

Le basi del Data Base

Data Base Management System: il modello semantico dei dati

di Andrea de Prisco

Terza puntata sui Data Base. Parleremo questo mese del modello semantico dei dati: un particolare modo di descrivere la conoscenza, senza allontanarsi troppo dalla realtà. Come dicevamo nella prima puntata, un Utente è un utente, una persona, e come tale va trattata. Avrà un nome, un recapito, un numero di telefono, un'età.

Quando dovremo ritrovarne uno nella base di dati, basterà chiedere al sistema:

dammi l'utente di nome tizio o col telefono 345678. Nient'altro.

Nulla del tipo: Utente A4Z23...

Costruiamo una classe

Anche se con un mese di ritardo rispetto alla tabella di marcia annunciata sul n. 33 (vi avevamo detto che avremmo affrontato l'argomento sul numero scorso) è giunta l'ora di mostrare come si costruisce una classe dati in Basic-micatanto e come è possibile operare su essa per individuare insiemi di oggetti o apportare modifiche. Per chi non avesse sottomano il n. 33, diamo nuovamente la definizione di due tipi di dato che useremo molto spesso nel seguito.

L'ennupla è un insieme finito di coppie (identificatore, valore) sul quale è possibile tramite opportuni operatori selezionare i singoli campi. Un esempio di ennupla è:

```
Persona = (Nome = "Mario", Cognome = "Alpini", Età = 45)
```

per selezionare i vari campi si usa l'operatore "of":

```
PRINT Nome of Persona  
Restituirà "Mario".
```

La sequenza è un multinsieme finito di valori dello stesso tipo. È assimilabile, come struttura, ad un array monodimensionale Basic. Le uniche differenze sono che non è necessario alcun dimensionamento (può variare dinamicamente il numero dei suoi elementi) e che ogni elemento può non essere un tipo semplice (intero, stringa, reale), ma anche un'ennupla complicata quanto si vuole.

Per selezionare un elemento di una sequenza si opera come sugli array: se Lista è una sequenza, il quinto elemento sarà Lista (5). Per costruire una sequenza, si usano gli operatori "[" e "]".

```
Lista = [4,3,6,18,27,3,41]
```

se si vuol unire due sequenze, si usa l'operatore "+". Ad esempio:

```
AltraLista = Lista + [45,7,12]  
NuovaLista = AltraLista + [11]
```

si noti (secondo caso mostrato) che per aggiungere anche un solo elemento ad una sequenza, è necessario racchiuderlo tra parentesi quadre.

Nell'esempio visto, si è aggiunta alla sequenza di partenza una sequenza formata da un solo elemento: 11.

Questo perché l'operatore "+" agisce tra sequenze e non tra una sequenza e un elemento.

Per sapere se un elemento è contenuto in

uno di questi nostri multinsiemi, è disponibile l'operatore "isin":

```
IF 7 isin AltraLista THEN <Qualcosa>  
(è eseguito il <Qualcosa> se 7 è contenuto in AltraLista; nel nostro caso sì).
```

Per visitare tutti gli elementi di una sequenza, in Basic-micatanto, è disponibile una versione speciale del comando FOR. La sua sintassi è:

```
FOR <variabile> in <sequenza>  
.  
.  
.  
NEXT <variabile>
```

e non fa altro che scorrere i vari elementi, assegnandoli uno per volta alla <variabile>. Ad esempio:

```
10 FOR X in AltraLista  
20 Somma = Somma + X  
30 NEXT X
```

calcola la somma degli elementi di AltraLista. Per conoscere il numero di elementi di una sequenza, si usa la funzione:

```
Count(<sequenza>)
```

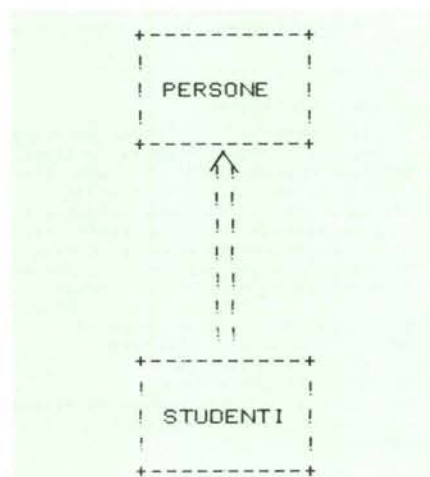


Figura 2 - Sottoclasse sottoinsieme.

