BASI DI DATI RELAZIONALI

F. M. Malvestuto

A. A. 2015/2016

INTRODUZIONE

Lo sviluppo dei mezzi e delle tecniche per la registrazione delle informazioni ha sempre accompagnato il progetto di costruire archivi di conoscenze e di esperienze. Si ritiene che storicamente l'idea di raccogliere ed organizzare informazioni scritte abbia avuto origine in due regioni, la Mesopotamia e l'Egitto, circa 5000 anni fa. Le prime raccolte sumere ed egizie di informazioni registrate in cuneiforme su tavolette ed in geroglifico su papiro, contenevano informazioni di natura legale ed economica — leggi e trattati, contratti, rogiti notarili, operazioni fiscali e lasciti testamentari. In questi ed altri documenti è difficile distinguere i concetti di archivio e di biblioteca. Dal Medio Oriente il concetto di raccolta di documenti entrò nel mondo greco e romano. I re romani istituzionalizzarono i censimenti delle popolazioni e delle proprietà fin dal VI secolo a. C. La grande Biblioteca di Alessandria d'Egitto, fondata nel III secolo a. C., conteneva un'immensa raccolta di papiri dove erano registrati atti catastali e fiscali. Di fatto, si tratta di un *sistema informativo* ante litteram. Lo splendore della cultura araba dall'VIII al XIII secolo lo si può in larga parte attribuire alla disponibilità di biblioteche pubbliche e private. La *Bayt al-Hikmah* ("La Casa della Saggezza"), fondata nell'830 a Bagdad, conteneva una biblioteca pubblica con una voluminosa raccolta di materiale su argomenti diversi, e la biblioteca del califfo al-Hakam a Cordova vantava più di 400.000 volumi.

Il rapido sviluppo (anche se tardivo) delle biblioteche europee a partire dal XVI secolo seguì l'invenzione della stampa. Fin dall'inizio del XVII secolo, la pubblicazione letteraria è diventata il mezzo principale per la diffusione delle conoscenze. L'espressione *letteratura primaria* è usata per designare informazioni di contenuto originale stampate in formato vario: giornali e periodici, monografie, atti di convegni, rapporti, brevetti, bollettini. La rivista scientifica, classico mezzo di comunicazione nelle comunità scientifiche, nasce nel 1665. Alla fine del XX secolo, si è stimato che il numero di riviste scientifiche stampate nel mondo superi i 60.000 titoli, riflesso non solo della crescita del numero degli appassionati di scienza e dell'espansione del corpo delle conoscenze, ma anche del credito riconosciuto alle pubblicazioni. Per rendere questo patrimonio di conoscenze più accessibile e gestibile, una porzione sempre più grande viene conservata anche in versione elettronica.

La quantità dell'informazione stampata ha impedito per qualche tempo di trarne il massimo profitto in senso conoscitivo. Strumenti come indici e sommari, che aiutano ad identificare e localizzare informazioni d'interesse nella letteratura primaria, sono stati in uso sin dal XVI secolo e portarono allo sviluppo di quella che nel XIX secolo

è stata chiamata *letteratura secondaria*, che risponde all'esigenza di filtrare le fonti informative primarie.

Il fiorire della tecnologia digitale dalla metà del XX secolo ha rivoluzionato il modo di registrare e di diffondere le informazioni. Nei primi anni '60, per la prima volta i computer vennero usati per registrare testi con il proposito di ridurre i costi ed i tempi richiesti dalla pubblicazione di due repertori, l'*Index Medicus* della National Library of Medicine ed i *Scientific and Technical Aerospace Reports* della NASA. Dalla fine degli anni '60, queste basi di dati bibliografici e numerici sono diventate una nuova fonte informativa che è proliferata anche fuori degli ambiti istituzionali tradizionali e si è diffusa grazie al progresso tecnologico.

L'enorme quantità delle informazioni in formato digitale oggi esistenti pongono in maniera drammatica il problema della loro utilizzazione, che trova soluzione solo se esse sono organizzate con sistematicità e se esistono meccanismi per la ricerca ed il reperimento di singoli elementi su cui di volta in volta si appunta l'interesse dell'utente.

Il complesso delle rappresentazioni digitali delle informazioni in un moderno sistema informativo viene chiamato il suo *insieme di dati* ("data set"). Tipicamente, ogni insieme di dati si compone di una o più *basi di dati*, ed ogni base di dati è un insieme di file mutuamente connessi. Gli insiemi di dati sono organizzati in diverse *strutture di dati* che ne facilitano la creazione, l'accesso e l'aggiornamento ed ottimizzano la gestione delle risorse fisiche. Dal punto di vista della registrazione dell'informazione in forma digitale, è utile distinguere tra dati strutturati, come avviene negli elenchi e nei cataloghi dove gli oggetti sono rappresentati da brevi stringhe di simboli e da numeri, e dati non strutturati come nei testi scritti in un linguaggio naturale o nelle immagini figurative. L'obiettivo principale è quello di facilitare l'elaborazione dei dati sulla base delle loro relazioni. La scelta di una particolare struttura di dati è dettata dalla ricchezza espressiva che essa consente in vista dei compiti elaborativi assegnati al sistema informativo. Nei sistemi informativi i cui archivi contengono basi di dati non strutturati (ad esempio, documenti scritti), l'obiettivo è quello di ricercare record o loro porzioni in base alla presenza di certe parole od espressioni linguistiche oggetto dell'interrogazione. Grazie ad un indice (un altro file) che fornisce l'informazione necessaria a localizzare parole ed espressioni nei record della base di dati, le relazioni d'interesse (ad esempio, l'adiacenza di parole) possono essere desunte dall'indice; così, il testo stesso può essere registrato come un semplice file di record ordinato e sequenziale. Nelle basi di dati strutturati, le relazioni tra i dati individuali sono rappresentate nella struttura dei record e, fino agli anni '70, due soli tipi di struttura venivano comunemente usati: quella "gerarchica" e quella "reticolare". Dagli anni Settanta, si è affermato prepotentemente l'uso delle strutture

"relazionali" il cui successo è dovuto alla sua semplicità concettuale, al rigore della sua teoria (l'algebra relazionale) ed alla disponibilità di programmi software per il trattamento delle relazioni tra i dati con una logica indipendente dalla rappresentazione fisica.

I sistemi informativi sono oggi il veicolo primario per la raccolta, l'organizzazione, la registrazione, l'elaborazione e la presentazione delle informazioni. In linea di principio, un sistema informativo è un qualsiasi sistema che conservi registrazioni — ad esempio, una rubrica telefonica. Ma l'aspetto che distingue i moderni sistemi informativi è la loro dimensione digitale, che consente il trattamento veloce ed automatico di dati registrati in forma digitale e la loro traduzione da ed in formato analogico. Dare una classificazione dei sistemi informativi non è facile a causa della loro varietà e della continua evoluzione delle loro funzioni e delle loro strutture. La distinzione tra manuali ed automatici, oppure tra interattivi ("on-line") e non ("off-line") oppure ancora tra quelli che operano in "real-time" e quelli in "batch" non è più molto utile. È forse più ragionevole distinguerli dal punto di vista funzionale, cioè in base alle loro applicazioni; li distingueremo così in due categorie: i sistemi informativi che operano in contesti organizzativi ed i sistemi informativi che espletano servizi di pubblica utilità. La prima categoria comprende i sistemi informativi utilizzati per lo svolgimento sia delle funzioni direttive che delle attività operative. La seconda categoria comprende i prodotti dei cosiddetti "venditori di basi di dati", vale a dire basi di dati che contengono documenti testuali, dati aziendali ed industriali, statistiche e cataloghi di prodotti e servizi.

