La Progettazione dei database relazionali

Crescenzio GALLO

Università di Foggia

PARTE PRIMA
Il Modello Logico dei Dati

1. I DATI E LA LORO FORMALIZZAZIONE

Durante il loro utilizzo in uno schema informativo organizzato, i dati subiscono diverse modificazioni: la loro analisi fornisce alcuni spunti interessanti per la comprensione della natura degli stessi. Si possono individuare tre stadi distinti attraverso i quali i dati transitano: ***dati di input***, prima che essi vengano introdotti stabilmente nello schema, quando sono ancora isolati e le loro associazioni non sono ancora stabilizzate; a caricamento effettuato i dati occupano il posto loro destinato nello schema informativo ed attivano le associazioni logiche da essi realizzate: in questa fase si parlerà di ***dati operazionali***. Quando lo schema informativo viene interrogato, e vengono ricercati i dati opportuni navigando attraverso lo schema e le sue associazioni logiche, ed infine i dati richiesti vengono presentati abbiamo i ***dati di output***.

In prima approssimazione possiamo dire che un database è una collezione di dati operazionali memorizzati ed utilizzati dalle applicazioni di un'organizzazione. I dati operazionali possono riguardare prodotti, fatti amministrativi, pazienti di un ospedale, studenti di un'università, pianificazioni di un'organizzazione. I dati di input diventano operazionali quando si integrano col resto del contenuto del database. I dati di output sono derivati da quelli operazionali per elaborazione (ordinamento, classificazione, aggregazione, selezione, etc.). I dati operazionali sono tali perché collegati tra di loro tramite associazioni, che sono dunque parte integrante dei dati.

I dati sono concettualmente separati nelle loro componenti costitutive essenziali: entità ed associazioni. Le entità sono gli oggetti fisici, i documenti, le persone; le associazioni i fatti che le coinvolgono. Il modello ispirato a queste considerazioni è noto come *"modello entità-associazioni"* (Entity-Relationship Model, la cui data di nascita risale al 1976, anno della pubblicazione su ACM TODS di un famoso articolo di Peter Chen) ed è uno dei più potenti nel catturare il significato dei dati, indipendentemente dal modello concettuale di database (gerarchico, reticolare, relazionale) impiegato, e consente di progettare le basi di dati con un margine molto ridotto di indeterminazione semantica.

2. L'ARCHITETTURA ANSI / SPARC

Nel 1972 il comitato X3 dell'ANSI (American National Standard Institute) su proposta dello SPARC (System Planning and Requirements Committee) ha costituito un gruppo di studio con lo scopo di fornire alcuni standard nella realizzazione degli aspetti funzionali di un DBMS (Database Management System): senza entrare nel dettaglio delle implementazioni fisiche, il gruppo ANSI/SPARC ha infine nel 1978 fornito una visione unitaria delle interfacce dei DBMS e dei meccanismi di visibilità logica delle entità fisiche che compongono un database (vedi fig.1) nel noto documento *"The ANSI/X3/SPARC DBMS Framework: Report of the Study Group on Database Management Systems"*. L'indicazione fondamentale derivata da tale studio è il concetto di *architettura a tre livelli* per i database: il livello di riferimento è la definizione (schema) concettuale del database; a questo corripondono (da parti opposte) il livello del database fisico ospite della macchina e del suo software di base, ed il livello esterno attraverso il quale il database viene percepito dagli utenti.

3. IL MODELLO ENTITÀ-ASSOCIAZIONI

Lo scopo del modello entità-associazioni (Entity/Relationship, E/R) è quello di permettere la descrizione dello schema concettuale, senza preoccuparsi dell'efficienza o della progettazione del database fisico. Un'***entità*** è un qualcosa che può essere distintamente riconosciuto. Le entità possono essere classificate in *tipi di entità*, come ad es. PARTI e PROGETTI in fig.2; nei diagrammi E/R un'entità è rappresentata da un rettangolo. Ci sono molte "cose" nel mondo reale: alcune di interesse per l'organizzazione da modellare, altre no. È responsabilità del progettista del DB decidere quali sono di interesse e quali no per la base dei dati da rappresentare e gestire. Le ***associazioni*** (da non confondere con le tabelle o relazioni del modello di database relazionale, come si vedrà nella parte seconda) possono esistere tra entità. Per esempio l'associazione MATRIMONIO è una relazione tra due tipi di entità di persone. Le associazioni possono essere classificate in *tipi di associazioni*. Ad esempio, PROG-IMP e PROG-DIR sono due differenti tipi di associazioni tra due tipi di entità (impiegati e progetti). Nei diagrammi E/R un'associazione è rappresentato da una figura romboidale con delle linee connesse alle entità relazionate. Possiamo considerare vari tipi di associazioni, come evidenziato in fig.2. La nozione di *n* e di *1* presente nell'associazione PROG-DIR indica che il progetto ha un solo direttore, ma che un impiegato può essere il diret-

tore di più progetti (i direttori sono degli impiegati). Sempre considerando tale figura, la *m* e la *n* nell'associazione PROG-IMP indicano che c'è una corrispondenza *molti-a-molti*: cioè ogni progetto può coinvolgere molti impiegati, ed ogni impiegato può partecipare a più di un progetto. Ad esempio, l'associazione MATRIMONIO è una corrispondenza *uno-a-uno* tra le entità PERSONA.
È possibile definire un'associazione tra più di due tipi di entità. Ad esempio, l'associazione PARTI-FORN-PROG di fig.3 descrive quali parti sono fornite da un particolare fornitore a determinati progetti. Anche qui, non sempre un'associazione connettente tre entità può essere rappresentata con due o persino tre associazioni binarie perché talvolta si possono generare "non fatti", come si può evincere in fig.3.

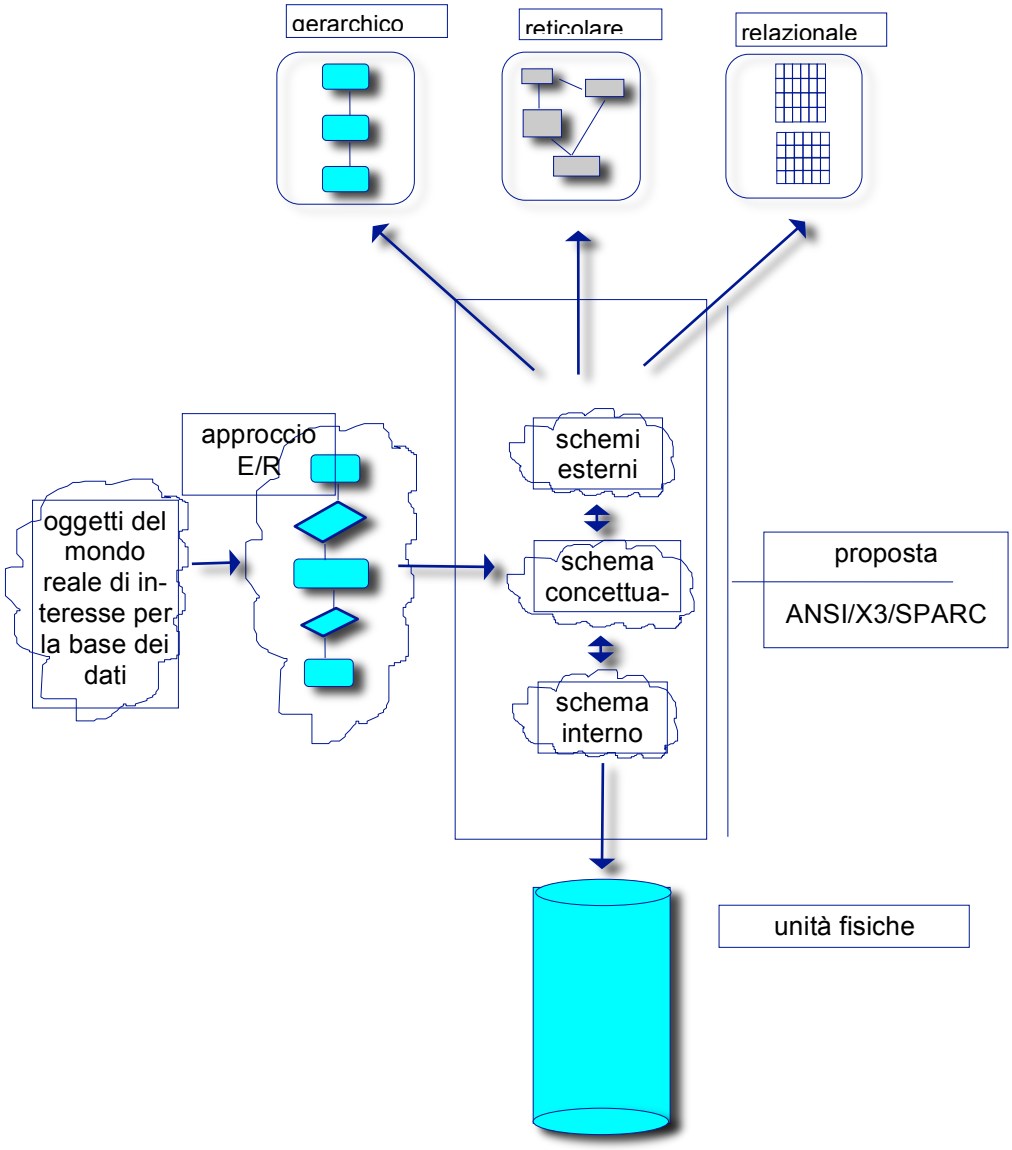


Fig. 1 - Introduzione all'approccio E/R

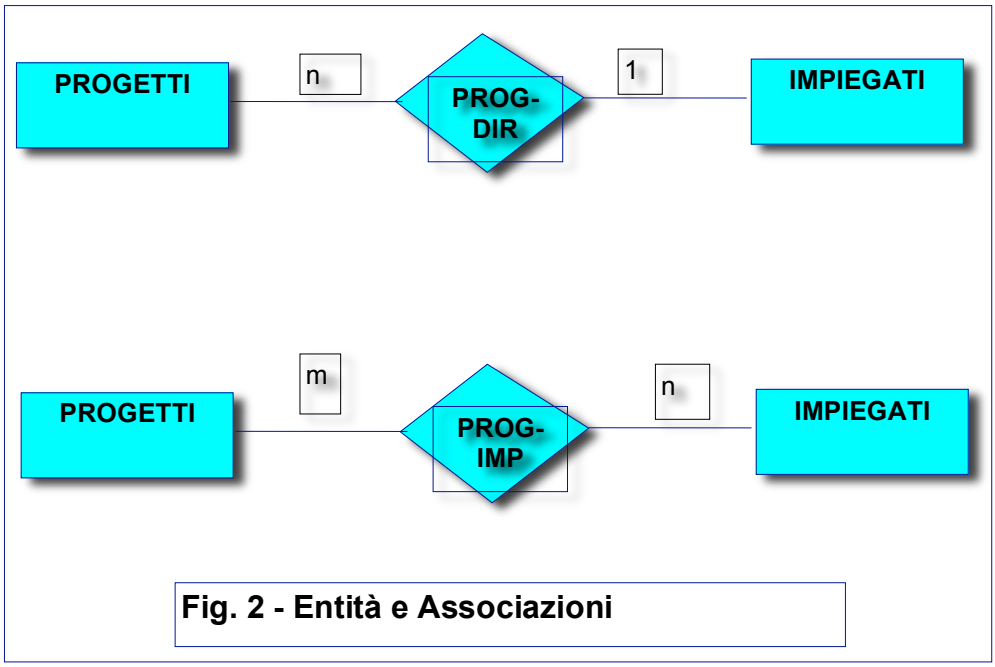


Fig. 2 - Entità e Associazioni

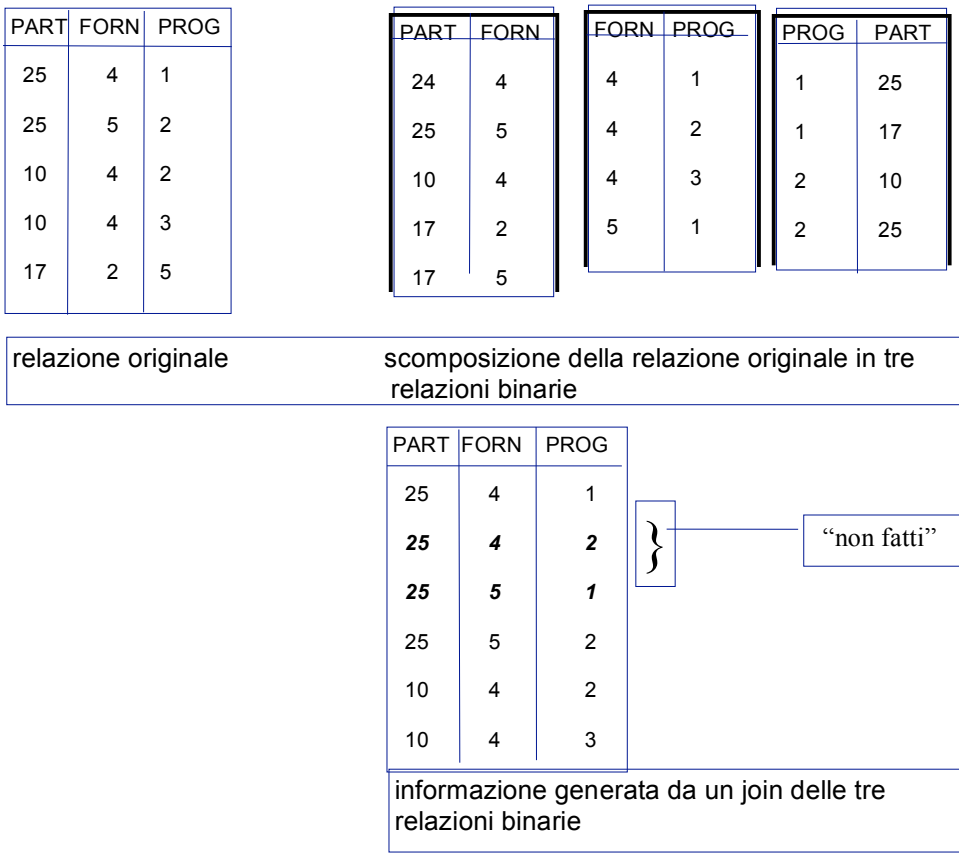


Fig. 3 - Scomposizione di relazioni

Entità e associazioni hanno delle proprietà che possono essere espresse in termini di attributi/valori. Cioè, quando si dice che "l'età di un impiegato è 24" allora "età" è un ***attributo*** dell'impiegato e 24 è il ***valore*** dell'attributo età. I valori possono essere classificati in differenti tipi di valore o domini, come "numero di anni", "quantità" e "colore". Nei diagrammi E/R un dominio è rappresentato da un cerchio e un attributo è rappresentato da una freccia diretta dal tipo di entità al desiderato tipo di valore (fig.4). In alcuni casi un attributo può avere più di un valore per una data entità. Ad esempio, il "numero telefonico" di un certo impiegato può avere più valori: in questi casi si mette *1:n* sulla freccia per indicare che è un attributo multivalore (fig.4). Per semplicità si omette *1:1* per attributi con valori unici.

Talvolta sono anche interessanti le proprietà delle associazioni. Ad esempio, se si vuole conoscere "quando un dato impiegato ha cominciato a lavorare ad un dato progetto" la DATA_DI_INIZIO non è nè un attributo di "impiegato" nè un attributo di "progetto", bensì il suo valore dipende sia dal particolare impiegato che dal particolare progetto. Perciò la DATA_DI_INIZIO è un attributo dell'associazione, cioè di PROG-IMP. Il concetto di attributo dell'associazione è importante per capire la semantica dei dati. Tale concetto è simile alle "relazioni tra i dati" nel sistema CODASYL reticolare e alle "intersezioni tra i dati" nel sistema IMS di tipo gerarchico.

Naturalmente c'è bisogno di *identificare* le entità. Un comune metodo per identificare le entità è quello di usare gli attributi ed i valori. Ogni entità ha vari attributi: quali scegliere? La risposta è che gli attributi da scegliere sono quelli che possono identificare le entità. Cioè si può, ed esempio, considerare l'attributo NOME per identificare gli impiegati di una piccola impresa, ma non in una grande impresa. Gli attributi scelti sono detti ***identificatori delle entità***. Talvolta è difficile trovare un attributo che sia un identificatore; quello che si può fare in tali casi è creare un attributo artificiale che possa identificare positivamente le entità. Tale concetto è molto simile al concetto di chiave primaria. Le associazioni sono identificate utilizzando gli identificatori inclusi nelle entità. Ad esempio, se un progetto è identificato dal suo "numero di progetto" ed un impiegato dal "numero di matricola", l'associazione PROG-IMP è identificata da entrambi gli identificatori delle entità. In alcune associazioni ci sono due occorrenze dello stesso tipo di entità. Per esempio, MATRIMONIO è un'associazione definita tra due occorrenze dello stesso tipo di entità, PERSONA. Per identificare positivamente questo tipo di associazione non si usano gli identificatori delle entità, ma si indica il ruolo che le entità hanno nell'associazione, cioè MARITO e MOGLIE al posto di NOME.

Nell'approccio E/R esistono alcuni tipi di ***dipendenze*** tra le entità. Una di queste è la ***dipendenza di esistenza***. Ad esempio, l'esistenza dell'entità BAMBINI dipende dall'esistenza dell'associato impiegato. In altre parole, se un impiegato lascia l'azienda non rimarrà traccia (nella base dati, ovviamente) dei suoi bambini. La fig.5 indica tale situazione: si nota che BAMBINI è in un doppio rettangolo, e la ***E*** nell'associazione indica la dipendenza di esistenza la cui direzione è data dalla freccia.

È anche possibile che tale dipendenza di esistenza sia di tipo molti-a-molti. Si ha una ***dipendenza ID*** quando un'entità non è univocamente determinata dai suoi propri attributi e deve essere determinata anche da altre entità. Ad esempio, una strada è unica solo in una città, una città solo in uno stato. Per cui, per identificare l'indirizzo di un luogo, oltre al nome della strada, bisogna specificare il nome della città ed il nome dello stato.

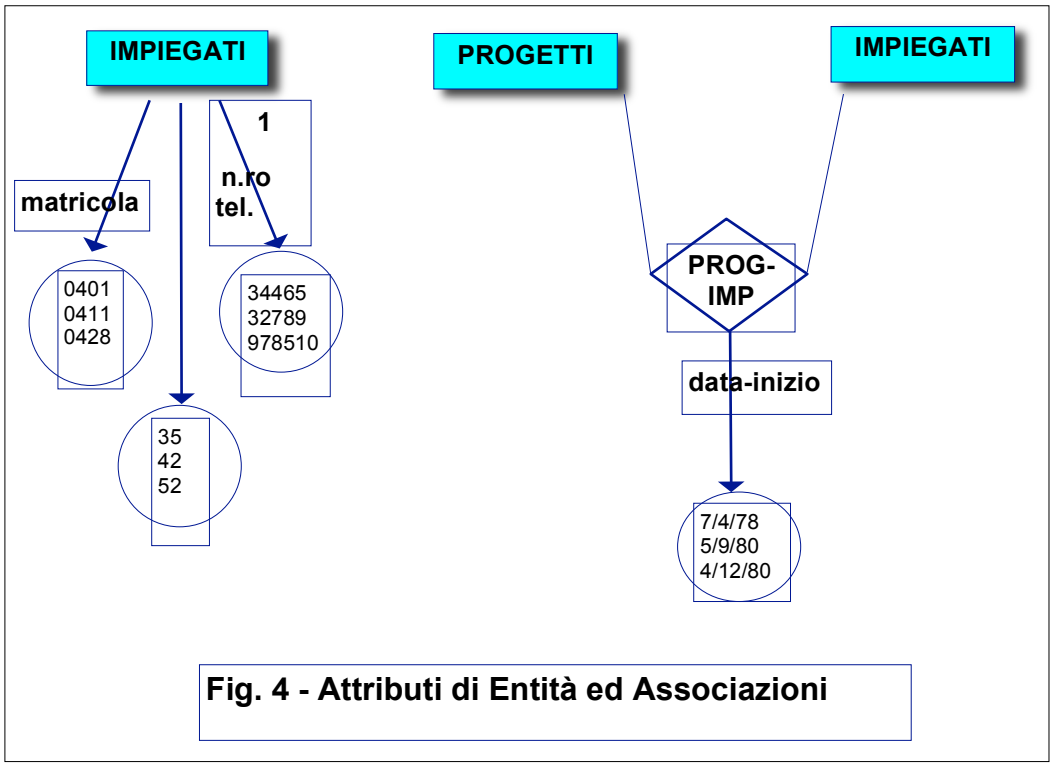


Fig. 4 - Attributi di Entità ed Associazioni

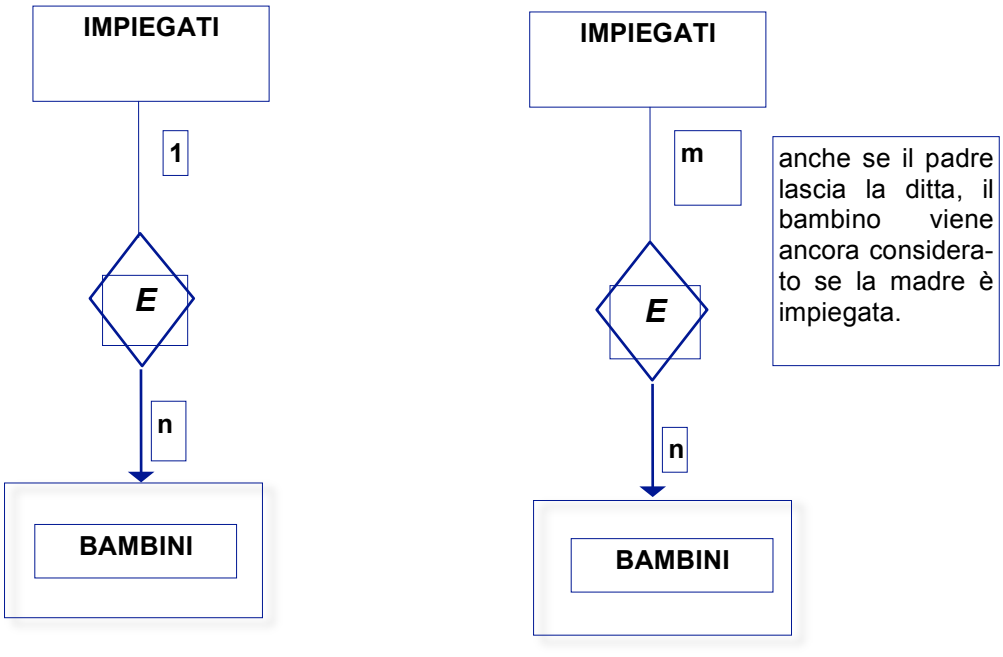


Fig. 5 - Dipendenza di Esistenza

La fig.6a indica una dipendenza ID. Di solito la dipendenza ID comprende la dipendenza di esistenza, ma non è vero il contrario. Diciamo infine che *A **isa** B* se il tipo di entità B è una generalizzazione di entità del tipo A, o equivalentemente se A è un tipo particolare di B. Lo scopo delle dipendenze di tipo **isa** consiste nella possibilità di descrivere le classi di oggetti. **Isa** significa letteralmente "è un" e si riferisce al caso in cui un'entità è una sottoclasse di un'altra. Per esempio, i dirigenti sono una sottoclasse dei dipendenti di una società, come indicato in fig.6b.