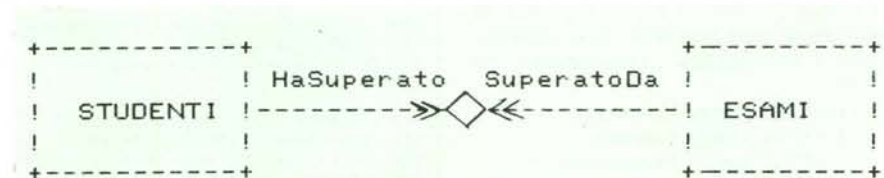


Figura 1 - Associazione con diretta e inversa multiple.

Un altro tipo di dato molto usato in linguaggi di programmazione un tantino più evoluti del Basic, è il tipo Booleano. Una variabile booleana può assumere solo i valori logici true (vero) o false (falso). Leggere una variabile booleana è valutare una espressione logica.

Le classi sono "quasi" delle sequenze: l'accesso agli oggetti (gli elementi) di una classe avviene tramite opportuni operatori e non indicando tra parentesi, accanto al nome della classe, il numero d'ordine.

Per usare una classe, è necessario definire la sua struttura, come è fatto ogni suo elemento, specificando il nome e il tipo di ogni attributo. Esempio: definiamo la classe Amici, nella quale inseriremo indirizzo e telefono di un po' di gente.

```
class Amici <->  
(Nome: string,  
Cognome: string,  
Indirizzo: string,  
Città: string,  
Telefono: int)
```

Abbiamo così definito una classe denominata Amici, in cui ogni elemento è una ennupla formata dai campi Nome, Cognome, Indirizzo, Città di tipo stringa e Telefono di tipo intero.

Per inserire oggetti in classe, si usa il costruttore Make, seguito dal nome della classe e dall'elemento.

```
Make Amici (Nome = "Maria Rosaria",  
Cognome = "D'Alessandro", Indirizzo = "Via Cisanello",  
Città = "Pisa", Telefono = 474747)
```

inserirà il primo oggetto. Continuiamo:

```
Make Amici (Nome = "Virginia",  
Cognome = "Ravenna", Indirizzo = "via Vesalio",  
Città = "Pisa", Telefono = 484848)
```

```
Make Amici (Nome = "Paolo",  
Cognome = "Zunino", Indirizzo = "Via Rossi",  
Città = "Cosenza", Telefono = 494949)
```

Abbiamo inserito 3 elementi. Vediamo gli operatori per "ripescarli". Si distinguono 2 casi:

— recuperare un elemento.

— recuperare un insieme di elementi.
 Esistono per la fattispecie 2 operatori: "get" e "all", il primo da non confondere col GET del Basic standard. La sintassi è comune:

<comando> <classe> with <condizione> <comando> è get o all, a seconda dei due casi.

<classe> indica il nome della classe in cui effettuare la ricerca.

<condizione> è il parametro di selezione col quale, specificando opportuni valori di attributi, indichiamo a quali elementi siamo interessati.

In 2 casi il sistema genera fallimento:
 — non esistono oggetti che soddisfano la condizione

— si è usato il get (volevamo un elemento) ed esistono più oggetti che soddisfano la condizione specificata. Il sistema non sa quale restituire ... e non è giusto che restituisca il primo che gli capita. Facciamo qualche esempio:

UnaPersona = get Amici with Nome = "Maria Rosaria"
 dopo questa linea, UnaPersona è l'ennupla inserita che ha campo Nome = "Maria Rosaria".

Se avessimo scritto:

UnaPersona = get Amici with Città = "Pisa"
 si sarebbe generato fallimento, dato che si sono inseriti 2 oggetti "pisani".

Il comando all, restituisce una sequenza:

ListaPersone = all Amici with Città = "Pisa"
 raggiungeremo i singoli elementi con ListaPersone (1) e ListaPersone(2). Inutile dire che all va bene anche se l'elemento che soddisfa la condizione è unico. Senza dimenticare però che sarà restituita in ogni caso una sequenza, anche se formata da un solo elemento. Esempio:

UnAltraPersona = all Amici with Nome = "Paolo"
 È una sequenza, il cui unico elemento è UnAltraPersona(1).