Il crescente interesse nel trattamento dell'informazione e nei sistemi informativi ha dato vita ad un campo di studi interdisciplinare che va sotto il nome di *scienza dell'informazione* e dove confluiscono discipline come l'informatica, le scienze cognitive, l'intelligenza artificiale, l'ingegneria, la matematica, le scienze sociali e del comportamento, la linguistica e l'archivistica. Come i fenomeni che essa studia, la scienza dell'informazione è dunque un campo composito d'indagine la cui struttura è in continua evoluzione.

CAPITOLO 1

BASI DI DATI RELAZIONALI

1.1 Sistemi informativi

Sotto il profilo squisitamente tecnico, un sistema informativo è un complesso di dati organizzati fisicamente in una memoria secondaria e gestiti in maniera tale da curarne la creazione, l'aggiornamento e l'interrogazione. I dati sono organizzati concettualmente in aggregati di informazioni omogenee che costituiscono le componenti del sistema informativo, e ogni operazione di aggiornamento ha per oggetto un singolo aggregato mentre un'interrogazione può coinvolgerne uno o più.
Comunque, le caratteristiche tecniche seppure eccellenti di un sistema informativo non valgono nulla se le informazioni in esso contenute non sono veritiere, se cioè non c'è corrispondenza fra l'immagine del mondo fornita dal sistema informativo e l'effettivo stato delle cose nel mondo stesso. Possiamo così riassumere la questione:

1. C'è un mondo che vogliamo rappresentare.
2. Vogliamo rappresentarlo usando un sistema informativo.
3. Il problema è di stabilire se le informazioni riflettono lo stato reale del mondo.
4. Per verificare la corrispondenza fra le informazioni ed il modo in cui stanno realmente le cose, abbiamo bisogno di un linguaggio per esprimere la conformità della rappresentazione.

1.2 Relazioni

Sul piano concettuale, l'informazione contenuta in un sistema informativo si riferisce a oggetti (fatti, cose, persone, etc.) che chiamiamo *entità* (entities, in inglese), e a loro legami o nessi concettuali che chiamiamo *relazioni* (relationships, in inglese). Ciascuna delle entità e delle relazioni è specificata da un certo numero di caratteristiche che chiamiamo *attributi* e ne descrivono le proprietà "rilevanti" (a giudizio del progettista del sistema informativo). Entità e relazioni sono concetti astratti che, in ogni istante (per così dire della vita del sistema informativo), si concretano rispettivamente in un insieme di oggetti concreti ed in legami fattuali tra questi.
C'è da dire che la differenza tra entità e relazioni non è così netta e, talora, svanisce. Così in un archivio di una persona a cui piace il cinema possiamo trovare un catalogo di Film (entità) con gli attributi

titolo anno regista produttore protagonista

Mentre nell'archivio di un cinefilo probabilmente troveremo un'entità Attore con attributi

nome_d'arte nome_anagrafico nascita morte nazionalità

un'entità Regista con attributi

nome nascita morte nazionalità

e la relazione Film tra Attore e Regista con attributi

titolo anno nome produttore protagonista

Senza insistere oltre sui costrutti concettuali, ci occuperemo nelle basi di dati relazionali nelle quali sia le entità che le relazioni sono formalmente trattate come tabelle (relations, in inglese), alle quali d'ora in poi riserveremo il nome di *relazioni* e che andiamo a definire.

Un *attributo* è definito da un nome A e da un insieme finito o infinito ma numerabile, che chiamiamo il *dominio* dell'attributo A e indichiamo con $dom(A)$.
Sia R un insieme di attributi. Un'*ennupla* (tuple, nell'inglese gergale) *su* R è una funzione definita su R che associa ad ogni attributo A in R un elemento di $dom(A)$. Se t è un'ennupla su R ed A è un attributo in R, allora con $t(A)$ indicheremo il valore assunto dalla funzione t in corrispondenza dell'attributo A.

Esempio 1.1 Consideriamo quattro attributi volo, giorno, pilota e ora che hanno i seguenti domini:

dom(volo)	=	insieme di codici numerici formati da tre cifre decimali
dom(giorno)	=	{lunedì, martedì, mercoledì, giovedì, venerdì, sabato, domenica}
dom(pilota)	=	insieme di stringhe alfabetiche
dom(ora)	=	insieme di stringhe del tipo "k:n" dove k ∈ {00, …, 23} e n ∈ {00, …, 59}

Un esempio di ennupla sull'insieme R = {volo, giorno, pilota, ora} è dato dalla funzione t che assume i seguenti valori:

t(volo) = 112 t(giorno) = lunedì t(pilota) = Rossi t(ora) = 10:30

Se t è un'ennupla su R ed X è un sottoinsieme proprio di R, allora l'ennupla t' su X così definita

$t'(A) = t(A)$ per ogni $A \in X$

prende il nome di *restrizione* di t ad X, e la indicheremo con $t[X]$. Si osservi che, se Y è a sua volta un sottoinsieme proprio di X, la restrizione di t a Y coincide con la restrizione di $t[X]$ a Y, cioè $t[Y] = (t[X])[Y]$. Diremo infine che due ennuple t_1 e t_2 su R *concordano* su un sottoinsieme X di R se $t_1[X] = t_2[X]$.

Esempio 1.1 (seguito) Data l'ennupla t

t(volo) = 112 t(giorno) = lunedì t(pilota) = Rossi t(ora) = 10:30

sia X = {volo, giorno, pilota}. La restrizione $t[X]$ di t ad X è

$t[X]$(volo) = 112 $t[X]$(giorno) = lunedì $t[X]$(pilota) = Rossi

Sia poi Y = {volo, pilota}. La restrizione $t[Y]$ di t a Y è

$t[Y]$(volo) = 112 $t[Y]$(pilota) = Rossi

e dunque $t[Y] = (t[X])[Y]$.

Se $(A_1, \ldots, A_k)$ è un qualsiasi ordinamento degli attributi in R, per pura comodità nel seguito potremo indicare l'ennupla t su R con $(A_1 = a_1, \ldots, A_k = a_k)$ o anche, più semplicemente, con $(a_1, \ldots, a_k)$. L'insieme (eventualmente infinito) di tutte le possibili ennuple su R prende il nome di *dominio* di R che indichiamo con $DOM(R)$. Una *relazione* con schema R è un sottoinsieme finito (eventualmente vuoto) di $DOM(R)$. Solitamente, una relazione con schema R viene rappresentata come una tabella, le cui colonne corrispondono agli attributi in R (presi in un ordine qualsiasi) e le cui righe corrispondono alle ennuple della relazione (anch'esse prese in un ordine qualsiasi).

Esempio 1.1 (seguito) Una relazione con schema R è riportata in Tabella 1.1.

volo	giorno	pilota	ora
112	lunedì	Rossi	10:30
112	mercoledì	Verdi	10:30
200	giovedì	Verdi	12:00
112	venerdì	Rossi	10:30

Tabella 1.1 Una relazione con schema {volo, giorno, pilota, ora}

1.3 Tabelle relazionali

Una *tabella relazionale* è una variabile insiemistica che ha un nome proprio e i cui valori sono relazioni. Più precisamente, ad ogni tabella relazionale sono associati

- un insieme di attributi che ne definisce lo *schema*, e
- un insieme di *vincoli* (*d'integrità semantica*) che riflettono le proprietà semantiche degli attributi presenti nello schema.

Esempio 1.2 Consideriamo una tabella relazionale di nome `Partenze` che abbia schema {`volo`, `giorno`, `pilota`, `ora`}. Per essa è ragionevole imporre i seguenti vincoli

> «Ogni volo parte sempre alla stessa ora»
>
> «Esiste un solo volo con un dato pilota, in un dato giorno ad una data ora»
>
> «C'è un solo pilota di un dato volo in un dato giorno».