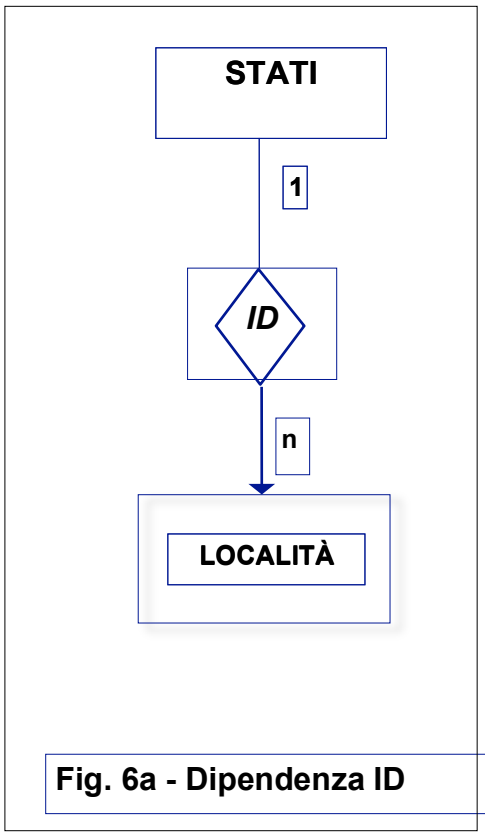


Fig. 6a - Dipendenza ID

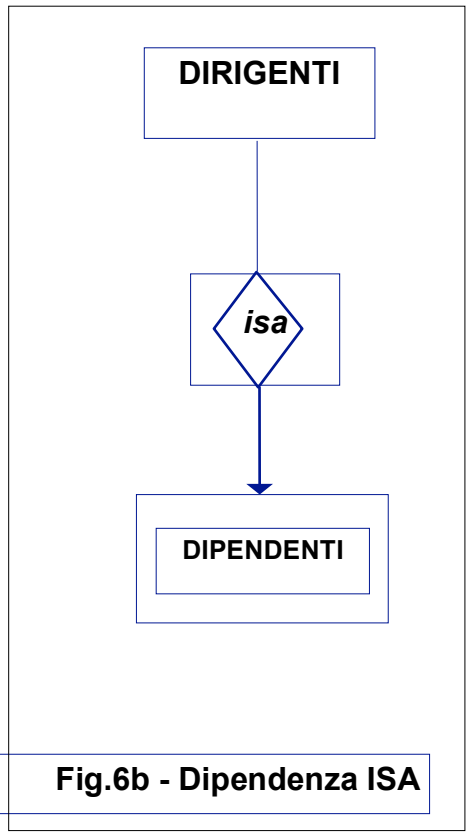


Fig.6b - Dipendenza ISA

4. TRADUZIONE DEL MODELLO ENTITÀ-ASSOCIAZIONI

La traduzione del diagramma E/R in un diagramma per strutture di dati avviene molto semplicemente. Questa volta si useranno solo forme rettangolari per tipi di record (intesi nella terminologia classica dei file) o tabelle (nella terminologia dei database relazionali). La freccia in tali casi rappresenta gli insiemi di strutture dei dati e connette sempre due tipi di record. Il record in cui la freccia è originata viene detto record owner, e quello in cui la freccia termina viene detto member della struttura di dati. In questi insiemi di strutture il record member ha esattamente un solo record owner, mentre il record owner può avere zero, uno o più record member. Ci sono diverse regole basilari per tale traduzione, basate sul tipo di relazione tra entità. Esse possono essere così riassunte.

4.1. Relazioni definite su due differenti tipi di entità

a) Se la relazione è uno-a-molti (oppure uno-a-uno) come quella mostrata in fig.7a, essa può essere trasformata in una equivalente come quella in fig.7b. In questo caso i tipi di entità sono trattati come record, mentre la relazione è rappresentata da una freccia.

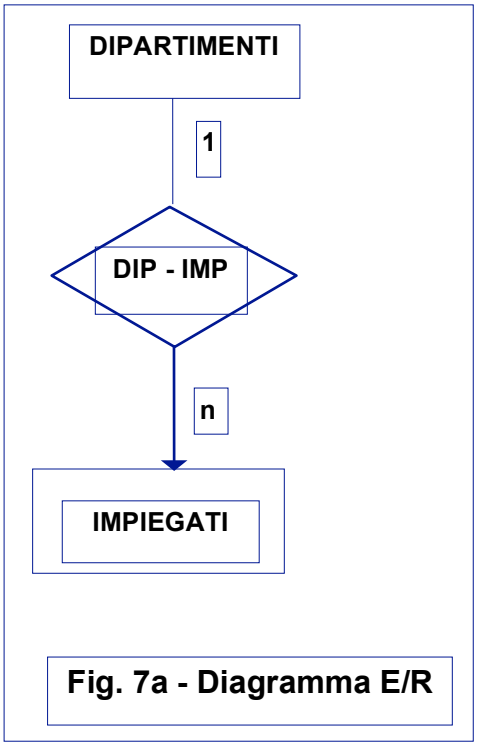


Fig. 7a - Diagramma E/R

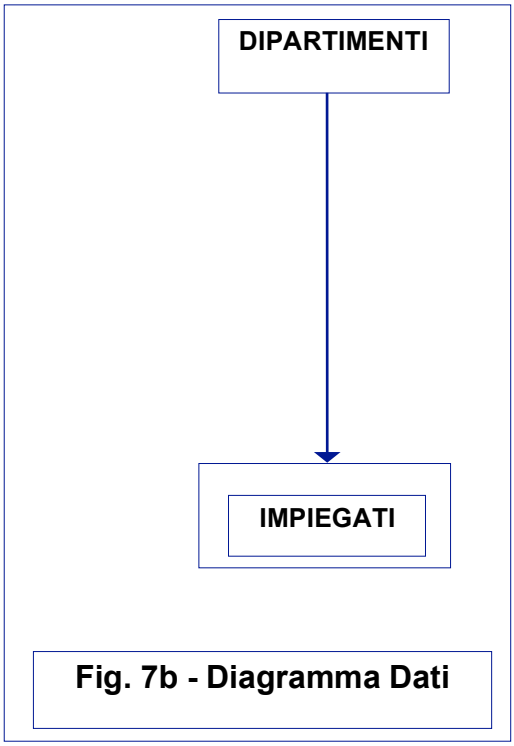


Fig. 7b - Diagramma Dati

b) Se la relazione è di tipo molti-a-molti (come la relazione in fig.8a) essa verrà trasformata in una relazione equivalente come quella in fig.8b. In questo caso il tipo di relazione non è trasformato in una freccia, ma piuttosto come un tipo di record. Cioè se un tipo di relazione è una corrispondenza molti-a-molti essa sarà tradotta in un tipo di record con due frecce che provengono dalle entità relazionate.

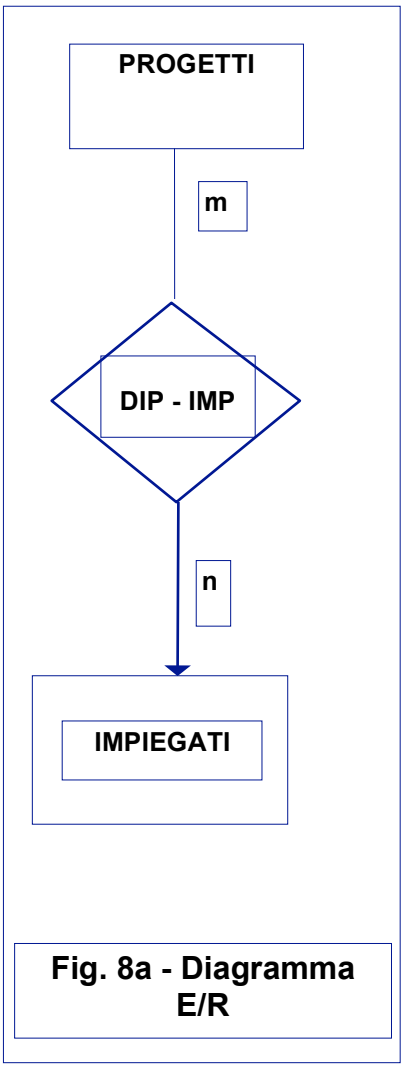


Fig. 8a - Diagramma E/R

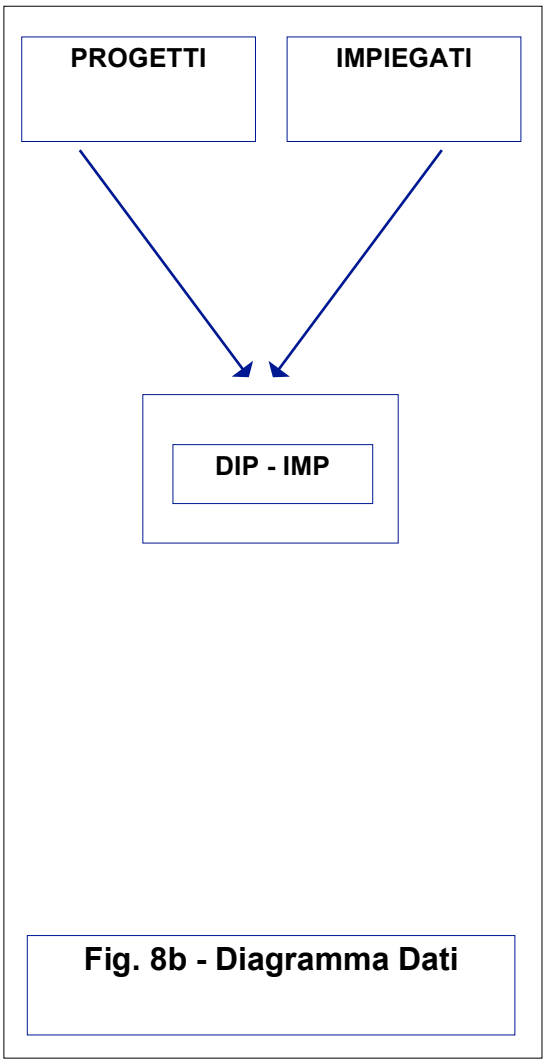


Fig. 8b - Diagramma Dati

4.2. Relazioni definite tra più di due entità

In questo caso la relazione del diagramma E/R sarà tradotta in un tipo di record. Ciò è illustrato dalle figg. 9a e 9b.

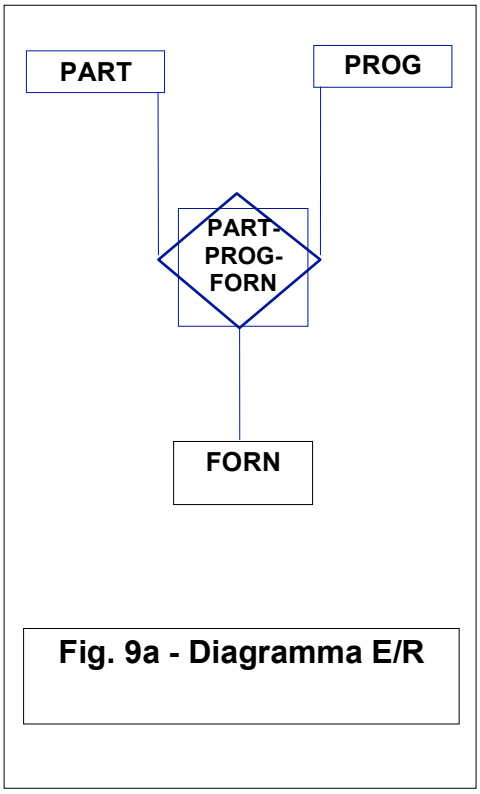


Fig. 9a - Diagramma E/R

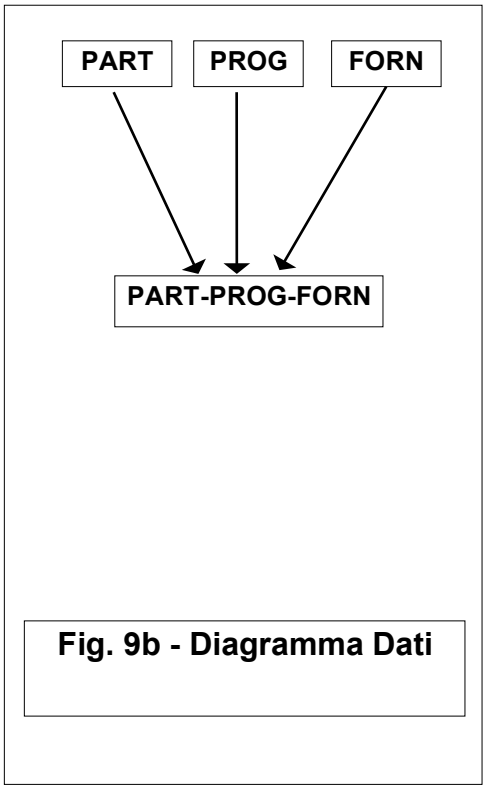


Fig. 9b - Diagramma Dati

4.3. Relazioni binarie definite sullo stesso tipo di entità

Una relazione binaria definita sullo stesso tipo di entità (come la relazione MATRIMONIO in fig.10a) potrà essere trasformata in un diagramma di strutture dati in due diverse maniere, come in fig.10b e fig.10c. Nel caso in cui una particolare implementazione di databse non permetta allo stesso tipo di record (tabella) di essere usato allo stesso tempo sia come owner che come member, si farà solo uso della fig.10c per tradurre la situazione descritta in fig.10a. Essa si usa anche nel caso di relazioni molti-a-molti.

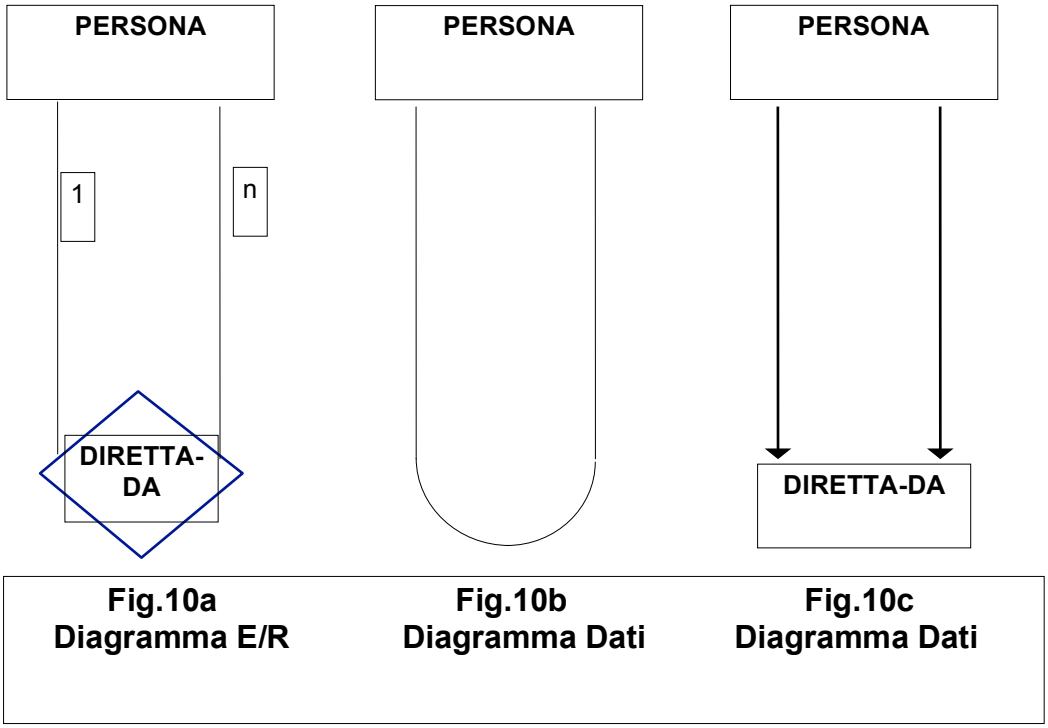


Fig.10a Diagramma E/R | **Fig.10b Diagramma Dati** | **Fig.10c Diagramma Dati**

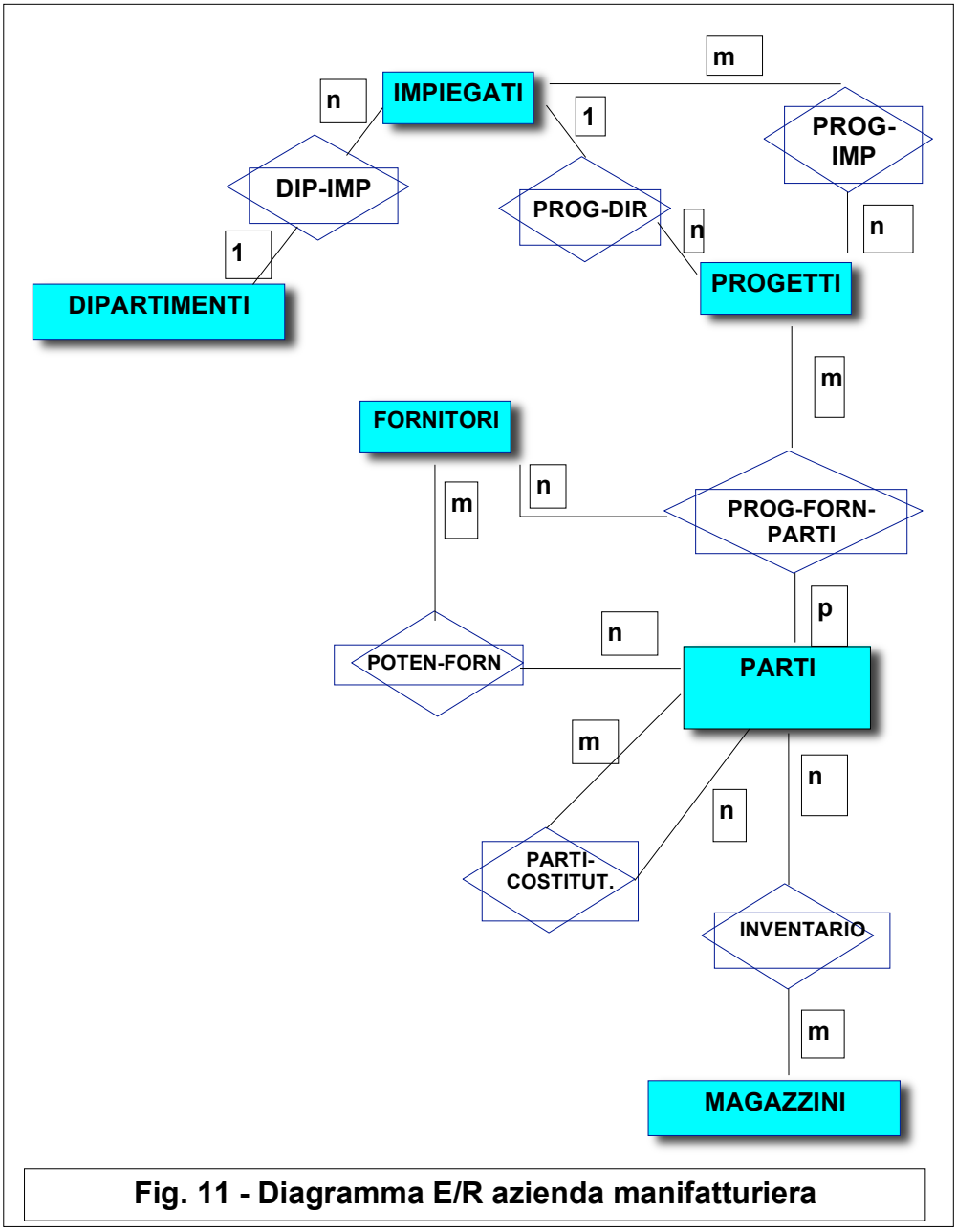


Fig. 11 - Diagramma E/R azienda manifatturiera

5. IL PROGETTO LOGICO DI UN DATABASE

I passi fondamentali per il progetto logico di un DB sono:

1) identificare i tipi di entità;
2) identificare i tipi di relazioni;
3) disegnare un diagramma E/R con tipi di entità e relazioni;
4) identificare i tipi di valori (domini) e gli attributi;
5) tradurre il diagramma E/R in un diagramma di struttura dei dati;
6) progettare i formati dei record (strutture delle tabelle).

