E. F. Codd

Un modello relazionale dei dati per grandi banche dati condivise

Introduzione

Alla fine degli anni sessanta l'informatica viveva ancora la sua era pionieristica, i "Centri Meccanografici" si stavano pian piano trasformando in "Centri Elaborazione Dati" e si stava entrando nella fase di crescita che avrebbe assegnato, col passar degli anni, ai sistemi informatici il ruolo centrale nel business che oggi rivestono indiscutibilmente.

Per rendere sostenibile questa crescita, i pionieri hanno dovuto affrontare alcune questioni che, se non risolte, avrebbero potuto sicuramente rallentare, se non frenare lo sviluppo del settore.

Sono stati brillantemente affrontati problemi fondamentali come il costo di produzione e di gestione dell'hardware, l'usabilità dei sistemi (si pensi al passaggio dalle schede perforate alle moderne periferiche di input/output), la semplicità di programmazione degli elaboratori (passaggio dal linguaggio macchina ai moderni linguaggi ad oggetti, passando per l'assembly ed il COBOL) e la gestione delle informazioni.

Proprio in quest'ultimo campo ha apportato il suo contributo decisivo Edgar Frank (Ted) Codd.

Nato in Inghilterra nel 1923, Codd si trasferì negli Stati Uniti nel 1948 per lavorare in IBM con la qualifica di "Matematico Programmatore". Nella seconda metà degli anni sessanta Codd affrontò il problema della gestione centralizzata delle informazioni, argomento che, proprio in quegli anni, generava un grande fermento. Era infatti evidente che il modello secondo cui ogni sistema dovesse autonomamente gestire i propri dati non stava più in piedi. Questo modello, difatti, generava duplicazioni di informazioni e problemi di sicurezza.

Era quindi partita la corsa alla definizione di un modello di gestione centralizzata dei dati che potesse essere soddisfacente per tutte le tipologie di applicazione che si trovassero a doverlo utilizzare. Già a metà degli anni sessanta erano disponibili prodotti commerciali per la gestione strutturata dei dati, tra questi di distinguevano IDS ed IMS (dell'IBM). Per iniziativa di Charles Bachman, autore di IDS, in seno al CODASYL, il gruppo che si occupava della standardizzazione del COBOL, nacque una specifica task force dedicata ai database che pubblicò il proprio standard nel 1971. I database conformi in tutto od in parte a questo standard erano basati su strutture gerarchiche o reticolari in cui il programmatore doveva navigare tra i dati utilizzando appositi link, opportunamente previsti in fase di progettazione della base dati, per arrivare all'informazione di cui aveva bisogno. Questi database, che per questa caratteristica erano detti anche "navigazionali", riscontrarono un discreto successo, portando Bachman ad aggiudicarsi il Turing award nel 1973 (la relazione letta da Bachman in quell'occasione si intitolava appunto "The programmer as a navigator").

E' esattamente in questo contesto che si inserisce l'opera di Codd. Alla fine degli anni sessanta egli si dedicò infatti a definire un nuovo modello (relazionale) per la gestione dei dati evidenziando in vari lavori i limiti dei sistemi a quel tempo diffusi e delineando le sfide che dovevano essere intraprese per la realizzazione di sistemi basati su questo nuovo modello.

Il culmine di quest'attività di ricerca, continuata poi fino agli anni novanta, è senza dubbio l'articolo "A Relational Model of Data for Large Shared Data Banks" vera e propria bibbia della teoria relazionale in cui si gettano le basi non solo dei moderni database relazionali ma anche del linguaggio strutturato di interrogazione (SQL). L'articolo, scritto nel 1969, fu pubblicato nel giugno del 1970 nel numero 6, volume 13 del periodico "Communications of the ACM" pubblicato appunto dall'"Association for Computing Machinary".

Giunta al quarantesimo anniversario quest'opera è ancora straordinariamente attuale, il che è praticamente un miracolo vista la velocità con cui l'informatica si evolve.

Troppo spesso si utilizzano gli strumenti messi a disposizione dall'informatica ignorandone la genesi e non riconoscendo il giusto merito a chi li ha pensati. Per questo motivo ho deciso di riprendere oggi, nel 2010, il lavoro di Codd, traducendolo in italiano e diffondendolo mediante Internet. Nella speranza di aumentare di qualche unità il numero di persone che, quando scrivono una istruzione di SELECT, sono consapevoli di chi ha reso possibile l'esecuzione di questo (apparentemente) semplice gesto.

In coda all'articolo ho tentato di riassumere i principali passi che ci hanno portato dall'articolo di Codd alle ultime versioni dei moderni database relazionali.

Massimo Ruocchio
http://oracleitalia.wordpress.com

Un modello relazionale dei dati per grandi banche dati condivise

E. F. Codd
Laboratori di Ricerca IBM, San Jose, California

I futuri utilizzatori di grandi banche dati non dovranno preoccuparsi di sapere come i dati sono organizzati a livello fisico (cioè della rappresentazione interna). Un servizio d'aiuto interattivo che fornisce informazioni sull'organizzazione fisica dei dati non è una soluzione soddisfacente. L'attività degli utenti ai terminali e la maggior parte dei programmi applicativi non devono essere impattati quando la rappresentazione interna dei dati cambia ed anche quando cambiano soltanto alcuni aspetti della rappresentazione esterna dei dati. I cambiamenti alla rappresentazione dei dati sono spesso dovuti ai cambiamenti delle query, degli aggiornamenti, della quantità di accessi ed al naturale aumento delle tipologie di informazioni archiviate.
I sistemi esistenti (non inferenziali) per la gestione dei dati strutturati forniscono agli utenti file gerarchici o modelli reticolari poco più generici. Nella Sezione 1, si tratta l'inadeguatezza di questi sistemi. Si introducono un modello basato su relazioni n-arie, una forma normale per le relazioni di database ed i fondamenti di un linguaggio universale dei dati. Nella Sezione 2 alcune operazioni sulle relazioni (oltre all'inferenza logica) vengono discusse ed applicate ai problemi della ridondanza e della consistenza del modello dati.

1. Modello Relazionale e Forma Normale

1.1. Introduzione

Quest'articolo tratta l'applicazione della teoria elementare delle relazioni ai sistemi che forniscono l'accesso condiviso a grandi banche di dati strutturati. Ad eccezione di un articolo di Childs [1], la principale applicazione delle relazioni ai sistemi di dati è stata ai sistemi deduttivi domanda-risposta. Levein e Maron [2] forniscono numerose referenze a lavori in questo settore.

Al contrario, i problemi trattati qui sono quelli dell'*indipendenza dei dati* – l'indipendenza dei programmi applicativi e delle attività dei terminali dall'aumento dei tipi di dato e dal cambiamento della rappresentazione dei dati – ed alcuni tipi di *inconsistenza dei dati* che ci si aspetta creino problemi anche nei sistemi non deduttivi.

La vista (o modello) relazionale dei dati descritta in Sezione 1 sembra superiore da diversi punti di vista rispetto al modello reticolare o a grafi [3,4] attualmente molto diffuso per sistemi non inferenziali. Esso fornisce un modo per descrivere i dati solo attraverso le loro caratteristiche naturali – cioè senza sovrapporre strutture fittizie ai fini della rappresentazione fisica. Allo stesso tempo questa fornisce le basi per un linguaggio dati di alto livello che consentirà di ottenere la massima indipendenza tra i programmi da una parte e la rappresentazione fisica e l'organizzazione dei dati dall'altra.

Un ulteriore vantaggio della vista relazionale è che costituisce una solida base per trattare derivabilità, ridondanza e consistenza delle relazioni – queste vengono discusse nella Sezione 2. Il modello reticolare, d'altra parte, ha generato svariati malintesi, come l'equivoco tra la derivazione delle connessioni e la derivazione delle relazioni (vedi le note in Sezione 2 sulla "trappola delle connessioni").

Infine, la vista relazionale consente una valutazione più chiara della portata e delle limitazioni logiche dei sistemi di dati strutturati attuali, ed anche dei vantaggi (da un punto di vista logico) di avere allo stesso tempo differenti rappresentazioni dei dati all'interno dello stesso sistema. Esempi di questo punto di vista più chiaro sono citati in varie parti di quest'articolo. L'implementazione di sistemi per supportare il modello relazionale non è discussa.

1.2. Dipendenza tra Dati nei Sistemi Esistenti

La disponibilità di tabelle che descrivono i dati nei sistemi recenti rappresenta un importante sviluppo verso la meta dell'indipendenza dei dati [5,6,7]. Queste tabelle facilitano il cambiamento di alcune caratteristiche della rappresentazione fisica dei dati nella banca dati. In ogni caso, il numero di caratteristiche della rappresentazione fisica che possono essere cambiate *senza impatto sui programmi applicativi* è ancora abbastanza limitato. Inoltre, il modello dei dati con cui l'utente interagisce non è ancora sgombro dalle proprietà di rappresentazione, specialmente nel caso di collezioni di dati (e non di singole informazioni). Tre dei principali tipi di dipendenza da eliminare sono: dipendenza dall'ordinamento, dipendenza dall'indicizzazione e dipendenza dal percorso di accesso. In alcuni sistemi queste dipendenze non sono neanche nettamente separate tra loro.

1.2.1. *Dipendenza dall'ordinamento.* I dati elementari in una banca dati possono essere archiviati in molti modi, alcuni di questi non tengono conto dell'ordinamento, alcuni consentono ad ogni elemento di partecipare in un unico ordinamento, altri permettono ad ogni elemento di partecipare a diversi ordinamenti. Prendiamo in considerazione quei sistemi esistenti che permettono o richiedono che i dati elementari siano archiviati in almeno un ordinamento complessivo che è strettamente associato all'ordinamento degli indirizzi dipendente dall'hardware. Per esempio, i record di un file che tratta pezzi di ricambio potrebbe essere archiviato in ordine ascendente rispetto al numero seriale del pezzo. Questi sistemi normalmente permettono ai programmi applicativi di dare per scontato che l'ordine di estrazione dei record dai file di questo tipo sia identico (o sia un sotto-ordinamento) all'ordinamento predefinito. Questi programmi applicativi che sfruttano l'ordinamento fisico del file hanno buone possibilità di dare risultati errati se per

qualche motivo bisogna cambiare l'ordinamento fisico con un altro. Lo stesso tipo di ragionamento si applica anche agli ordinamenti ottenuti mediante puntatori.