Per cancellare un elemento si usa il comando:

remove <elemento>
 dove <elemento> è un'ennupla contenuta nella classe. Concludendo, mostriamo come accedere ai vari campi:

PRINT Telefono of UnaPersona restituirà
 474747

PRINT Indirizzo of ListaPersone(2) restituirà

via Vesalio
 PRINT Città of UnAltraPersona(1) restituirà
 Cosenza

Il Modello semantico dei Dati

Ciò che ci interessa maggiormente è la conoscenza: non vogliamo fare filosofia, questa è informatica. Analizzeremo tre aspetti:

- La Conoscenza Concreta.
 - La Conoscenza Astratta.
 - La Conoscenza Procedurale.
- dando per ognuno di questi i relativi meccanismi di astrazione atti a modellarli.

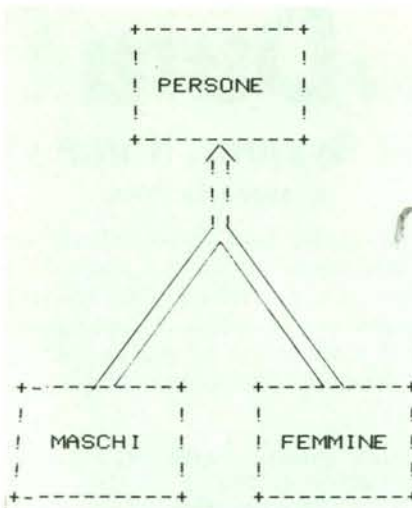


Figura 3 - Sottoclasse partizione.

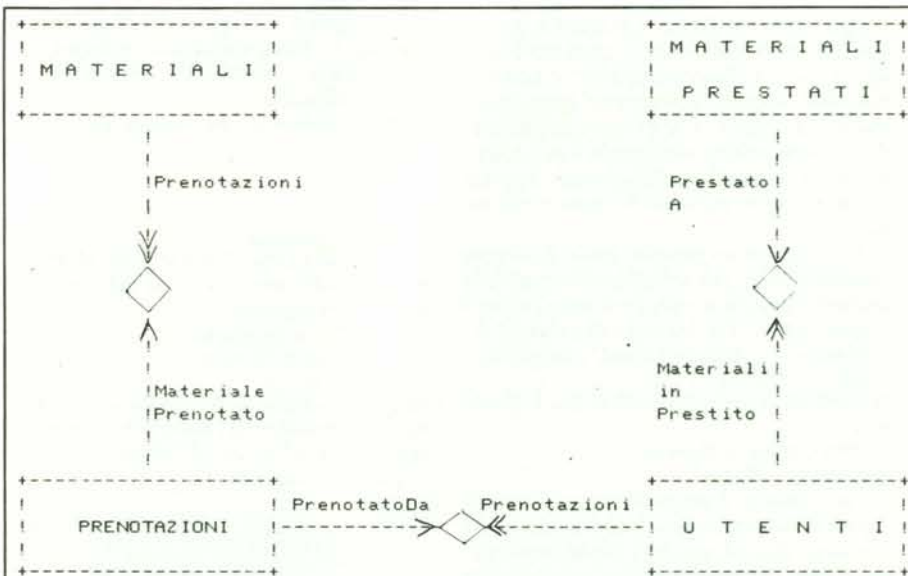
La conoscenza concreta riguarda ciò che si vuole rappresentare, che appartiene cioè al mondo osservato. Penseremo ai seguenti fatti.

Le Entità: le cose che ci interessano, ad esempio l'amico Paolo, il libro Knuth, l'utente Rossi.

Le Associazioni tra dati: le abbiamo viste sul numero scorso, ciò che stabilisce un legame logico tra entità.

Le Proprietà: ciò che descrive le entità. Una proprietà dell'amico Paolo è che abita a Cosenza, ad esempio.

La Classificazione è il meccanismo di astrazione mediante il quale entità diverse vengono considerate omogenee, da inserire in una stessa classe. Nell'esempio visto in precedenza, entità diverse (almeno si spera!) come Paolo e Virginia sono considerati appartenenti alla classe Amici (identica è la loro struttura).