Per illustrare come ciò sia realizzato, consideriamo come esempio un'azienda manifatturiera.

1) I tipi di entità di interesse per l'impresa sono MAGAZZINI, PARTI, IMPIEGATI, DIPARTIMENTI, PROGETTI e FORNITORI.
2) I tipi di relazioni esistenti tra queste entità possono essere:
   - DIP-IMP, che descrive il dipartimento per gli impiegati (uno-a-molti);
   - PROG-IMP, che descrive i progetti associati agli impiegati. È una corrispondenza molti-a-molti perché un impiegato può lavorare a più di un progetto, ed ogni progetto può includere più di un impiegato;
   - PROG-DIR, che identifica i direttori dei progetti (uno-a-molti); ogni progetto ha un solo direttore, ma ogni impiegato (come direttore) può essere associato con più di un progetto;
   - PROG-FORN-PARTI, che è una relazione che descrive quali fornitori forniscono certe parti per certi progetti; è una relazione molti-a-molti-a-molti, cioè per una particolare parte c'è più di un progetto. Analogamente, ogni progetto può usare più di una parte, che può avere più di un fornitore. Ancora, ogni fornitore può fornire ad un progetto più di una parte;
   - POTEN-FORN, che descrive una lista di potenziali fornitori ed è una corrispondenza molti-a-molti. Infatti ogni parte può avere più di un potenziale fornitore, ed ogni fornitore può essere in grado di fornire più di una parte;

   la relazione INVENTARIO conosce quali parti sono in magazzino ed è una relazione molti-a-molti.

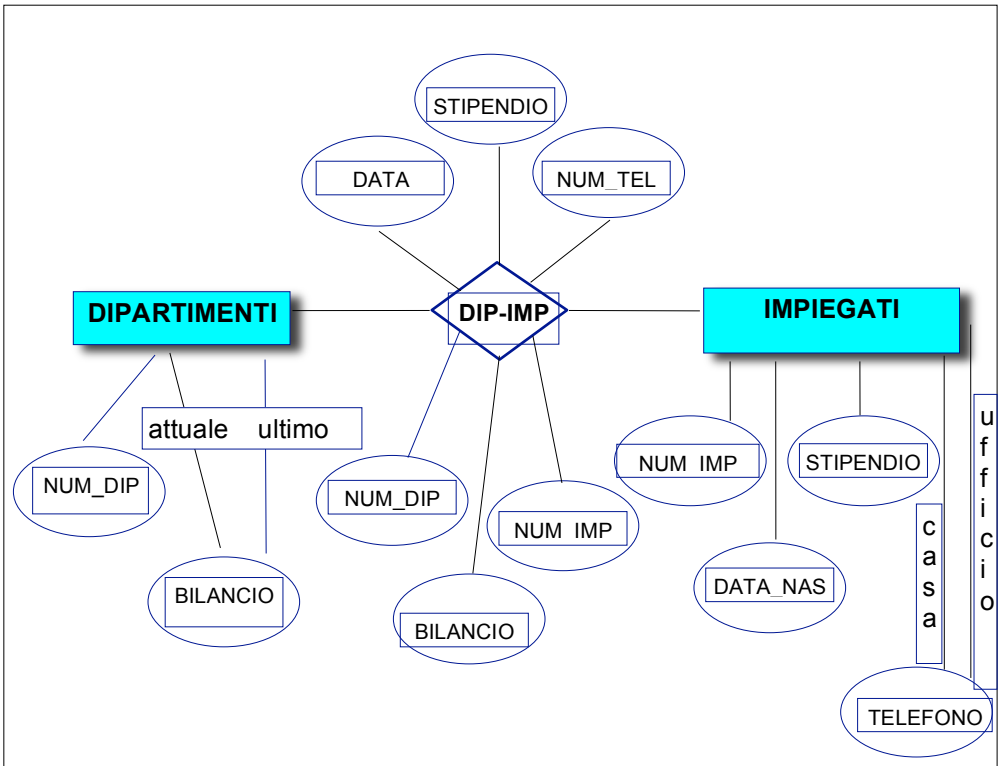


Fig. 12 - Attributi per le entità DIPARTIMENTI e IMPIEGATI e l'associazione DIP-IMP

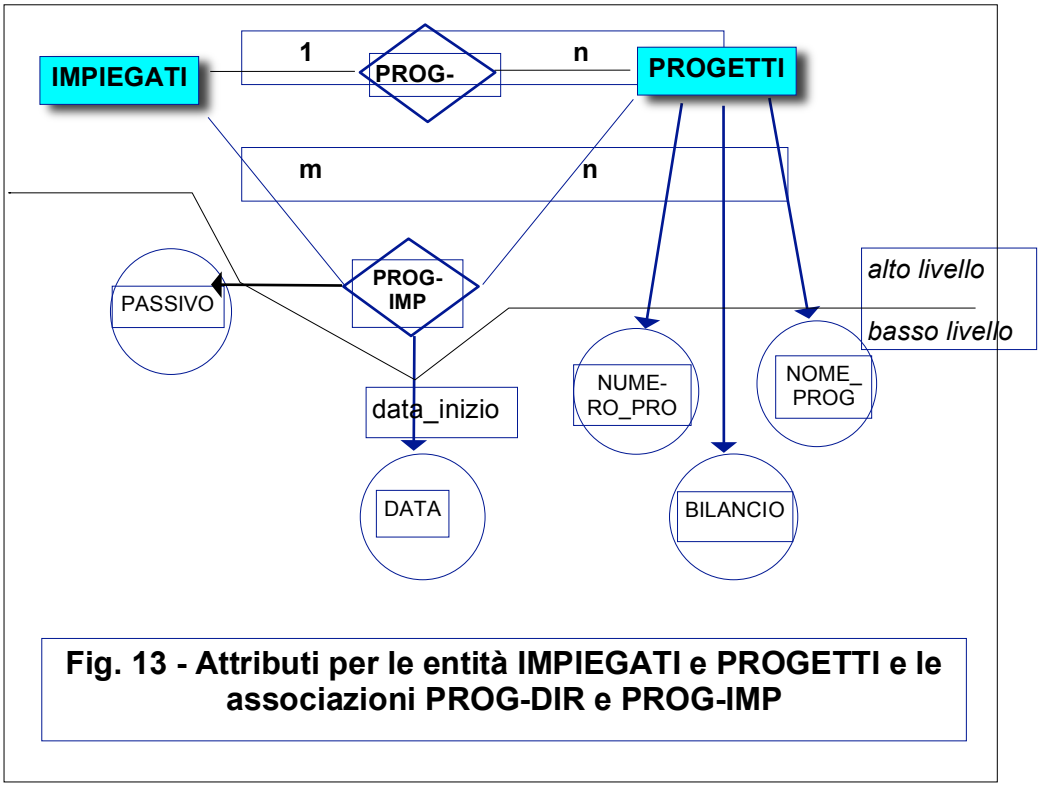


Fig. 13 - Attributi per le entità IMPIEGATI e PROGETTI e le associazioni PROG-DIR e PROG-IMP

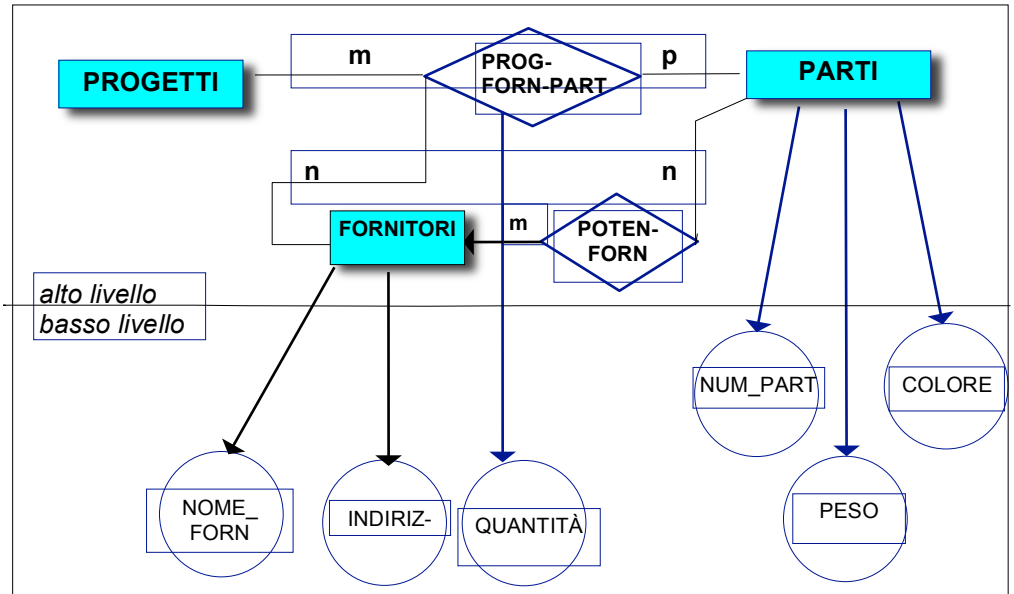


Fig. 14 - Attributi per le entità PROGETTI, PARTI e FORNITORI e le associazioni PROG-FORN-PART e POTEN-FORN

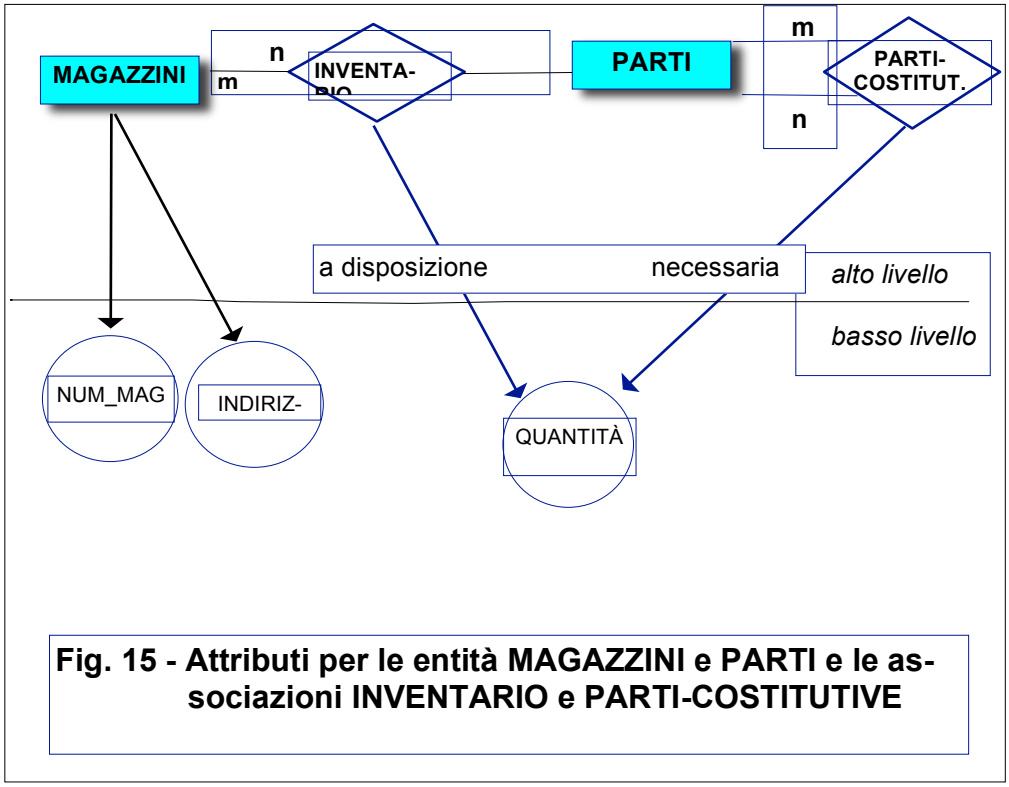


Fig. 15 - Attributi per le entità MAGAZZINI e PARTI e le associazioni INVENTARIO e PARTI-COSTITUTIVE

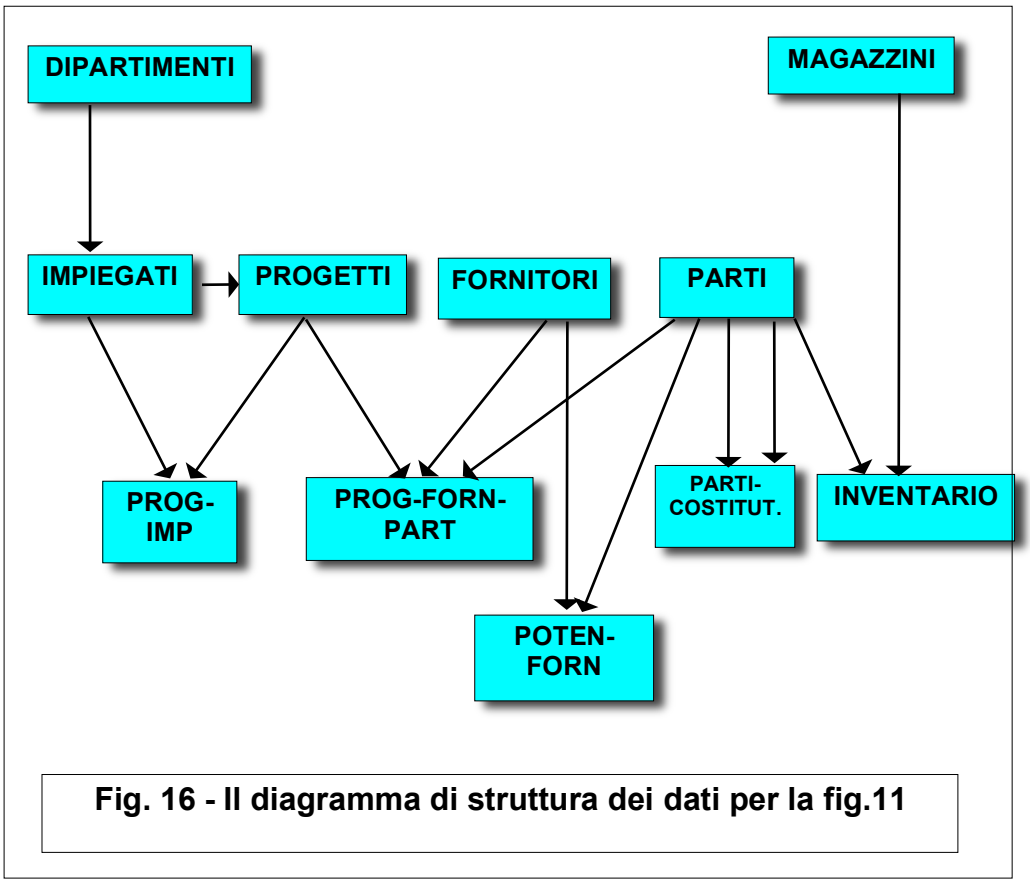


Fig. 16 - Il diagramma di struttura dei dati per la fig.11

NUM_DIP	ATTUALE_BILANCIO	ULTIMO_BILANCIO

Fig. 17 - Record DIPARTIMENTI

NUM_IMP	DATA_NAS	STIPENDIO	TEL_CASA	TEL_UFFICIO	NUM_DIP

NUM_PROG	NOME_PROG	BILANCIO

Fig. 18 - Record IMPIEGATI e PROGETTI

NUM_IMP	NUM_PROG	DATA_INIZIO	PASSIVO

Fig. 19 - Record PROG-IMP

NUM_PROG	NUM_FORN	NUM_PART	QUANTITÀ

Fig. 20 - Record PROG-FORN-PART

3) Questo terzo passo è mostrato in fig.11. Si noti che è stata inserita l'associazione PARTI-COSTITUTIVE perché le parti in genere sono costituite da altre parti: si tratta di un'associazione molti-a-molti.
4) La fig.12 illustra gli attributi per i tipi di entità DIPARTIMENTI e IMPIEGATI e la loro associazione DIP-IMP. Per tale relazione sono stati identificati i seguenti tipi di valore: NUM_DIP, BILANCIO, NUM_IMP, DATA, SALARIO, NUM_TEL. DIPARTIMENTI ha tre attributi: NUM_DIP, ATTUALE_BILANCIO, ULTIMO_BILANCIO. IMPIEGATO ha cinque attributi: NUM_IMP, DATA_NAS, SALARIO, TEL_CASA e TEL_UFFICIO. Si nota che gli attributi possono non avere lo stesso nome, ma possono avere lo stesso tipo di valore (dominio). Per esempio, ATTUALE_BILANCIO e ULTIMO_BILANCIO usano lo stesso tipo di valore. La stessa cosa vviene per TEL_CASA e TEL_UFFICIO che utilizzano lo stesso dominio NUM_TEL.

Per semplificare la figura si è omesso il nome degli attributi del diagramma se essi sono unici con il loro tipo di valore. Le figg. 13, 14 e 15 illustrano valori ed attributi per il resto del diagramma E/R.

5) Per quanto detto prima il diagramma E/R di fig.11 verrà trasformato nel diagramma di struttura dei dati di fig.16. Tutti i tipi di entità diventano tipi di record (tabelle), mentre le relazioni *1:n* (uno-a-molti) vengono indicate con una freccia; quelle *m:n* (molti-a-molti) con tipi di record (tabelle). Anche la relazione PARTI-COSTITUTIVE viene tradotta in un tipo di record (tabella) perché è una relazione definita sullo stesso tipo di entità, e quindi verranno usate due frecce che provengono dallo stesso tipo di record (tabella) PARTI.
6) Per quanto riguarda il progetto dei formati dei record (strutture delle tabelle) essi conterranno tutti gli attributi. Bisogna però decidere come implementare queste strutture. Le regole basilari da seguire sono le seguenti:
   - tutti gli attributi di un'entità saranno posti nello stesso tipo di record/tabella (fig.12 e fig.17);
   - se la relazione è uno-a-molti allora gli attributi di tale relazione verranno trattati come campi del record member (cioè della tabella principale, fig.12 e fig.18);
   - se la relazione è tradotta in tipo di record/tabella, allora gli attributi di tale relazione saranno trattati come campi di un record (colonne di una tabella): è il caso delle figg. 13, 14, 19 e 20.

6. CONSIDERAZIONI FINALI

Le regole di traduzione di un diagramma E/R in una struttura di dati viste precedentemente, sono comunemente usate ma non sono le sole. Si possono usare delle regole ancora più semplificate, ma il diagramma risultante sarà meno efficiente in termini di ritrovamento ed adattamento. Quindi vanno bene per DB che difficilmente cambiano nel tempo. Inoltre si possono usare degli accorgimenti per aumentare l'efficienza, che però dipendono caso per caso dalla struttura dei dati da ottimizzare.

Da quanto detto e dall'esempio riportato, appare evidente che l'approccio E/R al progetto di schema concettuale ha grossi vantaggi; questi essenzialmente si possono riassumere in una grossa potenza nel rappresentare i fatti del mondo reale di interesse per la base dei dati. Ma oltre a questi vantaggi, che sono comunque fondamentali, esistono altri vantaggi che non possono essere tralasciati. Ad esempio, la divisione del lavoro e le funzioni in due fasi rende il progetto del DB più semplice e meglio organizzato. Lo schema E/R è più facile da disegnare piuttosto che uno schema utente, dato che non è ristretto dalle capacità del database ed è indipendente da considerazioni di spazio ed efficienza. Inoltre lo schema E/R è più stabile rispetto agli schemi utenti: infatti, se si vuole cambiare un DB in un altro probabilmente si deve cambiare lo schema utente, ma non lo schema E/R, dato che quest'ultimo è indipendente dal DBMS utilizzato. Quello che deve essere fatto è rimappare lo schema E/R ad uno schema utente adatto per il nuovo sistema. Analogamente, se si vuole cambiare uno schema utente per ottimizzare una nuova esigenza applicativa, non è necessario cambiare lo schema E/R, ma piuttosto rimappare tale schema in un nuovo schema utente.

Lo schema concettuale così concepito è probabilmente la migliore soluzione per una metodologia che renda il processo di organizzazione dei dati più facile da capire. Infatti, tale progetto si interessa esclusivamente alla rappresentazione dei fatti del mondo reale di interesse per la base dei dati al di fuori da qualsiasi restrizione come le condizioni di efficienza e di spazio che rendono talvolta difficile e tedioso il lavoro stesso.

7. ALCUNE OBIEZIONI AL MODELLO E/R

Fin dal suo apparire il modello E/R è stato accolto con entusiasmo dal mondo EDP, principalmente per i formalismi grafici di cui è stato dotato. Ma, contemporaneamente, veniva definito da Chris Date *"a thin layer on top of the Relational Model"* (un sottile strato sovrapposto al modello relazionale), mentre due autorevoli professionisti della modellazione dei dati - D.C.Tscihritzis e F.H.Lochovsky - lo consideravano una semplice generalizzazione dei modelli gerarchico e reticolare. Secondo E.F. Codd, il padre del modello relazionale, ripartire gli oggetti nel data-base in due grandi categorie, le entità e le associazioni, è un rifarsi al passato prerelazionale, nel quale i record venivano trattati uno alla volta. L'approccio E/R manca inoltre - sempre secondo Codd - di precise definizioni, di un chiaro livello di astrazione e di disciplina mentale: deve essenzialmente la sua popolarità alla moltitudine di interpretazioni che ammette ed all'uso di schemi di pensiero obsoleti.