Non è necessario portare ad esempio un sistema specifico perché tutti i più noti sistemi che sono oggi sul mercato non riescono a distinguere chiaramente tra l'ordinamento della presentazione dei risultati di una ricerca da una parte e l'ordinamento fisico dei dati dall'altra. Alcune significative problematiche nell'implementazione devono essere risolte per poter ottenere questo tipo di indipendenza.

1.2.2. *Dipendenza dall'indicizzazione.* Nel contesto dei dati strutturati, si pensa ad un indice come un componente della rappresentazione dei dati utile soltanto al miglioramento delle performance. Esso serve a migliorare i tempi di risposta delle query e degli aggiornamenti e, allo stesso tempo, rallenta inserimenti e cancellazioni. Dal punto di vista del contenuto informativo, un indice è un componente ridondante della rappresentazione dei dati. Se un sistema fa uso di indici e vuole ottenere buone performance in un ambiente in cui le modalità di accesso ai dati sono varie, la possibilità di creare ed eliminare gli indici di tanto in tanto potrebbe probabilmente essere richiesta. Sorge quindi una domanda: Possono i programmi applicativi e le attività da terminale non essere impattati se gli indici vanno e vengono?

I sistemi di dati strutturati esistenti utilizzano approcci molto differenti all'indicizzazione. TDMS [7] fornisce un'indicizzazione incondizionata su tutti gli attributi. L'ultima versione rilasciata di IMS [5] dà all'utente la possibilità di scegliere, per ogni file, se non indicizzare affatto (organizzazione gerarchica sequenziale) oppure indicizzare solo sulla chiave primaria (organizzazione gerarchica sequenziale indicizzata). In nessuno dei due casi la logica applicativa dell'utente è dipendente dall'esistenza degli indici forniti incondizionatamente. IDS [8], in ogni caso, permette al progettista dei file di selezionare gli attributi da indicizzare e di incorporare gli indici nella struttura del file per mezzo di collegamenti addizionali. I programmi applicativi che sfruttano i benefici nelle performance di questa indicizzazione devono far riferimento a questi collegamenti addizionali per nome. Questi programmi non funzioneranno correttamente se l'indicizzazione sarà rimossa in un secondo momento.

1.2.3. *Dipendenza dal percorso di accesso.* Molti dei sistemi di dati strutturati esistenti forniscono all'utente file strutturati ad albero (gerarchici) o modelli dati reticolari poco più generici. I programmi applicativi sviluppati per lavorare con questi sistemi tendono ad essere logicamente impattati se la struttura gerarchica o i reticoli cambiano. Di seguito un semplice esempio.

Si supponga che la banca dati contenga informazioni sugli articoli ed i progetti. Per ogni articolo sono archiviati il numero dell'articolo, il nome dell'articolo, la quantità disponibile e la quantità ordinata. Per ogni progetto sono invece registrati il numero del progetto, il nome del progetto e la descrizione del progetto. Laddove un progetto faccia uso di un determinato articolo, viene anche registrata la quantità impegnata di quell'articolo da utilizzare per quello specifico progetto. Si supponga che il sistema richieda all'utente o al progettista dei file di definire i dati in termini di strutture gerarchiche. Può quindi essere adottata una qualunque delle cinque strutture gerarchiche seguenti.

Struttura 1. Progetti subordinati agli articoli

File	*Segmento*	*Campi*
F	ARTICOLO	Numero articolo
		Nome articolo
		Descrizione articolo
		Quantità disponibile
		Quantità ordinata
	PROGETTO	Numero progetto
		Nome progetto
		Descrizione progetto
		Quantità impegnata

Struttura 2. Articoli subordinati ai Progetti

File	*Segmento*	*Campi*
F	PROGETTO	Numero progetto
		Nome progetto
		Descrizione progetto
	ARTICOLO	Numero articolo
		Nome articolo
		Descrizione articolo
		Quantità disponibile
		Quantità ordinata
		Quantità impegnata

Struttura 3. Progetti ed Articoli alla pari
Relazione di impegno subordinata ai progetti

File	*Segmento*	*Campi*
F	ARTICOLO	Numero articolo
		Nome articolo
		Descrizione articolo
		Quantità disponibile
		Quantità ordinata
G	PROGETTO	Numero progetto
		Nome progetto
		Descrizione progetto
	ARTICOLO	Numero articolo
		Quantità impegnata

Struttura 4. Progetti ed Articoli alla pari
Relazione di impegno subordinata agli articoli

File	*Segmento*	*Campi*
F	ARTICOLO	Numero articolo
		Nome articolo
		Descrizione articolo
		Quantità disponibile
		Quantità ordinata
	PROGETTO	Numero progetto
		Quantità impegnata
G	PROGETTO	Numero progetto
		Nome progetto
		Descrizione progetto

Struttura 5. Progetti ed Articoli alla pari
Relazione di impegno alla pari

File	*Segmento*	*Campi*
F	ARTICOLO	Numero articolo
		Nome articolo
		Descrizione articolo
		Quantità disponibile
		Quantità ordinata
G	PROGETTO	Numero progetto
		Nome progetto
		Descrizione progetto
H	IMPEGNO	Numero articolo
		Numero progetto
		Quantità impegnata

Ora, consideriamo il problema di stampare il numero dell'articolo, il nome dell'articolo e la quantità impegnata per ogni articolo utilizzato nel progetto il cui nome progetto è "alpha". Le osservazioni seguenti sono indipendenti da quale sistema gerarchico si scelga per affrontare questo problema. Se un programma P è sviluppato per far fronte a questo problema utilizzando una delle cinque strutture summenzionate – cioè, P non effettua alcun test per determinare qual è la struttura in uso – allora P andrà in errore su almeno tre delle rimanenti strutture. Più specificamente, se P funziona con la struttura 5, andrà in errore con tutte le altre; se P funziona con le strutture 3 o 4, andrà in errore con almeno le strutture 1, 2 e 5; se P funziona con le 1 o 2, andrà in errore almeno con la 3, 4 e 5. Il motivo è semplice in tutti i casi. In assenza di un test per determinare qual è la struttura in uso, P va in errore perché tenterà di accedere ad un file inesistente (i sistemi esistenti trattano questo tentativo di accesso come un errore) oppure non accederà ad un file che contiene informazioni necessarie. Il lettore che non è convinto può provare a sviluppare un programma d'esempio per risolvere questo semplice problema.

Poiché, in generale, è scomodo sviluppare programmi applicativi che fanno un test tra tutte le strutture gerarchiche possibili in quel sistema, questi programmi sono destinati ad andare in errore se diventa necessario cambiare la struttura.

I sistemi che forniscono all'utente un modello reticolare dei dati devono affrontare difficoltà simili. Sia nel caso gerarchico che reticolare, l'utente (o il suo programma) è obbligato a sfruttare una collezione di percorsi d'accesso ai dati. Non gli interessa se questi percorsi sono in stretta corrispondenza con i percorsi definiti dai puntatori nella rappresentazione fisica – in IDS la corrispondenza è molto semplice, in TDMS è esattamente il contrario. La conseguenza, a prescindere dalla rappresentazione fisica, è che le attività da terminale ed i programmi diventano dipendenti dal fatto che i percorsi d'accesso utilizzati continuino ad esistere.

Una soluzione è adottare la convenzione che una volta che è stato definito un percorso d'accesso utente, esso non può essere considerato obsoleto fino a quando non diventa obsoleto il programma applicativo che lo utilizza. Questa convenzione non è praticamente applicabile visto che il numero dei percorsi d'accesso nel modello complessivo di tutti gli utenti di una banca dati potrebbe diventare eccessivamente ampio.

1.3. UNA VISTA RELAZIONALE DEI DATI

Il termine *relazione* è utilizzato qui nel suo significato comunemente accettato in matematica. Dati gli insiemi S_1, S_2, … , S_n (non necessariamente distinti), R è una relazione su questi *n* insiemi se è un insieme di n-uple ognuna delle quali ha il suo primo elemento preso da S_1, il secondo preso da S_2, e così via[1]. Ci riferiremo a S_j come il j-esimo *dominio* di R. Definita in questo modo, si dice che R ha *grado n*. Relazioni di grado 1 sono spesso chiamate *unarie*, di grado 2 *binarie*, di grado 3 *ternarie* e di grado *n n-arie*.

Per ragioni espositive, faremo un frequente utilizzo della rappresentazione vettoriale delle relazioni, ma bisogna ricordare che questo particolare tipo di rappresentazione non è una parte essenziale della vista relazionale che stiamo esponendo. Un vettore che rappresenta una relazione *n-aria* R ha le seguenti proprietà:

(1) Ogni riga rappresenta una n-upla di R.
(2) L'ordinamento delle righe è immateriale
(3) Tutte le righe sono distinte
(4) L'ordinamento delle colonne non è significativo – esso corrisponde all'ordinamento S_1, S_2, … , S_n dei domini su cui R è definita (si vedano, comunque, le note che seguono a proposito di relazioni a domini ordinati e relazioni a domini non ordinati).
(5) Il significato di ogni colonna è parzialmente espresso etichettandola con il nome del dominio corrispondente.

L'esempio in Figura 1 illustra una relazione di grado 4, chiamata *fornitura*, che rappresenta le spedizioni in corso di articoli da specifici fornitori per specifici progetti in specifiche quantità.

fornitura	*(fornitore*	*articolo*	*progetto*	*quantità)*
	1	2	5	17
	1	3	5	23
	2	3	7	9
	2	7	5	4
	4	1	1	12

FIG. 1. Una relazione di grado 4.