BIBLIOTECA

```

10 class Materiali <-> (Autore:string, Titolo:string, Posizione:int,
  Prestabile:bool, Prenotazioni:seq Prenotazioni, NumeroPrestiti:int)
20 class Prenotazioni <-> (MaterialePrenotato:Materiali, PrenotatoDa:Utenti
  Giorno:int, Mese:int, Anno:int)
30 class Utenti <-> (NomeCognome:string, Recapito:string, Documento:string,
  Ritardatario:bool, HaPrenotato:seq Prenotazioni,
  HaInPrestito:seq MaterialiPrestati)
40 class MaterialiPrestati <-> (Autore:string, Titolo:string, PrestatoA:Utenti,
  Giorno:int, Mese:int, Anno:int)

50 procedure "Inserisci Utente"
60 INPUT "Nome e Cognome":A$
70 INPUT "Recapito":B$
80 INPUT "Documento":C$
90 make Utenti (NomeCognome=A$,Recapito=B$,Documento=C$,Ritardatario=false,
  HaPrenotato=[], HaInPrestito=[])
100 RETURN

110 procedure "Consulta Libro"
120 INPUT "Autore,Titolo":A$,B$
130 Libro = get MaterialiPrestati with Autore=A$ AND Titolo=B$
140 IF fail THEN 170
150 Libro = get Materiali with Autore=A$ AND Titolo=B$
160 PRINT "Posizione": Posizione of Libro:RETURN
170 PRINT "Libro non disponibile"
180 RETURN

190 procedure "Restituisci Libro"
200 INPUT "Autore,Titolo":A$,B$
210 Libro = get MaterialiPrestati with Autore=A$ AND Titolo=B$
220 Utente = PrestatoA of Libro
230 Giorni = GiorniTrascorsi(Giorno of Libro, Mese of Libro, Anno of Libro)
    
```


L'Aggregazione è il meccanismo di astrazione atto a definire la struttura degli oggetti di una classe e quindi le associazioni tra entità diverse. La struttura di un elemento si descrive dichiarando un insieme finito di proprietà (Nome, Cognome, Città, Età, Professione, ecc.).

Una proprietà è detta Chiave, se un determinato suo valore è presente al più in un elemento della classe.

Per intenderci: la proprietà matricola degli elementi appartenenti alla classe Studenti, può essere considerata una chiave, non potendo esistere due studenti con uguale matricola.

Una proprietà è detta Costante se il suo valore è invariante nel tempo, altrimenti è detta Modificabile. Ad esempio, il nome di una persona è costante, il suo recapito no: nella definizione di una classe, è possibile fare questo tipo di distinzione.

Una proprietà è detta Derivata se il suo valore può essere ricavato da altre proprietà con determinate regole.

esempio:

Età = derived AnnoDomini - AnnoDiNascita

Consentiteci di affermare sinceramente che ormai è opinione diffusa che l'età di un individuo si "ricava" dalla sottrazione tra l'anno in cui ci poniamo il problema e l'anno in cui è nato.

Una proprietà è detta associazione se ha per valore elementi di altre classi. Descrive cioè correlazioni tra entità. Una delle salienti caratteristiche del modello semantico dei dati è di modellare le associazioni tra dati come se fossero caratteristiche delle entità. Facciamo un esempio di sapore vagamente universitario. Si vuole descrivere l'associazione mostrata in figura 1: due classi: Studenti e Esami, una l'associazione

tra le classi (con diretta e inversa multipla):

Ogni studente ha superato un certo numero di esami.

Ogni esame è stato superato da un certo insieme di studenti.

Basta. Di fatto non è necessario aggiungere altro. Definiremo le due classi così:

```
class Studenti <->
(NomeCognome: string,
Matricola:int,
Recapito:string,
HaSuperato:seq Esami)
```

```
class Esami <->
(NomeEsame:string,
Docente:string,
Codiceesame:int,
SuperatoDa:seq Studenti)
```

Con i meccanismi visti finora, è anche possibile automatizzare le associazioni tra dati. Sfrutteremo la possibilità di definire proprietà derivate. Facciamo un esempio, sempre a carattere universitario. Potremmo definire il campo HaSuperato degli Studenti, invece che come seq Esami visto sopra, con l'espressione:

```
HaSuperato = all Esami with this is in SuperatoDa
```

a parole vuol dire proprio ciò che vogliamo descrivere: HaSuperato sono tutti gli esami della classe Esami che hanno lo studente in questione ("this") tra gli studenti elencati nella proprietà SuperatoDa. In questo modo, quando uno studente supera un nuovo esame, è sufficiente aggiornare solo la classe esami (l'esame Blablabla è stato superato anche da Caio) perché sia aggiornata anche la classe Studenti.

