delle basi di dati a svolgere molta ricerca. Alcuni tentativi di soluzione saranno discussi nell'articolo sui linguaggi formali per basi di dati; invece, per quanto riguarda i linguaggi commerciali, una possibile soluzione è costituita dai linguaggi e i DBMS orientati agli oggetti. Questi sistemi offrono alle aree applicative più avanzate, servite talvolta in modo insoddisfacente dai sistemi tradizionali, vantaggi indubbi, sia dal lato della ricchezza di strutture messe a disposizione dal modello dei dati, sia da quello della capacità semantica di modellare dati e comportamenti in maniera compatta. Per esempio, nei DBMS orientati agli oggetti il disaccoppiamento di impedenza viene superato dal fatto che il paradigma del linguaggio di programmazione è lo stesso del linguaggio di interrogazione, perciò o sono entrambi orientati alle tuple o entrambi orientati agli insiemi. D'altra parte, mentre negli ultimi anni i linguaggi di programmazione orientati agli oggetti, in particolare Java, si sono diffusi nell'uso comune, proprio a causa della loro estrema flessibilità e capacità di modellazione, i sistemi di gestione di basi di dati orientate agli oggetti non hanno acquistato lo stesso peso nel mondo dell'informatica. Essi presentano infatti una difficoltà di implementazione che supera di gran lunga quella di un linguaggio di programmazione: in aggiunta alla complessità del linguaggio stesso, un sistema di gestione di basi di dati deve gestire le informazioni in memoria persistente, e quindi risolvere gli usuali problemi di efficienza, consistenza, di condivisione di dati, di sicurezza, di privatezza, di eventuale distribuzione dei dati su più siti ecc.. Anche le difficoltà teoriche non sono banali: si pensi per esempio al concetto di incapsulamento nei linguaggi di programmazione orientati agli oggetti, e a come questo contrasti proprio con il principio fondante i linguaggi di interrogazione, che è quello di poter reperire tutte le informazioni necessarie in modo diretto, basandosi su proprietà logiche, senza passare per uno specifico metodo. Un altro punto teorico un po' critico è costituito dal ritorno, nei linguaggi a oggetti, del collegamento tra concetti basato su riferimenti, che contraddice l'approccio dichiarativo così caro agli specialisti delle basi di dati.

5. Conclusioni

Per concludere, vorremmo far notare l'importanza della comprensione, per tutti gli informatici, dei concetti fondamentali delle basi di dati e dei loro linguaggi. Questo mondo, infatti, mentre da una parte condivide molti principi di quello dei linguaggi di programmazione, spesso se ne discosta per le necessità specifiche dell'obiettivo che si propone: si pensi alla necessità di linguaggi dichiarativi, strettamente legata all'esigenza di mantenere l'indipendenza fisica, o alle necessità di ottimizzazione affatto diverse da quelle di ottimizzazione dei programmi.

Bibliografia

Elencare i riferimenti ai concetti, modelli e linguaggi introdotti comporterebbe una lista infinita; indichiamo qui soltanto alcuni testi classici sulle basi di dati, dai quali il lettore potrà desumere ulteriori riferimenti bibliografici.

Atzeni P., Ceri S., Paraboschi S., Torlone R.: B*asi di dati: modelli e linguaggi di interrogazione*. 2ª edizione, Mc Graw Hill Italia, 2006.

Elmasri R., Navathe S.: *Sistemi di basi di dati. Fondamenti*. 4ª edizione. Pearson/Addison Wesley, 2004.

Letizia Tanca è Professore Ordinario di Basi di Dati presso il Politecnico di Milano, e presidente del Consiglio di Corso di Studi di Ingegneria Informatica del Politecnico di Milano, campus Milano Leonardo. È autrice di diverse pubblicazioni internazionali sulle basi di dati e sulla teoria delle basi di dati, e del libro "Logic Programming and Databases", scritto con S. Ceri e G. Gottlob. Ha partecipato a vari progetti nazionali e internazionali. I suoi interessi di ricerca riguardano tutta la teoria delle basi di dati, in particolare le basi di dati deduttive, attive e orientate agli oggetti, i linguaggi a grafi per basi di dati, la rappresentazione e l'interrogazione di informazione semistrutturata, le basi di dati per piccoli dispositivi.
E-mail: tanca@elet.polimi.it