Tali obiezioni si fondano, al di là della visione di parte che le sostiene (modello relazionale contro modello entità/associazioni), su argomenti che non possono essere trascurati e che possiamo così di seguito riassumere.

- Gli aspetti negativi riguardano innanzitutto la dubbia distinzione nel modello E/R tra entità ed associazioni. Nell'esempio dell'associazione MATRIMONIO, essa è un'associazione tra due entità PERSONA o un'entità a sè stante? Non è semplice deci-

dere: infatti, a seconda dei punti di vista potrebbe essere accettata l'una o l'altra interpretazione.

- A differenza delle entità, un'associazione nasce senza proprietà immediate, e cioè senza attributi che la riguardino direttamente. Ma l'associazione potrebbe acquisire successivamente queste proprietà: sempre nell'esempio citato, un'associazione MATRIMONIO potrebbe acquisire nel tempo un attributo DURATA_DEL_MATRIMONIO, trasformandosi così in una vera e propria entità. In questo caso ci vedremo costretti a rimaneggiare lo schema dei dati, che dovrebbe essere invece invariante nel tempo, con tutte le conseguenze ed i fastidi che si possono facilmente immaginare. Per la verità, forse per ovviare proprio a questo inconveniente, Chen ha introdotto nel 1985 un nuovo elemento nel proprio modello: l'entità composita, che può essere dotata di attributi propri.
- Resta infine il problema degli operatori. Se entità ed associazioni (per non parlare delle entità composite) corrispondono a due diverse categorie di informazioni, dovrebbero disporre anche di due set distinti di operatori per l'INSERT, il DELETE, etc. (come accade per i DBMS che aderiscono al modello CODASYL). Se invece sotto il profilo manipolativo non vengono fatte distinzioni tra entità ed associazioni, perché considerarle cose a sè stanti?

In conclusione, comunque, si può dire che il modello E/R - nonostante le ragionevoli obiezioni che gli si possono muovere - sia uno strumento valido per la modellazione dei dati, specialmente nella fase iniziale preliminare alla progettazione specifica nel modello di DBMS scelto (gerarchico, reticolare o relazionale che sia).

PARTE SECONDA
I Database Relazionali

1. INTRODUZIONE

Fin dagli anni ‘60 i progettisti si erano resi conto dei limiti e degli inconvenienti dei metodi allora utilizzati per la gestione dei dati (ad esempio, i dati erano fisicamente legati alle caratteristiche delle memorie, non potevano essere facilmente condivisi da più utenti, nè adeguatamente protetti da eventuali alterazioni) e si erano messi a cercare delle soluzioni più valide.

I sistemi per la gestione di basi di dati o DBMS (Database Management System) nacquero quindi dall’esigenza di trovare un modo per organizzare e gestire meglio grandi volumi di dati, in termini di integrità, sicurezza, possibilità di condivisione fra più utenti e rispetto di standard.

Tutti obiettivi relativamente abbordabili, rispetto a quello dell’indipendenza dei dati, ossia della trasparenza delle applicazioni ai cambiamenti dell’ambiente fisico e logico. Ad esempio, nei programmi tradizionali la gestione degli archivi veniva in larga misura affidata alle risorse del sistema operativo, ma qualsiasi cambiamento della struttura fisica dei supporti di memorizzazione (variazione delle dimensioni dei blocchi o della disposizione dei campi) comportava dei pesanti interventi di adattamento. Di conseguenza i DBMS si propongono innanzitutto di isolare le applicazioni dalla struttura dei supporti di memorizzazione e dalle vie di accesso ai dati.

Ma, per arrivare al vero obiettivo, bisogna prendere in considerazione anche la strategia di accesso, che dipende dalle relazioni logiche intercorrenti tra i dati. Per potersi orientare all’interno di un database bisogna infatti comprendere a fondo anche la struttura con cui vengono organizzati (ossia ordinati e raggruppati) i record, dato che, come una qualsiasi variazione strutturale dei supporti di memorizzazione, anche un cambiamento della struttura logica - andando a toccare le vie di accesso ai dati - può comportare delle modifiche di programma.

Il modello relazionale (sviluppato da Codd nel 1970) protegge però le applicazioni anche da questo secondo tipo di cambiamento e le rende quindi ancor più indipendenti dai dati, a differenza degli altri modelli (gerarchico e reticolare) disponibili a quell’epoca (come l’IMS di IBM e la proposta CODASYL) che richiedono di incorporare nell’applicazione la struttura dei campi e la via d’accesso ai dati e quindi di conoscerne a fondo l’organizzazione.

Un data-base deve invece essere in grado di espandersi - e cioè di accettare sia la ridefinizione di un campo (che più ad esempio passare da due a quattro byte) che l’aggiunta di un nuovo campo o di un nuovo tipo di record - senza interferire con le applicazioni esistenti.

L’obiettivo del modello relazionale è quindi quello di affrancare le applicazioni da qualsiasi forma di dipendenza dai dati, svincolando sia dalla loro organizzazione che dalle loro vie d’accesso.

2. IL MODELLO LOGICO

I database relazionali si distinguono specialmente per la notevole flessibilità e l'estrema chiarezza concettuale, che consente agevolmente di manipolare i dati con operazioni di tipo insiemistico. Il modello logico relazionale trae dalla teoria matematica degli insiemi una delle sue caratteristiche più significative ed attraenti: e cioè un'estrema chiarezza concettuale, che facilita la formulazione di un linguaggio ben definito per la manipolazione dei dati, basato interamente sulle operazioni dell'insiemistica.

Esaminiamo ora più in dettaglio la struttura di questo modello, la sua integrità e le operazioni che ne regolano il funzionamento.

2.1. La struttura.

Nel modello relazionale i dati si sviluppano intorno a delle tabelle bidimensionali, dette per l'appunto "relazioni". Ciascuna riga di una tabella rappresenta una *n-pla* della relazione e ciascuna colonna è detta *attributo*. I valori degli attributi vengono scelti nell'ambito di classi o gamme di valori accettabili (dominii).

Le relazioni possono essere rappresentate graficamente, come si può vedere dalla fig.1, che esemplifica una tabella chiamata CLIENTI con sei righe, ciascuna delle quali contiene quattro attributi: CODICE,NOME, CITTÀ e ZONA che possono assumere - come si è già detto - una certa gamma di valori (ad esempio per ZONA sono NW, CE, CN, NO, SI): queste sono le entità di base, gli oggetti distinguibili sui quali il database contiene delle informazioni.

I dati possono essere reperiti attraverso il nome della relazione, il nome dell'attributo ed il suo valore, ma non mediante un meccanismo fisico, come il numero d'ordine di una colonna o una catena di puntatori. A differenza dei loro predecessori i *database relazionali* risultano quindi basati su dei valori e non su degli indirizzi.

Una relazione tuttavia è qualcosa di più di una semplice tabella, perché deve obbedire ad una disciplina assicurata da un certo numero di proprietà. Ad esempio, l'ordinamento delle righe e degli attributi non è significativo e ciascun valore deve essere di tipo atomico, cioè elementare. Non sono pertanto ammessi nè vettori mono o pluridimensionali nè strutture composte.

Se non contiene gruppi del genere, una relazione si presenta nella cosiddetta "prima forma normale", che tuttavia può ancora contenere delle dipendenze funzionali e dre quindi luogo a delle anomalie in occasione di operazioni di inserimento, cancellazione, aggiornamento.

Figura 1 - La tabella CLIENTI

CODICE	NOME	CITTÀ	ZONA
C01	Rossi	Genova	NW
C04	Bianchi	Ancona	CN
C02	Verdi	Roma	CE
C12	Finzi	Firenze	CN
C34	Carli	Torino	NW
C09	Fazio	Milano	NO

Quindi, come tecnica di progettazione, conviene quando necessario suddividere una relazione in due o più "tronconi": questo processo di suddivisione e/o rimozione delle anomalie prende il nome di *normalizzazione dei dati*, come si vedrà meglio in seguito.

Va poi ancora sottolineato che questo modello logico rappresenta il modo in cui l'utente percepisce i dati, e che un database non è relazionale se non viene percepito dall'utente come un insieme di relazioni.

Ma perché questa struttura accresce l'indipendenza dai dati? Innanzitutto questo modello consente di descrivere i dati senza dover ricorrere ad alcuna rappresentazione fisica. Inoltre l'ordine con cui si susseguono gli attributi di una relazione non è significativo e quindi se ne possono facilmente aggiungere dei nuovi. Anzi, essendo basata sulla teoria degli insiemi, una relazione non può infatti contenere alcun concetto di ordinamento dei dati, neanche per quanto riguarda le righe.

Con i DBMS non relazionali invece il progettista di un'applicazione è speso tenuto a conoscere e rispettare una sequenza fisica dei dati. Se tale ordinamento contiene un significato, non può non scontrarsi con l'obiettivo dell'indipendenza dai dati.

Consideriamo ad esempio un file costituito da delle misure di temperatura, mantenute nell'ordine in cui sono state effettuate. In questo caso l'ordine stesso reca con sè delle informazioni sulla cronologia delle rilevazioni, che andrebbero perse se venisse cambiato, ma eliminando il vincolo della successione temporale (ed eventualmente introducendo come attributo o campo del record la data di rilevazione) si ottiene una struttura dei dati più semplice, più facile da capire e più flessibile.

2.2. L'integrità

Codd aveva definito due regole generali: l'*integrità delle entità* e l'*integrità referenziale*, applicabili a tutti i DBMS che dichiarino di richiamarsi al modello relazionale. Ciascuna implementazione potrà poi contenere altre regole oltre a queste due, senza per questo incrinare la definizione generale.

L'INTEGRITÀ REFERENZIALE

Per realizzare l'integrità referenziale, occorre innanzitutto specificare una chiave primaria in una tabella (che sarà la *tabella primaria*). Una **chiave primaria** è costituita da una colonna (o insieme di colonne) il cui valore è unico in ogni riga, ma che non è più tale se una delle colonne viene eliminata.

Poi, occorre specificare la dipendenza di chiave esterna da una seconda tabella a quella primaria. Una **chiave esterna** è un insieme di colonne corrispondenti in ampiezza e tipo di dati (cioè nello stesso dominio) della chiave primaria di un'altra tabella. Una dipendenza di chiave esterna è una regola (cioè una specie di asserzione) specificante che, a meno che la chiave esterna non sia nulla, essa deve sempre corrispondere ad un valore della chiave primaria.

Le dipendenze di chiavi esterne sono rafforzate dall'imposizione di regole sulle operazioni di manipolazione dei dati, quali:

REGOLE PER L'INSERIMENTO

No action: non permette l'aggiunta di un record a meno che la chiave esterna sia nulla o vi sia una corrispondente chiave primaria nella tabella primaria. Si noti che è possibile specificare che il valore nullo non è ammesso in una colonna, incluse quelle della chiave esterna.

REGOLE PER L'AGGIORNAMENTO

No action: non permette di dare un valore non nullo ad una chiave esterna a meno che vi sia una corrispondente chiave primaria nella tabella primaria. Non permette l'aggiornamento di una chiave primaria se esistono righe con una corrispondente chiave esterna.

Restrict: identico a NO ACTION, eccetto che il controllo è immediato.

REGOLE PER LA CANCELLAZIONE

No action: non permette la cancellazione di una riga nella tabella primaria se esiste una riga con chiave esterna corrispondente.

Restrict: identico a NO ACTION, eccetto che il controllo è immediato.

Cascade: quando una riga nella tabella primaria viene cancellata, vengono cancellate anche tutte le righe con corrispondente chiave esterna.

Set null: imposta al valore nullo le colonne delle chiavi esterne che possono assumere tale valore quando viene cancellata la riga con la corrispondente chiave primaria.

Set default: imposta al valore di default tutte le colonne della chiave esterna quando la riga della corrispondente chiave primaria viene cancellata.

Le regole dell'integrità sono strettamente abbinate al concetto delle chiavi di accesso e devono essere attentamente rispettate. Le *chiavi primarie* sono degli identificatori univoci di righe e sono costituite dal numero minimo di attributi necessari per differenziare ciascuna riga della tabella da tutte le altre (di chiavi primarie ce ne possono essere più di una, ed in tal caso la scelta spetta al progettista). Le chiavi primarie non possono contenere valori nulli e devono avere un significato. Per salvaguardare l'integrità delle entità, bisogna infatti che nessuno degli attributi presenti in una chiave primaria contenga un valore nullo. Inoltre, nessuno degli attributi che costituiscono una chiave primaria può essere eliminato senza rinunciare all'univocità dell'identificazione. Con l'inclusione di questi aspetti semantici, il modello relazionale costituisce quindi qualcosa di più di un semplice costrutto sintattico, cioè puramente formale.

Ma perché l'integrità delle entità accresce l'indipendenza dai dati? Perché è l'unico modo sistematico di individuare in modo univoco una determinata riga, partendo dal nome della tabella (relazione) e dal valore della chiave primaria. Ciò non significa che le chiavi primarie rappresentino l'unica via d'accesso alle tabelle, ma solo che sono le uniche a non fornire mai in risposta più di una riga e quindi a svolgere una funzione di identificazione. Se per assurdo una chiave primaria contenesse solo valori nulli, rappresenterebbe un'entità senza alcuna identità, ma in realtà una chiave primaria non può contenere un valore nullo. Ad esempio, se nella fig.1 possiamo essere sicuri che i valori di CODICE non vengono ripetuti, non sono mai nulli e sono sufficienti ad identificare in modo univoco una riga, allora CODICE può essere una chiave primaria.

L'altra regola è quella dell'integrità referenziale ed assicura non solo che le tabelle siano correlate tra loro, ma che lo siano in modo non ambiguo. Ogni valore di una chiave deve infatti esistere come valore di una chiave primaria in un'altra tabella o essere nullo. Una *chiave esterna* è un attributo (o una combinazione di attributi) di una tabella i cui valori devono corrispondere a quelli della chiave primaria di un'altra tabella o essere nulli, ossia non avere alcun valore. Le chiavi esterne devono essere definite nell'ambito della stessa gamma di valori o dominio.

2.3. Un esempio

A titolo di esempio, prendiamo in esame le due tabelle CLIENTI (fig.1) e ZONE (fig.2). Nella fig.1 ZONA è una chiave esterna e l'insieme dei valori di ZONA in CLIENTI (fig.1) deve esistere nella tabella ZONE (eccettuati i valori di ZONA in CLIENTI che possono essere nulli). Si noti che in questo caso tutti i valori di zona in CLIENTI esistono anche nell'insieme di ZONA nella tabella ZONE. Però esiste un valore ("**SI"**) che non è correlato ad alcuna riga della tabella CLIENTI.

Le chiavi esterne e le chiavi primarie rappresentano dei riferimenti o delle relazioni logiche che servono, per così dire, a "tenere insieme" le relazioni. Quale contributo

Figura 2 - La tabella ZONE

ZONA	DESCR	QUOTA
NW	Nord-Ovest	1.000
CE	Centro Italia	2.300
CN	Centro-Nord	2.300
NO	Nord Italia	4.400
SI	Sud e Isole	3.400

Figura 3 - Proiezione della tabella ZONE

ZONA
NW
CE
CN
NO
SI

Figura 4 - Fusione delle tabelle CLIENTI e ZONE

CODICE	NOME	CITTÀ	ZONA	DESCR	QUOTA
C01	ROSSI	GENOVA	NW	Nord-Ovest	1.000
C04	BIANCHI	ANCONA	CN	Centro-Nord	2.300
C02	VERDI	ROMA	CE	Centro Italia	2.300
C12	FINZI	FIRENZE	CN	Centro-Nord	2.300
C34	CARLI	TORINO	NW	Nord-Ovest	1.000
C09	FAZIO	MILANO	NO	Nord Italia	4.400

può dare l'integrità referenziale all'indipendenza dei dati? I DBMS non relazionali rappresentano i rapporti tra i gruppi di record con catene di puntatori logici, che devono essere costruite, seguite e comprese dal progettista di un'applicazione. Qualsiasi modifica dei loro rapporti comporta una modifica delle catene di puntatori e quindi si ripercuote su tutti i programmi applicativi che se ne servono.

Il modello relazionale invece non si basa su alcuna forma di concatenamento, ma collega le tabelle per mezzo dei valori di chiavi esterne. Nel nostro esempio, l'elemento che crea il collegamento è l'attributo (che funge da chiave esterna) ZONA. Qualsiasi valore nel database che non rispetti queste regole è per definizione errato. L'aggiunta nella tabella CLIENTI di una riga con valore ZONA="**E**" non sarebbe ad esempio consentito, perché l'insieme dei valori ammessi per ZONA è: **SW**, **NE**, **CN**, **SO** e **NW**.

Il modo in cui il modello relazionale viene implementato dipende poi dall'interpretazione data dal produttore del particolare DBMS. Come si possono evitare delle situazioni non corrette? Un DBMS relazionale può respingere qualsiasi operazione che violi le regole oppure accettarla, ma effettuando al tempo stesso altre operazioni di carattere compensativo, per evitare che l'integrità possa essere violata. Ad esempio, se la riga della tabella ZONE che contiene il valore "**SW**" nella colonna ZONA dovesse essere rimossa, quale azione (ossia, accettazione o rigetto) dovrebbe essere intrapresa nella tabella CLIENTI per salvaguardare l'integrità referenziale?

Un buon DBMS relazionale dovrebbe consentire di specificare le operazioni che possono essere accettate e quelle da respingere. La forma e la funzione di queste operazioni relazionali sono state delineate da Codd in "The Relational Model for Database Management: Version 2".

2.4. Le operazioni

I linguaggi per la manipolazione dei dati (DML, Data Manipulation Language) ed in particolare il sottoinsieme utilizzato per le interrogazioni (QL, Query Language) consentono di manipolare la struttura di un database. Il modello relazionale usa le operazioni dell'***albegra relazionale*** di selezione di righe, fusione di tabelle e proiezione di colonne (select, join e project) per reperire o estrarre i dati delle tabelle: tali operatori manipolano tabelle e restituiscono tabelle, cioè insiemi di righe e colonne.

L'operazione di **selezione** consente di estrarre un sottoinsieme di righe da una tabella, condizionate da un'espressione logica da soddisfare. L'operazione di **proiezione** consente di eliminare degli attributi non richiesti, e dà come risultato una nuova tabella costituita dalle colonne restanti, con eliminazione delle eventuali righe duplicate che si dovessero generare. Ad esempio, se si proietta ZONA dalla tabella ZONE, il risultato - come per tutte le operazioni relazionali - sarà un'altra relazione che, in questo caso, avrà un unico attributo (vedi fig.3). L'operazione di **fusione** consente invece di unificare due relazioni intorno ad un attributo comune, ottenendo sempre una nuova relazione. Ad esempio, per collegare CODICE con QUOTA (vedi fig.4) è necessaria un'operazione di fusione, che unifi-

Il trigger è uno dei due strumenti che possono permettere al DBMS di rafforzare l'integrità del database. L'altro strumento è noto come asserzione. Per comprenderli, si pensi ai valori ed alle associazioni di tutte le colonne in tutte le tabelle, nello stato del database in un determinato istante.

In generale un ***trigger*** dice: *"Ogni volta che lo stato del database è sottoposto (o sta per essere sottoposto) ad uno specifico tipo di cambiamento, esegui questa determinata azione"*.

Una ***asserzione*** dice: *"Il database deve essere SEMPRE in questo stato"*.

Un esempio di asserzione è la parola chiave UNIQUE dell'istruzione CREATE INDEX dell'SQL. Essa specifica che due o più righe della tabella su cui viene costruto l'indice non possono mai avere (nelle colonne specificate) lo stesso valore di chiave. Una violazione dell'asserzione procude un errore run-time. Una asserzione definita dall'utente è costituita da un predicato (cioè la specifica di una relazione tra colonne o righe) che deve sempre essere vero. Per esempio, la dipendenza di una chiave esterna è una asserzione secondo la quale certe colonne nella tabella B devono sempre avere uno dei valori della chiave primaria della tabella A (oppure, facoltativamente, essere nulli).

Per l'operazione di DELETE è possibile specificare la regola di "cascata". Essa è una specie di trigger tale che ogni volta che viene cancellata una riga nella tabella primaria, tutte le righe nella tabella dipendente con una corrispondente chiave esterna saranno cancellate automaticamente. Trigger definiti dall'utente permettono di estendere le possibilità delle regole interne di integrità referenziale.

chi le tabelle CLIENTI e ZONE sull'attributo ZONA, agganciandole l'una all'altra per mezzo di un attributo comune, ossia con gamme di valori compatibili. Oltre ai tre operatori base, ve ne sono anche altri (di derivazione insiemistica) come **unione**, **intersezione**, **prodotto**, **differenza** e **divisione** tra tabelle, che nel loro complesso costituiscono la cosiddetta algebra relazionale.