Il meccanismo delle sottoclassi

La Generalizzazione è il meccanismo di astrazione della conoscenza concreta mediante il quale è possibile organizzare insieme di classi in una gerarchia definita da una relazione di ordinamento parziale. È il discorso delle sottoclassi, lievemente anticipato due numeri fa. Col meccanismo delle sottoclassi, si migliora notevolmente la visione utente di una base di dati, senza appesantire l'organizzazione interna: le sottoclassi sono solo un modo diverso di vedere le stesse cose. Per intenderci: se abbiamo la classe Italiani e la sottoclasse Toscani (e perché no, la sottoclasse Livornesi), il fatto che un abitante Livornese si ritrovi in tre classi in gerarchia, non vuol dire che è rappresentato tre volte nella base di dati. L'abitante è sempre lo stesso. Prova ne è il fatto che cancellandolo dalla classe Italiani, sparirà definitivamente anche dai Toscani e dai Livornesi.

Distinguiamo tre specie di sottoclasse:

— Sottoclasse Sottoinsieme: un esempio è la classe Studenti, vista come sottoinsieme di una classe più ampia denominata Persone. È ovvio che uno studente è una Persona, ma non è sempre vero il viceversa. Ciò significa che non potremo avere un elemento in Studenti senza averlo anche in Persone. Avere nel senso logico: ripetiamo, l'oggetto rappresentato è lo stesso, tutt'al

```
240 EventualeRitardo = false
250 IF Giorni > 12 THEN EventualeRitardo=true
260 LibriInPrestito = HaInPrestito of Utente
270 NuovaLista = []
280 for X in LibriInPrestito
290 IF X <> Libro THEN NuovaLista=NuovaLista + [X]
300 next X
310 make Utenti (NomeCognome=NomeCognome of Utente, Recapito=Recapito of Utente,
Documento=Documento of Utente, Ritardatario=EventualeRitardo,
HaPrenotato=HaPrenotato of Utente, HaInPrestito=NuovaLista)
320 remove Utente;remove Libro
330 RETURN

340 procedure "Prenota Libro"
350 INPUT "Autore,Titolo":A#,B#
360 INPUT "Nome e Cognome":C#
370 Libro = get Materiali with Autore=A# AND Titolo=B#
380 Utente = get Utenti with NomeCognome=C#
390 LaPrenotazione = (MaterialePrenotato=Libro, PrenotatoDa=Utente,
Giorno=Giorno of Oggi,Mese=Mese of Oggi,Anno=Anno of Oggi)
400 make Prenotazioni (LaPrenotazione)
410 make Materiali (Autore=Autore of Libro,Titolo=Titolo of Libro,
Posizione=Posizione of Libro, Prestabile=Prestabile of Libro,
Prenotazioni=Prenotazioni of Libro + [LaPrenotazione],
NumeroPrestiti=NumeroPrestiti of Libro)
420 make Utenti (NomeCognome=NomeCognome of Utente,Recapito=Recapito of Utente,
Documento=Documento of Utente,Ritardatario=Ritardatario of Utente,
HaPrenotato=HaPrenotato of Utente + [LaPrenotazione],
HaInPrestito=HaInPrestito of Utente)
430 remove Utente;remove Libro
440 RETURN

450 procedure "Presta Libro"
460 INPUT "Autore,Titolo":A#,B#
470 INPUT "Nome e Cognome":C#
480 Libro = get MaterialiPrestati with Autore=A# AND Titolo=B#
490 IF fail THEN 510
500 PRINT "Materiale non disponibile":RETURN
510 Libro = get Materiali with Autore=A# AND Titolo=B#
520 Utente = get Utenti with NomeCognome=C#
530 IF Prestabile of Libro = false THEN PRINT "Testo non prestabile":RETURN
540 IF Ritardatario of Utente = true THEN PRINT "Utente ritardatario":RETURN
550 NumeroLibri = count(HaInPrestito of Utente)
560 IF NumeroLibri = 2 THEN PRINT "L'Utente ha già due libri":RETURN
570 Prenotazione = get Prenotazioni with MaterialePrenotato=Libro
580 IF fail THEN 620
590 NumeroGiorni = GiorniTrascorsi(Giorno of Prenotazione,Mese of Prenotazione,
Anno of Prenotazione)
600 IF NumeroGiorni > 1 THEN 620
610 PRINT "Materiale prenotato":RETURN
620 Prestito = (Autore=A#,Titolo=B#,PrestatoA=Utente,Giorno=Giorno of Oggi,
Mese=Mese of Oggi,Anno=Anno of Oggi)
630 make MaterialiPrestati(Prestito)
640 make Utenti (NomeCognome=NomeCognome of Utente,Recapito=Recapito of Utente,
Documento=Documento of Utente,Ritardatario=false,
HaPrenotato=HaPrenotato of Utente,
HaInPrestito=HaInPrestito of Utente)+[Prestito]
650 make Materiali (Autore=Autore of Libro,Titolo=Titolo of Libro,
Posizione=Posizione of Libro,Prestabile=true,
Prenotazioni=Prenotazioni of Libro,
NumeroPrestiti=NumeroPrestiti of Libro + 1 )
660 remove Utente;remove Libro
670 RETURN
```