Il modello di `Partenze` è la coppia $M = [R, F]$ dove R = {`volo`, `giorno`, `pilota`, `ora`} ed F è l'insieme dei tre vincoli. Un esempio di relazione conforme al modello di `Partenze` è la relazione riportata nella Tabella 1.1.

Esempio 1.3 Consideriamo una tabella relazionale di nome `Regno` che abbia schema {`sovrano`, `inizio`, `fine`}. Per essa è ragionevole imporre i seguenti vincoli

> «Esistono un solo inizio e una sola fine del regno di ogni sovrano»
>
> «La fine di un regno non è anteriore al suo inizio».

Esempio 1.4 Consideriamo una tabella relazionale di nome `Personale` che abbia schema {`matricola`, `nome`, `ufficio`, `direttore`, `matricola_direttore`}. Per essa è ragionevole imporre i seguenti vincoli

> «Per un impiegato con una data matricola, c'è un solo nome, c'è un solo ufficio, c'è un solo direttore e c'è una sola matricola di direttore».
>
> «La matricola di ogni direttore è tra le matricole degli impiegati».

Il vincolo

> «La fine di un regno non è anteriore al suo inizio»

è un esempio delle cosiddette *dipendenze su singole ennuple*.

I vincoli

> «Ogni volo parte sempre alla stessa ora»
>
> «Esiste un solo volo con un dato pilota, in un dato giorno ad una data ora»
>
> «C'è un solo pilota di un dato volo in un dato giorno»
>
> «Esistono un solo inizio e una sola fine del regno di ogni sovrano»

«Per un impiegato con una data matricola, c'è un solo nome, c'è un solo ufficio, c'è un solo direttore e c'è una sola matricola di direttore»

sono esempi delle cosiddette *dipendenze funzionali*.

Il vincolo

«La matricola di ogni direttore è tra le matricole degli impiegati»

è un esempio dei cosiddetti *vincoli d'inclusione* (altrimenti detti *vincoli referenziali*).

Nei prossimi due paragrafi, daremo le definizioni rigorose delle dipendenze funzionali e dei vincoli d'inclusione.

1.4 Dipendenze funzionali

Una dipendenza funzionale stabilisce un particolare legame semantico tra due insiemi non-vuoti di attributi X e Y; tale vincolo si scrive $X \rightarrow Y$ e si legge

X determina Y;

qui l'insieme X è detto il *determinante* della dipendenza funzionale e l'insieme Y il *dipendente*.

Ad esempio, i vincoli sulla tabella relazionale `Partenze` dell'Esempio 1.2 si esprimono con le tre dipendenze funzionali

volo → ora
{giorno, pilota, ora} → volo
{volo, giorno} → pilota

così come il primo vincolo sulla tabella relazionale `Regno` dell'Esempio 1.3 si esprime con la dipendenza funzionale

sovrano → {inizio, fine}

o, in maniera equivalente, con le due dipendenze funzionali

sovrano → inizio sovrano → fine.

Diremo che una relazione r con schema R *soddisfa* la dipendenza funzionale $X \rightarrow Y$ se

(*i*) la dipendenza funzionale $X \rightarrow Y$ è *applicabile* ad R, nel senso che sia il determinante (X) sia il dipendente (Y) sono sottoinsiemi di R;

(*ii*) le ennuple in r che concordano su X concordano anche su Y, vale a dire che per ogni coppia di ennuple t_1 e t_2 in r

$$t_1[X] = t_2[X] \quad \Rightarrow \quad t_1[Y] = t_2[Y].$$

Vediamo ora come decidere se una relazione r con schema R soddisfa o meno una dipendenza funzionale $X \rightarrow Y$ che supponiamo applicabile ad R. Il risultato dell'algoritmo è un valore della variabile logica *test* che assumerà il valore VERO se e solo se la relazione r soddisfa la dipendenza funzionale $X \rightarrow Y$.

(1) *test* := VERO.

(2) Con il criterio lessicografico ordinare le ennuple in r usando X come base dell'ordinamento. Sia L la lista risultante.

(3) Per ogni coppia t_1 e t_2 di elementi consecutivi della lista L,

se $t_1[X] = t_2[X]$ e $t_1[Y] \neq t_2[Y]$, allora *test* := FALSO ed Uscire.

Si osservi che, se $X \cup Y = R$, l'Istruzione 3 può semplificarsi come segue

(3′) Per ogni coppia t_1 e t_2 di elementi consecutivi della lista L, se $t_1[X] = t_2[X]$, allora *test* := FALSO ed Uscire.

1.5 Vincoli d'inclusione

Un vincolo d'inclusione stabilisce un legame di subordinazione di un insieme di attributi X ad un altro insieme di attributi Y e richiede che ogni valore di X sia già presente come valore di Y; tale vincolo si scrive $X \rhd Y$ e si legge

X rimanda ad Y.

Ad esempio, il secondo vincolo sulla tabella relazionale `Personale` dell'Esempio 1.4 si esprime con il vincolo d'inclusione

`matricola_direttore` $\rhd$ `matricola`

Diremo che una relazione r con schema R *soddisfa* il vincolo d'inclusione (o *vincolo autoreferenziale*) $X \rhd Y$ se

(*i*) il vincolo referenziale $X \rhd Y$ è *applicabile* ad R, cioè $X \cup Y \subseteq R$;

(*ii*) per ogni ennupla t_1 in r esiste un'ennupla t_2 in r tale che

$$t_1[X] = t_2[Y].$$

1.6 Modelli

Se R ed F sono rispettivamente lo schema e l'insieme dei vincoli su di una tabella relazionale di nome `Tab`, un *valore valido* di `Tab` è una qualsiasi relazione con schema R che soddisfi tutti i vincoli specificati in F. Chiamiamo la coppia $M = [R, F]$ il *modello* (*semantico*) della tabella relazionale `Tab` e diremo che una relazione con schema R è *conforme* al modello M se è un valore valido di `Tab`. Se indichiamo con $SAT_R(F)$ l'insieme di tutte le relazioni con schema R che soddisfano i vincoli definiti in F, una relazione r è conforme al modello M se appartiene a $SAT_R(F)$. Talora è utile la seguente analogia. Lo schema R di M è l'analogo dell'insieme delle variabili reali (incognite) di un sistema S di equazioni lineari, F è l'analogo di S, e $SAT_R(F)$ è l'analogo delle insieme delle soluzioni di S.

1.7 Chiavi

Dato il modello $M = [R, F]$ di una tabella relazionale, un sottoinsieme (non-vuoto) X dello schema R è una *sovrachiave* (superkey) per M se ogni relazione conforme ad M soddisfa la dipendenza funzionale $X \rightarrow A$ per ogni attributo $A \in R–X$. In termini equivalenti, X è una sovrachiave per M se nessuna relazione conforme ad M contiene due distinte ennuple t_1 e t_2 in r che concordano su X e, dunque, per ogni relazione r conforme ad M e per ogni coppia di ennuple distinte t_1 e t_2 in r, si ha $t_1[X] \neq t_2[X]$. Un esempio banale di sovrachiave per M è l'insieme R stesso.

Una *chiave* (key) per M è una sovrachiave minimale; così, se X è una sovrachiave, perché X sia una chiave deve aversi che nessun sottoinsieme proprio di X sia una sovrachiave.

Ad esempio, per il modello associato alla tabella relazionale `Partenze` gli insiemi

{volo, giorno} {giorno, pilota, ora} {volo, giorno, pilota, ora}

sono esempi di sovrachiave, ma solo gli insiemi {volo, giorno} e {giorno, pilota, ora} sono chiavi. E per il modello associato alla tabella relazionale `Regno` gli insiemi

{sovrano} {sovrano, inizio} {sovrano, fine} { sovrano, inizio, fine}

sono esempi di sovrachiave, ma solo l'insieme {sovrano} è chiave.

