Le basi del Data Base

Data Base Management System: Galileo/J, atto secondo

di Andrea de Prisco

Nona parte

Il mese scorso abbiamo presentato il listato del Galileo/J, mini data base per Commodore 64, con una rapida spiegazione dei comandi.
Come promesso, su questo numero con più calma dedicheremo tutto il nostro tempo all'interfaccia utente-sistema, all'organizzazione interna dei dati e a qualche ulteriore esempio chiarificativo.

L'interfaccia utente-sistema

Come già descritto il mese scorso, dando Run al programma Galileo/J dopo qualche attimo appare sullo schermo, in alto a sinistra, una "E" maiuscola seguita dai due punti e dal cursore lampeggiante. Il sistema, in stato di pronto o di ready se preferite, è in grado di eseguire comandi. Sempre sul numero scorso, raccomandavamo all'utente di non dimenticare mai che tutte le volte che si digita qualcosa in Galileo/J si è in ambiente INPUT del Basic standard, con le dovute limitazioni proprie del Basic del 64 (uso molto limitato dei tasti cursore). Abbiamo anche visto che per introdurre input più lunghi di 2 linee di schermo è sufficiente battere [Return] e continuare sulla linea successiva. Fra le cose più importanti da tenere presente per non impazzire davanti al video, sottolineiamo il fatto che tutti i comandi del Galileo/J terminano col carattere separatore "punto e virgola". Se al termine della digitazione di un comando dimentichiamo di metterlo, il sistema al [Return] come sempre attenderà una nuova linea mostrando nuovamente il cursore lampeggiante. In casi del genere, basterà semplicemente battere il punto e virgola mancante per far accettare al sistema il comando appena digitato. Le linee 50-960 del listato pubblicato lo scorso mese implementano l'interfaccia utente-sistema del Galileo/J: vediamo linea per linea il suo funzionamento. La prima istruzione è un OPEN 10,0 e serve per aprire un file di tipo input con la tastiera (ricordiamo che per il 64 anche il video e la tastiera possono essere viste come periferiche del computer stesso). Questo metodo è usato principalmente per non mostrare (in alcuni casi antiestetici) punti interrogativi sullo schermo al momento del Prompt: chiaramente, come mostrato dalla linea 110, occorrerà usare il comando INPUT#10 e non INPUT per ricevere i comandi dall'esterno. L'uso delle variabili T1$ e T2$ sarà spiegato più avanti. La linea 100 inizializza alcune variabili di servizio e pulisce il video. Il carattere di controllo corrispondente a CHR$(14) passa al set di caratteri minuscolo/ maiuscolo; quel segnaccio tra apici non è altro che una "E" maiuscola del set alternativo seguita dai due punti. In generale, Q$ contiene sempre ciò che si deve mostrare prima dell'INPUT vero e proprio (la sequenza "E:" o semplicemente 2 spazi). Alla linea 110 vediamo che ogni INPUT è per la variabile A$ e questa, ogni volta, viene riversata nella variabile B$ che correntemente contiene la porzione di comando già digitata. È chiaro che ogni comando non potrà essere più lungo di 255 caratteri essendo questa la massima capienza di una stringa Basic, ma tale limitazione non ha effetto rilevante. Alla linea 120, la domanda di fondamentale importanza: quanto inserito termina col punto e virgola?