Un metodo alternativo è quello del ***calcolo relazionale***. Mentre l'algebra fornisce un insieme di operazioni esplicite che danno come risultato un'unica tabella, il calcolo relazionale offre semplicemente una notazione per definire la relazione desiderata mediante le tabelle a disposizione. Spetta poi al sistema decidere esattamente quali operazioni di algebra relazionale debbano essere eseguite per costruire la nuova tabella. Un esempio di calcolo relazionale equivalente alle operazioni descritte in precedenza (vedi fig.4) consiste nel prendere gli attributi CODICE, NOME, CITTÀ, ZONA, DESCR e QUOTA in modo che esistano righe con il medesimo valore ZONA nelle tabelle ZONE e CLIENTI:

```
select CODICE,NOME,CITTÀ,ZONA,DESCR,QUOTA
from ZONE,CLIENTI
where ZONE.ZONA=CLIENTI.ZONA
```

L'algebra ed il calcolo relazionale sono semplicemente modi differenti - ma perfettamente equivalenti - di esprimere le operazioni effettuabili sulle relazioni. Ma com'è che anche le operazioni sulle relazioni contribuiscono ad accrescere l'indipendenza dei dati? Grazie alla semplicità del modello relazionale, per estrarre o manipolare dei dati si possono usare solo le operazioni previste dalla teoria degli insiemi. Dato che non ci sono catene logiche di puntatori che uniscano le vie d'accesso ai dati e che l'accesso non deve essere necessariamente basato su una chiave primaria, non vi sono limitazioni alle vie d'accesso ai dati. Si possono quindi progettare delle basi di dati semplicemente in base alla loro semantica ("data-driven design"), anzichè alle esigenze di accesso come avviene di solito. Inoltre, dato che la struttura di una tabella non impone alcuna sequenza alle righe, l'accesso ai dati può essere effettuato in modo non procedurale, cioè in base ai dati da reperire e non al modo in cui gli stessi si possono ottenere. A questo punto, l'accesso ai dati è molto più facile e flessibile.

2.5. Implementazioni

Sono pochi i DBMS che hanno implementato integralmente il modello relazionale, anche se molti vi si sono avvicinati. Certo, vi sono considerevoli differenze tra il modello relazionale elaborato da Codd (specialmente nella Versione 2 del 1990) e quelli implementati dalle varie case produttrici, ma non è detto che un sistema debba supportare il modello relazionale nella sua interezza per potersi a sua volta definire relazionale.

In realtà, vi sono vari livelli di implementazione, in rapporto alla struttura, all'integrtà ed alle operazioni: alcuni sistemi supportano solo le tabelle, altri tabelle e ope-

VERSO LA PROGETTAZIONE GUIDATA DAGLI EVENTI

I trigger costituiscono un modo di realizzare uno dei principali concetti alla base del disegno object-oriented: l'***incapsulamento di comportamenti***. Questa espressione significa che la risposta di un oggetto ad uno stimolo esterno ad esso è memorizzata insieme alla definizione dell'oggetto stesso. In questo modo, è possibile che diversi oggetti rispondano in modo diverso alla stessa situazione.

Un concetto sorrelato al precedente è la possibilità che la stessa situazione abbia effetti diversi su differenti tipi di oggetti. Questo concetto è talvolta chiamato polimorfismo, un termine generale che può includere anche l'incapsulamento di comportamenti. Termini tecnici più precisi sono ***polimorfismo operazionale*** e ***overloading funzionale***. I trigger realizzano l'overloading funzionale permettendo che le operazioni di gestione dei dati abbiano effetti diversi sulle diverse tabelle. Siccome essi nascondono (incapsulano) le azioni internamente alle definizioni delle tabelle, il comportamento di ogni operazione di manipolazione dei dati può variare da una tabella ad un'altra.

Il nascondere le azioni si realizza collegando un insieme di istruzioni (il codice che realizza la procedura trigger) alla definizione della tabella. Siccome la codifica della procedura trigger è svincolata dalla codifica che la richiama, la procedura trigger è detta *guidata dagli eventi* (event-driven), modo di dire per esprimere che la sua esecuzione dipende dall'accadimento di qualche altro evento e non da istruzioni esplicite codificate in qualche altro programma.

ratori di algebra relazionale, altri ancora tabelle, operatori ed entrambe le regole di integrità. Ad esempio, molte implementazioni pratiche di sistemi relazionali offrono la possibilità di introdurre un ordinamento nelle tabelle, accogliendo una richiesta che viene spesso fatta in molte applicazioni reali, senza contraddire le regole di Codd, perché l'ordinamento non diventi un requisito indispensabile del sistema, ma rimanga opzionale.

Per finire, occorre tenere presente che la qualifica "relazionale" non è di per sè stessa una garanzia di qualità. Il modello relazionale costituisce senza dubbio una valida base su cui costruire dei sistemi più indipendenti dai dati di quanto non possano esserlo i database gerarchici e reticolari, ma non bisogna dimenticare che fra la definizione di un modello e la sua implementazione c'è sempre una certa differenza e che il livello di indipendenza dai dati è direttamente orrelato al livello di implementazione raggiunto.

2.6. La potenza del modello relazionale

Svincolando lo sviluppo di un'applicazione dall'organizzazione fisica dei dati, il modello relazionale permette di raggiungere un grado assai elevato di indipendenza dei dati e di flessibilità nella rappresentazione delle interrelazioni. I sistemi relazionali si distinguono quindi per la loro flessibilità, cioè per la loro capacità di adeguarsi alle richieste, anche imprevedibili, di un ambiente di lavoro dinamico.

Supponiamo ad esempio di dover aggiungere un nuovo attributo, SCONTO, alla relazione CLIENTI (vedi fig.5). Se SCONTO dovesse diventare una chiave esterna, basterebbe aggiungere la nuova relazione SCONTI, i cui attributi sarebbero SCONTO, SDESCR e PERCENTUALE (vedi fig.6).

L'introduzione di chiavi esterne preserva l'integrità dei dati e dà la possibilità di aggiungere altre informazioni, senza dover modificare le applicazioni. L'indipendenza dei dati rimane quindi uno degli aspetti più importanti del modello relazionale e continuerà ad esserlo in ogni sua futura estensione o revisione. Inoltre, la facilità con cui si possono vedere e manipolare i dati per mezzo di operazioni non procedurali ad alto livello giustifica l'attesa di un incremento di produttività.

La necessità ed i ritmi degli attuali ambienti di lavoro continuano a crescere, e l'esigenza di sviluppare nuove applicazioni o di modificare quelle già esistenti per adeguarle alle nuove necessità informative è chiaramente avvertita nel mondo informatico. Indubbiamente questo è un problema che può essere affrontato solo con sistemi flessibili e dinamici, e solo utilizzando la potenza del modello relazionale si può pensare di risolverlo.

Figura 5 - La nuova tabella CLIENTI

CODICE	NOME	CITTÀ	ZONA	SCONTO
C01	ROSSI	GENOVA	NW	3
C04	BIANCHI	ANCONA	CN	2
C02	VERDI	ROMA	CE	1
C12	FINZI	FIRENZE	CN	2
C34	CARLI	TORINO	NW	2
C09	FAZIO	MILANO	NO	3

Figura 6 - La tabella SCONTI

SCONTO	SDESCR	PERCENTUALE
2	medio	34%
3	alto	67%
1	basso	17%

3. I PASSI PROCEDURALI PER IL DISEGNO DI DATABASE RELAZIONALI

Il disegno del database è una parte fondamentale nello sviluppo delle applicazioni. La fase del disegno produce uno schema generale di base per la realizzazione, valido in tutte le situazioni: per ottenere un disegno efficace è utile seguire un metodo che prevede un lavoro suddiviso in passi sequenziali.

Vi sono diversi modi per affrontare il disegno del database; fra di essi ve ne sono alcuni basati sul modello relazionale. Tale modello considera tabelle per rappresentare le entità da trattare, con le colonne destinate a rappresentarne le proprietà. Le nozioni matematiche alla base del modello relazionale sono potenti e ben definite, e sia gli sviluppatori di applicazioni che gli utenti finali sembrano a loro agio nel trattare con tabelle astratte concetti che corrispondono a cose reali trattate dai processi aziendali.

Premessa

Il termine *"disegno del database"* viene qui usato per indicare due dei componenti fondamentali nello sviluppo delle applicazioni: il **disegno logico del database** (o modello dei dati) e il **disegno fisico del database** (o disegno realizzativo). Il primo produce una descrizione nella prospettiva delle attività del sistema da modellare, mentre il secondo tiene conto delle caratteristiche hardware e software del sistema. La realizzazione converte la descrizione fisica del database in appropriati comandi per creare file, programmi ed altre aprti di un'applicazione. In termini concettuali, il disegno logico del database determina il disegno fisico che a sua volta precede la realizzazione. Nella realtà, naturalmente, vi è spesso una notevole attività svolta in parallelo nelle tre aree.

Le prime quattro fasi di seguito descritte per il progetto di un database sono dedicate al disegno logico; l'ultima è dedicata al disegno fisico. Una sola fase dedicata al disegno fisico non deriva dal fatto che esso è una semplice e quasi meccanica conseguenza del disegno logico, bensì ciò è dovuto al fatto che in esso giocano una parte rilevante le caratteristiche hardware e software del sistema su cui si lavora, che non sono oggetto di questa esposizione. A mano a mano che si procede nei vari passi sarà più chiara la distinzione tra disegno logico e fisico. Il disegno del database va di pari passo con il **disegno del processo**, ed entrambi costituiscono fasi essenziali nella realizzazione di un progetto. Il disegno del processo tratta elementi ed aspetti quali le modalità con cui le attività da modellare avvengono nel sistema reale. Ovviamente, la descrizione di un processo richiede riferimenti ad un disegno di database per identificare l'origine dei dati su cui si basano i calcoli, dove sono stati memorizzati i dati di input e dove vanno memorizzati i risultati dei calcoli. Il disegno del database può anche riportare al processo definizioni per azioni che vanno attivate allorchè si verificano determinati cambiamenti nel database.

Non è obiettivo della presente trattazione descrivere i passi per il disegno dei processi; un metodo diffuso che potrebbe essere utilizzato è quello dei diagrammi di flusso dati

(DFD, Data Flow Diagram) oppure quello delle reti di Petri, che si accoppiano bene con i metodi di disegno dei database relazionali.

Ciò premesso, iniziamo la descrizione delle cinque fasi di progetto, che saranno di seguito indicate con le lettere A-E, e ciascuna si articolerà a sua volta in sottofasi progressivamente numerate.

Passo A1	***Definire le regole per i nomi ed un dizionario per il disegno.***	**Passo C2**	***Applicare le regole. Determinare i limiti di integrità aggiuntivi.***
Passo A2	***Parlare con gli utenti e registrare le loro opinioni.***	**Passo C3**	***Armarsi di cautela e determinare le norme di sicurezza.***
Passo B1	***Informazioni sui dettagli. Determinazione dei più importanti attributi delle entità.***	**Passo D1**	***Ottenere la panoramica generale: integrare viste d'utente multiple in un quadro generale.***
Passo B2	***Determinare la chiave primaria per ogni entità.***	**Passo D2**	***Eliminare il superfluo: togliere le ridondanze dalle tabelle base dello schema.***
Passo B3	***Esaminare come i vari elementi si combinano. Determinare le associazioni tra oggetti ed eventi.***	**Passo D3**	***Il momento della revisione: determinare le viste dei sottoschemi.***
Passo C1	***Conservare l'integrità. Determinare le regole sulle chiavi esterne per ogni associazione.***	**Passo E1**	***L'aspetto fisico: il disegno per la realizzazione fisica del database.***

Fig. 1 - Le cinque fasi di disegno ed i relativi passi

FASE (A) -- ANALISI PRELIMINARE

Passo A1: ***definire le regole per i nomi***

Prima di iniziare il progetto di disegno del database occorre fare bene due cose: stabilire le convenzioni da seguire nel creare nomi e le modalità con cui registrare le informazioni sul disegno.

Molte applicazioni comportano centinaia o migliaia di nomi, e definirli in modo chiaro e lineare è molto spesso quasi la metà del lavoro nel disegno di un database. Vanno pertanto definite norme per le abbreviazioni, per i nomi degli oggetti del database e per i processi, quali le tabelle, le colonne ed i programmi.

Una volta stabilite le norme per i nomi e su come tenerne traccia, si giunge al momento in cui occorre creare un **dizionario** correlato a questo argomento. Un dizionario di progettazione non è una sola cosa, bensì un insieme di oggetti disparati, di "contenitori" usati per conservare cose da ricordare durante la fase di progetto. Questi contenitori possono essere documenti di word-processor, file, directory od ogni altro tipo di oggetto utile.

È bene definire le proprie regole per nomi e tabelle prima di cominciare a creare oggetti che devono essere poi registrati perché, nel procedere della progettazione, si scoprirà quanto sia importante il poter seguire una norma razionale di creazione dei nomi. Ciò significa in sintesi avere un metodo ed un posto in cui conservare registrazioni ed informazioni in modo tale che esse possano essere rintracciate, e poter distribuire - quando occorre - informazioni *aggiornate* in merito al progetto.

Passo A2: ***intervistare gli utenti***

È naturalmente l'utente finale il miglior conoscitore, in una organizzazione, di ciò che si deve realizzare. Pertanto, si abbia cura di registrare accuratamente una descrizione chiara dei vari oggetti, eventi, associazioni e processi di interesse per i vari utenti.

In termini relazionali, un oggetto (ad esempio un cliente) ed un evento (ad esempio la definizione di un ordine) sono due tipi di ***entità*** (cose in merito alle quali si devono registrare fatti precisi) e sono rappresentati tramite ***tabelle*** (oggetti ed eventi complessi possono essere rappresentati tramite gruppi di tabelle). Una ***vista*** è invece la prospettiva che un utente ha su oggetti ed eventi, ed è definita in termini di operazioni dell'algebra relazionale quali ad es. il join tra tabelle.

La differenza tra tabelle e viste è che le prime rappresentano fatti essenziali e non ridondanti in merito alle entità, mentre le viste rappresentano un modo di vedere tali fatti.

Poichè in questa fase si opera sulle prospettive dei vari utenti, certamente non si è in grado di stabilire sempre se quanto pensa l'utente su un certo oggetto od evento dovrà in realtà tradursi in una tabella o in una vista sulle tabelle. Pertanto, è opportuno registrare semplicemente le prospettive degli utenti usando tabelle per gli oggetti e gli eventi, operando tutte le strutturazioni che siano ovvie o richieste dai passi successivi. Ad esempio, se l'utente descrive un ordine, si può inizialmente rappresentare questo tramite una tabella base degli ordini. Quando poi si identificano gli attributi di un ordine (passo B1), sarà possibile descrivere una nuova tabella (RIGHE_ORDINI) per rappresentare le proprietà a valori

multipli trovate sulle righe dell'ordine. A questo stadio, non si presti cura eccessiva alla purezza in termini relazionali di quanto si va registrando (Codd non si offenda...): in fasi successive ci si occuperà di normalizzazione e di altre rifiniture.

Procedendo con l'identificazione delle entità, si identificano anche le **gerarchie di entità**. Per esempio, se alcuni componenti sono prodotti in loco ed altri sono invece acquistati, i due insiemi costituiranno dei *sottotipi* (dipendenza "isa" nel modello E/R) delle entità tipo 'componente'. Come altro esempio, 'capo' e 'sottoposto' sono sottotipi del tipo di entità 'impiegato'. Un sottotipo ha tutte le proprietà e le regole di integrità di un supertipo. Per esempio, i sottotipi 'capo' e 'sottoposto' hanno tutte le proprietà (quali matricola, nome, mansione) del supertipo 'impiegato'. Ciò rende promettente il terreno di ricerca dei cosiddetti OODBMS (Object-Oriented Database Management System), che potranno sicuramente incorporare in maniera naturale tali concetti nell'ambito più generale delle classi di oggetti, nonchè i processi come "metodi" incapsulati negli oggetti stessi e "messaggi" tra gli oggetti del database.

FASE (B) -- DISEGNO DELLE ENTITÀ E DELLE ASSOCIAZIONI

Passo B1: ***determinare gli attributi delle entità***

In termini relazionali, le proprietà di un oggetto o di un evento sono conosciute come **attributi** e sono rappresentate come colonne di una tabella. È importante determinare quali attributi hanno un valore unico (ad esempio, la data di nascita) e quali possono avere occorrenze multiple (ad esempio, i figli di una persona). Gli attributi con valori multipli - in un database relazionale normalizzato - vengono rappresentati in una tabella separata. È anche necessario individuare gli attributi dipendenti dal tempo. Per esempio, qualora si debba conoscere il costo di una componente in date diverse, allora tale costo è una proprietà variabile con il tempo. Per ogni attributo va anche individuato l'eventuale valore assunto da usarsi quando non viene definito un valore esplicito in occasione di nuove aggiunte.

Passo B2: ***determinare la chiave primaria per ogni entità***

In un corretto disegno di database, bisogna poter identificare ogni occorrenza di entità tramite i valori di uno o più dei suoi attributi. Nel modello dei dati, quando si rappresentano entità mediante tabelle occorre specificare la **chiave primaria** della tabella: essa è costituita da una colonna o gruppo di colonne che, con i loro valori, identificano in modo univoco ogni riga della tabella stessa. Vi possono essere parecchi modi per identificare in modo univoco le righe di una tabella, ed ogni colonna o gruppo di colonne che possono servire come chiave primaria sono dette **chiavi candidate**. Occorre scegliere tra di esse quella che sarà usata quale chiave primaria. Ogni chiave candidata (e quindi anche la primaria) deve soddisfare ai seguenti requisiti:

- i valori di una chiave candidata devono essere unici per ogni riga della tabella;
- in ogni riga i valori della chiave candidata non possono mancare o essere incompleti;

- ogni chiave primaria o candidata non deve usare altre colonne oltre quelle della stessa per identificare in modo univoco una riga.

Una chiave primaria deve inoltre soddisfare anche queste condizioni:

- deve avere il solo significato di identificatore;
- il suo valore non deve essere soggetto a cambiamenti;
- il numero dei valori possibili di una chiave primaria dovrebbe essere illimitato;
- per ogni tabella si dovrebbe specificare una sola chiave primaria.

Spesso, la scelta migliore per la chiave primaria di una tabella è data da valori interi progressivi assegnati automaticamente (contatori). In tali casi si introduce una **chiave fittizia** quando non vi sono attributi naturali soddisfacenti i criteri della chiave primaria.

Vi sono due casi in cui una chiave primaria composta di più attributi può essere usata in modo efficace al posto di una chiave primaria costituita da una sola colonna. Primo: la chiave primaria di tabelle che rappresentino associazioni del tipo molti-a-molti può essere la combinazione di **chiavi esterne** che individuano le tabelle associate. Una **associazione** tra entità è una corrispondenza tra le tabelle rappresentanti le entità. Una chiave esterna è data da uno o più attributi i cui valori per una determinata riga corrispondono al valore della chiave primaria di qualche altra riga (nella stessa od altra tabella). Per esempio, una tabella ORDINI potrebbe avere una colonna ID_CLIENTE quale chiave esterna. Ogni colonna ID_CLIENTE della tabella ORDINI dovrebbe (di norma) contenere il valore di una chiave primaria della tabella CLIENTI. Pertanto essa identifica il cliente che ha emesso l'ordine. Finchè le chiavi esterne di un'associazione non sono definite come valori nulli, ogni loro combinazione avrà le stesse proprietà di chiavi fittizie a colonna singola. Per esempio, in una tabella COMPONENTI_FORNITE, la chiave primaria potrebbe essere una combinazione delle chiavi esterne CODICE_COMPONENTE e CODICE_FORNITORE.

Il secondo caso è per tabelle il cui unico scopo consiste nella definizione di proprietà a valori multipli. Per tali tabelle, un numero di sequenza che sia unico all'interno della chiave primaria "madre" può essere combinato con la chiave esterna indicante la tabella madre. Per esempio, una tabella RIGHE_ORDINI può avere la combinazione delle colonne NUMERO_ORDINE e NUMERO_RIGA_ORDINE quale chiave primaria, dove NUMERO_ORDINE è una chiave esterna indicante il relativo identificativo dell'ordine nella tabella madre ORDINI, e NUMERO_RIGA_ORDINE è il numero di sequenza delle varie righe all'interno di un ordine.

Passo B3: ***determinare le associazioni tra le entità***

Gli oggetti e gli eventi rilevati non sono mai isolati: essi sono interrelati o associati. Quando si identifica un'associazione occorre determinare la sua molteplicità (o cardinalità). Vi sono le seguenti relazioni di molteplicità:

- **uno-a-uno** (o corrispondenza biunivoca). Per esempio, ogni auto aziendale può avere associato un determinato posto macchina e viceversa ogni posto macchina del parcheggio è assegnato ad una sola auto aziendale;
- **uno-a-molti** (o corrispondenza univoca). Per esempio, un cliente può avere molti ordini in essere, ma ciascun ordine è associato ad un solo cliente;
- **molti-a-molti** (o corrispondenza multipla). Per esempio, molti fornitori possono fornire diverse componenti ed ogni componente può essere fornita da parecchi fornitori.

Nel modello relazionale è possibile rappresentare un'associazione semplice quale quella uno-a-uno o quella uno-a-molti tramite una chiave esterna. Un'associazione molti-a-molti dovrebbe invece essere rappresentata mediante una nuova tabella riflettente (due o più) associazioni univoche. Per esempio, è possibile rappresentare un'associazione molti-a-molti tra le tabelle COMPONENTI e FORNITORI tramite una tabella COMPONENTI_FORNITE. Le tabelle COMPONENTI e FORNITORI avrebbero allora una corrispondenza uno-a-molti con la tabella COMPONENTI_FORNITE.