Quando viene creata tabella relazionale vengono specificati non solo il modello ma anche una particolare chiave, che prende il nome di *chiave primaria* (primary key); questa guiderà l'organizzazione fisica dei dati (v. capitolo 11).

1.8 Basi di dati relazionali

Una *base di dati* (*relazionale*) è un insieme di tabelle relazionali $\mathcal{D} = \{\mathtt{Tab}_1, \ldots, \mathtt{Tab}_n\}$ con l'aggiunta di vincoli d'integrità inter-relazionali. Tra questi il più importante è il vincolo d'inclusione (o *vincolo referenziale*)

$$\mathtt{Tab}_i\,(X) \;\triangleright\; \mathtt{Tab}_j\,(Y)$$

in cui X e Y sono rispettivamente sottoinsiemi degli schemi delle variabil relazionali $\mathtt{Tab}_i$ e $\mathtt{Tab}_j$, $i \neq j$. Se poi Y è una chiave del modello di $\mathtt{Tab}_j$, allora X è detta essere una *chiave esterna* (foreign key) per il modello di $\mathtt{Tab}_i$.

Quello che segue è un esempio di base di dati che verrà frequentemente richiamato nei prossimi capitoli.

Esempio 1.5 STUART è il nome di una base di dati (le informazioni riguardano la dinastia scozzese degli Stuart, che iniziò dopo la morte di Elisabetta I della dinastia dei Tudor la quale non ebbe eredi) formata dalle due tabelle relazionali:

Regno	con schema	{sovrano, inizio, fine}
Dinastia	con schema	{nome, sesso, nascita, morte}.

Qui sovrano è la chiave (primaria) del modello di Regno, e nome è la chiave (primaria) del modello di Dinastia. Inoltre, vale il vincolo referenziale

«il nome di ogni sovrano deve essere incluso nell'elenco dei nomi dei membri della dinastia»,

cioè, Regno (sovrano) $\triangleright$ Dinastia (nome). Si osservi che, siccome nome è la chiave primaria del modello di Dinastia, allora sovrano è anche una chiave esterna del modello di Regno.

Uno *stato* della base di dati $\mathcal{D} = \{\mathtt{Tab}_1, \ldots, \mathtt{Tab}_n\}$ è un insieme di relazioni una per ogni tabella relazionale in $\mathcal{D}$. Se $d = \{r_1, \ldots, r_n\}$ è uno stato di $\mathcal{D}$ ed r_i è la relazione corrispondente alla tabella relazionale $\mathtt{Tab}_i$, $1 \leq i \leq n$, allora

a) ogni r_i è una relazione conforme al modello di $\mathtt{Tab}_i$, ed è chiamata la *relazione di nome* $\mathtt{Tab}_i$ *contenuta in d*,

b) le relazioni $r_1, \ldots, r_n$ soddisfano i vincoli d'integrità inter-relazionali (se ve ne sono).

Ecco un altro esempio di base di dati.

Esempio 1.6 AEROPORTO è il nome di una base di dati formata dalle quattro tabelle relazionali:

Inventario	con schema	{A#, parte_di, nome}
Dotazione	con schema	{A#, vettore, numero}
Scorta	con schema	{A#, aerostazione, livello}
Partenze	con schema	{volo, giorno, pilota, ora}.

Supponiamo che lo stato attuale *d* della base di dati AEROPORTO contenga le quattro relazioni riportate nelle Tabelle 1.2, 1.3, 1.4 e 1.5.

A#	parte_di	nome
211	0	poltrona di II classe
2114	211	fodera per poltrona
216	211	cintura di sicurezza
21163	2116	gancio per cinture
21164	2116	attacco per cinture
206	0	pannello superiore
2066	206	dispositivo luminoso
2068	206	dispositivo d'areazione

Tabella 1.2 La relazione di nome Inventario contenuta in *d*

A#	vettore	numero
211	707	86
211	727	134
2114	707	86
2114	727	134
2116	707	244
2116	727	296
21164	707	488
21164	727	592

Tabella 1.3 La relazione di nome Dotazione contenuta in *d*

A#	aerostazione	livello
211	Fiumicino	106
211	Ciampino	28
211	Linate	6
2114	Malpensa	57

Tabella 1.4 La relazione di nome Scorta contenuta in *d*

volo	giorno	pilota	ora
112	lunedì	Rossi	10:30
112	mercoledì	Verdi	10:30
200	giovedì	Verdi	12:00
112	venerdì	Rossi	10:30

Tabella 1.5 La relazione di nome Partenze contenuta in *d*

Esempio 1.5 (seguito) Supponiamo che lo stato attuale *d* della base di dati STUART contenga le due relazioni riportate nelle Tabelle 1.6 e 1.7.

sovrano	inizio	fine
Giacomo I	1603	1625
Carlo I	1625	1649
Carlo II	1660	1685
Giacomo II	1685	1688
Maria II	1689	1694
Anna	1702	1714

Tabella 1.6 La relazione di nome Regno contenuta in *d*

Nome	sesso	nascita	morte
Giacomo I	M	1566	1625
Elisabetta	F	1590	1662
Carlo I	M	1600	1649
Carlo II	M	1630	1685
Maria	F	1631	1659
Giacomo II	M	1633	1701
Enrichetta A.	F	1640	1670
Maria II	F	1662	1694
Anna	F	1665	1714
Giacomo E.	M	1686	1766

Tabella 1.7 La relazione di nome Dinastia contenuta in *d*

Il lettore attento avrà certamente notato che nella relazione di nome Regno (vedi Tabella 1.6) curiosamente dopo i regni di Carlo I e di Maria II c'è un interregno rispettivamente di 11 e 8 anni. Il fatto è che Carlo I fu deposto, processato e messo a morte da Cromwell che istaurò una dittatura, e che alla morte di Cromwell il potere tornò nelle mani della monarchia. Quanto al regno di Maria II, il fatto è che essa successe a Giacomo II assieme al marito Guglielmo della dinastia degli Orange, il quale regnò col nome di Guglielmo III e morì 8 anni dopo la moglie.

Vediamo infine come viene creata una base di dati in un comune sistema di gestione di basi dati relazionali. Va subito detto che le tabelle relazionali sono semplicemente chiamate TABLES.

```
CREATE DATABASE CINEMA;

CREATE TABLE Film
(
titolo VARCHAR(20),
anno SMALLINT(4),
regista VARCHAR(30),
durata SMALLINT(3),
PRIMARY KEY (titolo, anno)
);


CREATE TABLE Attore
(
nome VARCHAR(30) PRIMARY KEY,
nascita SMALLINT(4) NOT NULL
);


CREATE TABLE Cast
(
film VARCHAR(20) NOT NULL,
anno SMALLINT(4) NOT NULL,
attore VARCHAR(30) NOT NULL REFERENCES Attore (nome),
ruolo VARCHAR(10),
PRIMARY KEY (film, anno, attore),
FOREIGN KEY (film, anno) REFERENCES Film (titolo, anno),
);
```

Va osservato che nella definizione delle tabelle relazionali Attore e Cast compare il vincolo che un attributo *A* sia NOT NULL, che sta a significare che il valore di *A* appartenga sempre a *dom*(*A*). Questo perché si ammettono anche relazioni in cui il valore di un attributo sia assente (NULL) sempreché quell'attributo non sia stato dichiarato NOT NULL. Va comunque precisato che la dichiarazione di una chiave sottintende che tutti gli attributi presenti nella chiave siano NOT NULL.