Se sì possiamo eseguire il comando impostato (GOSUB 500) altrimenti si torna alla linea 110 per continuare l'INPUT. Le variabili T1$ e T2$ contengono gli ultimi due comandi dati al sistema: in questo modo, tramite una semplice serie di scambi di variabile è possibile realizzare un meccanismo in alcuni casi molto comodo: battendo solo il punto e virgola verrà rieseguito l'ultimo comando dato. Ciò può risultare abbastanza utile quando si inseriscono più elementi in una stessa classe o si ricercano questi ultimi a colpi di next. Sia nel primo caso che nel secondo, basterà battere una sola volta il comando e proseguire semplicemente digitando punto e virgola [Return]. Non è molto chiaro quanto appena detto? Facciamo un esempio: immaginiamo di dover inserire tre indirizzi nella classe Amici: il comando è make Amici. Il sistema, come già detto lo scorso numero, chiederà i valori dei vari campi (Nome, Recapito, Residenza, ecc). Terminato l'inserimento del primo nominativo, per inserire il secondo dovremo digitare nuovamente make Amici: basterà battere il solo punto e virgola dato che il sistema conserva automaticamente traccia dell'ultimo comando impostato (leggi: dell'ultima cosa digitata a seguito della richista "E:").

```
FAILURE : MISSING <->
FAILURE : MISSING (
FAILURE : TOO MANY CLASSES
FAILURE : MISSING ←
FAILURE : ILLEGAL TYPE
FAILURE : MISSING )
FAILURE : MISSING 'IN' NEAR EXTKEY
FAILURE : MISSING KEY
FAILURE : UNKNOWN KEY
FAILURE : SYNTAX ERROR
FAILURE : INVALID LEN
FAILURE : TOO MANY FIELDS
FAILURE : RECORD TOO LONG
FAILURE : MISSING CLASS NAME
FAILURE : CLASS UNKNOWN
FAILURE : TYPE MISMATCH
FAILURE : KEY EXISTS
FAILURE : MISSING WITH
FAILURE : NEXT NOT FOUND
FAILURE : KEY NOT INDEXED
FAILURE : NEXT WITHOUT FIND
FAILURE : MISSING =
FAILURE : INVALID KEY
FAILURE : CLASS FULL
```

Figura 1 - Messaggi d'errore del Galileo/J.

Per finire, a partire dalla linea 500 in poi, a seconda di quale istruzione è stata digitata, il controllo è passato a quella porzione di programma che l'implementa: questo molto in generale. Per essere più precisi, prima di fare questo è necessario "spacchettare" il comando contenuto in B$. Si tratta (linee 500-540) di trasferire ogni parola del comando in un elemento dell'array CO$(I), semplicemente individuando i separatori tra parole (linea 520).

È chiaro che se in B$ non c'è un comando Galileo/J sarà segnalato il Syntax Error (linea 900) grazie alla routine di trattamento errori, locata a partire alla linea 15000, che descriveremo in seguito.

L'organizzazione interna

Chiusa questa piccola parentesi, ricominciamo tutto daccapo: dando Run al programma, notiamo che il driver si mette in funzione prima di dare la "E:" di pronto: il sistema legge, se presente, un file di servizio contenente varie informazioni sullo stato della base di dati. È ovvio che se tale file non esiste (è il Galileo/J stesso che lo crea ogni volta che si termina una sessione di lavoro con quit) vorrà dire che nessun dato è stato inserito, né alcuna classe dichiarata.

Tale meccanismo serve per migliorare l'interattività del sistema: se terminiamo una sessione di lavoro in un qualsiasi momento, quando decideremo di continuare ci troveremo automaticamente nello stesso punto in cui avevamo interrotto, semplicemente dando Run al programma come abbiamo fatto la prima volta.

Lo stato del sistema, memorizzato nel file di servizio, consiste in un insieme di informazioni che normalmente vengono mantenute in Ram sotto forma di array numerici e stringa. Sono il nome di tutte le classi dichiarate sino a quel momento, il tipo delle ennuple, la lunghezza degli elementi, gli attributi chiave di ogni classe, e le chiavi di tutti gli elementi inseriti.

Sappiamo infatti che per accedere a una registrazione di un file relativo, occorre specificarne la posizione: dato che in Galileo/J abbiamo a che fare solo con accessi per chiave, la trasformazione chiave-posizione avviene semplicemente cercando l'elemento nell'array apposito: la posizione di questo che conterrà la chiave cercata sarà la stessa nel file relativo che conterrà l'intera registrazione. Analogamente, per inserirne una nuova, basterà trovare una

Questo programma è disponibile su disco presso la redazione. Vedere l'elenco dei programmi disponibili e le istruzioni per l'acquisto a pag. 131.