Ci si potrebbe chiedere: Se le colonne sono etichettate col nome del dominio corrispondente, perché l'ordinamento delle colonne dovrebbe essere significativo? Come mostra l'esempio in Figura 2, due colonne potrebbero avere identiche testate (il che indica identici domini) ma possedere significati distinti rispetto alla relazione. La relazione raffigurata si chiama *componente*. È una relazione ternaria, i cui primi due domini si chiamano *articolo* ed il terzo dominio si chiama *quantità*. Il significato di *componente (x, y, z)* è che l'articolo *x* è un componente diretto (o sub-assemblato) dell'articolo *y*, e *z* unità di articolo *x* sono necessarie per assemblare un'unità di articolo *y*. E' una relazione che gioca un ruolo critico nel problema della distinta base.

[1] Più brevemente, R è un sottoinsieme del prodotto cartesiano $S_1 \times S_2 \times \ldots \times S_n$.

componente	*(articolo*	*articolo*	*quantità)*
	1	5	9
	2	5	7
	3	5	2
	2	6	12
	3	6	3
	4	7	1
	6	7	1

FIG. 2. Una relazione con due domini identici.

È un fatto degno di nota che molti sistemi informatici esistenti (principalmente quelli basati su file gerarchici) non riescano a fornire una rappresentazione dei dati per le relazioni due o più domini identici. La versione attuale di IMS/360 [5] è un esempio di questo tipo di sistemi.

La totalità dei dati in una banca dati può essere vista come una collezione di relazioni (di grado assortito) che cambiano nel tempo. Col passar del tempo, ogni relazione n-aria può essere soggetta all'inserimento di n-uple addizionali, alla cancellazione di quelle esistenti, ed alla modifica di componenti di ciascuna delle n-uple esistenti.

In molte banche dati commerciali, governative e scientifiche, comunque, alcune relazioni sono di grado piuttosto alto (il grado 30 non è affatto infrequente). Normalmente gli utenti non devono essere gravati dell'onere di ricordare l'ordine dei di ogni relazione (ad esempio, l'ordinamento dei domini *fornitore*, poi *articolo*, poi *progetto*, poi *quantità* nella relazione *fornitura*). Per questo la nostra proposta è che gli utenti debbano avere a che fare non con le relazioni che sono ordinate in base ai domini, ma con le *correlazioni* che sono la loro controparte non ordinata in base ai domini[2]. Per far questo i domini devono essere univocamente identificabili al meno all'interno di ogni singola relazione, senza utilizzare la posizione. Quindi, quando ci sono due o più domini identici, sarà richiesto in tutti i casi che il nome del dominio sia qualificato utilizzando un *nome di ruolo* distintivo che serva ad identificare il ruolo giocato da quel dominio in quella specifica relazione. Per esempio, nella relazione *componente* di Figura 2, il primo dominio *articolo* può essere qualificato con il nome di ruolo *sub*, il secondo con *super*, in modo che gli utenti possano trattare la relazione *componente* ed i suoi domini – *sub.articolo*, *super.articolo*, *quantità* – senza alcun riguardo per l'ordine dei domini.

Riassumendo, si propone che gli utenti interagiscano con un modello relazionale dei dati consistente in una collezione variabile nel tempo di correlazioni (piuttosto che relazioni). Ogni utente non ha bisogno di sapere nient'altro di ogni correlazione che il suo nome ed il nome dei suoi domini (qualificati con un ruolo laddove necessario)[3]. Queste informazioni potrebbero essere offerte dal sistema (previi controlli di sicurezza e riservatezza) in forma di menù a richiesta dell'utente.

Ci sono solitamente molti modi alternativi in cui un modello relazionale può essere definito per ogni banca dati. Al fine di discutere una modalità preferenziale (o forma normale), dobbiamo preliminarmente introdurre alcuni concetti ulteriori (dominio attivo, chiave primaria, chiave esterna, dominio non-semplice) e stabilire alcuni collegamenti con la terminologia correntemente in uso nell'ambito della programmazione dei sistemi informatici. Nel resto di quest'articolo, non faremo più distinzione tra relazione e correlazione ad eccezione dei casi in cui esplicitare la differenza offra un vantaggio espositivo.

Consideriamo un esempio di banca dati che includa relazioni che trattano articoli, progetti e fornitori. Una relazione chiamata *articolo* è definita sui seguenti domini:

(1) numero dell'articolo
(2) nome dell'articolo
(3) colore dell'articolo
(4) peso dell'articolo
(5) quantità disponibile
(6) quantità ordinata

e magari anche altri domini. Ognuno di questi domini è , praticamente, un insieme di valori, alcuni dei quali possono essere presenti nella banca dati in ogni determinato istante. Mentre è plausibile che, in qualche istante particolare, tutti i colori degli articoli sono presenti, è improbabile che lo siano anche tutti i possibili pesi, nomi e numeri. Definiremo l'insieme dei valori presenti in un determinato istante *dominio attivo* a quell'istante.

Normalmente, un dominio (o una combinazione di domini) di una data relazione ha valori che identificano univocamente ogni elemento (*n-upla*) di quella relazione. Questo dominio (o combinazione) è detto una *chiave primaria*. Nell'esempio suddetto, il numero dell'articolo potrebbe essere una chiava primaria, mentre il colore dell'articolo non lo potrebbe essere. Una chiave primaria è *non-ridondante* se è o un dominio semplice (non una combinazione) oppure una combinazione in cui nessuno dei domini semplici che la compongono sia superfluo per

[2] In termini matematici, una correlazione è la classe d'equivalenza di tutte le relazioni che sono equivalenti a meno di una permutazione dei domini (vedi Sezione 2.1.1).

[3] Naturalmente, quando bisogna caricare o leggere dati da un sistema informatico, l'utente normalmente riesce a fare un uso molto più produttivo dei dati se ne capisce il significato.

l'individuazione univoca degli elementi. Una relazione può possedere più di una chiave primaria non-ridondante. Questo caso si verificherebbe nell'esempio se ad articoli differenti fossero associati sempre nomi differenti. Quando una relazione ha due o più chiavi primarie non-ridondanti, una di queste è scelta arbitrariamente e definita *la* chiave primaria di quella relazione.

Un requisito comune è che gli elementi di una relazione abbiano riferimenti incrociati con altri elementi della stessa relazione o con elementi di altre relazioni. Le chiavi forniscono un modo orientato all'utente (ma non l'unico modo) di esprimere tali riferimenti incrociati. Definiremo un dominio (o una combinazione di domini) della relazione R una *chiave esterna* se essa non è la chiave primaria della tabella ma i suoi elementi sono valori della chiave primaria di una qualche relazione S (non escludendo la possibilità che R ed S siano identiche). Nella relazione *fornitura* di Figura 1, la combinazione di *fornitore*, *articolo*, *progetto* è la chiave primaria, mentre ognuno di questi tre domini separatamente è una chiave esterna.

Nei lavori precedenti c'è stata una forte tendenza a trattare i dati in una banca dati come formati da due parti, una parte consistente delle descrizioni delle entità (ad esempio, le descrizioni dei fornitori) e l'altra parte consistente nelle relazioni tra le varie entità o tipi di entità (per esempio la relazione di *fornitura*). Questa distinzione è difficile da gestire laddove si possono avere chiavi esterne dappertutto ed in qualunque relazione. Nel modello relazionale dell'utente non si scorge nessun vantaggio nel continuare a fare questa distinzione (ci può essere qualche vantaggio, ad ogni modo, quando si applica il concetto relazionale alla rappresentazione fisica dell'insieme delle correlazioni dell'utente).

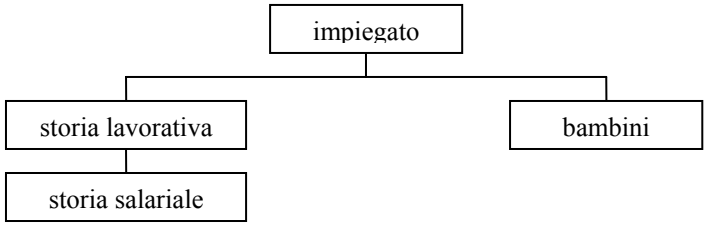
Fin qui abbiamo discusso esempi di relazioni che sono definite su domini semplici – domini i cui elementi sono valori atomici (non decomponibili). Anche i valori non atomici possono essere discussi nella logica relazionale. Quindi, alcuni domini possono avere relazioni come elementi. Queste relazioni possono, a loro volta, essere definite su domini non semplici, e così via. Per esempio, uno dei domini su cui la relazione *impiegato* è definita può essere la *storia salariale*. Un elemento della storia salariale è una relazione binaria definita sul dominio *data* e sul dominio *salario*. Il dominio *storia salariale* è un insieme di queste relazioni binarie. Ad ogni preciso istante nel tempo ci sono tante istanze della relazione *storia salariale* nella banca dati per quanti sono gli impiegati. Viceversa c'è una sola istanza della relazione *impiegato*.

I termini "attributo" e "gruppo ripetitivo" nell'accezione oggi comunemente utilizzata per le basi di dati sono pressoché analoghi rispettivamente a "dominio semplice" e "dominio non semplice". Molta della confusione nella terminologia correntemente utilizzata è dovuta alla mancata distinzione tra tipo ed istanza (come in "record") e tra componenti e modello utente dei dati da una parte e le loro controparti nella rappresentazione fisica dall'altra (di nuovo, citiamo "record" come esempio).

1.4. FORMA NORMALE

Una relazioni i cui domini sono tutti semplici può essere rappresentata fisicamente da un vettore bidimensionale a colonne omogenee del tipo discusso prima. Strutture dati più complesse sono necessarie quando nella relazione sono presenti uno o più domini non semplici. Per questo motivo (ed altri che citeremo più avanti) vale la pena di investigare la possibilità di eliminare domini non semplici[4]. Esiste, in pratica, una semplicissima procedura d'eliminazione che chiameremo normalizzazione.