È importante anche capire più specificamente in cosa consiste la *cardinalità* di un'associazione. In genere, dopo aver suddiviso le associazioni molti-a-molti in associazioni uno-a-molti, si dovrebbe determinare:

- per il lato "uno" dell'associazione, se esso indica esattamente "uno" o non piuttosto "uno o nessuno";
- per il lato "molti", se esso significa "nessuno o molti" piuttosto che "almeno uno o diversi".

Se vi sono norme più specifiche sulle associazioni (per esempio, per ogni squadra vi devono essere esattamente undici giocatori), anche queste devono essere rilevate e registrate. Le ragioni più importanti per determinare con esattezza le corrispondenze multiple riguardano il conoscere se un lato dell'associazione è obbligatorio o facoltativo ed anche il numero massimo od esatto di ogni entità correlata (non sarebbe accettabile realizzare un database del campionato di calcio in cui si permette ad un allenatore di mandare in campo più di undici giocatori per partita).

Nel processo di identificazione delle associazioni bisogna anche determinare quali di esse sono *n-arie* (cioè coinvolgono più di due entità). Per esempio, un'associazione per cui alcuni costruttori forniscono alcune componenti per specifici progetti è del tipo a tre vie. Anche le associazioni n-arie dovrebbero essere rappresentate tramite nuove tabelle.

FASE (C) -- DEFINIRE L'INTEGRITÀ E LA SICUREZZA

Passo C1: ***determinare le regole sulle chiavi esterne per ogni associazione***

Per ogni associazione registrata una delle tabelle relative deve contenere una chiave esterna che punta a specifiche righe della tabella referenziata. Per esempio, una tabella ORDINI dovrebbe avere una colonna ID_CLIENTE quale chiave esterna per indicare la

riga della tabella CLIENTI associata a quel particolare ordine. Per ogni associazione si divrebbe identificare quali colonne nella tabella referenziata rappresentano la chiave esterna.

Si deve inoltre per ogni chiave primaria od esterna specificare le regole relative a *cancellazioni/aggiornamenti/inserimenti*. Questo compito è suddiviso in due parti. Dapprima, occorre determinare quali azioni intraprendere quando, cancellando o modificando specifici valori di chiavi primarie in una tabella, si lasciano "orfane" di riferimento determinate righe delle tabelle associate che non hanno più riferimenti specifici a colonne della tabella modificata. Vi sono parecchie azioni possibili in tali casi:

- rifiutare la cancellazione o modifica (e segnalare l'errore);
- trattare il caso con una procedura personalizzata (*trigger* o *stored-procedure*);
- propagare l'azione di cancellazione o modifica a tutte le tabelle "figlie" associate;
- impostare le chiavi esterne associate a valori nulli o di default (quando possibile).

Occorre poi determinare quali azioni devono essere intraprese se un aggiornamento o un inserimento nella tabella referenziata origina un valore di chiave esterna non nullo e senza corrispondenza nelle tabelle correlate. Si può:

- rigettare l'inserimento o la modifica (e segnalare l'errore);
- trattare il caso tramite una procedura personalizzata (*trigger* o *stored-procedure*);
- creare una riga "genitore" di default nella tabella in questione e dare ad essa la stessa chiave primaria corrispondente al nuovo valore di chiave esterna;
- ripristinare la chiave esterna nuova o modificata ad un valore nullo o di default (se possibile).

Passo C2: ***determinare i limiti di integrità aggiuntivi***

Nei passi B2 e C1 si sono definite due importanti regole: l'integrità delle chiavi primarie ed esterne. Definendo le regole che la successiva fase di realizzazione dovrà seguire al fine di assicurare che il database abbia sempre valori di chiavi primarie ed esterne valide si evitano identificazioni ambigue od incomplete delle entità memorizzate nel database e delle interrelazioni tra queste entità.

Si dovrebbero anche identificare diverse altre categorie di limiti di integrità, ivi inclusi *domini*, *predicati* e *limiti di transizione*. Le regole devono essere accurate e complete, se si vuole evitare che il database diventi un insieme di dati senza significato e senza controllo. Gli utenti spesso descrivono le regole in modo incoerente od incompleto. Bisogna porvi attenzione: se dal database, in conseguenza dell'applicazione di regole approssimate, si ottenessero dati inconsistenti la colpa ricadrebbe sul progettista. Ecco alcune regole per una corretta progettazione dell'integrità del database.

Per primo, determinare il **dominio** per ogni colonna. Un dominio definisce due aspetti di una colonna: i valori relativi permessi e le operazioni ammissibili. Con il definire esplicitamente i dominii si ottengono linee guida per assicurarsi che la realizzazione non permetterà che vengano inseriti valori non validi in una colonna e che non siano usati impropriamente i valori della colonna in un'operazione. Per esempio, una chiara definizione

dei dominii degli attributi PAGA_ORARIA e SALARIO_MENSILE assicura che entrambe le colonne non possano contenere valori negativi e che la PAGA_ORARIA non possa essere sommata direttamente (senza opportuna conversione) al SALARIO_MENSILE.

Di norma, non si esprimono *esplicitamente* le varie operazioni permesse in un dominio poichè tale definizione richiederebbe di considerare un grandissimo numero di operazioni tra dominii. Invece, bisogna definire i dominii in modo che i loro valori permessi ed il *significato* siano chiari e che le operazioni permesse siano conseguenza della definizione parziale. Si può iniziare con il registrare una descrizione sintetica e chiara di ogni dominio, ivi inclusi i valori permessi ed i limiti nelle operazioni. Parecchie caratteristiche di un dominio aiutano nel definire ulteriormente il suo significato. Si dovrebbe indicare, in termini comprensibili all'utente, il *tipo base* del dominio (ad esempio: valore intero, stringa di caratteri, data). Si può poi limitare maggiormente l'insieme dei valori possibili di un dominio usando vari modi di annotazione di insiemi. Ad esempio, nel caso di numeri interi, si può usare l'intervallo (o range: per es. GIACENZA>0); per le enumerazioni è possibile usare una lista di valori ammessi, completa o indicante solo gli estremi (ad esempio: GIORNO_LAVORATIVO=[Lunedì,Martedì,Mercoledì,Giovedì,Venerdì]).

Nel definire i dominii può capitare per un certo insieme la necessità di dare valori di un certo tipo insieme ad uno o più valori di un altro tipo. Per esempio, nello specificare le date limite per eventi dipendenti dal tempo sis potrebbe voler definire un dominio che includa certe date valide più valori specifici di tipo "aperto" per indicare intervalli temporali.

Due distinti valori possono essere assunti per qualsiasi dominio: nullo e non-applicabile. **Nullo** significa "*sconosciuto, ma può esistere"*. Per esempio, il valore nullo potrebbe essere usato per rappresentare il numero telefonico, ignoto, di un cliente. Nullo non significa, pertanto, zero o spazi o non-applicabile. **Non-applicabile** (N/A) significa che per quella colonna non esiste alcun valore che abbia significato. Per esempio, una tabella COMPONENTI può avere una colonna COLORE ed una componente senza colore potrebbe avere in tale colonna il valore N/A. I dominii con un valore N/A sono connessi direttamente con dipendenze gerarchiche di entità (cfr. la dipendenza *isa* del modello E/R). Si potrebbe evitare il ricorso al valore N/A suddividendo la tabella in tabelle separate che rappresentino i sottotipi ed omettendo in queste tabelle derivate le colonne in cui tutte le righe avrebbero il valore N/A. Ricorrere a tale valore o ricorrere a tabelle derivate per sottotipi è una scelta decisionale a livello di progetto: si scelga il metodo che facilita maggiormente il lavoro. In generale, si dovrebbero usare i sottotipi quando vi è tra di essi una chiara distinzione nella prospettiva dei processi del sistema da modellare o dell'utente finale. Pertanto, si dovrebbe per esempio ricorrere ai sottotipi per distinguere tra componenti prodotte localmente rispetto a quelle acquistate, mentre si potrebbe usare un dominio che permetta il valore non-applicabile per distinguere parti senza colore rispetto ad altre con colore.

Le unità di misura ed i tassi (ad esempio, chilogrammi o lire/mese) sono altri strumenti per chiarire il significato di un dominio. Si tenga inoltre presente che, come è possibile definire gerarchie di tipi di entità, parimenti è possibile definire i dominii come

gerarchie. Per esempio, DATA è un dominio molto generico. PRIMO_DEL_MESE è un sottodominio del dominio DATA limitato a quelle sole date corrispondenti al primo di un mese.

Dopo aver definito i dominii per limitare i valori possibili per ogni colonna, possono essere rappresentate altre regole mediante le specifiche di **predicati** di riga. Un predicato di riga è una limitazione che può essere valutata tramite i valori di una singola riga di una tabella. Per esempio, se una tabella IMPIEGATI ha le colonne DATA_ASSUNZIONE e DATA_CESSAZIONE, si potrebbe definire un predicato di riga come:

(DATA_CESSAZIONE=null) OR (DATA_ASSUNZIONE≤DATA_CESSAZIONE)

Altri limiti di integrità si possono definire tramite la nozione generale di asserzione e procedura condizionata. Le **asserzioni** sono espressioni logiche che devono essere vere per un valido database. Per esempio, il modello dei dati potrebbe stabilire che in ogni anno fiscale gli incentivi di un qualsiasi impiegato non possono superare il suo stipendio complessivo. Le **procedure condizionate** (triggered) sono azioni che si verificano quando si avverano specifiche condizioni. Ad esempio, il modello dei dati potrebbe decidere che ogniqualvolta un cliente effettua un ordine di valore superiore a 10 milioni si deve inviare una certa nota al responsabile vendite. È facile anche incontrare norme organizzative che impongono al modello dei dati **limiti di transizione**. Essi limitano le modifiche ai dati. Per esempio, le norme aziendali possono limitare le modifiche al credito concesso per oscillazioni non superiori al 50%.

Passo C3: ***determinare le norme di sicurezza***

Le **regole di sicurezza** costituiscono un caso speciale di limiti di integrità dipendenti dall;utente. È possibile usare per queste regole le stesse notazioni usate per le altre norme organizzative. Nello specificare le norme di sicurezza vanno identificate le seguenti classi di sicurezza:

- regole *indipendenti dal valore*: esse sono restrizioni su tabelle, viste o colonne basate solamente sul tipo di dati e non su particolari valori. Per esempio, taluni utenti possono non avere il diritto di accesso ai dati dei clienti;
- regole *dipendenti dal valore*: sono basate su valori specifici. Per esempio, un certo utente può essere autorizzato a vedere la colonna STIPENDIO solo quando il valore non supera un certo importo;
- regole *statistiche*: esse limitano le inferenze che possono essere ottenute dai valori del database mediante funzioni quali SUM e COUNT. Per esempio, gli utenti non autorizzati a vedere gli stipendi dei dipendenti possono avere anche il divieto di ottenere totalizzazioni in merito, di qualunque tipo, in quanto da esse potrebbero derivare informazioni specifiche sui singoli individui;

- regole *dipendenti dal contesto*: sono regole definite in termini di funzioni o valori da cui dipendono. Per esempio, gli utenti potrebbero essere limitati nell'aggiornamento delle colonne della tabella IMPIEGATI per i soli dipendenti che a loro riferiscono.

FASE (D) -- COMPLETARE IL DISEGNO LOGICO

Passo D1: ***integrare le viste d'utente in un quadro generale***

Dopo aver determinato oggetti, eventi, associazioni, attributi e limiti di integrità per ogni vista d'utente queste dovrebbero essere combinate in modello logico di dati unico, non contraddittorio, non ridondante. Le viste d'utente sono talvolta denominate **sottoschemi** (o *schemi concettuali*). L'integrazione delle viste multiple comporta:

- l'identificazione delle colonne che sono identiche nelle viste multiple;
- la risoluzione degli eventuali conflitti a livello di dominio e di altri limiti tra colonne e tabelle multiple;
- la sintesi delle tabelle base dello schema dalle tabelle e viste multiple dei sottoschemi.

In pratica, il processo di integrazione avviene quando si registrano le viste d'utente. Nella produzione dello schema integrato, si scoprono contraddizioni o ambiguità nelle liste d'utente che saranno risolte lavorando assieme agli utenti. Il processo di integrazione richiede una buona convenzione sui nomi ed un efficace dizionario del disegno. Gli utenti spesso usano diversi sinonimi per la stessa entità o attributo, mentre è necessario avere un unico nome "ufficiale" da usarsi in tutte le tabelle e definizioni relative all'integrità.Una tabella dei sinonimi nel dizionario del progetto permette di correlare la terminologia degli utenti con i nomi "ufficiali". Il dizionario del progetto dovrebbe anche riportare la fonte dei vari limiti di integrità.

Passo D2: ***eliminare le ridondanze***

Questo passo consiste in un controllo del risultato dell'integrazione delle viste d'utente multiple. Ci si deve assicurare che le tabelle base dello schema non contengano fatti ridondanti. Per esempio, una tabella COMPONENTI_FORNITE non dovrebbe di norma contenere la colonna INDIRIZZO_FORNITORE perché questo facilmente apparirebbe in parecchie righe della tabella.

Fig. 2 - Le cinque forme normali

Ogni forma normale è un prerequisito per quella successiva, tranne la 5NF che richiede solo la 1NF.

Prima forma normale (1NF)

Ogni dominio di colonna è atomico (cioè non ripetitivo e non ulteriormente suddivisibile senza perdita di significato).

Seconda forma nsormale (2NF)

La tabella è in 1NF ed ogni colonna che non sia parte della chiave primaria è funzionalmente dipendente dall'intera chiave primaria.

Terza forma normale (3NF)
La tabella è in 2NF e nessuna colonna è funzionalmente dipendente da ogni colonna che non sia parte di una chiave candidata. Non sono permesse dipendenze transitive (cioè una colonna A che dipende da una colonna B che a sua volta dipende dalla chiave primaria).
Quarta forma normale (4NF)
La tabella è in 3NF e contiene non più di una dipendenza a valori multipli (il figlio di un dipendente, in una relazione familiare, è un esempio di una dipendenza a valori multipli). Questa norma proibisce la combinazione di due o più di tali dipendenze nella stessa tabella.
Quinta forma normale (5NF)
La tabella non può essere decomposta in due o più delle sue proiezioni (una proiezione è un sottoinsieme di colonne di una tabella), ognuna con una diversa chiave primaria, senza perdere qualche informazione. In altre parole, la tabella non può essere ricostruita dalle sue proiezioni.

Un metodo per eliminare le ridondanze consiste nell'analizzare ogni tabella per vedere se essa viola qualcuna delle regole della normalizzazione (vedi fig.2). Il modello logico dei dati dovrebbe essere completamente normalizzato (e cioè soddisfare le cinque regole). Questa è la forma migliore per comunicare con accuratezza il modello dell'organizzazione usando quale strumento di notazione le tabelle. Quando si progetterà la realizzazione si potrà scegliere di progettare file che non corrispondono esattamente alle tabelle di dati del modello logico. Tale *ridondanza controllata* è perfettamente accettabile, ma quale parte del disegno di realizzazione, non della fase del disegno logico dei dati. Questo è un punto del disegno del database spesso frainteso: *le forme normali non hanno nulla a che vedere con il disegno fisico dei dati*. Le forme normali sono una convenzione di disegno del database relazionale che riflettono dipendenze funzionali.

Un secondo modo per eliminare le ridondanze consiste nel controllare tutte le colonne per vedere se esse possono essere *derivate* da altre colonne. Se una colonna può essere derivata non dovrebbe essere inclusa nella tabella, ma piuttosto in una lista (si veda il

passo D3). Per esempio, un TOTALE_ORDINE è di norma ottenibile dall'insieme delle righe ordine associate. Pertanto, TOTALE_ORDINE è una colonna che appartiene ad una vista, non ad una tabella base.

Passo D3: ***determinare le viste dei sottoschemi***

Ottenuto lo schema generale si dovrebbe ritornare alle originali viste d'utente (sottoschemi) e controllare quanto può essere desunto dalle tabelle base dello schema. Questo passo produce un insieme revisionato delle viste d'utente definite tramite le tabelle base dello schema. Attraverso le viste potrebbe essere necessario esprimere i seguenti tipi di conclusioni:

- restrizioni (sottoinsiemi) delle righe;
- eliminazione di colonne;
- combinazioni di righe (ad esempio unioni) da più di una tabella;
- derivazione di *colonne virtuali* che sono funzioni di colonne delle tabelle base (per esempio: TOTALE_RIGA_ORDINE=PREZZO_UNITARIO*QUANTITÀ);
- ordinamento (selezione) delle righe in una vista.

Compiuto un ciclo attraverso queste prime quattro fasi, si potrebbe rendere necessario ripeterle parecchie volte per ottenere il disegno logico appropriato. Inoltre, al cambiare del modello gestionale occorrerà procedere alla revisione del modello logico dei dati. Comunque, l'uso di queste fasi fornisce una metodologia per analizzare e registrare sia il modello originale dei dati che le revisioni successive.

FASE (E) -- REALIZZARE IL DISEGNO FISICO

Passo E1: ***il disegno per la realizzazione fisica del database***

Sebbene le attività per implementare il disegno fisico del database possano procedere in parallelo con quelle per implementare il disegno logico, concettualmente il disegno fisico segue quello logico: bisogna determinare *cosa* si vuole fare prima di decidere *come* farlo. Il progetto di realizzazione fisica è basato su quattro elementi fondamentali:

- il *modello logico dei dati* e il *modello di processo* che definiscono cosa la realizzazione deve fare;
- i *volumi* e il *profilo degli accessi* previsti; per esempio, il numero dei clienti e degli ordini e la frequenza delle ricerche e degli aggiornamenti su un ordine in essere;
- le *caratteristiche prestazionali* del sistema hardware e del software; per esempio, il tempo relativo alla ricerca di record su chiave rispetto alla ricerca di tipo sequenziale;
- le *caratteristiche funzionali* del software di sistema; per esempio, se il linguaggio di interrogazione del DBMS supporta operazioni aritmetiche sui dati.

Il disegno della realizzazione *non può* essere appropriato senza un modello logico dei dati completo, una stima delle modalità d'accesso ed una comprensione delle caratteri-

stiche hardware e software del sistema. Per esempio, per decidere se includere in un file un campo ridondante è necessario rispondere alle seguenti domande:

- quale sarà la frequenza con cui il campo derivato sarà ricercato?
- sarà tale campo sempre ricercato tramite strumenti d'utente finale piuttosto che tramite programmi applicativi scritti in linguaggi ad alto livello?

Due regole approssimate possono aiutare nel prendere decisioni in merito al disegno di realizzazione:

- la ridondanza controllata di norma rende la *ricerca* più veloce e semplice;
- la ridondanza controllata di norma rende l'*aggiornamento* più lento e più complesso.

Decisioni specifiche in merito al progetto di realizzazione richiedono una profonda conoscenza dei compromessi sulle prestazioni e sulla complessita. Il miglior approccio consiste nel separare il progetto di database in due parti concettuali: la modellazione logica dei dati ed il progetto di realizzazione fisica. Poi, al momento della realizzazione fisica, è possibile concentrarsi sui dettagli della piattaforma e disegnare una realizzazione efficiente senza dover ripensare a cosa il sistema deve fornire.

PARTE TERZA
Esempio di Progettazione di un Database Relazionale

1. INTRODUZIONE

Un buon modo per iniziare la progettazione di un database consiste nel'abbozzare su carta un modello logico dei dati da gestire; raffinare successivamente il modello e, infine realizzare il modello mediante un DBMS (database management system) di adeguata potenza e flessibilità.

In questa sezione viene descritto un esempio di come si progetta in particolare un database relazionale.

Fig.1 - Il modello dei dati dell'azienda ALFACOM

Tabella **VENDITORI**

COGNOME_VENDITORE	NOME_VENDITORE	DATA_ASSUNZIONE
CARLI	Pietro	26/11/1983
HERNANDEZ	Ernesto	28/09/1982
SEMPRINI	Maria	01/06/1984
SESSA	Giovanni	15/09/1983

Tabella **VENDITE**

COGNOME VENDITORE	RAG_SOC CLIENTE	DESCRIZ PRODOTTO	QUANTITÀ VENDUTA	DATA VENDITA
CARLI	Computer Distributors	PC portatile standard	10	12/03/1989
HERNANDEZ	Industrial Computers	PC avanzato a colori	8	12/03/1989
SEMPRINI	Industrial Comcepts	PC classico a colori	10	13/03/1989
SESSA	Computer Mountain	PC portatile standard	25	13/03/1989

2. DISEGNO DEL MODELLO DEI DATI.

Quando si gestiscono delle informazioni, si utilizza un modello concettuale che aiuta a tenere traccia delle informazioni di cui si ha bisogno. Questo modello può essere dinamico, in modo da adattarsi al mutare delle esigenze. Consideriamo un esempio. L'azienda ALFACOM produce e distribuisce computer. Il direttore delle vendite, Giovanni Guglielmi, decide di metter su un archivio di informazioni per rispondere alle esigenze informative del settore che dirige.

Delle persone, eventi e cose con cui ha a che fare, queste categorie sono evidenti:

- I computer prodotti e venduti dalla ALFACOM;
- I clienti che acquistano i prodotti;
- I venditori che li vendono.

Molte domande che sorgono nel lavoro quotidiano ricevono risposta dall'interrogazione di una o più delle categorie suesposte. Per tenersi informato, il direttore delle vendite costruisce un modello dei dati.

La fig.1 mostra un semplice modello dei dati con una tabella per ciascuna categoria. Insiemi Individuali di dati correlati sono rappresentati da righe nelle tabelle, e le informazioni specifiche che si intendono gestire sono rappresentate dalle colonne nelle tabelle.