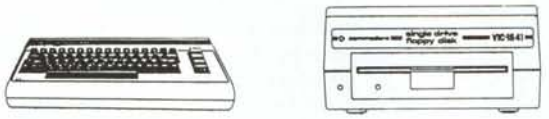

Figura 2 - Durante l'uso, il Galileo/J conserva in memoria centrale le chiavi utilizzate e mantiene su disco le registrazioni inserite.

Array Chiavi

1	RM
2	NA
3	PI
4	BA

File Relativo Citta'

	Sigla	Localita'	CAP	Prefisso
1	RM	Roma	00100	06
2	NA	Napoli	80100	081
3	PI	Pisa	56100	050
4	BA	Bari	70100	080

posizione libera nell'array per individuare la posizione nel file relativo da occupare.

Anche quando si tratta di cancellare un elemento, il problema si riduce a marcare opportunamente l'array delle chiavi ponendo un carattere "-" al posto della chiave da cancellare: si tratta cioè di cancellazione logica del dato e non fisica.

Per quanto riguarda l'accesso per chiavi non primarie (interrogazioni del tipo all NomeClasse with Attributo = Costante) appositi indici sono mantenuti su disco, sotto forma di file sequenziali. Ad esempio, se abbiamo nella nostra base un insieme di indirizzi, come chiave useremo il campo NomeCognome, ma nulla ci vieterà di richiedere tutte le registrazioni con Recapito = "Piazza Crispi". Né il sistema si divertirà ad accedere una per una a tutte le registrazioni controllando il Recapito: accederà semplicemente al file Recapito (che si è creato da solo al momento opportuno) cercando in questo la dicitura "Piazza Crispi" alla quale seguiranno tutte le posizioni del file relativo con tale indirizzo.

Vengono indicizzati (= creati gli opportuni indici) tutti gli attributi non chiave ad eccezione dei campi di tipo extkeys (attenzione, al plurale!); potremo cioè chiedere quasi di tutto: nell'esempio dell'indirizzario visto sul numero scorso, anche interrogazioni del tipo: all Amici with Varie = y; o del tipo: all Amici with Città = NA (chiave esterna per la classe Città).

A questo punto è ovvio pensare che l'inserimento di un elemento in classe è un'operazione tutt'altro che banale: in effetti coinvolge più passi:

1) controllare che la chiave della registrazione non sia stata già adoperata.

2) Trovare una posizione libera nell'array delle chiavi.

3) Inserire la registrazione nel file relativo (nella posizione trovata al passo 2).

4) Aggiornare gli indici delle chiavi secondarie.

L'operazione 4 è certamente la più lunga: proprio per questo, fintantochè si aggiungono elementi nella stessa classe, l'operazione viene rimandata. Solo quando al posto del solito make <Classe> si esegue qualsiasi altro comando (compreso l'inserimento di elementi in altre classi) si aggiornano uno per uno tutti gli indici delle chiavi secondarie, tenendo per qualche secondo in più il driver impegnato. Alla luce di questo nuovo fatto, sembra superfluo raccomandare di non saltare da una classe all'altra durante l'inserimento dei vari elementi: nei limiti del possibile (solo per risparmiare del tempo, altrimenti fate come volete) inserite i dati classe per classe (ad esempio, prima tutti gli indirizzi, poi tutte le città).

Facciamo un'istantanea

Per comprendere meglio il funzionamento dei file indice, scattiamo un'istantanea: vediamo cosa è mantenuto in memoria (disco + Ram), ad un certo istante, durante l'inserimento di alcuni dati. Supponiamo di dichiarare due classi (le solite: Amici e Città) e immaginiamo di introdurre i seguenti quattro indirizzi:

Amilcare Pallisi, Via del Rospo, Roma

Zaccaria Modelli, Piazza Matusalemme, Napoli

Cesare Zebedeo, Via Kinzica, Pisa

Achille Baccelli, Largo Piattola, Bari

Per non dilungarci eccessivamente, non consideriamo le varie strutture della classe Città, ma ci limiteremo a tener presente che l'arcinota associazione Amici-Città è realizzata col meccanismo delle chiavi esterne e che la chiave per gli elementi della classe Città è la sigla automobilistica.