Consideriamo, ad esempio, la collezione delle relazioni mostrate in Figura 3(a). *Storia lavorativa* e *bambini* sono domini non semplici della relazione *impiegato*. *Storia salariale* è un dominio non semplice della relazione *storia lavorativa*. L'albero di Figura 3(a) mostra proprio queste interrelazioni tra i domini non semplici.



impiegato (*matricola*, nome, dataNascita, storiaLavorativa, bambini)
storiaLavorativa (*dataAssunzione*, titolo, storiaSalariale)
storiaSalariale (*dataSalario*, salario)
bambini (*nomeBambino*, annoNasciata)

FIG. 3(a). Insieme non normalizzato

Impiegato′ (*matricola*, nome, dataNascita)
storiaLavorativa′ (*matricola*, *dataAssunzione*, titolo)
storiaSalariale′ (*matricola*, *dataAssunzione*, *dataSalario*, salario)
bambini′ (*matricola*, *nomeBambino*, annoNasciata)

FIG. 3(b). Insieme normalizzato

[4] M. E. Sanko di IBM, San Jose, ha determinato indipendentemente l'opportunità di eliminare i domini non semplici.

La normalizzazione procede come segue. Partendo dalla relazione posta alla radice dell'albero, si prende la sua chiave primaria e si espande ognuna delle relazioni immediatamente subordinate aggiungendo il dominio (o la combinazione di domini) della chiave primaria. La chiave primaria di ogni relazione che è stata espansa sarà composta dalla chiave primaria com'era prima dell'espansione più il dominio (la combinazione di domini) appena aggiunto. Ora si possono eliminare dalla relazione superiore tutti i domini non semplici, rimuovere la radice dell'albero e continuare con tutti i nodi rimanenti.

Il risultato della normalizzazione della collezione delle relazioni di Figura 3(a) è la collezione in Figura 3(b). La chiave primaria di ogni relazione è in corsivo per evidenziare come queste chiavi vengono espanse dalla normalizzazione.

Perché la normalizzazione come appena descritta sia applicabile è necessario che la collezione non normalizzata delle relazioni soddisfi le seguenti condizioni:

(1) Il grafo delle interrelazioni dei domini non semplici deve essere una collezione di alberi.

(2) Nessuna chiave primaria deve essere basata su un dominio non semplice.

Chi scrive non è a conoscenza di alcuna applicazione che richieda un indebolimento di queste condizioni. Ulteriori operazioni di normalizzazione sono possibili. Queste non sono discusse in questo articolo.

La semplicità della rappresentazione vettoriale, che diventa fattibile quando tutte le relazioni sono state ricondotte alla forma normale, non solo è vantaggiosa per la memorizzazione fisica ma anche per comunicare grandi volumi di dati tra sistemi che utilizzano rappresentazioni molto diverse dei dati. Il formato di comunicazione potrebbe essere una versione opportunamente compressa della rappresentazione vettoriale ed avrebbe i seguenti vantaggi:

(1) Sarebbe privo di puntatori (basati su indirizzi o spostamenti).

(2) Eviterebbe tutte le dipendenze dagli schemi di indirizzamento hash.

(3) Non conterrebbe indici né liste d'ordinamento.

Se il modello relazionale utente è in forma normale, i nomi delle singole informazioni della banca dati possono assumere una forma semplificata. Un nome generico assumerà una forma del tipo

R(g).r.d

dove R è un nome di relazione; g è un identificativo di generazione (opzionale); r è un nome di ruolo (opzionale); d è un nome di dominio. Poiché g è necessario solo quando esistono, o sappiamo che esisteranno, molteplici generazioni della medesima relazione, ed r è necessario solo quando la relazione R ha due o più domini di nome d, la forma semplificata R.d sarà normalmente sufficiente.

1.5. ALCUNI ASPETTI LINGUISTICI

Adottare un modello relazionale dei dati, come appena descritto, consente lo sviluppo di un sublinguaggio universale per i dati basato sul calcolo dei predicati applicato. Un calcolo dei predicati di primo ordine è sufficiente se la collezione delle relazioni è in forma normale. Questo linguaggio sarebbe l'unità di misura delle capacità linguistiche di tutti i linguaggi dati e sarebbe esso stesso un forte candidato per essere integrato (con appropriate modifiche di sintassi) in una varietà di linguaggi (di programmazione, di linea di comando, specifici di ambiti particolari). Sebbene non sia tra gli obiettivi di questo documento la descrizione dettagliata di un tale linguaggio, le caratteristiche salienti dovrebbero essere le seguenti.

Denotiamo il sublinguaggio dati con R ed il linguaggio in cui si integra con H. R consente la dichiarazione di relazioni e dei loro domini. Ogni dichiarazione di una relazione identifica la chiave primaria della relazione. Le relazioni dichiarate sono aggiunte al catalogo di sistema, utilizzabile da qualunque utente del sistema che disponga delle opportune autorizzazioni. H consente di effettuare dichiarazioni aggiuntive che indichino, magari in modalità meno permanente, come le relazioni debbano essere rappresentate a livello fisico. R consente formulare richieste di estrazione di qualunque sottoinsieme di dati dalla banca dati. Le possibili azioni su questi dati estratti sono soggette a controlli di sicurezza.

L'universalità del sublinguaggio risiede nella sua capacità descrittiva (non nella sua capacità elaborativa). In una grande banca dati ogni sottoinsieme dei dati ha un alto numero di possibili (e significative) descrizioni, anche quando si parte dal presupposto (come stiamo facendo) che ci sia solo un insieme finito di funzioni a cui il sistema ha accesso da utilizzare per qualificare i dati da estrarre. Quindi, la classe delle espressioni di qualificazione che possono essere utilizzate quando si specifica un insieme devono avere la capacità descrittiva della classe delle formule ben-formate del calcolo dei predicati applicato. È ben noto che per preservare questa capacità descrittiva non è necessario esprimere (in qualunque sintassi si scelga) ogni formula del calcolo dei predicati selezionato. Per esempio, sono sufficienti solo le formule nella forma normale prenex[9].

Le funzioni aritmetiche possono essere necessarie nella qualificazione o in altre parti dell'istruzione di ricerca. Queste funzioni possono essere definite in H ed invocate in R.

Un insieme specificato in questo modo può essere estratto in sola lettura oppure può essere riservato per apportare modifiche. Gli inserimenti consistono nell'aggiunta di nuovi elementi alle relazioni senza alcuna attenzione all'ordinamento che può essere presente nella rappresentazione fisica. Le cancellazioni valide per l'intera comunità degli utenti (e non per singoli utenti o gruppi) consistono nella rimozione di elementi dalle relazioni dichiarate. Alcune cancellazioni ed aggiornamenti possono essere automaticamente causate da altre, se in R sono state dichiarate delle regole di cancellazione ed aggiornamento tra relazioni dipendenti.

Un effetto importante che la vista dati adottata ha sul linguaggio di ricerca utilizzato è sulla nomenclatura dei singoli dati e degli insiemi. Alcuni aspetti di questo fatto sono stati discussi nella sezione precedente. Con la consueta vista reticolare, gli utenti sono spesso costretti a coniare ed utilizzare più nomi di relazioni di quanti siano effettivamente necessari, poiché i nomi sono associati ai percorsi (o alle tipologie di percorso) piuttosto che alle relazioni.

Una volta che l'utente è a conoscenza del fatto che una relazione è stata creata, egli si aspetterà di poterla sfruttare[5] utilizzando una qualunque combinazione dei suoi argomenti come "conosciuti" e gli altri come "sconosciuti", visto che l'informazione è lì. Questa è una funzionalità (mancante in molti sistemi informatici attuali) che definiremo (logicamente) *sfruttamento simmetrico* delle relazioni. Naturalmente non bisogna attendersi una simmetria anche nelle performance.

Per supportare lo sfruttamento simmetrico di una singola relazione binaria, sono necessari due percorsi diretti. Per una relazione di grado *n*, il numero dei percorsi da nominare e controllare è *n* fattoriale.

Nuovamente, se si adotta una vista relazionale in cui ogni relazione n-aria (n > 2) deve essere espressa dall'utente come un'espressione innestata tra relazioni esclusivamente binarie (si veda il sistema LEAP di Feldman[10], per esempio) allora devono essere coniati 2n-1 nomi invece di soli n + 1 che si hanno con la notazione n-aria descritta nella sezione 1.2. Per esempio, la relazione quaternaria *fornitura* di Figura 1, che comporta 5 nomi in notazione n-aria, sarebbe rappresentata, in notazione binaria innestata, nella forma

P (*fornitore*, Q (*articolo*, R (*progetto*,*quantità*)))

e dunque richiederebbe 7 nomi.

Un ulteriore svantaggio di questo tipo di espressione è il fatto che è asimmetrica. Sebbene quest'asimmetria non impedisca lo sfruttamento simmetrico, essa certamente rende alcune interrogazioni molto scomode da esprimere (si consideri, ad esempio, una query che intenda estrarre gli articoli e le quantità associati ad un dato progetto via Q ed R).

1.6. Relazioni esprimibili, Denominate e Memorizzate

Associate con una banca dati troviamo due collezioni di relazioni: l'*insieme delle relazioni denominate* e l'*insieme delle relazioni esprimibili*. L'insieme delle denominate è la collezione di tutte quelle relazioni che la comunità degli utenti può identificare semplicemente per mezzo di un nome (o identificatore). Una relazione R diviene membra dell'insieme delle denominate quanto un opportuno utente autorizzato dichiara R; essa esce dall'insieme quando un opportuno utente autorizzato elimina la dichiarazione di R.

L'insieme delle esprimibili è la collezione totale delle relazioni che possono essere determinate utilizzando espressioni del linguaggio dati. Queste espressioni sono costruite utilizzando i nomi delle relazioni denominate; i nomi delle generazioni, ruoli e domini; i connettori logici; i quantificatori del calcolo dei predicati[6]; ed alcune relazioni simboliche predefinite come =, >. Le relazioni denominate sono un sottoinsieme delle esprimibili, normalmente un piccolissimo sottoinsieme.