più aumentato di qualche campo quando è visto come *Studiante*. Ciò perché, nella definizione di sottoclasse, è possibile aggiungere campi alle ennuple della classe da cui si parte. Ad esempio, uno studente è una persona con in più un numero di matricola. Se cercheremo Caio tra le Persone, conosceremo il suo indirizzo, il suo telefono ed altro. Se cercheremo Caio tra gli studenti, otterremo oltre alle proprietà delle Persone, anche la sua matricola.

— Sottoclasse Partizione: si costruiscono a partire da una classe eseguendo una partizione (immaginate di tagliare una torta...). Per fare un esempio, potremmo partizionare la classe *Persone* in maschi e femmine. In generale, l'unione delle sottoclassi della stessa partizione, è un sottoinsieme della classe di partenza.

— Sottoclasse Restrizione: contengono tutti e soli gli elementi che soddisfano una determinata condizione su uno o più attributi costanti. Esempio:

Francesi <-> restriction of Europei class with nazione = "France"
costruisce automaticamente la sottoclasse *Francesi*, aggiornandola ad ogni inserimento in *Europei* di un elemento con campo *Nazione* = "France".

Conoscenza astratta e conoscenza procedurale

Per modellare la conoscenza astratta si impongono determinati vincoli di integrità. I vincoli di integrità servono per limitare l'evolvere della conoscenza concreta, descrivendo fatti generali non direttamente riconducibili ai vincoli impliciti del modello semantico dei dati. Un esempio di vincolo è che le matricole degli studenti di una università sono tutte diverse, che uno studente non può superare due volte lo stesso esame, che abbia almeno 17 anni. Insomma, i vincoli dovrebbero impedire o almeno limitare la possibilità di raccontare frottole alla base di dati. Il tutto per migliorare le prestazioni, sfruttando al massimo questa benedetta informazione di cui si dispone. Vincoli più interessanti sono quelli che riguardano insieme diversi. A livello di biblioteca, ad esempio, far controllare al sistema che se un *Utente* ha in prestito un certo libro, nella classe *materiali prestatati* nel campo *PrestatoA*, del libro di cui sopra, ci sia appunto l'utente da cui siamo partiti. O più semplicemente: se *Ermengildo* è sposato con *Ermengarda*, deve essere anche che *Ermengarda* è sposata con *Ermengildo*, altrimenti "qualcosa" non torna.

I vincoli di integrità si definiscono al momento di creare una classe, di seguito alla descrizione dell'ennupla. Ad esempio, imponiamo la restrizione che l'età di una persona deve essere maggiore di zero:

```

classe Persone <->
(NomeCognome:string,
Recapito:string,
Telefono:int,
Età:int)
assert Età > 0
se si costruirà un elemento persona con età

```

= 12 il sistema genererà fallimento, non consentendo di sporcare la base di dati con dati (almeno in questo senso) inconsistenti. La conoscenza procedurale si modella col meccanismo delle funzioni. Nel Basic-micatanto con porzioni di programma etichettate in cima con un nome simbolico.

Si sta parlando delle linee di programma che si stendono dopo aver definito una

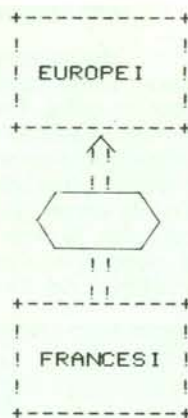


Figura 4 - Sottoclasse restrizione.

base di dati, per l'automazione di determinate procedure. Nell'esempio di organizzazione di una biblioteca mostrato in queste pagine si è "procedurizzato" le funzioni di prestito materiale, riconsegna, inserzione di un utente, ed altro.