Se introduciamo i 4 personaggi nell'ordine visto sopra, questi occuperanno il primo, la prima posizione del file relativo; il secondo, la seconda posizione; il terzo, la terza; il quarto la quarta. Scattiamo la nostra istantanea: lo stato del sistema in questo momento è il seguente: in un apposito array stanno memorizzate le 4 chiavi (Amilcare Pallisi, Zaccaria Modelli, Cesare Zebedeo, Achille Baccelli). L'indice Recapito conterrà le seguenti informazioni:

Via del Rospo
1
Piazza Matusalemme
2
Via Kinzica
3
Largo Piattola
4

l'indice Residenza le seguenti:

RM
1
NA
2
PI
3
BA
4

Notare il formato di tali indici: è presente il valore di un attributo, seguito dalla posizione occupata nel file relativo. Se introduciamo un quinto elemento:

Bartolomeo Fibbiana, Piazza Vettovaglie, Pisa

i due indici cambieranno nel seguente modo:

Indice Recapito:
Via del Rospo
1
Piazza Matusalemme
2
Via Kinzica
3
Largo Piattola
4
Piazza Vettovaglie
5

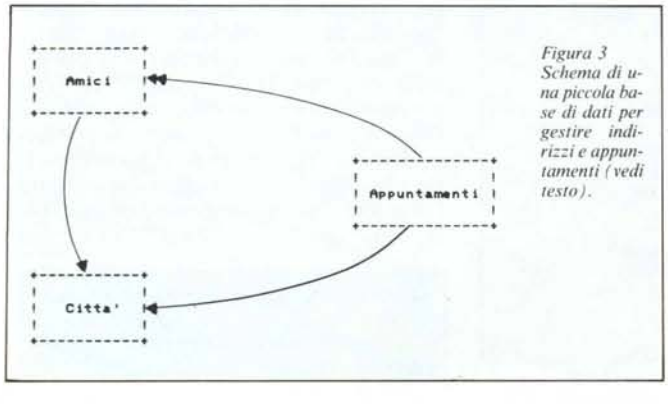

Figura 3 Schema di una piccola base di dati per gestire indirizzi e appuntamenti (vedi testo).

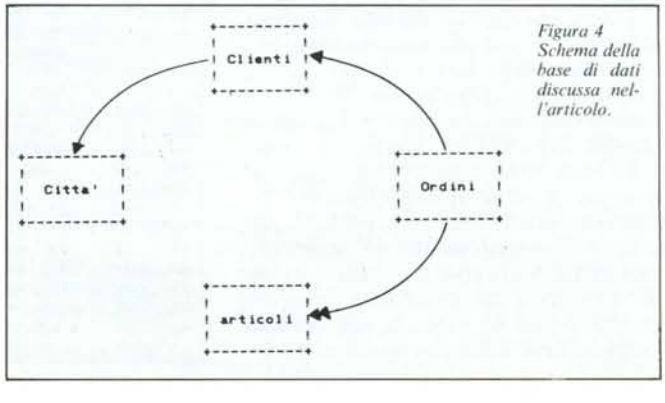

Figura 4 Schema della base di dati discussa nell'articolo.

Indice Residenza:
RM
1
NA
2
PI
5
3
BA
4

Se chiediamo tutti gli Amici con Residenza = PI, accedendo al file Residenza sapremo velocemente che dobbiamo prendere le registrazioni 5 e 3; cercando gli Amici con Indirizzo = Piazza Matusalemme, accedendo al file Recapito scopriamo che questo occupa la posizione 2.