Poiché alcune relazioni denominate possono essere combinazioni indipendenti dal tempo di altre dello stesso insieme, può essere utile associare all'insieme delle denominate una collezione di istruzioni che definisce queste regole invarianti nel tempo. Rinvieremo ulteriori approfondimenti su questo tema a quando avremo introdotto alcune operazioni sulle relazioni (si veda Sezione 2).

Uno dei maggiori problemi che deve affrontare il progettista di un sistema per la gestione dei dati che sta supportando il modello relazionale è la determinazione della classe delle rappresentazioni fisiche da supportare. Idealmente, la varietà delle rappresentazioni ammesse dovrebbe essere sufficiente a coprire la gamma dei requisiti di performance di tutte le installazioni. Una varietà così ampia causa un eccesso di memorizzazione ed una continua reinterpretazione delle descrizioni per le strutture correntemente definite.

Per ogni data classe di rappresentazioni fisiche i sistemi di gestione dei dati devono fornire un modo per tradurre le richieste utente espresse nel linguaggio dati del modello relazionale nelle corrispondenti – ed efficienti – azioni sulla rappresentazione fisica effettiva. Per un linguaggio dati di alto livello questo rappresenta un grosso problema di cui tenere conto in fase di progettazione. Cionondimeno, è un problema che deve essere risolto – quando molti utenti accedono contemporaneamente ad una grande banca dati, la responsabilità di fornire risposte efficienti e la capacità produttiva si spostano dal singolo utente al sistema.

2. Ridondanza e Consistenza

2.1. Operazioni sulle Relazioni

Poiché le relazioni sono insiemi, tutte le operazioni insiemistiche consuete sono applicabili ad esse. Cionondimeno, il risultato potrebbe non essere una relazione; ad esempio, l'unione di una relazione binaria ed una relazione ternaria non è una relazione.

---

[5] Lo sfruttamento di una relazione include ricerca, aggiornamento e cancellazione.

[6] Poiché ogni relazione in una banca dati reale è un insieme finito ad ogni istante di tempo, i quantificatori esistenziali ed universali possono essere espressi in termini di una funzione che conta il numero degli elementi in ogni insieme finito.

Le operazioni discusse nel seguito sono specifiche per le relazioni. Le loro applicazioni principali sono nel campo dei sistemi non inferenziali – sistemi che non forniscono servizio di inferenza logica – sebbene la loro applicabilità non cessi necessariamente quando questi servizi vengono aggiunti.

La maggior parte degli utenti non saranno interessati direttamente a queste operazioni. I progettisti di sistemi informatici e persone demandate al controllo della banca dati dovrebbero, comunque, averne una discreta conoscenza.

2.1.1. *Permutazione.* Una relazione binaria ha una rappresentazione vettoriale con due colonne. Scambiando queste due colonne si ottiene la relazione inversa. Più in generale, se si applica una permutazione alle colonne di una relazione n-aria, la relazione risultante è detta essere una *permutazione* della relazione data. Esistono, ad esempio, 4! = 24 permutazioni della relazione *fornitura* in Figura 1, se includiamo la permutazione identica che lascia immutato l'ordinamento delle colonne.

Poiché il modello relazionale utente consiste di una collezione di correlazioni (relazioni a domini non ordinati), la permutazione non ha alcun effetto sul singolo modello. È però rilevante rispetto alla rappresentazione fisica del modello. In un sistema che mette a disposizione l'esplorazione simmetrica delle relazioni, l'insieme delle ricerche effettuabili su una relazione definita è identico all'insieme delle ricerche eseguibili su ogni sua permutazione. Sebbene non sia logicamente necessario memorizzare sia una relazione che qualcuna delle sue permutazioni, ciò potrebbe essere consigliabile per ragioni di performance.

2.1.2. *Proiezione.* Si supponga adesso di selezionare alcune colonne di una relazione (ignorando le rimanenti) e successivamente di rimuovere dal vettore dei risultati tutte le righe duplicate. Il vettore risultante rappresenta una relazione che si dice essere una *proiezione* della relazione data.

Per ottenere tutte le permutazioni, proiezioni o combinazioni di queste due operazioni su una relazione si utilizza un operatore di selezione π. Quindi, se L è una lista di k indici[7] $L = i_1, i_2, \ldots, i_k$ e R è una relazione n-aria ($n \geq k$), allora $\pi_L(R)$ è la relazione k-aria la cui j-esima colonna è la colonna i_j di R ($j = 1,2, \ldots, k$) ad eccezione del fatto che le righe duplicate vengono rimosse. Si consideri la relazione *fornitura* di Figura 1. Una proiezione permutata di questa relazione è mostrata in Figura 4. Si noti che, in questo caso particolare, la proiezione ha meno n-uple della relazione da cui è derivata.

2.1.3. *Join.* Si supponga che siano date due relazioni binarie, le quali hanno qualche dominio in comune. In quali casi possiamo combinare queste relazioni per formare una relazione ternaria che preservi tutte le informazioni delle relazioni di partenza?

L'esempio in Figura 5 mostra due relazioni R, S che sono combinabili senza perdita di informazioni, mentre Figura 6 mostra la join di R con S. Una relazione binaria R è *combinabile* con una relazione binaria S se esiste una relazione ternaria U tale che $\pi_{12}(U) = R$ e $\pi_{23}(U) = S$. Ciascuna di tali relazioni ternarie è detta *join* di R con S. Se R, S sono relazioni binarie tali che $\pi_2(R) = \pi_1(S)$, allora R è combinabile con S. Una join che esiste sempre è la join naturale di R con S definita da

$$R*S = \{(a,b,c):R(a,b) \Lambda S(b,c)\}$$

dove R(a,b) ha il valore *vero* se (a,b) è un membro di R ed allo stesso modo S(b,c). È immediato che

$$\pi_{12}(R*S) = R$$

e

$$\pi_{23}(R*S) = S.$$

Si noti che la join mostrata in Figura 6 è la join naturale di R con S di Figura 5. Un'altra join è mostrata in Figura 7.

Π31 (fornitura)	*(progetto*	*Fornitore)*
	5	1
	5	2
	1	4
	7	2

FIG. 4. Una proiezione permutata della relazione in Figura 1.

R	*(fornitore*	*Articolo)*	*S*	*(articolo*	*progetto)*
	1	1		1	1
	2	1		1	2
	2	2		2	1

FIG. 5. Due relazioni combinabili.

[7] Nel lavorare con le correlazioni, utilizzeremo i nomi dei domini (qualificati con i ruoli laddove necessario) invece delle posizioni degli stessi.

*R*S*	*(fornitore*	*articolo*	*Progetto)*
	1	1	1
	1	1	2
	2	1	1
	2	1	2
	2	2	1

FIG. 6. La join naturale di R con S (da Figura 5).

U	*(fornitore*	*articolo*	*Progetto)*
	1	1	2
	2	1	1
	2	2	1

FIG. 7. Un'altra join di R con S (da Figura 5).

Da una lettura di queste relazioni risulta evidente che un elemento (elemento 1) del dominio *articolo* (il dominio su cui si effettua la join) ha la proprietà di possedere più di un padre sia nella relazione R che in S. È questo elemento che causa l'aumento del numero delle n-uple nelle join. Un elemento del genere è detto *punto di ambiguità* rispetto alla join di R con S.

Se almeno una tra $\pi_{21}(R)$ ed S è una funzione[8], non si può verificare alcun punto di ambiguità durante la join di R con S. Si noti che la reiterata puntualizzazione "di R con S" è necessaria perché S potrebbe essere combinabile con R (così come R con S), ma quest'altra join meriterebbe tutt'altre considerazioni rispetto a quelle fatte per la join di R con S. In Figura 5, nessuna delle relazioni R, $\pi_{21}(R)$, S, $\pi_{21}(S)$ è una funzione.

Le ambiguità nella join di R con S possono essere talvolta risolte per mezzo di altre relazioni. Si supponga che sia data, o si possa derivare indipendentemente da R ed S, una relazione T sui domini *progetto* e *fornitore* con le seguenti proprietà:

$$
\begin{aligned}
&(1) \quad \pi_1(T) = \pi_2(S),\\
&(2) \quad \pi_2(T) = \pi_1(R),\\
&(3) \quad T(j,s) \rightarrow \exists p(R(s,p) \wedge S(p,j)),\\
&(4) \quad R(s,p) \rightarrow \exists j(S(p,j) \wedge T(j,s)),\\
&(5) \quad S(p,j) \rightarrow \exists s(T(j,s) \wedge R(s,p)),
\end{aligned}
$$

allora è possibile formare una join a tre direzioni tra R, S e T; cioè una relazione ternaria tale che

$$\pi_{12}(U) = R, \qquad \pi_{23}(U) = S, \qquad \pi_{31}(U) = T.$$

Una join di questo tipo sarà chiamata *3-join ciclica* per distinguerla da una *3-join lineare* che sarebbe una relazione quaternaria V tale che

$$\pi_{12}(V) = R, \qquad \pi_{23}(V) = S, \qquad \pi_{34}(V) = T.$$

È possibile che esista più di una 3-join ciclica (si vedano le figure 8, 9 per un esempio) solo se si verificano regole molto più restrittive di quelle che servono per avere una pluralità di 2-join.

R	*(s*	*p)*	*S*	*(p*	*j)*	*T*	*(j*	*s)*
	1	a		a	d		d	1
	2	a		a	e		d	2
	2	b		b	d		e	2
				b	e		e	2

FIG. 8. Relazioni binarie con una pluralità di 3-join cicliche.

U	*(s*	*p*	*j)*	*U'*	*(s*	*p*	*j)*
	1	a	d		1	a	d
	2	a	e		2	a	d
	2	b	d		2	a	e
	2	b	e		2	b	d
					2	b	e

FIG. 9. Due 3-join cicliche delle relazioni in Figura 8.

[8] Una funzione è una relazione binaria uno-a-uno o molti-a-uno ma non uno-a-molti.

In particolare, le relazioni R, S, T devono possedere punti di ambiguità nelle join di R con S (ad esempio il punto x), S con T (punto y) e T con R (punto z), e, inoltre, y deve essere padre di x nella relazione S, z padre di y nella relazione T e x padre di z nella relazione R. Si noti che nella Figura 8 i punti x = a; y = d; z = 2 verificano queste regole.