La fig.1 mostra che ci sono quattro venditori. I fatti che sono registrati nella tabella **VENDITORI** sono COGNOME_VENDITORE, NOME_VENDITORE e DATA_ASSUNZIONE**.** Informazioni riguardanti le vendite sono registrate nella tabella **VENDITE**, che contiene fatti riguardanti: COGNOME_VENDITORE, COGNOME_ACQUIRENTE, RAGIONE_SOCIALE_ACQUIRENTE, RAGIONE_SOCIALE_CLIENTE, DESCRIZIONE_COMPUTER, QUANTITÀ_VENDUTA, DATA_VENDITA. Nonostante la semplicità del modello, il direttore vendite può rispondere a domande come le seguenti:

- Chi sono i venditori della ALFACOM?
- Quali vendite sono state effettuate nel mese di marzo del 1989?
- Quali sono i venditori che lavorano da più tempo nell'azienda?
- Quali venditori hanno venduto più PC Portatili?
- Quanti PC Avanzati sono stati venduti?
- Qual è stato l'ultimo acquisto fatto dall'Industrial Computers?
- La Industrial Concepts ha mai acquistato il Pc Portatile Standard?

Ad alcune domande si può rispondere osservando una sola tabella, poichè la tabella si riferisce ad un unico tipo di informazione. Altre richiedono l'esame di entrambe le tabelle, poichè le informazioni in termini dei valori che le tabelle hanno in comune. Nell'esempio della fig.1, il dato comune che lega la tabella **VENDITORI** alla tabella **VENDITE** è il cognome del venditore.

I buoni modelli non vengono fuori da soli. Più attentamente si progetta, più vicino alla realtà sarà il risultato ottenuto. Occorre fare attenzione nella progettazione di un modello accurato: solo così sarà possibile rispondere successivamente al maggior numero di domande.

3. CARATTERISTICHE DELLE TABELLE USATE PER I MODELLI DEI DATI

La tabella raffigurata in fig.2 presenta varie caratteristiche:

Fig.2 - La tabella VENDITORI

Tabella **VENDITORI**

COGNOME_VENDITORE	NOME_VENDITORE	DATA_ASSUNZIONE
CARLI	Pietro	26/11/1983
HERNANDEZ	Ernesto	28/09/1982
SEMPRINI	Maria	01/06/1984
SESSA	Giovanni	15/09/1983

- Ogni riga ha lo stesso numero di "fatti": **COGNOME, NOME e DATA_ASSUNZIONE**
- Ciascuna colonna contiene lo stesso genere di fatti in ciascuna riga. per esempio, la **DATA_ASSUNZIONE**. Se uno dei vanditori lascia l'azienda e viene successivamente riassunto, bisognerà prendere una decisione in merito. Il direttore delle vendite registra la data di assunzione originaria o la più recente? O aggiunge un nuovo fatto alla tabella per esprimere l'eventuale DATA_RIASSUNZIONE? La risposta dipende dal tipo di informazioni che il direttore delle vendite desidera estrarre dal modello dei dati.

In questo esempio, non vi sono due righe identiche. Anche se l'azienda avesse due venditori con lo stesso cognome e nome, le righe sarebbero distinguibili dalle informazioni presenti nelle altre colonne. Questa caratteristica ne implica un'altra: **l'ordine delle righe non è importante**. Per ottenere informazioni su Giovanni Sessa non è necessario sapere che numero di riga gli corrisponda. Tutto quello che è necessario sapere è un fatto o combinazione di fatti su di lui che lo identifichi in maniera univoca. La sua riga è individuabile tramite il suo cognome o data di assunzione, dato che nessun altro venditore ha gli stessi fatti in comune. Una tabella come questa ha anche un'altra caratteristica: **l'ordine delle colonne non è importante**. In questo caso, DATA_ASSUNZIONE può tranquillamente precedere COGNOME_VENDITORE.

Un corretto modello di dati contiene i giusti tipi di fatti e relazioni tra i vari tipi di dati. I modelli sono approssimazioni della realtà che rappresentano. Essi contengono solo il dettaglio necessario. L'obiettivo del disegno di un modello dei dati è di includere sufficiente dettaglio in maniera tale che l'unica domanda a cui non si possa rispondere è quella che non sarà mai posta.

4. FASI NEL DISEGNO DI UN DATABASE

Per esaminare le fasi coinvolte nella creazione di un efficace progetto di un database, consideriamo nuovamente l'esempio dell'ufficio vendite dell'azienda ALFACOM. Il direttore delle vendite ha bisogno di conoscere le risposte alle domande come:

- Che tipo di computer ha venduto l'azienda?
- Quanti ne sono immediatamente disponibili per la vendita?
- Chi sono i clienti?
- Quanti computers ha venduto ciscun venditore?
- Quali sono state le vendite totali di Gennaio?

Per iniziare, consideriamo che tipo di dati desideriamo registrare e che tipo di informazioni desideriamo estrarre dal nostro database. Le tre fasi principali nella progettazione di un database sono:

- Elencare le infomazioni basilari da gestire;
- Decidere quali fatti sono importanti;
- Rappresentare le relazioni.

4.1. Elencare le informazioni basilari da gestire

Il direttore delle vendite Giovanni Guglielmi ha bisogno di informazioni sulle vendite effettuate, i computers venduti, i clienti che li hanno acquistati ed i venditori che hanno venduto i computers ai clienti. Cioè le tabelle: **VENDITE, PRODOTTI, CLIENTI, VENDITORI.**

4.2. Decidere quali fatti sono importanti

Successivamente, occorre decidere quali fatti sono necessari per rispondere alle domande che saranno poste.

Per rispondere a domande riguardanti le vendite, il direttore ha bisogno di sapere quali e quanti computers sono stati venduti, chi li ha venduti, chi e quando li ha acquistati ed a che prezzo.

Per rispondere a domande riguardanti i prodotti che la ditta ALFACOM vende, il direttore ha bisogno di conoscere il nome di ogni prodotto ed il suo prezzo.

Individuato il genere di domande che il direttore delle vendite si aspetta di dover esaudire, tutto quello che serve a questo punto è il nome dell'acquirente, la ragione sociale dell'azienda cliente, l'indirizzo ed il numero di telefono.

A questo punto, tutto ciò che occorre al direttore riguardo ai venditori è semplicemente il cognome. La fig. 3 elenca i tipi di informazioni richieste dall'ufficio vendite, con le relative tabelle di appartenenza individuate.

Fig.3 - La lista delle tabelle per l'Ufficio Vendite

VENDITE (Descrizione prodotto, Quantità, Venditore, Cliente, Data, Prezzo di vendita)
PRODOTTI (Descrizione prodotto, Prezzo di listino)
CLIENTI (Nome, Ragione sociale, Indirizzo, Numero di telefono)
VENDITORI (Cognome, Nome)

4.3. Rappresentare le relazioni

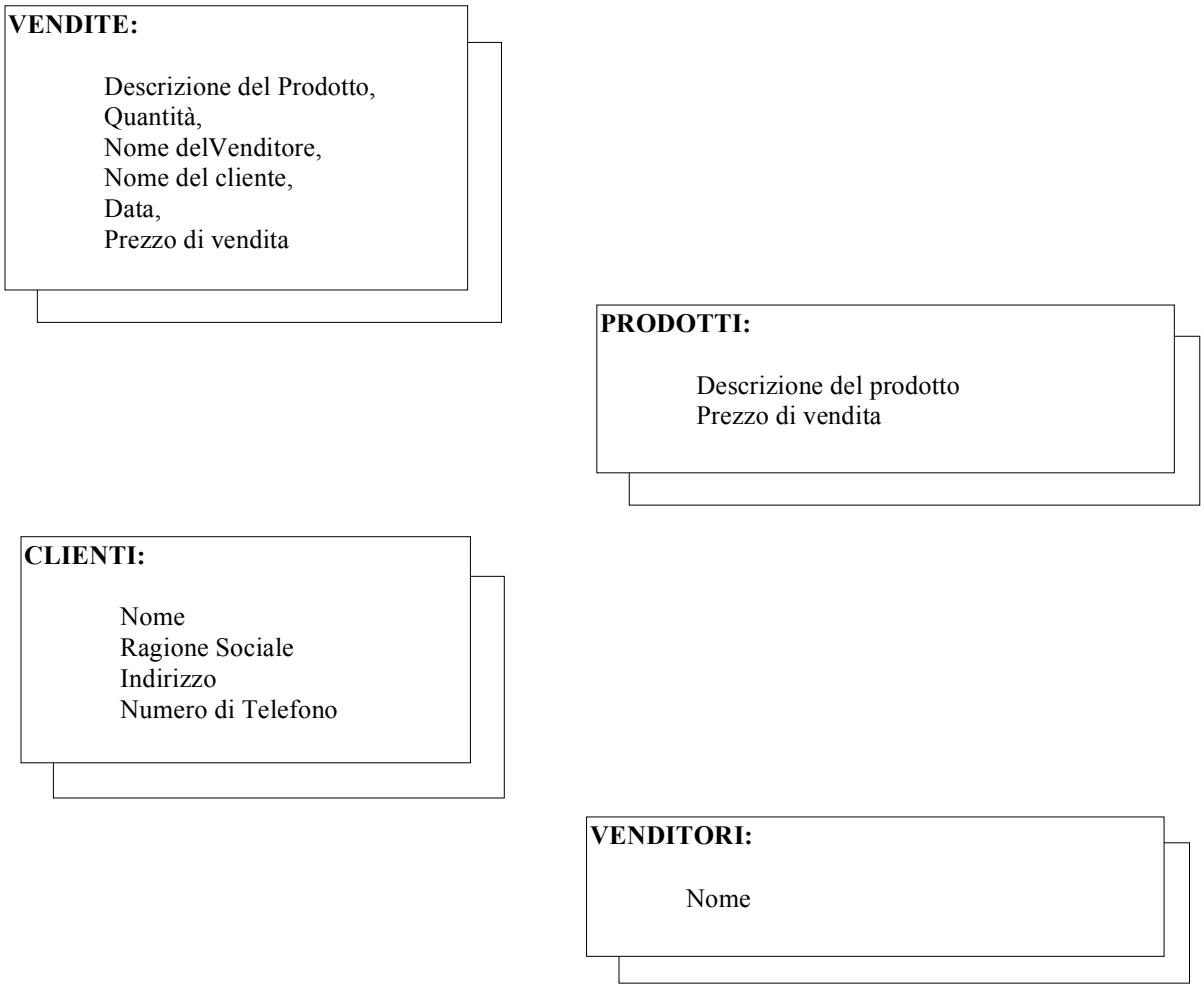
Occorre essere sicuri che il database rifletta le appropriate relazioni tra le varie tabelle di nomi, cose ed eventi. Nell'esempio, il modo giusto di procedere consiste

nell'osservare le tabelle **VENDITE, PRODOTTI, CLIENTI e VENDITORI** ed assicurarsi che il modello rappresenti le reali relazioni fra i dati.

Quali relazioni implicano le vendite? Le **VENDITE** sono correlate ai **PRODOTTI**. La loro correlazione può essere descritta come **uno-a-molti**. Cioè, vi possono essre molte vendite per ciascun prodotto, ma un solo prodotto ècoinvolto in ciascuna vendita. dato che le relazioni tra le tabelle sono rappresentate dai fatti che hanno in comune, il collegamento coi prodotti è rappresentato dalla descrizione del prodotto.

Vi sono anche relazioni uno-a-molti tra le **VENDITE** ed i **CLIENTI** e tra le **VENDITE** ed i **VENDITORI.** Ciascuna vendita è realizzata da un venditore, ma un venditore può effettuare più vendite. La fig. 4 mostra queste relazioni includendo la descrizione del prodotto, la ragione sociale del cliente ed il cognome del venditore nella tabella **VENDITE.**

Fig.4 - Relazioni nel Modello dei dati



5. MIGLIORAMENTO DEL PROGETTO

Avendo costruito un modello base, tentiamo di trovare dei modi per migliorarlo. Il metodo per migliorarlo è lo stesso usato per lo sviluppo iniziale attraverso le stesse tre fasi di progetto: elencare le informazioni basilari da gestire; decidere quali fatti sono importanti; rappresentare le relazioni tra questi oggetti.

5.1. Elencare le informazioni basilari da gestire

Nell'esempio dell'ufficio vendite della ALFACOM, il direttore vendite ha determinato che per rispondere a domande inerenti la sua attività gli occorrono informazioni sulle vendite effettuate, i prodotti venduti, i clienti ed i venditori. Il compito principale per migliorare il progetto consiste nell'esaminare più strettamente tali componenti informative ed espanderle laddove necessario.

5.2. Decidere quali fatti sono importanti

Il principio basilare consiste nel fatto che una tabella deve contenere informazioni su una sola categoria. La tabella **VENDITE**, ad esempio, dovrebbe contenere solo le informazioni necessarie per registrare con accuratezza una vendita effettuata. Consideriamo i seguenti dettagli inclusi nella tabella.

Il nome del cliente è un'informazione di vitale importanza. Risponde alla domanda: chi ha acquistato? Per avere una registrazione più completa, il direttore delle vendite ha anche bisogno di conoscere la ragione sociale dell'azienda per cui lavora l'acquirente, l'indirizzo ed il telefono. Sebbene il direttore delle vendite possa aver bisogno di conoscere tali informazioni, esse non necessitano di essere registrate nella tabella relativa alle vendite. Difatti, è bene non includere nulla di più dello stretto necessario in una tabella. La ridondanza di informazioni porta ad errori e, in particolare, rende estremamente difficili gli aggiornamenti. La soluzione è di avere una tabella separata che contiene una lista completa dei clienti e delle relative informazioni. Tutto quello che occorre nella tabella **VENDITE** è un collegamento che identifichi l'acquirente, l'azienda di appartenenza, l'indirizzo ed il telefono.

Per rispondere a più domande, due di queste categorie (il nome dell'acquirente e l'indirizzo) sono suddivise in maniera da specificare ulteriori dettagli. La tabella **CLIENTI** ora ha queste colonne: COGNOME_ACQUIRENTE, NOME_ACQUIRENTE, RAGIONE_SOCIALE_CLIENTE, INDIRIZZO, CAP, LOCALITÀ, PROVINCIA, TELEFONO.

La tabella può ora fornire più informazioni sui clienti, e può mantenere le informazioni insieme dove esse possono essere più facilmente aggiornate.

Per la tabella **VENDITE,** è in qualche maniera necessario identificare la persona che ha effettuato la vendita. Il direttore delle vendite, comunque, ha bisogno di altre informazioni sulla forza vendite, così si rende necessario espandere la tabella **VENDITORI** per contenere le colonne: COGNOME_VENDITORE, NOME_VENDITORE, INDIRIZZO,

CAP, LOCALITÀ, PROVINCIA, TELEFONO_ABITAZIONE, INTERNO, DATA_ASSUNZIONE.

Le informazioni contenute nella tabella **VENDITORI** rispondono a domande come le seguenti:

- A quale indirizzo l'azienda deve inviare il premio produzione di Semprini?
- Da quanto tempo Hernandez lavora per l'azienda?

Insieme alla tabella **VENDITE**, essa può servire a rispondere a domande come:

- A quanto ammontano le vendite mensili medie di Sessa dall'inizio e nell'ultimo anno?

La domanda fondamentale sui prodotti che la tabella **VENDITE** deve soddisfare è:

- Quali prodotti sono stati venduti?

Ciascun prodotto, perciò, deve essere identificato univocamente. la descrizione del prodotto e le informazioni addizionali che lo riguardano possono essere registrate nella tabella **PRODOTTI.**

Per rispondere a domande relative alle vendite, il direttore ha bisogno di rispondere a domande come queste:

- Quanto costa all'azienda produrre questo prodotto?
- Qual è il suo prezzo di listino?
- Quanti pezzi abbiamo in stock?

Per rispondere a domande sui prodotti stessi, il direttore delle vendite ha bisogno di conoscere il codice del prodotto e
sua descrizione. La tabella **PRODOTTI** ora include le seguenti colonne: COSTO, PREZZO, ESISTENZA, CODICE_PRODOTTO, DESCRIZIONE_PRODOTTO.

Il direttore delle vendite ha anche bisogno di conoscere le componenti principali incluse in ciascun prodotto. Questo perché i prodotti che l'azienda commercializza (personal computer) hanno tre componenti principali: un video, un'unità centrale ed una tastiera. Per questi prodotti, l'azienda offre la scelta di due video, due tastiere e sei unità centrali.

Per tenere facilmente sotto costrollo le varie configurazioni, il direttore delle vendite aggiunge al modello una nuova tabella, contenente informazioni riguardanti le componenti dei prodotti. La tabella **COMPONENTI** ha le seguenti colonne: CODICE_PRODOTTO, CODICE COMPONENTE, DESCRIZIONE_COMPONENTE.

5.3. Rappresentare le relazioni

Quando i modelli di dati diventano complicati, è necessario usare qualche metodo per evidenziare le relazioni. Un modo è quello di rappresentarle in rettangoli collegati fra loro come mostrato in fig.5. L'unica relazione che coinvolge i clienti è quella che collega il cliente alla vendita. Per mostrare che esiste una tale relazione, si disegna una linea che unisce i relativi rettangoli. Ora, occorre individuare di che tipo è la relazione che collega la tabella **CLIENTI** alla tabella **VENDITE**: è di tipo uno-a-molti o molti-a-molti? La risposta deriva dalla natura stessa della relazione in esame e dal ruolo che in essa riveste il cliente. Ciascun cliente può essere presente in più transazioni di vendita; cioè, ogni cliente può ac-

quistare più volte. In ciascuna vendita, comunque, vi è un solo cliente che ha effettuato l'acquisto. La relazione è, dunque, del tipo **uno_a_molti:** un cliente a molte vendite. La fig.5 illustra un modo di raffigurare tale relazione.

Si osservi la freccia singola sul rettangolo CLIENTI (indicante che questo è il lato "uno" della relazione), e la freccia doppia sul rettangolo VENDITE (indicante che questo è il lato "molti" della relazione).

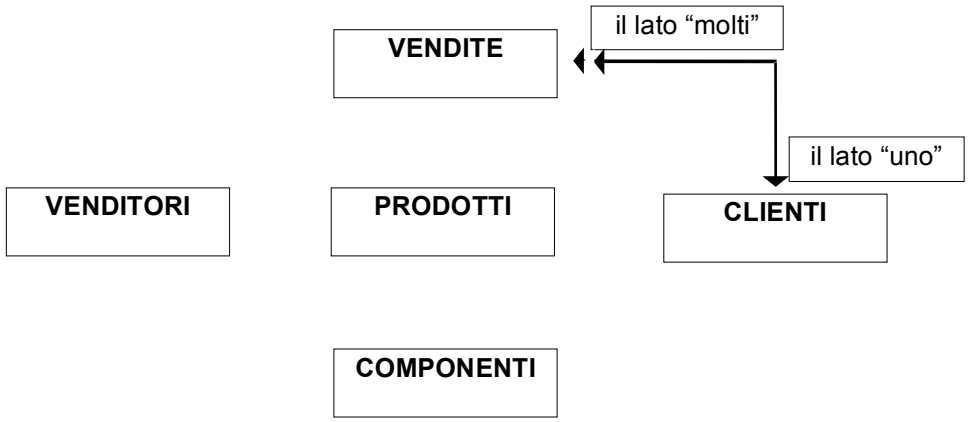


Fig.5 - Il lato "uno" ed il lato "molti" della relazione VENDITE-CLIENTI

L'importanza di determinare la natura di questa relazione diventerà presto chiara non appena il direttore delle vendite avrà bisogno di informazioni da entrambe le tabelle. Le risposte alle domande spesso coinvolgono più di una tabella. Per esempio, si osservi la fig.6. Se il direttore delle vendite ha bisogno di conoscere il nome della persona che ha acquistato cinque PC Standard a colori il 17 Settembre 1989, ricava il nome della riga relativa all'azienda nella tabella **CLIENTI.** Alla domanda si può rispondere solo se c'è una riga nella tabella **CLIENTI** per l'azienda Industrial Computers.

