di Raffaello De Masi

Che cosa è un Database?

Lo studio delle caratteristiche generali del Prolog si sta avviando rapidamente al termine; abbiamo fino a oggi analizzato i tool a disposizione di questo strano linguaggio, così diverso da qualunque altro, anche nell'ambito di quelli dedicati alla Intelligenza Artificiale, e abbastanza facile da usare tanto da soppiantare, come sta facendo, molti ben più noti e blasonati.
Ma per assolvere alla sua vera struttura di manipolatore di dati, Prolog deve «sposarsi» a strutture di dati, a database contenenti conoscenza! È la sorte di ogni linguaggio di programmazione dedicato alla Intelligenza Artificiale

Negli ultimi anni la manipolazione e la strutturazione dei dati è divenuta una delle maggiori aree di interesse a mano a mano che le capacità di manipolare masse di dati complesse da parte delle macchine sempre più potenti presenti sul mercato, per uno strano (anche se non incomprensibile) fenomeno di autoesaltazione aumentavano i campi di applicazione dei database.

Vediamo come questo viene fatto da Turbo Prolog. Questo linguaggio, è costruito in modo da mettere a disposizione i tool ideali per la programmazione dei database, in particolare per la costruzione e manipolazione di database relazionali che, allo stato attuale dell'arte, sono le strutture più efficienti disponibili oggi.

Nella sua più semplice forma, una base di dati è una collezione di fatti. Generalmente questa collezione copre un solo argomento, ma non si tratta di una condizione vincolante; ciononostante un database che manipola diversi argomenti non sempre è utile.

Poiché un database consiste in una collezione di fatti, e linguaggi come Prolog sono costruiti di clausole che maneggiano fatti e regole (si tenga conto che le regole sono una particolare forma di fatti), appare implicito che Prolog è lo strumento ideale per manipolare strutture di dati.

Ma che cosa è un database relazionale? Vediamo la risposta attraverso una definizione e un esempio. Un DB relazione generalmente usa, per i suoi calcoli, più di una struttura principale di dati; tanto, per capirci, se tutti i dati su cui è necessario lavorare sono compresi in un solo file, la relazionalità non è necessaria; se invece occorre prelevare dati da due o più file diversi, per legarli insieme magari in uno nuovo, allora la relazionalità diviene praticamente insostituibile.

E passiamo ad un esempio; nel 1970 E.F. Codd, un ricercatore della IBM formulò per la prima volta il concetto di relazionalità, così come è oggi intesa.

Come la maggior parte delle grandi scoperte (la relazionalità è una delle più grandi, nel campo della manipolazione di file), anche questa scaturì da una osservazione nel complesso banale. Codd si accorse semplicemente che le relazioni tra singoli parti di un record e i campi in esso presenti possono essere rappresentate utilizzando un semplice array bidimensionale, chiamato tabella, o tavola. Un esempio di struttura di semplice database è rappresentato nella figura a.

La matrice è il sistema più semplice per disegnare e manipolare una tabella. Codd individuò una tabella come composta di entità (righe della tabella, corrispondenti ai record del file), e attributi (il contenuto delle colonne, in altri termini, i campi del record).

Ogni entità, nella tabella dell'esempio, ha quattro elementi (tradotto in linguaggio di database, ogni record ha quattro campi); in gergo diremo che la tabella è di grado 4. Essa può essere utilizzata come tabella di corrispondenza di tipo cartesiano per individuare il contenuto di quel campo in quel record. Ma la cosa interessante, nella tabella è che è possibile mettere insieme dati di diverse entità in base a caratteristiche comuni, come:

- ogni riga rappresenta fedelmente un esempio della struttura del file.
- Ogni colonna contiene un solo tipo di dati.
- Ogni riga, sebbene costruita secondo uno schema comune, è individuale e specifica nelle sue caratteristiche.

L'ordine con cui le colonne sono sistemate non è significativo né determinante. Vediamo adesso, sempre attraverso un esempio, come un database, divenendo relazionale, aumenta la sua efficienza e razionalità.