La 3-join lineare naturale di tre relazioni binarie R, S, T è data da

$$R*S*T = \{(a,b,c,d):R(a,b) \Lambda S(b,c) \Lambda T(c,d)\}$$

Dove le parentesi non sono necessarie sul lato sinistro dell'uguaglianza perché la 2-join naturale è associativa. Per ottenere la controparte ciclica introduciamo l'operatore γ che produce una relazione di grado n-1 da una relazione di grado n appariglandone gli estremi. Quindi, se R è una relazione n-aria ($n \geq 2$), l'appariglamento di R è definito dall'equazione

$$\gamma(R) = \{(a_1, a_2, \ldots, a_{n-1}):R(a_1, a_2, \ldots, a_{n-1}, a_n) \Lambda a_1=a_n\}.$$

Possiamo adesso rappresentare la 3-join ciclica naturale di R, S, T con l'espressione

$$\gamma(R*S*T).$$

L'estensione delle nozioni di 3-join lineare e ciclica e delle loro controparti naturali alle join di *n* relazioni binarie (con $n \geq 3$) è ovvia. Qualcosa però bisogna dire a proposito delle join tra relazioni che non siano necessariamente binarie. Si consideri il caso di due relazioni R (grado r), S (grado s) che devono essere messe in join su p dei loro domini ($p < r$, $p < s$). Per semplicità, che i p domini siano esattamente gli ultimi p dei r domini di R, ed i primi p degli s domini di S. Se non fosse così potremmo sempre applicare una permutazione per metterci in questa situazione. Sia A il prodotto cartesiano dei primi r-p domini di R. Sia B il prodotto cartesiano degli ultimi p domini di R. Sia C il prodotto cartesiano degli ultimi s-p domini di S.

Si può trattare R come se fosse una relazione binaria sui domini A, B. Allo stesso modo S si può considerare una relazione binaria su sui domini B, C. A questo punto le nozioni di 3-join ciclica e naturale sono applicabili. Un approccio simile si può utilizzare per definire le n-join cicliche e lineari di grado arbitrario.

2.1.4. *Composizione.* Il lettore probabilmente ha familiarità con la nozione di composizione applicata alle funzioni. Noi discuteremo una generalizzazione di questo concetto applicandolo dapprima alle relazioni binarie. Le nostre definizione di composizione e componibilità sono strettamente collegate alle definizioni di join e combinabilità date in precedenza.

Si supponga che siano date due relazioni R, S. T è una *composizione* di R con S se esiste una join U di R con S tale che $T = \pi_{12}(U)$. Quindi, due relazioni sono componibili se e solo se sono combinabili. In ogni caso, però, l'esistenza di più di una join di R con S non implica l'esistenza di più di una composizione di R con S.

Parallelamente alla join naturale di R con S c'è la *composizione naturale*[9] di R con S definita da

$$R \cdot S = \pi_{12}(R*S)$$

Prendendo le relazioni R, S da Figura 5, la loro composizione naturale è mostrata in Figura 10 ed un'altra composizione è mostrata in Figura 11 (derivata dalla join mostrata in Figura 7).

R·S	*(progetto*	*fornitore)*
	1	1
	1	2
	2	1
	2	2

FIG. 10. La composizione naturale di R con S (da Figura 5).

T	*(progetto*	*fornitore)*
	1	2
	2	1

FIG. 11. Un'altra composizione di R con S (da Figura 5).

Quando esistono due o più join, il numero delle composizioni distinte è compreso tra uno ed il numero delle join distinte. La Figura 12 mostra un esempio di due relazioni che hanno svariate join ma solo una composizione. Si noti che il punto di ambiguità *c* si perde nella composizione di R con S, a causa delle associazioni non ambigue fatte attraverso i punti *a*, *b*, *d*, *e*.

[9] Altri autori tendono ad ignorare tutte le composizioni oltre alla naturale, e di conseguenza fanno riferimento a questa particolare composizione come *la* composizione – si veda ad esempio la "Topologia Generale" di Kelley.

R	*(fornitore*	*Articolo)*	*S*	*(articolo*	*progetto)*
	1	a		a	g
	1	b		b	f
	1	c		c	f
	2	c		c	g
	2	d		d	g
	2	e		e	f

FIG. 12. Molte join, un'unica composizione.

L'estensione della composizione a coppie di relazioni che non siano necessariamente binarie (e che possono essere di grado differente) segue lo stesso schema mostrato per l'estensione delle join alle stesse relazioni.

Una mancata comprensione della composizione relazionale ha fatto sì che molti progettisti di sistemi cadessero in quella che hanno chiamato *trappola della connessione*. Questa trappola può essere descritta con l'esempio seguente. Si supponga che la descrizione di ogni fornitore sia è legato attraverso puntatori alla descrizione di ogni articolo fornito da quel fornitore, e che ogni descrizione di articolo sia collegata allo stesso modo alla descrizione di ciascun progetto che fa uso di quell'articolo. Si arriverà adesso ad una conclusione che generalmente è errata: vale a dire che se si seguono tutti i possibili percorsi da un fornitore attraverso gli articoli forniti per arrivare ai progetti che fanno uso di quegli articoli, si ottiene un insieme valido di tutti i progetti forniti da quel fornitore. Questa conclusione è corretta solo nel caso molto particolare in cui la relazione finale tra progetti e fornitori è la composizione naturale di due altre relazioni – e dovremmo a questo punto aggiungere la frase "ad ogni istante" visto che questa è normalmente presente quando si discute di tecniche di "inseguimento di percorso".

2.1.5. *Restrizione.* Un sottoinsieme di una relazione è una relazione. Un modo in cui una relazione S può agire su una relazione R per generare un sottoinsieme di R è attraverso l'operazione di *restrizione* di R attraverso S. Quest'operazione è una generalizzazione della restrizione di una funzione ad un sottoinsieme del suo dominio, ed è definita come segue.

Siano L, M liste di indici di pari lunghezza tali che $L = i_1, i_2, \ldots, i_k$, $M = j_1, j_2, \ldots, j_k$ dove $k \leq$ grado di R e $k \leq$ grado di S. Allora la restrizione L, M di R attraverso S denotata $R_L|_M S$ è il sottoinsieme massimale R′ di R tale che

$$\pi_L(R') = \pi_M(S).$$

L'operazione è definita solo se è applicabile l'uguaglianza tra gli elementi di $\pi_{i_h}(R)$ da una parte e $\pi_{j_h}(S)$ dall'altra per tutti gli h = 1, 2, …, k.

Le tre relazioni R, S, R′ di Figura 13 soddisfano l'equazione $R' = R_{(2,3)}|_{(1,2)}S$.

R	*(s*	*p*	*j)*	*S*	*(p*	*j)*	*R′*	*(s*	*p*	*j)*
	1	a	A		a	A		1	a	A
	2	a	A		c	B		2	a	A
	2	a	B		b	B		2	b	B
	2	b	A							
	2	b	B							

FIG. 13. Esempio di restrizione.

Possiamo adesso prendere in considerazione varie applicazioni di queste operazioni sulle relazioni.

2.2. RIDONDANZA

Bisogna distinguere tra la ridondanza nell'insieme delle relazioni denominate e la ridondanza nell'insieme delle rappresentazioni fisiche. Siamo principalmente interessati alla prima. Tanto per cominciare abbiamo bisogno di una definizione precisa di derivabilità per le relazioni.

Si supponga che θ sia una collezione di operazioni sulle relazioni e che ogni operazione abbia la proprietà che applicando i suoi operandi essa genera una relazione unica (quindi la join naturale è applicabile, ma la join no). Una relazione R è θ-derivabile da un insieme S di relazioni se esiste una sequenza di operazioni della collezione θ che, ad ogni istante, genera R dai membri di S. La frase "ad ogni istante" è presente perché stiamo trattando con relazioni dipendenti dal tempo e ci interessa la derivabilità che si conserva per un periodo di tempo significativo. Per l'insieme denominato delle correlazioni in sistemi non inferenziali sembra che una collezione θ_1 adeguata contenga le seguenti operazioni: proiezione, join naturale, apparigliamento e restrizione. La permutazione è irrilevante e non c'è bisogno di includere la composizione naturale perché può essere ottenuta eseguendo prima una join naturale e poi una proiezione. Per l'insieme delle rappresentazioni memorizzate un'adeguata collezione θ_2 di operazioni dovrebbe includere la permutazione ed ulteriori operazioni che consentano di ottenere sottoinsiemi e fusioni di relazioni, ordinamenti e connessioni.

2.2.1. *Ridondanza forte.* Un insieme di relazioni è *fortemente ridondante* se contiene almeno una relazione che possiede una proiezione che è derivabile da altre proiezioni di relazioni nell'insieme. I due esempi seguenti intendono dimostrare perché la ridondanza forte è definita in questo modo, e dimostrarne l'utilizzo pratico. Nel primo esempio la collezione delle relazioni consiste solo della seguente relazione:

impiegato (*numeromatricola*, *nome*, *matricolamanager*, *nomemanager*)

dove numeromatricola è la chiave primaria e matricolamanager è una chiave esterna. Denotiamo il dominio attivo con Δ_t e supponiamo che

$$\Delta_t(\text{matricolamanager}) \subset \Delta_t(\text{numeromatricola})$$

e

$$\Delta_t(\text{nomemanager}) \subset \Delta_t(\text{nome})$$

Ad ogni istante t. In questo caso la ridondanza è ovvia: il dominio *nomemanager* è superfluo Per verificare che si tratta di una ridondanza forte come definito prima osserviamo che

$$\pi_{34}(\text{impiegato}) = \pi_{12}(\text{impiegato}) \,{}_1|_1\, \pi_3(\text{impiegato}).$$

Nel secondo esempio la collezione delle relazioni include una relazione S che descrive i fornitori con chiave primaria f#, una relazione D che descrive i dipartimenti con chiave primaria d#, una relazione J che descrive i progetti con chiave primaria j#, e le seguenti relazioni:

P(f#, d#, …), Q(f#, j#, …), R(d#, j#, …),

Dove in tutti i casi … rappresenta altri domini diversi da f#, d#, j#. Supponiamo che la condizione C sia indipendente dal tempo: il fornitore f rifornisce il dipartimento d (relazione P) se, e solo se, il fornitore f rifornisce qualche progetto j (relazione Q) a cui è assegnato il dipartimento d (relazione R). In questo caso si può scrivere la seguente equazione

$$\pi_{12}(P) = \pi_{12}(Q) \cdot \pi_{21}(R)$$

che mostra una ridondanza forte.