I messaggi d'errore

In figura 1 sono mostrati i messaggi d'errore del Galileo/J: non sono molti, ma in qualche modo danno una mano quando qualcosa non va come dovrebbe. Li analizzeremo uno per uno. I primi 12 messaggi riguardano la dichiarazione di una classe, tramite l'apposito comando class: è questo

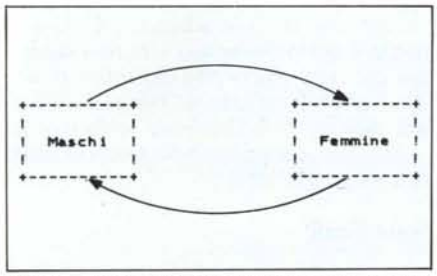

Figura 5 - Possiamo scomporre le nostre amicizie in due classi, Maschi e Femmine, correlando i due insiemi con l'associazione partner a valori nella classe adiacente. Nell'articolo le relative definizioni in Galielo/J.

il più articolato dei comandi, nonché quello più disposto a generare errori. La sua sintassi, come già visto lo scorso mese, è la seguente:

class NomeClasse <-> (Tipo Ennupla)Key (AttributoChiave) [Len(lunghezzaElementi)]

tra parentesi quadre la specifica della lunghezza degli elementi essendo questa facoltativa. Gli errori segnalati dal sistema sono:

CLASS EXISTS
la classe che si sta dichiarando esiste già: è ovvio che non è possibile chiamare due o più classi con lo stesso nome, per non causare ambiguità usando gli altri comandi.

MISSING <->
non si è usato l'operatore <-> tra il nome della classe e la descrizione dei suoi elementi.

MISSING (
nella dichiarazione di classe manca qualche apertura di parentesi.

TOO MANY CLASSES
si tenta di dichiarare più classi di quante il Galileo/J del 64 ne possa gestire. Tale limite è fissato a 8.

MISSING ←
manca il carattere "←" tra il nome di un attributo e il suo tipo.

ILLEGAL TYPE
si tenta di usare un tipo di dato non predefinito. I tipi di dato predefiniti del Galileo/J sono: int, string, page, extkey, extkeys.

MISSING)
nella dichiarazione di classe manca qualche chiusura di parentesi.

MISSING 'IN' NEAR EXTKEY
manca la parola 'in' nella dichiarazione di tipo extkey o extkeys.

MISSING KEY
non è stato dichiarato l'attributo chiave primaria.

Foto 1 e foto 2 - Alcune fasi della dichiarazione della base di dati di figura 4.

Foto 3 - Si richiedono tutti gli articoli immessi nella base di dati.

Foto 4 - Si richiede il primo articolo con campo (o attributo) costruttore = Commodore.

UNKNOWN KEY
l'attributo dichiarato chiave non compare tra quelli della dichiarazione di ennupla della classe oppure, nella condizione data in un comando find o all la chiave da cercare non è stata mai immessa.

SYNTAX ERROR
il comando impartito non ha sintassi corretta.

INVALID LEN
nella dichiarazione di classe, la lunghezza specificata degli elementi non è compresa tra 1 e 254.

TOO MANY FIELDS
si dichiara una classe con più di 8 (è numero massimo consentito) campi.

RECORD TOO LONG
in fase di make, l'elemento che si sta inserendo occupa più spazio di quanto specificato col comando len nella dichiarazione di classe.

MISSING CLASS NAME
in un comando di find, di all, di edit o di destroy manca il nome della classe.

CLASS UNKNOWN
in un comando di find, di all, di edit o di destroy la classe specificata non è mai stata dichiarata.

TYPE MISMATCH
durante l'operazione di make si inserisce un dato di tipo non compatibile col tipo dichiarato (es. stringhe al posto di interi, risposte diverse da y o n per attributi di tipo page, ecc.): in questo caso il sistema non fa abortire l'operazione, ma chiede nuovamente il valore dell'attributo.

KEY EXISTS
l'elemento che si sta inserendo ha chiave uguale a quella di un elemento già inserito.

MISSING WITH
manca il with in un comando di edit, di destroy, di find o di all (in questi ultimi due casi, solo quando non si usa la forma abbreviata che trova tutti gli elementi).

NEXT NOT FOUND
non esistono altri elementi che soddisfano la condizione data.

KEY NOT INDEXED
si usa un attributo non dichiarato o non indicizzato per un'operazione di ricerca per chiave non primaria.

NEXT WITHOUT FIND
si impartisce il comando next senza prima aver dato il find.