Immaginiamo di costruire un database per gli articoli di un magazzino di vendita per corrispondenza; possiamo provare a pensare a quello in figura, più ampliato e articolato; ogni articolo è caratterizzato da una serie di parametri, come numero di catalogo, descrizione, parte, colore, peso, lunghezza, larghezza, altezza, taglia, ecc. Se strutturassimo il database secondo un disegno convenzionale dovremmo per ogni articolo costruire un record contenente questi campi.

Ma non è detto che tali campi siano obbligatori per tutti gli articoli. Ad esempio, sarebbe inutile, probabilmente assurdo, includere tutti i dati descritti prima per qualcosa come un vestito o un tubetto di colla: nel primo caso potrebbe avere poca importanza il peso, nel secondo sarebbe assurdo parlare di taglia o di colore, per una chiave inglese probabilmente ben pochi dei campi sarebbero occupati e ne potrebbe servire un altro con le caratteristiche di apertu-

cat.	descriz.	cliente	n. pezzi
11.2245	matita	AAA	25000
12.455A	colla	AAB	120
12.44.51	carta	AAS	2000
AN.577	etich.	RRA	18000
Y455.U	filo	ASZ	1500

Figura a
Un esempio di base di dati strutturata in tabella.

ra della bocca dell'utensile. Tutto ciò porterebbe, utilizzando un database di struttura e tipo tradizionale, a inefficienza e ad uno spreco, magari enorme, di spazio. Viceversa se, sempre per motivi di spazio, si decidesse di rinunciare a certi campi, probabilmente alcune delle informazioni andrebbero perse e non sarebbero classificabili.

La soluzione è l'adozione di un database relazionale. Cosa vuol dire ciò: semplice, ragionando per assurdo, è sufficiente costruire non più un solo file ma tanti file ognuno per ogni parametro di registrazione; si avrà, da una parte, un armadio che magari occupa tutti o quasi tutti i campi necessari, dall'altro una caffettiera che ne occupa solo un paio. Sarà compito poi del programma accedere acconciamente a maggiori o minori informazioni a seconda della categoria merceologica analizzata.

Prolog è essenzialmente relazionale. In altre parole è costruito per manipolare relazioni presenti in diversi database. Quando, nel 1974 Codd descrisse il primo efficiente modello di database relazionale, non a caso si basò su una notazione abbastanza simile al Prolog. In linguaggio relazionale, il nostro catalogo di prodotti per corrispondenza sarebbe:

ARTICOLO (numero_di_catalogo, descrizione, caratteristiche, prezzo)
CARATTERISTICHE (lunghezza, larghezza, altezza, peso, colore)

Le strutture e le tabelle descritte in precedenza offrono una tecnica di gran pregio per la costruzione di database relazionali. Vediamo in che modo.

Secondo quanto abbiamo detto nelle puntate dedicate alla programmazione delle regole e dei fatti, è possibile aggiungere alla base di conoscenza nuove tecniche cognitive solo accedendo direttamente al sorgente e incrementando il numero di regole. Ciò non va bene, non solo per il problema intrinseco, ma anche perché solo una persona che comprende il Prolog può accedere al sorgente per aggiungere le nuove regole (ammesso che chi ha redatto il programma sia disposto a metterci a disposizione il listato sorgente).

Il problema, in fondo è che le strutture di dati sono dinamiche, vale a dire che i dati, le informazioni, per la loro stessa natura sono destinati a cambiare continuamente. Con ogni probabilità quello che era vero due anni fa è oggi falso o, non più vicino al vero come prima. Nel caso di database personali il nome, la data di nascita e l'altezza saranno sicuramente gli stessi, ma saranno cambiati molti altri dati, come l'età, il conto in banca, il numero dei figli, forse il numero di telefono, il modello di auto, e così via. Non è pensabile accedere ogni volta al programma per modificare le regole in esso contenute; è molto più semplice accedere e modificare un database.