Una ragione importante per l'esistenza della ridondanza forte nell'insieme delle correlazioni denominate è la comodità per l'utente. Un caso particolare del genere è la conservazione di relazioni parzialmente obsolete nell'insieme delle denominate in modo tale che i programmi che le utilizzano possano continuare a funzionare correttamente. Sapere che esistono ridondanze forti nell'insieme delle denominate consente all'amministratore di sistema o di database grande libertà nella scelta delle rappresentazioni fisiche per far fronte in maniera efficiente al traffico. Se le ridondanze forti nell'insieme delle denominate si rispecchiano direttamente in ridondanze forti nell'insieme delle rappresentazioni fisiche (oppure se ulteriori ridondanze forti sono introdotte nell'insieme delle rappresentazione fisiche), allora, parlando in generale, c'è un consumo aggiuntivo di spazio di memorizzazione e tempo d'aggiornamento con una potenziale caduta del tempo di ricerca per alcune query ed un appesantimento del lavoro della CPU.

2.2.2. *Ridondanza debole.* Un secondo tipo di ridondanza può esistere. A differenza della ridondanza forte non è rappresentata da un'equazione. Una collezione di relazioni è debolmente ridondante se contiene una relazione che ha una proiezione che non è derivabile da altri membri ma è in ogni istante una proiezione di qualche join di altre proiezioni di relazioni della collezione.

Possiamo mostrare una ridondanza debole prendendo il secondo esempio (citato prima) di ridondanza forte, ed assumendo che la condizione C non è valida ad ogni istante. Le relazioni $\pi_{12}(P)$, $\pi_{12}(Q)$, $\pi_{21}(R)$ sono relazioni complesse con la possibilità che si verifichino punti d'ambiguità di tanto in tanto quando due di queste vengono messe in join. Sotto questi prerequisiti nessuna di loro è derivabile dalle altre due. Esistono però delle regole tra loro siccome ognuna è una proiezione di qualche join ciclica delle tre. Una delle ridondanze deboli può essere caratterizzata dall'affermazione: ad ogni istante, $\pi_{12}(P)$ è una composizione di $\pi_{12}(Q)$ con $\pi_{21}(R)$. La composizione in questione può essere quella naturale in qualche particolare istante e una non naturale in altri istanti.

Parlando in generale, le ridondanze deboli sono inerenti alle necessità logiche della comunità degli utenti. Esse non possono essere rimosse dall'amministratore di sistema o di database. Se appaiono, esse appaiono sia nell'insieme delle denominate che nell'insieme delle rappresentazioni fisiche.

2.3. CONSISTENZA

Quando l'insieme delle denominate è ridondante in uno dei due sensi, assoceremo a questo insieme una collezione di regole che definiscono tutte le ridondanze che sussistono tra i membri delle relazioni indipendentemente dal tempo. Se il sistema informatico – com'è probabile – non ha tutte le informazioni semantiche di dettaglio sulle tutte le singole relazioni denominate, esso non può dedurre quali siano le ridondanze. Il sistema può, in una finestra temporale, fare tentativi di indurre le ridondanze, ma questi tentativi potrebbero non andare a buon fine.

Data una collezione C di relazioni variabili col tempo, un insieme Z di regole ed un valore istantaneo V di C, definiremo lo stato (C, Z, V) *consistente* o *inconsistente* in funzione del fatto che V soddisfi Z oppure no. Ad esempio date le relazioni R, S, T e la regola "$\pi_{12}(T)$ è una composizione di $\pi_{12}(R)$ con $\pi_{12}(S)$", possiamo controllare in alcuni istanti che i valori memorizzati per R, S, T soddisfino la regola. Un algoritmo per eseguire tale controllo dovrebbe esaminare le prime due colonne di ogni relazione (in qualunque modo essere siano fisicamente rappresentate nel sistema) e determinare se

(1) $\pi_1(T) = \pi_1(R)$,

(2) $\pi_2(T) = \pi_2(S)$,

(3) per ogni coppia di elementi (a,c) della relazione $\pi_{12}(T)$ esiste un elemento b tale che (a,b) sia nella relazione $\pi_{12}(R)$ e (b,c) sia in $\pi_{12}(S)$.

Ci sono problemi pratici (che non discuteremo qui) nel prendere una fotografia istantanea di una collezione di relazioni, alcune delle quali possono essere molto ampie e variabili.

È importante notare che la consistenza come appena definita è una proprietà dello stato transitorio di una banca dati ed è indipendente da come questo stato si sia verificato. Quindi, in particolare, non fa alcuna differenza se l'utente ha generato l'inconsistenza a causa di una omissione oppure di una precisa volontà. Esaminando un semplice esempio si evidenzierà la ragionevolezza di questo (magari non convenzionale) approccio alla consistenza.

Supponiamo che l'insieme delle denominate C includa le relazioni S, J, D, P, Q, R dell'esempio in Sezione 2.2 e che P, Q, R posseggano la ridondanza forte o la debole come descritto in quella Sezione (nel caso particolare che stiamo considerando adesso, non importa quale tipo di ridondanza occorra). Inoltre si supponga che in un certo istante t lo stato della banca dati sia consistente e non contenga alcun progetto *j* tale che il fornitore 2 rifornisca il progetto j e j sia assegnato al dipartimento 5. Di conseguenza non c'è l'elemento (2, 5) in $\pi_{12}(P)$. Adesso, un utente introduce l'elemento (2,5) in $\pi_{12}(P)$ inserendo in P un'appropriata n-upla. Lo stato della banca dati adesso è inconsistente. L'inconsistenza potrebbe essere stata creata da un'omissione se l'input (2, 5) è corretto ed esiste un progetto j tale che il fornitore 2 fornisca j e j sia assegnato al dipartimento 5. In questo caso, è probabile che l'utente intenda nel futuro immediato inserire elementi in Q ed R che avranno l'effetto di introdurre (2, j) in $\pi_{12}(Q)$ e (5, j) in $\pi_{12}(R)$. D'altra parte l'input (5, 2) potrebbe essere un errore. Potrebbe essere il caso in cui l'utente intendeva inserire qualche altro elemento in P – un elemento che avrebbe trasformato uno stato consistente in un altro stato consistente. Il punto è che il sistema normalmente non ha modo per di risolvere questa questione senza ulteriori informazioni esterne (magari fornite dall'utente che ha creato l'inconsistenza).

Ci sono, ovviamente, molti modi possibili in cui un sistema può rilevare le inconsistenze e reagire ad esse. In un approccio il sistema controlla possibili inconsistenze ad ogni inserimento, cancellazione od aggiornamento di chiave. Naturalmente questi controlli rallenteranno le operazioni. Se si è generata un'inconsistenza, i dettagli vengono registrati internamente e se non si pone rimedio in un tempo ragionevole avviene una notifica all'utente oppure ad un responsabile per la sicurezza e l'integrità dei dati. Un altro approccio è effettuare i controlli di consistenza in modalità batch una volta al giorno o meno frequentemente. I dati inseriti che causano inconsistenze ancora presenti nella banca dati al momento del controllo possono essere rintracciati se il sistema ha un registro di tutte le transazioni che ne hanno cambiato lo stato. Quest'ultimo approccio sarebbe certamente superiore se occorressero poche inconsistenze non transitorie.

2.4. RIEPILOGO

In Sezione 1 è stato proposto un modello relazionale dei dati per tutelare gli utenti di sistemi di dati strutturati dai cambiamenti potenzialmente distruttivi della rappresentazione fisica dei dati causati dalla crescita della banca dati e del traffico degli utilizzatori. È stata introdotta una forma normale per le collezioni di relazioni dipendenti dal tempo.

Nella Sezione 2 sono state definite ed applicate al problema di conservare la banca dati in uno stato consistente alcune operazioni sulle relazioni. Questo è destinato a diventare un importante problema pratico visto che molti tipi differenti di dati sono integrati tra loro in banche dati comuni.

Molte questioni sono state sollevate ma non hanno trovato risposta. Ad esempio solo alcune delle più importanti proprietà del sublinguaggio dati sono state menzionate in Sezione 1.4. Non sono stati discussi i dettagli linguistici di tale linguaggio né i problemi implementativi. Ciononostante, il materiale presentato dovrebbe essere adeguato per programmatori esperti di sistemi per visualizzare diversi possibili approcci. Si spera anche che quest'articolo possa contribuire ad aumentare la precisione nel lavoro sui sistemi di dati strutturati.

Ringraziamenti. È stato C. T. Davies di IBM Poughkeepsie a convincere l'autore della necessità di ottenere l'indipendenza dei dati nei futuri sistemi informatici. L'autore desidera ringraziare lui ed anche F. P. Palermo, C. P. Wang, E. B. Altman, e M. E. Senko dei laboratori di ricerca IBM di San Jose per le utili discussioni.