MISSING =
non si è usato l'uguale in un comando di find, all, edit,

INVALID KEY
si usa una chiave non primaria in un comando di edit o destroy.

CLASS FULL
si tenta di inserire più di 60 elementi in una classe: 60 è appunto la massima capacità.

Qualche esempio

La volta scorsa, come esempio di base di dati, è stato mostrato solo quello schematizzato in figura 3, relativo a una organizzazione strutturata di indirizzi e appuntamenti. La prima classe aveva la seguente struttura:

class Amici <-> (
NomeCognome←string and
Recapito←string and

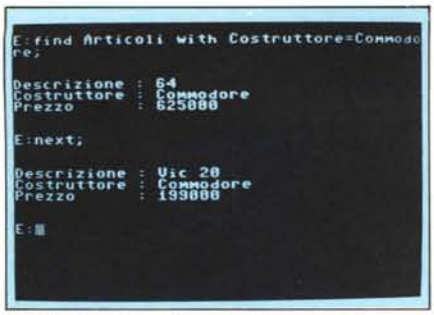

Foto 5 - Con l'operatore NEXT si ottiene il prossimo elemento che soddisfa la medesima condizione data col FIND precedente (costruttore = Commodore)

Foto 6 - Se non esistono altri elementi che soddisfano la condizione (nel nostro caso costruttore = Commodore) il sistema segnala fallimento col messaggio NEXT NOT FOUND.

```
  Residenza←extkey in Città and
  Telefono←int and
  Varie←page)
  key (NomeCognome)len(120);
```

e, come visto, permetteva di inserire indirizzo e telefono dei nostri conoscenti. Si noti l'attributo Residenza che, tramite il meccanismo delle chiavi esterne, permette l'associazione di tali elementi con la classe delle città:

```
class Città <-> (
Località←string and
Sigla←string and
CAP←string and
Prefisso←string)
key (Sigla)len(80);
```

Per ultima, la classe appuntamenti, serviva per associare una data a un luogo (associazione univoca con la classe Città) e a un insieme di amici (associazione multipla con la classe Amici). Questa la sua definizione:

```
class Appuntamenti <-> (
Data←string and
Luogo←extkey in Città and
Partecipanti←extkeys in Amici)
key (Data)len(254);
```

In figura 4 è mostrato lo schema di una base di dati riguardante un negozio di computer: abbiamo una classe Clienti, una classe Articoli, una classe Ordini e la classe Città. L'associazione Clienti-Città, è la solita: serve, come nel caso dell'indirizzario, per non ripetere più volte notizie riguardo una particolare località (Cap, Prefisso, Sigla) che può certamente essere presente più volte nella base di dati (es. più clienti di una stessa città).

La classe Ordini, correlata con le classi Articoli e Clienti, serve per mantenere traccia delle ordinazioni già evase, in merito al cliente interessato e agli articoli da lui ordinati. Nelle foto 1 e 2, è mostrata la fase di definizione delle quattro classi.

```
class Città <-> (
Località←string and
Sigla←string and
CAP←string and
Prefisso←string)
key (Sigla)len(80);
```

è uguale all'esempio precedente: serve per memorizzare le città di tutti i clienti.

```
class Clienti <-> (
Nome←string and
Recapito←string and
Piazza←extkey in Città and
Telefono←string)
key (Nome)len(120);
```

in questa classe si inseriscono nome, indirizzo, telefono e piazza (inteso come posto in cui si svolge un'attività) dei vari clienti del nostro computer shop.

```
class Articoli <-> (
Descrizione←string and
Costruttore←string and
Prezzo←int)
key (Descrizione)len(100);
```

serve per memorizzare gli articoli venduti dalla ditta.

```
class Ordini <-> (
Numero←int and
Data←string and
Cliente←extkey in Clienti and
Articoli←extkeys in Articoli)
key (Numero)len(254);
```

riguarda le ordinazioni evase. Volendo provare a fare qualche inserimento, non ci resta che scegliere da quale classe iniziare. Introduciamo degli articoli: il comando è:

```
make Articoli;
```

il sistema ci chiederà Descrizione: risponderemo, ad esempio, VIC 20. La richiesta successiva sarà Costruttore: digiteremo Commodore. Infine, Prezzo, la nostra risposta sarà 199000. Sentiremo il drive in movimento: la prima registrazione è stata inserita nella prima posizione del file relativo Articoli. Se vogliamo inserire qualche altro articolo non c'è da fare complimenti: dato che dovremmo digitare di nuovo make Articoli, basterà inviare il solo punto e virgola al sistema.