1.9 Interrogazione di una base di dati

Come già detto, una base di dati è essenzialmente una raccolta di dati accessibili ad una comunità di soggetti che formano l'utenza del sistema informativo. Oltre a

rendere reperibili per intero le relazioni contenute nello stato corrente della base di dati, i linguaggi di interrogazione permettono anche di selezionare porzioni di singole relazioni ed infine di combinare informazioni provenienti da distinte relazioni.

Esempio 1.6 (seguito) Ecco due possibili domande sulla base di dati AEROPORTO:

1. "Quali sono le parti dell'articolo di codice 211?"

Risposta:

nome
fodera per poltrona
cintura di sicurezza

Tabella 1.8

2. "Quante poltrone di II classe sono usate sul vettore 727?"
Risposta:

numero
134

Tabella 1.9

Esempio 1.5 (seguito) Ecco tre possibili domande sulla base di dati STUART:

1. "Chi furono i sovrani che precedettero Carlo II, in che anno salirono al trono e in che anno i loro regni ebbero termine?".
Risposta:

sovrano	inizio	fine
Giacomo I	1603	1625
Carlo I	1625	1648

Tabella 1.10

2. "Chi furono il primo re e la prima regina?"
Risposta:

sovrano
Giacomo I
Maria II

Tabella 1.11

3. “Chi fu l’ultimo sovrano e in che anno nacque?”
Risposta:

sovrano	nascita
Anna	1665

Tabella 1.12

Esistono due modelli formali per interrogare una base di dati relazionale: il primo è basato sulla logica dei predicati ed è un linguaggio “dichiarativo”; il secondo è basato sull’algebra delle relazioni ed è un linguaggio “procedurale”. Ad essi sono dedicati i Capitoli 2 e 3. Qui ci limitiamo a mostrare come vengono formulate alcune domande sulla base di dati CINEMA nel linguaggio SQL, senza alcuna pretesa di darne una completa descrizione. Il costrutto base è il comando SELECT- FROM-WHERE.

Cominciamo con una domanda sulla TABLE Film.

```
SELECT titolo
FROM Film
WHERE anno >= 1980 AND regista = ‘Fellini’;
```

Altre domande simili:

```
SELECT titolo
FROM Film
WHERE anno >= 1980 AND regista LIKE ‘%ini’;
```

```
SELECT titolo
FROM Film
WHERE anno >= 1980 AND regista LIKE ‘%in_’;
```

```
SELECT regista
FROM Film
WHERE anno >= 1980;
```

```
SELECT titolo AS title, anno AS year, regista AS director, durata AS length
FROM Film
```

```
WHERE anno >= 1980;

SELECT regista DISTINCT
FROM Film
WHERE anno >= 1980;


SELECT *
FROM Film
WHERE anno >= 1980;

SELECT COUNT(*)
FROM Film
WHERE anno >= 1980;

SELECT anno, COUNT(*)
FROM Film
WHERE anno >= 1980
GROUP BY anno;

SELECT AVG(durata)
FROM Film
WHERE regista = 'Fellini';
```

C'è da dire che, per la possibile presenza di attributi che non siano stati dichiarati NOT NULL, la logica delle condizioni contenute nella clausola WHERE è "a tre valori": Vero, Falso, Sconosciuto. Pertanto una condizione è vera per una data ennupla se e solo se il suo valore non è né falso né sconosciuto. Per determinare il valore di una condizione che include i connettivi logici, basta interpretare

— i tre valori come Vero = 1, Falso = 0, Sconosciuto = ½
— AND = *min*, OR = *max*, NOT (x) = $1 - x$

Ad esempio

Vero AND (Falso OR NOT (Sconosciuto)) = *min* (1, *max* (0, 1 – ½)) = ½

Così, dal momento che l'attributo durata non è stato dichiarato NOT NULL, la condizione

durata < 90 OR durata >= 90

è sempre vera per ogni ennupla di una relazione di nome Film in cui durata ha un valore diverso da NULL, ma non lo è più ad esempio per l'ennupla (La dolce vita, 1960, Fellini, NULL) perché il suo valore è sconosciuto: *max* (½,½) = ½.

Vediamo ora una domanda sulle due TABLES Cast e Film.

```
SELECT attore, ruolo
FROM Cast, Film
WHERE film = titolo AND regista = 'Fellini';
```

Vediamo un esempio un po' meno semplice. Trovare le coppie di film dirette dallo stesso regista senza produrre coppie come (*a*, *a*); inoltre, le coppie sono ordinate alfabeticamente cioè (*a*, *b*) esclude (*b*, *a*).

```
SELECT titolo, regista
FROM Film f1, Film f2
WHERE f1.regista = f2.regista AND f1.titolo < f2.titolo;
```

Vediamo infine domande "nidificate". Questo è un primo esempio: Trovare i film diretti dal regista di La dolce vita.

```
SELECT    titolo
FROM      Film,
          (SELECT    regista
          FROM       Film
          WHERE      titolo = 'La dolce vita') R
WHERE     Film.regista = R.regista;
```

```
SELECT    titolo
FROM      Film
```

```
WHERE    regista =
         (SELECT    regista
         FROM       Film
         WHERE      titolo = 'La dolce vita')
```

Questo è un altro esempio: Trovare gli attori che hanno recitato in almeno un film diretto dal regista di La dolce vita.

```
SELECT   attore
FROM     Cast,
         (SELECT    titolo
         FROM       Film
         WHERE      regista =
                    (SELECT    regista
                    FROM       Film
                    WHERE      titolo = 'La dolce vita')) T
WHERE    film  = T.titolo;
```

CAPITOLO 2

LINGUAGGIO D'INTERROGAZIONE DICHIARATIVO

Consideriamo una base di dati relazionale $\mathcal{D} = \{\texttt{Tab}_1, \ldots, \texttt{Tab}_n\}$ e indichiamo con R_i lo schema della tabella relazionale $\texttt{Tab}_i$, $1 \leq i \leq n$. L'insieme $U = R_1 \cup \ldots \cup R_n$ prende il nome di *universo del discorso*. L'interrogazione della base di dati relazionale $\mathcal{D}$ avviene formulando le domande con espressioni del *Calcolo Relazionale*, che è un'estensione della logica dei predicati del prim'ordine. Le espressioni contengono *variabili ennuple* (tuple variables), che chiamiamo semplicemente *variabili* per brevità, che stanno a rappresentare ennuple su sottoinsiemi dell'universo del discorso U: così, se R è un insieme di uno o più attributi presenti in U, allora utilizzeremo la scrittura $x(R)$ (o $y(R)$ o $z(R)$ …) per definire una variabile che rappresenti le ennuple su R, cioè gli elementi di $DOM(R)$. Inoltre, i nomi $\texttt{Tab}_1, \ldots, \texttt{Tab}_n$ delle tabelle relazionali compariranno nelle espressioni del Calcolo Relazionale con funzione di *predicati* e, dato uno stato d di $\mathcal{D}$, per selezionare le ennuple che appartengono alla relazione di nome $\texttt{Tab}_i$ contenuta in d scriveremo la condizione

$$\texttt{Tab}_i(x)$$

dove la variabile x deve essere stata definita precedentemente come $x(R_i)$. Prima di definire formalmente il nostro linguaggio, vogliamo illustrare queste nozioni servendoci di due esempi.

Esempio 2.1 Consideriamo la base di dati AEROPORTO e prendiamo la domanda "Quali sono le parti dell'articolo di codice 211?". Una sua formulazione logica è la seguente:

Quali sono i valori t della variabile x(nome) per cui

esiste un valore t_1 della variabile y(A#, parte_di, nome) tale che

l'ennupla t_1 (A# = b, parte_di = c, nome = d) appartiene alla relazione di nome Inventario,
t_1(parte_di) = 211,
t_1(nome) = t

?