RIFERIMENTI

1. CHILDS, D. L. Feasibility of a set-theoretical data structure -a general structure based on a reconstituted definition of relation. Proc. IFIP Cong., 1968, North Holland Pub. Co., Amsterdam, p. 162-172.
2. LEVEIN, R. E., AND MARON, M. E. A computer system for inference execution and data retrieval. Comm. ACM 10, 11 (Nov. 1967), 715-721.
3. BACHMAN, C. W. Software for random access processing. Datumation (Apr. 1965), 3641.
4. MCGEE, W. C. Generalized file processing. In Annual Review in Automatic Programming 6, 13, Pergamon Press, New York, 1969, pp. 77-149.
5. Information Management System/360, Application Description Manual H20-0524-1. IBM Corp., White Plains, N. Y., July 1968.
6. GIS (Generalized Information System), Application Description Manual H20-0574. IBM Corp., White Plains, N. Y., 1965.
7. BLEIER, R. E. Treating hierarchical data structures in the SDC time-shared data management system (TDMS). Proc. ACM 22nd Nat. Conf., 1967, MD1 Publications, Wayne, Pa., pp. 41-49.
8. IDS Reference Manual GE 625/635, GE Inform. Sys. Div., Pheonix, Ariz., CPB 1093B, Feb. 1968.
9. CHURCH, A. An Introduction to Mathematical Logic I. Princeton U. Press, Princeton, N.J., 1956.
10. FELDMAN, J. A., AND ROVNER, P. D. An Algol-based associative language. Stanford Artificial Intelligence Rep. AI-66, Aug. 1, 1968.

Da Codd ad Oracle 11gR2

Breve storia dei database relazionali

L'articolo di Codd non trovò subito applicazione pratica. Il mondo dell'informatica, come detto nell'introduzione, in quegli anni stava andando decisamente verso lo standard gerarchico/reticolare proposto dal CODASYL e fu in quella direzione che si concentrarono i maggiori sforzi realizzativi per sistemi di database.

Le prime implementazioni di database relazionale furono INGRES, realizzato a Berkeley da Eugene Wong e Michael Stonebrakerda (primi test nel 1973, rilascio ufficiale nel 1979, da questo DB sono nati molti prodotti commerciali tra cui Microsoft SQL Server nel 1985) e System R di IBM, le cui prime versioni non commerciali sono del 1974 ed i primi utilizzi commerciali del 1980. System R si è evoluto con gli anni dapprima in SQL/DS (1982) e poi in Database 2 (DB2, 1983).

Nel 1974 venne pubblicato il fondamentale articolo di Donald Chamberlin e Raymond Boyce (colleghi di Codd in IBM) "SEQUEL: A Structured English Query Language" che riprendeva i concetti introdotti da Codd in merito al linguaggio di query formalizzandolo ulteriormente e gettando le basi per il moderno SQL (ancora oggi pronunciato, appunto *sequel*). Pochi mesi dopo la pubblicazione di quest'articolo Boyce, noto anche per una forma normale che porta il suo nome, si ammalò gravemente e morì.

Nel 1976 venne depositata un'altra pietra miliare, Peter Chen pubblicò infatti l'articolo "The Entity-Relationship Model – Toward a Unified View of Data" in cui si delineava una tecnica, ancora oggi utilizzatissima, per la modellazione concettuale delle basi di dati relazionali.

Nel 1977 Larry Ellison, Bob Miner e Ed Oates, fondarono la Software Development Laboratories con l'intento di realizzare un database relazionale a cui diedero il nome di Oracle. La prima versione del prodotto non venne mai rilasciata. La seconda versione di Oracle vide invece la luce nel 1979, si trattò del primo database relazionale rilasciato sul mercato. L'azienda aveva intanto cambiato nome in Relational Software Inc. che poi nel 1982 si sarebbe definitivamente trasformato in Oracle.

Nel 1981 fu assegnato a Codd il Turing Award con la seguente motivazione "Per i sui contributi fondamentali e continui alla teoria ed alla pratica dei sistemi per la gestione delle basi di dati, con particolare riferimento ai database relazionali"

Nel 1983 Oracle rilasciò la versione 3 del DB che aveva la caratteristica di essere il primo RDBMS portabile su diverse piattaforme hardware essendo scritto in C (le precedenti versioni erano scritte in assembly).

Nel 1985 in due articoli ("Is Your DBMS Really Relational?" e "Does Your DBMS Run By the Rules?", entrambi pubblicati su Computer World), Codd elencò le sue famose dodici regole (in realtà sono tredici ma la regola zero è più che altro una definizione):

Regola 0: Il sistema deve potersi classificare come *relazionale*, *database*, e *sistema di gestione*.
Per potersi classificare come sistema di gestione relazionale dei dati (RDBMS), il sistema deve utilizzare esclusivamente le sue caratteristiche *relazionali* per *gestire* il *database*.
Regola 1: La *regola delle informazioni*:
Tutte le informazioni nel database devono essere rappresentate in un unico modo, come valori in colonne all'interno di righe di tabelle.
Regola 2: *Regola dell'accesso garantito*:
Tutti i dati devono essere accessibili. Questa regola è essenzialmente una riformulazione del requisito fondamentale delle chiavi primarie. Asserisce che ogni singolo valore scalare nel database deve essere individuabile univocamente specificando il nome della tabella che lo contiene, il nome della colonna in cui si trova ed il valore della chiave primaria della riga a cui appartiene.
Regola 3: *Trattamento sistematico dei valori nulli*:
Il DBMS neve consentire che ogni campo resti non valorizzato. Specificamente, esso deve supportare una rappresentazione delle "informazione mancante e informazione non applicabile" che sia sistematica, distinta da tutti i valori regolari (ad esempio, "distinta da zero e da ogni altro numero ", in caso di valori numerici), ed indipendente dal tipo di dato. Queste rappresentazioni devono essere gestite dal DBMS in modo sistematico.
Regola 4: *Catalogo in linea basato sul modello relazionale*:
Il sistema deve supportare un catalogo in linea delle relazioni che sia accessibile agli utenti autorizzati per mezzo del linguaggio di query ordinario. Cioè gli utenti devono accedere al catalogo utilizzando lo stesso linguaggio di query che utilizzano per accedere ai dati nel database.
Regola 5: *Regola del sublinguaggio globale*:
Il sistema deve supportare almeno un linguaggio relazionale che

1. Abbia una sintassi lineare
2. Possa essere utilizzato interattivamente ed all'interno di programmi,
3. Supporti istruzioni per la definizione dei dati (inclusa la definizione delle viste), istruzioni per la manipolazione dei dati (aggiornamenti e ricerche), sicurezza e regole d'integrità referenziale, ed operazioni per il controllo delle transazioni (begin, commit, e rollback).

Regola 6: *Regola dell'aggiornamento delle viste*:
Tutte le viste che teoricamente possono essere aggiornate devono essere rese aggiornabili dal sistema.
Regola 7: *Inserimento, aggiornamento e cancellazione di alto livello*:
Il sistema deve supportare delle istruzioni per eseguire in una singola operazione multipli inserimenti, aggiornamenti o cancellazioni. I risultati delle ricerche in un database relazionale sono insiemi composti da dati provenienti da diverse righe e/o diverse tabelle. Questa regola richiede che inserimenti, aggiornamenti e cancellazioni debbano essere supportati per ogni insieme leggibile dal database piuttosto che solo per una singola riga di una singola tabella.
Regola 8: *Indipendenza fisica dei dati*:
I cambiamenti a livello fisico (come sono memorizzati i dati, se in vettori o liste collegate ecc.) non devono richiedere una modifica delle applicazioni che utilizzano quelle strutture.
Regola 9: *Indipendenza logica dei dati*:
I cambiamenti a livello logico (tabelle, colonne righe e così via) non devono richiedere una modifica delle applicazioni che utilizzano quelle strutture. L'indipendenza logica dei dati è più difficile da ottenere dell'indipendenza fisica.
Regola 10: *Indipendenza dell'integrità*:
Le regole di integrità referenziale devono essere specificate separatamente dai programmi applicativi e conservate nel catalogo. Deve essere possibile cambiare queste regole quando serve senza necessariamente avere effetti sulle applicazioni esistenti.
Regola 11: *Indipendenza dalle distribuzioni*:
La distribuzione di parti di database in varie località deve essere trasparente per l'utente del database. Le applicazioni esistenti devono continuare ad operare senza problemi:

1. la prima volta che viene introdotta una versione distribuita del DBMS e
2. quando dati già distribuiti vengono ridistribuititi.

Regola 12: *Regola di non sovversione*:
Se il sistema fornisce un'interfaccia di basso livello (che consenta l'accesso ai singoli record), allora quell'interfaccia non si deve poter utilizzare per sovvertire il sistema, ad esempio bypassando una regola di sicurezza o di integrità referenziale.

Nel 1986 l'ANSI pubblicò lo standard SQL/86 acquisendo di fatto la versione di SQL utilizzata da IBM. Ulteriori standardizzazioni di SQL sono avvenute con le specifiche SQL/89, SQL/92 e SQL/2003 che però hanno avuto poco successo tra i produttori di DB che hanno continuato a sviluppare dialetti proprietari dell'SQL.

Guardando più nel dettaglio alle vicissitudini di Oracle, la versione 5, che per la prima volta consentiva query distribuite, fu rilasciata nel 1986; nel 1988 Oracle 6 introdusse il lock a livello di record (e non più a livello di tabella) ed il backup a caldo; nel 1992 Oracle 7 apportò un'autentica rivoluzione introducendo le stored procedure, i database triggers e l'integrità referenziale dichiarativa; Nel 1997 con Oracle 8 ci fu la (fugace) strizzata d'occhio ai database ad oggetti e poi, nel 1998, con oracle 8i il repentino cambio di direzione dovuto all'esplosione di Internet. Nel 2001 l'uscita di Oracle 9i portò come maggiore novità l'introduzione di Real Application Clusters, il sistema di clustering che garantisce l'alta affidabilità del database. Altra importante novità della versione 9i fu l'introduzione di XMLDB. La nuova moda del *grid computing*, trovò la sua concretizzazione in Oracle 10g, rilasciato nel 2003. La versione 11g è stata rilasciata nel luglio 2011, con tante piccole innovazioni ma nessuna grande rivoluzione. Per finire, a settembre 2009, è stata rilasciata l'ultima versione di oracle, la 11gR2 lo slogan dell'evento di presentazione è "Lower your IT costs", in linea con la crisi economica diffusa ci si concentra sul taglio dei costi.

Ted Codd è morto in Florida il 18 Aprile 2003 all'età di 79 anni.