Il modello deve includere il vincolo che nessuna riga può essere aggiunta alla tabella **VENDITE** per un dato cliente, fnchè lo stesso non sia stato inserito nella tabella **CLIENTI.**

Fig.6 - Le tabelle VENDITE e CLIENTI

Tabella **VENDITE**

DATA	RAGIONE_SOCIALE CLIENTE	PRODOTTO	QUANTITÀ	PREZZO	COGNOME VENDITORE
17/09/1989	Industrial Computers	PC standard a colori	5	1.530.000	SEMPRINI
24/09/1989	Industrial Computers	PC avanzato a colori	5	2.575.000	HERNANDEZ
10/10/1989	Computer Distributors	PC standard	10	1.800.000	SESSA
17/10/1989	PC Distribution	PC portatile classico	8	3.000.000	CARLI

Tabella **CLIENTI**

NOME ACQUIR.	COGN. ACQUIR.	RAG.SOC. CLIENTE	INDIRIZZO	CAP	LOCALITÀ	PR	TEL.
Anna	ADAMI	Sudest Computers	Via Garibaldi 5	00100	Roma	RM	1012578
Andrea	CHINI	St.Consulenti Associati	Via Sempione 15	20100	Milano	MI	2207659
Gianna	FERRI	Industrial Computers	P.zza Mille 105	70100	Bari	BA	331456
Walter	GIACOMI	Computer Distributors	Via Volta 140	10100	Torino	TO	210567
Sara	GIORGI	PC Distribution	SS.100 Km.650	71100	Foggia	FG	33910
Bernardo	GIANNINI	Midtown Computer	Piazza Nenni 23	71016	S.Severo	FG	72089
Bruno	GIANNINI	Computer Warehouse	Via Veneto 54	00100	Roma	RM	8190576
Stella	VALIATI	Industrial Concepts	Via Garibaldi 50	70100	Bari	BA	445671

La figura 7 mostra come possa essere illustrato tale requisito:

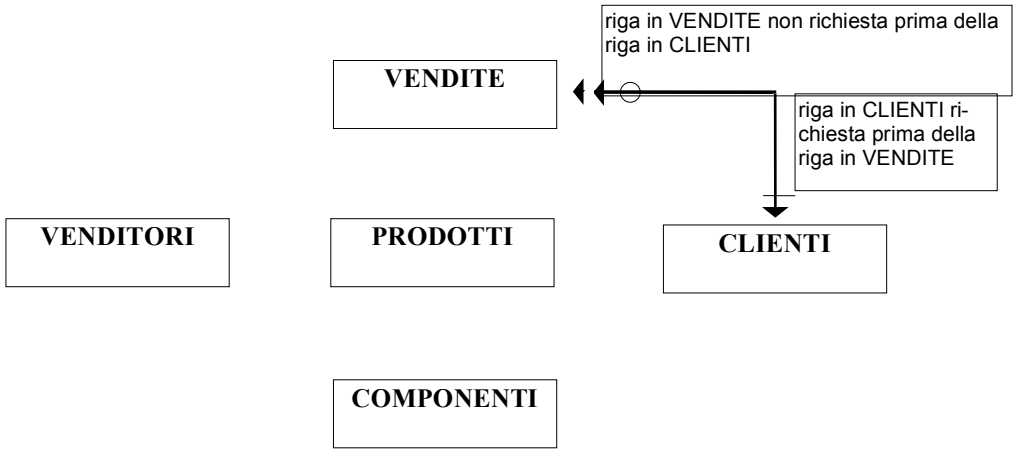


Fig.7 - I vincoli della Relazione VENDITE-CLIENTI

La linea sottile disegnata sulla freccia singola dal lato del rettangolo CLIENTI indica che deve esistere una riga nella tabella **CLIENTI** prima che qualsiasi riga ad essa relativa possa essere inserita nella tabella **VENDITE**. Il cerchietto vicino alla doppia freccia dal lato del rettangolo VENDITE indica che non è necessario che esista una riga nella tabella **VENDITE** per ogni riga della tabella **CLIENTI**. Si noti che il lato opzionale corrisponde al lato "molti" della relazione, mentre il lato richiesto corrisponde al lato "uno". In questo modo è possibile registrare clienti che non hanno ancora acquistato nulla (ad esempio per effettuare delle campagne promozionali).

L'unica relazione diretta che coinvolge i venditori è quella che lega il venditore alla transazione di vendita. Ciascun venditore può effettuare più vendite, ma ciascuna vendita implica un solo venditore. Così come nel caso delle tabelle **CLIENTI** e **VENDITE**, la relazione è del tipo uno-a-molti: un venditore a molte vendite. La tabella **VENDITORI**, una lista completa di tutti i venditori in forza all'ALFACOM, permette all'azienda di registrare informazioni su di essi indipendentemente dal fatto che abbiano o meno effettuato delle vendite. Prima che una riga possa essere registrata nella tabella **VENDITE** per un dato venditore, comunque, la relazione richiede che la riga relativa a tale venditore sia presente nella tabella **VENDITORI.** Questo vincolo è illustrato dalla lineetta vicino al rettangolo

VENDITORI (il che significa che una riga per il venditore deve essere presente in tale tabella prima che una riga nella tabella **VENDITE** faccia riferimento ad esso), e dal cerchietto vicino al rettangolo **VENDITE** (che illustra il fatto che una riga relativa ad un venditore può esistere nella tabella **VENDITORI** senza che egli debba necessariamente aver effettuato delle vendite).

La tabella **PRODOTTI** è collegata alla tabella **VENDITE** da una relazione uno-a-molti: ciascun tipo di computer può essere incluso in più transazioni di vendita, ma ciascuna vendita include un solo tipo di computer.

Similmente alla relazione **CLIENTI-VENDITE** ed alla relazione **VENDITORI-VENDITE**, anche questa relazione richiederà che nesuna riga possa essere inserita nella tabella **VENDITE** se le informazioni relative al prodotto venduto non sono già presenti nella tabella **PRODOTTI.** Il direttore delle vendite può registrare informazionis sui prodotti senza che questi debbano necessariamente essere stati venduti.

La tabella **COMPONENTI** è collegata alla tabella **PRODOTTI** da una relazione molti-a-molti: ciascuna componente può essere inclusa in più di un computer, e ciascun computer prodotto può avere sino a tre componenti (video, unità centrale, tastiera nei diversi modelli previsti).

Si osservi che nella fig.8 vi sono doppie frecce vicino ai rettangoli PRODOTTI e COMPONENTI di questa relazione. Il vincolo da rispettare consiste nel non inserire un prodotto nella tabella **PRODOTTI** se le sue componenti non sono già presenti nella tabella **COMPONENTI.**

Il disegno del database con tutte le relazioni stabilite è quindi quello rappresentato in fig.8.

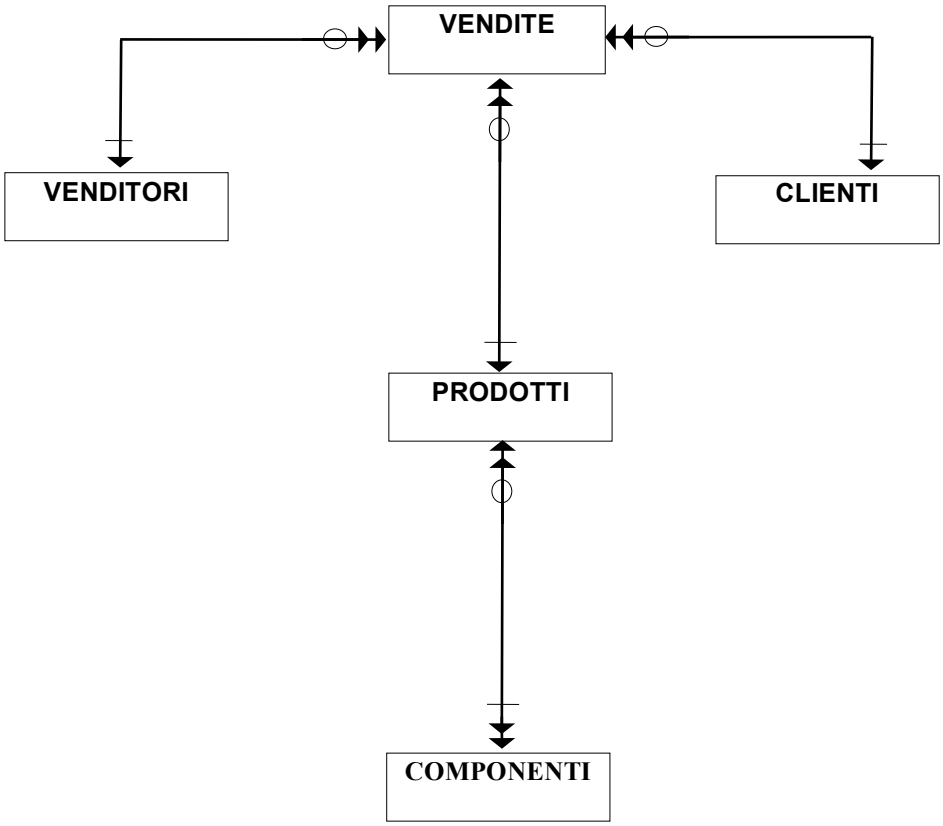


Fig.8 - Relazioni e vincoli: un progetto migliorato

6. GESTIONE DELLE RELAZIONI UNO-A-MOLTI

Il passo finale nel progetto del modello dei dati consiste nell'assicurarsi che esistano i collegamenti appropriati. Nell'esempio dell'azienda ALFACOM, tre relazioni sono del tipo uno-a-molti: un venditore, prodotto o cliente a più vendite. Una è del tipo molti-a-molti: più prodotti a più componenti.

6.1. Scelta dei collegamenti

La colonna o le colonne utilizzate per collegare due tabelle in una relazione uno-a-molti dovrebbero identificare univocamente una riga su un lato della relazione. Per vedere il perché, osserviamo nuovamente la relazione **CLIENTI-VENDITE.**

La fig.9 mostra alcune righe campione delle tabelle utilizzando solo il cognome dell'acquirente come collegamento.

Fig.9 - Il Cognome come collegamento

Tabella **VENDITE**

DATA	**COGNOME ACQUIRENTE**	PRODOTTO	QUANTITÀ	PREZZO	COGNOME VENDITORE
17/10/1989	**GIORGI**	PC portatile classico	8	3.000.000	SESSA
09/11/1989	**GIANNINI**	PC standard	15	1.800.000	SESSA

Tabella **CLIENTI**

NOME ACQUIR.	**COGNOME ACQUIRENTE**	RAG.SOC. CLIENTE	INDIRIZZO	CAP	LOCA-LITÀ	PR	TEL.
Bernardo	**GIANNINI**	Midtown Computer	P.zza Nenni 23	71016	S.Severo	FG	72089
Bruno	**GIANNINI**	Computer Warehouse	Via Veneto 54	00100	Roma	RM	819057 6

Supponiamo che il direttore delle vendite voglia trovare il numero di telefono del cliente che ha acquistato il PC Standard il 9 novembre 1989. Per ottenere il telefono, osserva prima la tabella **VENDITE** per trovare il collegamento. Visto che il valore del collegamento è GIANNINI, esamina successivamente la tabella **CLIENTI** per trovare la riga avente GIANNINI nella colonna COGNOME. Ora sorge un problema. Il cognome non identifica univocamente una riga nella tabella **CLIENTI.** Quale GIANNINI sta cercando? Bernardo o Bruno? Per risolvere l'ambiguità occorre considerare sia il cognome che il nome: questo perché, come già detto, la colonna o combinazione di colonne che servono da collegamento devono identificare univocamente le righe della tabella che viene collegata.

Per evitare il problema delle righe duplicate, il collegamento tra le tabelle **CLIENTI** e **VENDITE** deve essere o una combinazione di colonne (cognome e nome) o una colonna univoca. Una scelta migliore del cognome e nome dell'acquirente sarebbe la ragione sociale dell'azienda cliente. Nell'ambiente delle aziende come la ALFACOM, ciò probabilmente eliminerebbe la necessità di utilizzare combinazioni di colonne per ottenere un collegamento univoco.

Un'altra buona scelta è il numero di telefono. In questo caso, comunque, la scelta migliore sarebbe una colonna di collegamento creata ad-hoc: il **CODICE_CLIENTE.**
Esso offre i seguenti vantaggi:

- È **univoco**. Elimina qualsiasi ambiguità sul nome dell'acquirente o dell'azienda e la necessità di avere più di una colonna per identificare una riga.
- È **breve**. Nella maggior parte dei casi, sarà più corto del nome del cliente, della ragione sociale dell'azienda, o del numero di telefono. Sul lato "molti" della relazione, dove esso è usato spesso, occupa meno spazio.
- È **facile da mantenere.** Semplifica l'aggiornamento se, per esempio, Alfredo Harris sostituisce Guglielmo Giannini alla Computer Warehouse, o se la Midtown Computer decide di cambiare il suo nome in M.C.C.
- A differenza dei valori di tipo carattere, è più **facile introdurlo senza errori.** Per esempio, uno potrebbe ricordare il cognome di un cliente come Rosini, mentre in realtà è Rossini.

Una piccola azienda con pochi dipendenti potrebbe utilizzare il cognome come colonna di collegamento fra le tabelle **VENDITORI** e **VENDITE**. Una grande azienda, comunque, può incorrere negli stessi problemi con i nomi dei venditori così come successo con i nomi dei cienti. Due o più dipendenti possono avere lo stesso primo nome: Giovanni Maria Sessa e Giovanni Carlo Sessa possono essere due venditori. Un dipendente può cambiare nome: Maria Semprini può diventare, sposandosi Maria Tozzi. Come per la tabella **CLIENTI,** una soluzione migliore di quella dell'uso dei nomi (che possono essere non univoci) consiste nell'utilizzare un codice univoco, come ad esempio la matricola del dipendente od il suo codice fiscale. Per brevità (tre cifre invece di sedici caratteri) conviene adoperare la matricola del dipendente. Per il direttore delle vendite, questa è una buona scelta per la colonna di collegamento tra le tabelle **VENDITORI** e **VENDITE.**

Per facilità di registrazione e mantenimento delle informazioni e per una maggiore accuratezza, una buona scelta per il collegamento fra le tabelle **PRODOTTI** e **VENDITE** è la colonna del codice prodotto. Come il codice cliente e la matricola dipendente, è non ambiguo.

7. GESTIONE DELLE RELAZIONI MOLTI-A-MOLTI

Sebbene la colonna CODICE_COMPONENTE colleghi la tabella **PRODOTTI** e la tabella **COMPONENTI**, il direttore delle vendite ha ancora un problema di registrazione ed integrità delle informazioni.

Questa è una relazione molti-a-molti. Ciascun prodotto utilizza più componenti, e ciascuna componente è utilizzata in più prodotti. Nel modello attuale dei dati, ogni volta che una componente diversa è registrata nella tabella **PRODOTTI** per un dato prodotto, tutte le altre informazioni per quel prodotto devono essere nuovamente inserite.

Questo problema può essere risolto creando una **tabella di collegamento**. Invece di includere il codice componente nella tabella **PRODOTTI,** è opportuno creare una nuova tabella che includa due colonne di collegamento: una dalla tabella **PRODOTTI** (il CODICE_PRODOTTO) ed una dalla tabella **COMPONENTI** (il CODICE_COMPONENTE), come illustrato in fig.10.

Questa tabella di collegamento è denominata **COMPONENTI_USATE.** Le relazioni a cui partecipa tale tabella sono rappresentate in fig.11.

C'è una relazione uno-a-molti tra le tabelle **PRODOTTI** e **COMPONENTI_USATE** ed una relazione uno-a-molti tra **COMPONENTI** e **COMPONENTI_USATE.** Poichè non esistono coppie di valori identici nelle righe di quest'ultima tabella, non sono necessarie altre colonne per identificare univocamente una riga. Si osservino le analogie nella fig.10 tra la tabella **COMPONENTI_USATE** e la tabella **VENDITE**.

La tabella **VENDITE** è anche'essa in realtà una tabella di collegamento.

8. IL PROGETTO COMPLETO

La fig.10 mostra una lista di tabelle e colonne nel progetto completo, e la fig.11 mostra il diagramma finale del progetto.

VENDITE:
- Numero Progr.vo Vendita
- Matricola Venditore
- Codice Cliente
- Codice Prodotto
- Quantità venduta
- Data
- Prezzo

COMPONENTI:
- Codice componente
- Descrizione

PRODOTTI:
- Codice Prodotto
- Descrizione
- Giacenza
- Costo di Acquisto
- Prezzo di Listino
- Ubicazione

CLIENTI:
- Codice Cliente
- Cognome Acquirente
- Nome Acquirente
- Ragione Sociale
- Indirizzo
- C.A.P.
- Località
- Provincia
- Numero di Telefono

VENDITORI:
- Matricola Venditore
- Cognome
- Nome
- Indirizzp
- Telefono Abitazione
- Interno
- Data di Assunzione

COMPONENTI USATE:
- Codice Prodotto
- Codice componente

Fig. 10 - Il Progetto completo dei dati

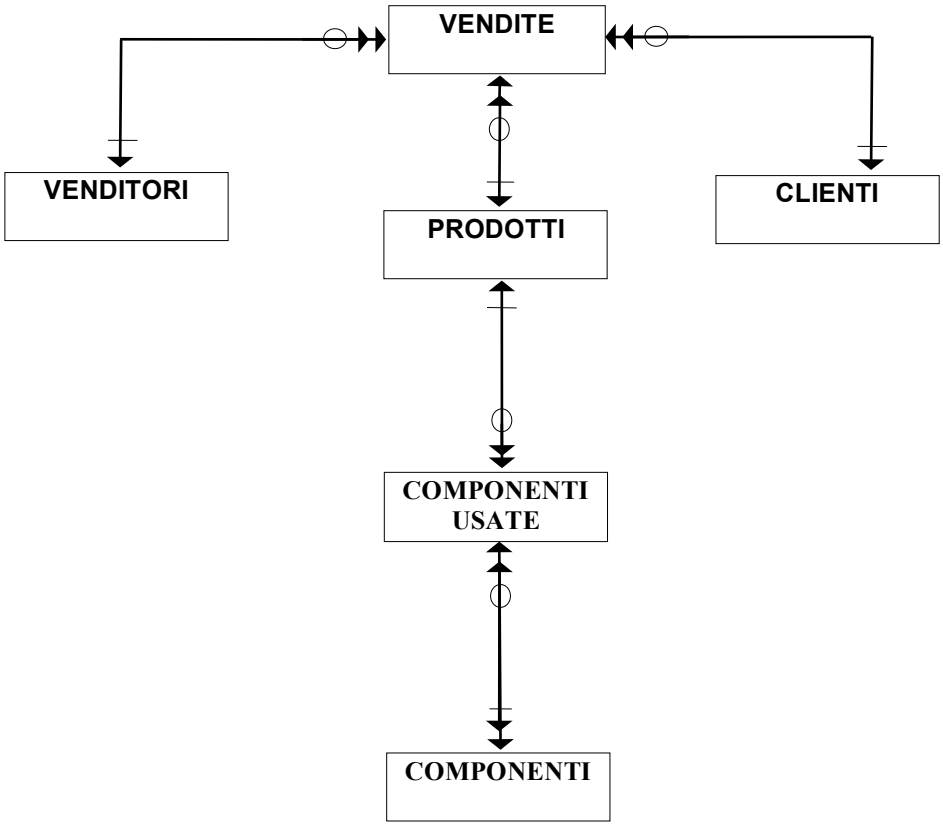


Fig.11 - Relazioni e vincoli: il progetto finale

BIBLIOGRAFIA

1. Bernstein, Hadzilacos, Goodman: *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, 1987.
2. Chen: *The Entity-Relationship Model - Toward a Unified View of Data*, ACM TODS, 1976.
3. Chen: *Entity-Relationship Approach to Information Modeling and Analysis*, North-Holland, 1983.
4. Codd: *The Relational Approach to Database Management - An Overview*, Third Annual Texas Conference on Computing Systems, Austin, 1974.
5. Codd: *Principles of Design of Database Management Systems*, San Jose, The Relational Institute, Technical Report EFC-18, 1987.
6. *SQL Reference Manual*, IBM Corporation, 1988.
7. Maier, Ullman, Vardi: *On the Foundations of the Universal Relational Model*, ACM TODS, Vol.9 n.2, 1984.
8. Vardi: *The Universal-Relational Data Model for Logical Independence*, IEEE Software, Marzo 1988.
9. Wiorkowski, Kull: *DB2 Design and Development Guide*, Addison-Wesley, 1988.
10. Codd: *The Relational Model for Database Management - Version 2*, Addison-Wesley, 1990.
11. *Draft proposed American National Standard Database Language SQL*, ANSI Inc., 1985.
12. Chao-Chih Yang: *Relational Databases*, Prentice-Hall, 1986.
13. Date: *An Introduction to Database Systems*, Addison-Wesley, 1986.
14. Teorey, Fry: *Design of Database Structures*, Pretice-Hall, 1982.
15. Tsichritzis, Klug: *The ANSI/X3/SPARC DBMS Framework - Report of the Study Group on Database Management Systems*, Information Systems 3, 1978.
16. Tsichritzis, Lochovsky: *Data Models*, prentice-Hall, 1982.
17. Ullman: *Principles of Database and Knowledge-Base Systems*, Computer Science Press, 1991.
18. Beech: *Groundwork for an Object Database Model*, Hewlett-Packard, 1987.
19. Brodie, Mylopoulos: *On Knowledge Base Management Systems*, Springer-Verlag, 1986.
20. Brodie, Mylopoulos, Schmidt: *On Conceptual Modeling*, Springer-Verlag, 1984.
21. Ceri, Pelagatti: *Distributed Databases - Principles and Systems*, McGraw-Hill, 1984.
22. *CODASYL Data Base Task Group April 71 Report*, ACM, 1971.
23. Fernandez, Summers, Wood: *Database Security and Integrity*, Addison-Wesley, 1980.
24. Frost: *Introduction to Knowledge Base Systems*, MacMillan, 1986.
25. Klug: *Equivalence of Relational Algebra and Relational Calculus QueryLanguages having Aggregate Functions*, J.ACM, 1981.
26. Korth, Silberschatz: *Database Systems Concepts,* McGraw-Hill, 1986.

27. Minker: *Foundations of Deductive Databases and Logic Programming*, Morgan-Kaufmann, 1988.
28. Olle: *The Codasyl Approach to Database Management*, John Wiley and Sons, 1978.
29. Papadimitriou: *The Theory of Database Concurrency Control,* Computer Science Press, 1986.
30. Tanimoto: *The Element of Artificyal Intelligence,* Computer Science Press, 1987.
31. Tsichritzis, Lochovsky: *Data Models*, Prentice-Hall, 1982.
32. Wiederhold: *Database Design,* McGraw-Hill, 1983.
33. Wiederhold: *File Organization for Database Design*, McGraw-Hill, 1987.