In maniera più formale, possiamo selezionare i valori t di x(nome) che entreranno a far parte della relazione cercata (vedi Tabella 1.2) con la formula

$$\exists\, y(\texttt{A\#}, \texttt{parte\_di}, \texttt{nome})$$
$$(\texttt{Inventario}(y) \wedge y(\texttt{parte\_di}) = 211 \wedge\ y(\texttt{nome}) = x(\texttt{nome})).$$

Prendiamo ora la domanda "Quante poltrone di II classe sono usate sul vettore 727?". Una formulazione logica della domanda è la seguente:

Qual è il valore t della variabile x(numero) per cui

esiste un valore t_1 della variabile y(A#, vettore, numero) tale che

l'ennupla t_1 appartiene alla relazione di nome Dotazione,
t_1(vettore) = 727,
t_1(numero) = t,
esiste un valore t_2 della variabile z(A#, parte_di, nome) tale che

l'ennupla t_2 appartiene alla relazione di nome Inventario,
t_2(A#) = t_1(A#),
t_2(nome) = poltrona di II classe

?

In maniera più formale, possiamo selezionare i valori di x(numero) che entreranno a far parte della relazione cercata (vedi Tabella 1.3) con la formula

$\exists$ y(A#, vettore, numero)
(Dotazione(y) $\wedge$ y(vettore) = 727 $\wedge$ y(numero) = x(numero) $\wedge$
$\exists$ z(A#, parte_di, nome)
(Inventario(z) $\wedge$ z(A#) = y(A#) $\wedge$ z(nome) = poltrona di II classe))

Esempio 2.2 Consideriamo la base di dati STUART e prendiamo la domanda "Chi furono i sovrani che precedettero Carlo II, e da che anno a che anno regnarono?". D'ora in poi supporremo di sapere che non ci siano stati sovrani saliti al trono lo stesso anno. Una formulazione logica della domanda è la seguente:

Quali sono i valori t della variabile x(sovrano, inizio, fine) tali che

l'ennupla t appartiene alla relazione di nome Regno,
esiste un valore t_1 della variabile y(sovrano, inizio, fine} tale che

l'ennupla t_1 appartiene alla relazione di nome Regno,
t_1(sovrano) = Carlo II,
t_1(inizio) > t(inizio)

?

In maniera più formale, possiamo selezionare i valori della variabile x(sovrano, inizio, fine) che entreranno a far parte della relazione cercata (vedi Capitolo 1) con la formula

Regno(x) $\wedge$ ($\exists$ y(sovrano, inizio, fine)
(Regno(y) $\wedge$ y(sovrano) = Carlo II $\wedge$ x(inizio) < y(inizio))

Prendiamo poi la domanda "Chi furono il primo re e la prima regina?". Una sua formulazione logica è la seguente:

Quali sono i valori t della variabile x(sovrano) per cui

esiste un valore t_1 della variabile y(sovrano, inizio, fine) tale che

l'ennupla t_1 appartiene alla relazione di nome Regno,
t_1(sovrano) = t,
per ogni valore t_2 della variabile z(sovrano, inizio, fine)

se

l'ennupla t_2 appartiene alla relazione di nome Regno e t_2(inizio) < t_1(inizio))

allora

esiste un valore t_3 della variabile u(nome, sesso, nascita, morte) tale che:

l'ennupla t_3 appartiene alla relazione di nome Dinastia,
t_3(nome) = t,
esiste un valore t_4 della variabile v(nome, sesso, nascita, morte) tale che

l'ennupla t_4 appartiene alla relazione di nome Dinastia,
t_4(nome) = t_2(sovrano),
t_4(sesso) ≠ t_3(sesso)

?

Si osservi che il requisito

per ogni valore t_2 della variabile z(sovrano, inizio, fine)

se (l'ennupla t_2 appartiene alla relazione di nome Regno e t_2(inizio) < t_1(inizio)) allora ……

è logicamente equivalente al seguente

per ogni valore t_2 della variabile z(sovrano, inizio, fine)

o l'ennupla t_2 non appartiene alla relazione di nome Regno,
o t_2(inizio) ≥ t_1(inizio),
o ……

In maniera più formale, possiamo selezionare i valori della variabile x(SOVRANO) che entreranno a far parte della relazione cercata (vedi Capitolo 1) con la formula

$\exists$ y(sovrano, inizio, fine)
(Regno(y) ∧ y(sovrano) = x(sovrano) ∧
$\forall$ z(sovrano, inizio, fine) (¬ Regno(z) ∨ y(inizio) ≤ z(inizio) ∨

$\exists$ u(nome, sesso, nascita, morte)
(Dinastia(u) ∧ u(nome) = x(sovrano) ∧

$\exists\, v$(nome, sesso, nascita, morte)
(Dinastia(v) $\wedge$ v(nome) = z(sovrano) $\wedge$

v(sesso) $\neq$ u(sesso)))))

Prendiamo infine la domanda "Chi fu l'ultimo sovrano e in che anno nacque?". Una formulazione logica della domanda è la seguente:

Quali sono i valori t della variabile x(sovrano, nascita) per cui

esiste un valore t_1 della variabile y(sovrano, inizio, fine) tale che

l'ennupla t_1 appartiene alla relazione di nome Regno,
t_1(sovrano) = t(sovrano),
per ogni valore t_2 della variabile z(sovrano, inizio, fine)

se l'ennupla t_2 appartiene alla relazione di nome Regno
allora t_2(inizio) $\leq$ t_1(inizio),

ed esiste un valore t_3 della variabile u(nome, sesso, nascita, morte) tale che
l'ennupla t_3 appartiene alla relazione di nome Dinastia,
t_3(nome) = t(sovrano),
t_3(nascita) = t(nascita)

?

In maniera più formale, possiamo selezionare i valori della variabile x(sovrano, nascita) che entreranno a far parte della relazione cercata (vedi Capitolo 1) con la formula

($\exists\, y$(sovrano, inizio, fine)
(Regno(y) $\wedge$ y(sovrano) = x(sovrano) $\wedge$
$\forall\, y'$(sovrano, inizio, fine)
($\neg$ Regno(y') $\vee$ y'(inizio) $\leq$ y(inizio))) $\wedge$

($\exists\, z$(nome, sesso, nascita, morte)

(Dinastia(z) $\wedge$ z(nome) = x(sovrano) $\wedge$ z(nascita) = x(nascita)))

2.1 Calcolo Relazionale

Passiamo ora a definire il nostro linguaggio logico di interrogazione su una base di dati relazionale $\mathcal{D}$, che chiamiamo *Calcolo Relazionale su* $\mathcal{D}$. Le espressioni del Calcolo Relazionale su $\mathcal{D}$ sono definite attraverso le nozioni di "formula" e di "formula ben formata". Per costruire tali formule, abbiamo anche bisogno dell'insieme Θ = {=, $\neq$, <, $\leq$, >, $\geq$} di operazioni di confronto tra valori di attributi nell'universo del discorso U. Se θ è un elemento di Θ, diremo che due attributi A e B sono *confrontabili* rispetto a θ (per brevità, θ-*confrontabili*) se l'operazione θ è definita sul prodotto cartesiano dei domini di A e B. Assumeremo sempre che ogni

attributo sia confrontabile con se stesso rispetto all'eguaglianza ed alla diseguaglianza.

2.2 Formule del Calcolo Relazionale

Una formula è definita ricorsivamente a partire da *formule atomiche*.