Proviamo ora a inserire qualche cliente: come nel caso precedente, dopo aver digitato make Clienti, non resterà che rispondere alle domande del sistema. Analogamente possiamo inserire un ordine: digiteremo make Ordini. In questo caso alla richiesta Articoli digiteremo, ad esempio, VIC 20. Dato che il tipo era extkeys (al plurale) nuovamente il sistema chiederà Articoli: digitiamo il secondo articolo ordinato, ad esempio Apple 2/C. Se non ci sono più articoli da inserire basterà premere semplicemente [Return].

Se chiediamo al sistema un determinato ordine, ci verranno restituite oltre al numero e alla data, anche tutte le notizie riguardo il cliente e gli articoli da lui ordinati.

In figura 5 è mostrato un altro modo di memorizzare dati riguardanti delle persone: come si può notare sono usate due classi, una per sesso. L'associazione doppia tra le due classi permette di correlare tutte le persone sposate (o fidanzate). In questo modo, accedendo a una registrazione della classe maschi sapremo contemporaneamente se e con chi è legata sentimentalmente; idem per il viceversa, accedendo alla classe femminile.

La dichiarazione sarà di questo tipo:

```
class Maschi <-> (
NomeCognome←string and
Recapito←string and
Residenza←string and
Telefono←string and
Partner←extkey in Femmine)
key (NomeCognome)len(120);

class Femmine <-> (
NomeCognome←string and
Recapito←string and
Residenza←string and
Telefono←string and
Partner←extkey in Maschi)
key (NomeCognome)len(120);
```

Si noti che, essendo obbligatorio specificare un partner quando si creano gli elementi, per quelle persone celibi o nubili, basterà indicare come relativo partner un nome simbolico (es. "nessuno", "?" o altro) che praticamente non attuerà l'associazione con la classe adiacente. Cattiveria finale: è ovvio che se una sera non sappiamo che fare, e ci capita davanti il nostro innocente 64, caricato il Galileo/J, possiamo digitare "all Femmine with Partner = nessuno" e scegliere nella lista chi andare a trovare. Chissà...

Note finali

Il programma Galileo/J è disponibile presso la nostra redazione su Floppy Disc. Sul medesimo dischetto è fornita sia la versione Basic, listata sul numero scorso, sia compilata col famosissimo PETSPEED del Commodore 64. Sarà conveniente usare la versione compilata in quanto se si somma l'esasperante lentezza del drive 1541 a quella non indifferente del Basic del 64, gestire un po' di dati col Galileo/J non diventa una cosa tanto divertente. Almeno così si annullano (praticamente) i tempi di elaborazione, quali modifica degli indici in memoria primaria e analisi sintattica dei comandi impartiti, lasciando solo al 1541 la colpa di risposte non troppo immediate.

L'ultimo appunto riguarda le reali possibilità di utilizzo del Galileo/J. Bisogna dire che sessanta elementi per classe in alcuni casi sono davvero pochini, così come la possibilità di usare solo 8 attributi limita un bel po' le possibili applicazioni. Come dicemmo lo scorso mese, il Galileo/J è usato in questo contesto per mostrare qualcosa di soltanto prossimo a sistemi più seri di gestione per basi di dati. Serve cioè per vedere un po' più da vicino le classi e "giocare" alle basi di dati (ricordate il Basal 2.1? con quello giocammo a BEGIN-END col Vic). Se il nostro 64 avesse avuto un po' di Ram in più e un 1541 dieci volte più veloce (e non ci sembra proprio di chiedere troppo) qualcosa di più serio si poteva fare.

Della serie:... buon divertimento! MC