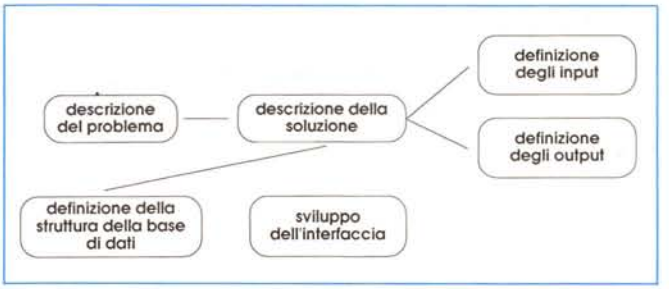

Figura b
Criterio di sviluppo di un algoritmo per la realizzazione di un database relazionale.

Il processo di definizione e ottimizzazione di un database relazionale è l'argomento su cui gli esperti hanno speso la maggior parte delle loro energie dopo la definizione formale del Prolog. Ovviamente non tutti i problemi incontrati nella programmazione sono problemi riconducibili a maneggio di basi di dati. Alcuni, come calcolo numerico, grafica, suono hanno poco a che vedere con la manipolazione dei dati; altri, come word processing, non abbisognano di manipolazione strutturata dei dati stessi. Ma in problemi specifici legati alla manipolazione di grandi masse di informazioni, come analisi delle vendite, inventari, gestioni di registri di offerte e acquisti, paghe di operai, gestione di catalogo e magazzino, ecc. la gestione relazionale dei dati è pressoché insostituibile.

A prescindere dalla complessità del problema, dal numero e dalla grandezza delle variabili in gioco, il sistema di gestione di un db statico è piuttosto rigido e statico. Lo sviluppo grafico dell'algoritmo è espresso dalla figura b. La sequenza non è obbligatoria ma permette uno sviluppo abbastanza logico del problema.

Analizzando in dettaglio il diagramma, fermiamoci un momento sulle caratteristiche dei singoli blocchi. Descrivere bene un problema è aver raggiunto già il 50% della soluzione. Purtroppo è questo uno step del problema che può essere acquisito solo attraverso la pratica e l'esperienza. Comunque si tenga conto che anche programmatori con esperienza vasta e avanzata impiegano circa la metà del tempo totale per eseguire una descrizione accurata del problema; tutto il tempo impiegato in questa fase rappresenta un notevole guadagno.

Viene poi tutto il processo di sviluppo della soluzione: questa fase generalmente si sviluppa in tre diversi processi, fase di input, fase di analisi, modifica e sviluppo di questi, e fase di output. Anche qui balza evidente agli occhi la grande peculiarità del nostro linguaggio; mentre in altri più convenzionali lo sforzo maggiore va applicato alla seconda fase, in Prolog questa fase può essere davvero ridotta al minimo, in quanto la più corretta tecnica di sviluppo viene articolata direttamente dal calcolatore.

Per quello che invece riguarda l'output esistono per la verità una serie di forme, ormai ampiamente collaudate dall'uso, che si sono dimostrate le migliori per la maggior parte delle problematiche. Tanto per fare un esempio, sistemi che maneggiano dati e base di dati, eseguono output sono forma di report, stampa di etichettari, sommari, rendicontazioni e medie, ecc. Tutto deve passare comunque attraverso una adeguata definizione della struttura del file; la maggior parte del lavoro di strutturazione è già eseguita in fase di descrizione degli input. Nella fase di strutturazione finale del file occorre però definire e costruire il numero e la forma delle relazioni che intercorrono sia tra i diversi record che tra i file che concorrono alla relazionalità del risultato finale. Quanti file? Oggi, con le potenzialità messe a disposizione dalle moderne memorie di massa, è possibile affermare che non esistono limiti al numero dei file connessi e alla loro tipologia di interconnessione. Ovviamente in tali tipi di interconnessione vanno compresi anche campi e record non direttamente inseriti dall'utente, come campi calcolati, indicizzati o campi autoaggiornantisi.

L'ultima fase è quella dello sviluppo di una interfaccia ben costruita: questo non significa solo creare fogli e campi di input ben costruiti, significa anche costruire un programma che funzioni secondo le nostre direttive e necessità.

E anche questa volta abbiamo concluso; è, credo, la terzultima puntata su questo linguaggio; parleremo la prossima volta di controllo di flusso dei database e, ancora di backtracking e verifica dei programmi. MC