> A1. Per ogni tabella relazionale `Tab` in $\mathcal{D}$ e per ogni variabile x, `Tab`(x) è una formula atomica.
>
> A2. Per ogni coppia di variabili x ed y (non necessariamente distinte), per ogni operazione θ in Θ e per ogni coppia di attributi A e B in U che siano θ-confrontabili,
>
> $$x(A)\ \theta\ y(B)$$
>
> è una formula atomica.
>
> A3. Per ogni variabile x, per ogni operatore θ in Θ e per ogni attributo A in U tale che θ sia definita sul dominio di A, se a è un elemento di $dom(A)$, allora
>
> $$a\ \theta\ x(A) \quad \text{ed} \quad x(A)\ \theta\ a$$
>
> sono formule atomiche.

Abbiamo già incontrato questi tre esempi di formule atomiche:

(A1) `Inventario`(y)
(A2) z(`nome`) = x(`sovrano`).
(A3) y(`parte_di`) = 211

Una *formula* è costruita a partire da formule atomiche utilizzando le parentesi, i connettivi logici ed i quantificatori esistenziale ed universale.

> F1. Ogni formula atomica è una formula.
>
> F2. Se f è una formula, allora (f) è una formula.
>
> F3. Se f è una formula, allora $\neg(f)$ è una formula.
>
> F4. Se f e g sono formule, allora (f) $\wedge$ (g) ed (f) $\vee$ (g) sono formule.

F5. Se x è una variabile, f è una formula che contiene il simbolo x ed R è un sottoinsieme di U, allora

$$\exists\, x(R)\ (f) \qquad\qquad \forall\, x(R)\ (f)$$

sono formule.

Per non appesantire la scrittura, delle parentesi per il momento faremo un uso parsimonioso e le utilizzeremo solo per evitare ambiguità.

Ecco alcuni esempi di formule non atomiche:

$y(\texttt{parte\_di}) = 211 \wedge\ y(\texttt{nome}) = x(\texttt{nome})$

$\texttt{Inventario}(y) \wedge y(\texttt{parte\_di}) = 211 \wedge\ y(\texttt{nome}) = x(\texttt{nome})$

$\exists\, y(\texttt{A\#\_di}, \texttt{parte\_di}, \texttt{nome})$
$\quad (\texttt{Inventario}(y) \wedge y(\texttt{parte\_di}) = 211 \wedge y(\texttt{nome}) = x(\texttt{nome}))$

Consideriamo ancora la seguente formula che abbiamo usato per chiedere alla base di dati STUART «chi furono il primo re e la prima regina ?»

$f = \exists\, y(\texttt{sovrano}, \texttt{inizio}, \texttt{fine})$
$(\texttt{Regno}(y) \wedge y(\texttt{sovrano}) = x(\texttt{sovrano}) \wedge$

$\exists\, u(\texttt{nome}, \texttt{sesso}, \texttt{nascita}, \texttt{morte})$
$\quad (\texttt{Dinastia}(u) \wedge u(\texttt{nome}) = x(\texttt{sovrano}) \wedge$

$\forall\, z(\texttt{sovrano}, \texttt{inizio}, \texttt{fine})$
$\quad (\neg\, \texttt{Regno}(z) \vee y(\texttt{inizio}) \leq z(\texttt{inizio}) \vee$

$\exists\, v(\texttt{nome}, \texttt{sesso}, \texttt{nascita}, \texttt{morte})$
$\quad (\texttt{Dinastia}(v) \wedge v(\texttt{nome}) = z(\texttt{sovrano}) \wedge$
$\quad v(\texttt{sesso}) \neq u(\texttt{sesso}))))$

2.3 Espressioni del Calcolo Relazionale

Introdurremo ora la nozione di "formula ben formata", la quale serve ad escludere formule del tipo

$\texttt{Dotazione}(x) \wedge x(\texttt{aerostazione}) = \text{Fiumicino}$

in cui la presenza della formula atomica $\texttt{Dotazione}(x)$ sottintende che i valori della variabile x siano le ennuple su {A#, vettore, numero} laddove la formula atomica $x(\texttt{aerostazione}) = \text{Fiumicino}$ vorrebbe che la variabile x fosse definita anche sull'attributo aerostazione.

Sia f una formula. Una variabile x presente in f potrà comparirvi una o più volte e le sue presenze in f sono dette le *occorrenze* di x in f. Vedremo che un'occorrenza di una variabile che compare in f può essere *libera* o *vincolata*. Inoltre, le variabili di f con almeno un'occorrenza libera sono dette *variabili reali*, e le altre *variabili apparenti*. La formula f è *aperta* se contiene almeno una variabile reale; altrimenti, f è una formula *chiusa*. Una variabile reale potrà essere di tre tipi: *ben strutturata*, *parzialmente strutturata* e *mal strutturata*. Infine, ad ogni variabile reale x di f che sia ben strutturata o anche parzialmente strutturata, faremo corrispondere un insieme (non vuoto) di attributi che chiameremo il *tipo* di x in f ed indicheremo con $T_f(x)$.

Riassumendo, se f è una qualsiasi formula, per le variabili che compaiono in f avremo:

- variabili reali:
  - — ben strutturate: $x_1, \ldots, x_n$ — $T_f(x_i)$ è definito
  - — parzialmente strutturate: $y_1, \ldots, y_m$ — $T_f(y_i)$ è definito
  - — mal strutturate: $z_1, \ldots, z_p$ — $T_f(z_i)$ non è definito
- variabili apparenti: $u_1, \ldots, u_q$ — $T_f(u_i)$ non è definito

Diamo ora la definizione (di natura ricorsiva) di *formula ben formata*, la quale essenzialmente richiede che nessuna sua variabile reale sia mal strutturata.
Cominciamo con le formule atomiche. Queste formule sono tutte aperte e, per definizione, sono tutte ben formate.

A1. $f = \texttt{Tab}(x)$
L'occorrenza della variabile x è libera in f. La variabile x è una variabile reale ben strutturata e, se R è lo schema della tabella relazionale di nome `Tab`, il tipo di x in f è proprio R, cioè $T_f(x) := R$.

Ad esempio, se $f = \texttt{Regno}(y)$, allora la variabile y è ben strutturata in f; inoltre $T_f(x) := \{\texttt{sovrano}, \texttt{inizio}, \texttt{fine}\}$.

A2. $f = x(A)\ \theta\ y(B)$
Entrambe le occorrenze delle variabili x ed y sono libere in f. Le variabili x ed y sono entrambe variabili reali parzialmente strutturate. I loro tipi in f sono $T_f(x) := \{A\}$ e $T_f(y) := \{B\}$ se $x \neq y$; altrimenti, $T_f(x) := \{A, B\}$.

Ad esempio, se $f = x(\texttt{fine}) \leq y(\texttt{inizio})$, allora le variabili x e y sono entrambe variabili reali parzialmente strutturate in f; inoltre, $T_f(x)$:= {fine} e $T_f(y)$:= {inizio}.

A3. $f = x(A)\ \theta\ a$ oppure $f = a\ \theta\ x(A)$.
L'occorrenza della variabile x è libera in f. La variabile x è una variabile reale parzialmente strutturata. Il tipo di x in f è $T_f(x) := \{A\}$.

Ad esempio, se f è $x(\texttt{sovrano})$ = Carlo II, allora la variabile x è una variabile reale parzialmente strutturata in f; inoltre, $T_f(x)$:= {sovrano}.

Passiamo ora alle formule non atomiche. Una condizione necessaria perché una formula non atomica sia ben formata è che ogni sua componente sia essa stessa una formula ben formata. Tale condizione è anche sufficiente per le formule non atomiche di tipo F2, $f = (g)$, e di tipo F3, $f = \neg(g)$. Per queste si ha inoltre che tutte le proprietà delle variabili in g restano immutate in f; vale a dire,

un'occorrenza di una variabile presente in g è libera in f se e solo se lo è in g,

le variabili reali di f sono esattamente le variabili reali di g,

una variabile reale di f è ben strutturata o parzialmente strutturata se e solo se lo è in g,

il tipo di una variabile reale di f che non sia mal strutturata in f è identico al suo tipo in g.