Scrivendo le opportune procedure si semplifica notevolmente la gestione della base di dati. Per esempio, se un utente vuole in prestito un libro, l'addetto ai prestiti basta che digiti sulla tastiera "Presta Libro" e risponda alle domande poste dal sistema (autore e titolo del libro, nome e cognome dell'utente). Se qualcosa non quadra, saranno segnalati automaticamente messaggi sul video, altrimenti sarà concesso il prestito, aggiornando opportunamente l'informazione mantenuta dalla base di dati. Tutto qui.

La nostra amata Biblioteca

Vedremo ora come è possibile organizzare in Basic-micatanto una base di dati per biblioteca, secondo il modello semantico dei dati. Sul numero scorso è stata data la specifica comportamentale: quali saranno le funzioni svolte dal sistema, a istallazione completata. Per ragioni di spazio non commenteremo tutto il listato, ma ci soffermeremo principalmente sui punti più interessanti.

Immaginiamo che il sistema abbia incorporato un calendario interno che si aggiorna automaticamente ogni mattina. L'ennupla *Oggi*, contiene giorno, mese e anno corrente. Per intenderci, per sapere che giorno è oggi è sufficiente digitare:

```

PRINT Giorno of Oggi

```

analogamente per il mese e l'anno.
Sempre facente parte del sistema, la funzione:

```

GiorniTrascorsi(Giorno, Mese, Anno)

```

che restituisce il numero di giorni che inter-

corrono tra la data di *Oggi* e la data passata come parametro.

Il programma *Biblioteca* è diviso in due parti: la parte dichiarativa (linee 10-40) dove sono definite le varie classi usate, e la parte procedurale, dove sono specificate le procedure usabili.

Per chiamare una procedura è sufficiente digitare il suo nome.

Le prime quattro linee contengono la definizione delle quattro classi adoperate: *Materiali*, *Prenotazioni*, *Utenti* e *Materiali Prestati*. Fra parentesi la descrizione delle ennuple contenute nelle classi. A partire dalla linea 50 fino alla linea 100, è listata la prima delle procedure: "Inserisci Utente". Si invoca quando un *Utente* accede per la prima volta alla biblioteca. Le linee 60,70 e 80 caricano tramite comunissimi *INPUT*, *Nome* e *Cognome*, *Recapito* e *Documento di identità* nelle variabili *A\$*, *B\$* e *C\$*. La linea 90 inserisce l'oggetto (...pardon, l'*Utente*...) in classe. Si noti il campo *Ritardatario* posto a false e i due campi sequenza posti a "[]", ossia a "sequenza vuota". Questo perché un utente, appena accede alla biblioteca, certamente non è ritardatario (ancora ha la coscienza pulita) e non ha né prenotazioni né libri in prestito.

Diamo ora uno sguardo alla procedura "Presta Libro" (linee 450 e segg.).

Dopo aver chiesto autore e titolo del libro che si chiede in prestito e nome e cognome dell'utente richiedente, il sistema controlla che il testo non sia già in prestito a qualcuno. Ciò avviene con le linee 480 e 490. Si cerca il testo tra i *Materiali Prestati* e solo se non c'è si continua la procedura, altrimenti si stampa il messaggio "Libro non disponibile". Il "solo se non c'è" si implementa col costrutto *IF fail THEN*, (se c'è un errore allora...) che funziona in virtù del fatto che si genera fallimento se si cerca un oggetto in una classe e non lo si trova.

Le linee 510-520 assegnano alle variabili *Libro* e *Utente* il testo desiderato e l'utente già memorizzato. Ciò per controllare (linee 530-560) che il testo sia prestabile, che l'utente non sia ritardatario o che non abbia già due libri in prestito. Di seguito (linee 570-610) con un truccetto simile a quello usato per sapere se il materiale era già in prestito a qualche altro utente, si controlla che il testo non sia citato in una prenotazione e che tale prenotazione (se esiste) non sia scaduta. Le linee 620-660 modificano la base di dati secondo il nuovo stato: il libro è tra i materiali prestatati, l'utente ha in prestito il libro di cui sopra, il materiale concesso in prestito è stato prestatato nuovamente a qualcuno. Per la modifica, si inseriscono i nuovi dati aggiornati e si cancellano i vecchi, non più significativi.

Bibliografia:

Per chi volesse saperne di più in merito a Basi di Dati e Modelli di Dati è disponibile il libro:

A. Albano, R. Orsini
I Sistemi per Basi di Dati
Editrice Boringhieri, C.so Vittorio Emanuele 86, Torino