Passiamo ora a considerare le formule non atomiche di tipo F4 ed F5.

F4. $f = (g) \wedge (h)$ oppure $f = (g) \vee (h)$
dove le componenti g e h di f sono formule ben formate. Per ogni variabile x presente in f, un'occorrenza di x in f è libera se e solo se la corrispondente occorrenza di x in g o in h è libera. Le variabili reali di f sono le variabili reali di g e le variabili reali di h.
Sia x una variabile reale di f. Se le occorrenze libere di x in f sono tutte e solo le sue occorrenze libere in g (rispettivamente, in h), allora tutte le proprietà di x in f verranno mutuate da g

(rispettivamente, da h). Supponiamo invece che x abbia occorrenze libere sia in g che in h. Distinguiamo i seguenti quattro casi:

> (1) La variabile x è ben strutturata sia in g che in h. Allora x è ben strutturata in f se $T_g(x) = T_h(x)$; altrimenti, è mal strutturata. Se x è ben strutturata in f, allora il suo tipo in f è definito come $T_f(x) := T_g(x)$.
>
> (2) La variabile x è ben strutturata in g ma lo è parzialmente in h. Allora x è ben strutturata in f se $T_h(x) \subseteq T_g(x)$; altrimenti, è mal strutturata. Se x è ben strutturata in f, allora il suo tipo in f è definito come $T_f(x) := T_g(x)$.
>
> (3) La variabile x è ben strutturata in h ma lo è parzialmente in g. Allora x è ben strutturata in f se $T_g(x) \subseteq T_h(x)$; altrimenti, è mal strutturata. Se x è ben strutturata in f, allora il suo tipo in f è definito come $T_f(x) := T_h(x)$.
>
> (4) La variabile x è parzialmente strutturata sia in g che in h. Allora x è parzialmente strutturata in f ed il suo tipo in f è definito come $T_f(x) := T_g(x) \cup T_h(x)$.

La formula f è ben formata se nessuna variabile reale è mal strutturata.

F5. $f = \exists\, x(R)\ (g)$ oppure $f = \forall\, x(R)\ (g)$

dove la componente g di f è una formula ben formata. Ogni occorrenza di x in f è vincolata dal quantificatore. Per ogni altra variabile y presente in f, le sue proprietà in f vengono mutuate da g. La formula f è ben formata se:

> x è una variabile reale di g,
> $T_g(x) = R$ nel caso che x sia ben strutturata in g,
> $T_g(x) \subseteq R$ nel caso che x sia parzialmente strutturata in g.

Esempio 2.1 (seguito). Consideriamo la formula g così definita

$$\texttt{Inventario}(y) \wedge y(\texttt{parte\_di}) = 211 \wedge y(\texttt{nome}) = x(\texttt{nome})$$

Tutte le occorrenze delle variabili x ed y sono libere in g. Inoltre, x è parzialmente strutturata in g ed il suo tipo è $T_g(x) = \{\texttt{nome}\}$. La variabile y è ben strutturata in g ed il suo tipo è $T_g(y) = \{\texttt{A\#}, \texttt{parte\_di}, \texttt{nome}\}$. Dunque, la formula g è ben formata. Consideriamo poi la formula f

$$\exists\, y(\texttt{A\#}, \texttt{parte\_di}, \texttt{nome})\ (g)$$

L'occorrenza della variabile x è libera in f, mentre le occorrenze della variabile y in f sono tutte vincolate dal quantificatore esistenziale. La variabile x è reale e parzialmente strutturata in f ed il suo tipo in f è $T_f(x) = T_g(x) = \{\texttt{nome}\}$. Dunque, anche la formula f è ben formata.

Con queste definizioni, definiamo un'*espressione* del Calcolo Relazionale sulla base di dati $\mathcal{D}$ una qualsiasi stringa della forma

$$\{x(R) \mid f\}$$

dove

R è un sottoinsieme di U,
f è una formula aperta e ben formata,
x è l'unica variabile reale di f e
$T_f(x) \subseteq R$, dove è richiesta l'uguaglianza se x è ben strutturata in f.

Consideriamo la domanda: "Chi fu il primo sovrano?". Un'espressione del Calcolo Relazionale per questa domanda è $\{x(\texttt{sovrano}) \mid f\}$ dove f è data dalla seguente formula

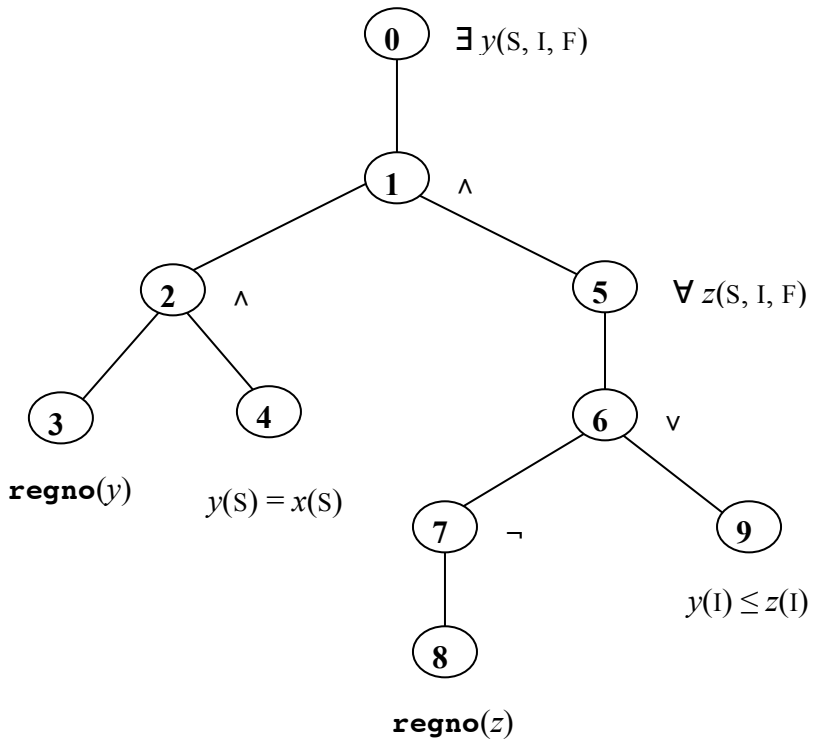
$$\begin{aligned}
&\exists\, y(\texttt{sovrano}, \texttt{inizio}, \texttt{fine}) \\
&\quad (((\texttt{Regno}(y)) \wedge (y(\texttt{sovrano}) = x(\texttt{sovrano}))) \wedge \\
&\qquad (\forall\, z(\texttt{sovrano}, \texttt{inizio}, \texttt{fine}) \\
&\qquad\quad ((\neg(\texttt{Regno}(z))) \vee (y(\texttt{inizio}) \leq z(\texttt{inizio}))))))
\end{aligned}$$

Data la base di dati $\mathcal{D}$ ed una qualsiasi stringa della forma

$$\{x(R) \mid f\}$$

sta al *compilatore* (un programma ad hoc del sistema di gestione della base di dati) decidere se la stringa è o meno un'espressione del Calcolo Relazionale. (Se non lo è, allora l'utente viene avvertito che la sua domanda non è corretta.) A tale scopo, come prima cosa il compilatore controlla che R sia un sottoinsieme di U. Se così, il compilatore costruisce l'*albero*

sintattico di *f*, che ne rappresenta graficamente la struttura sintattica. Nel nostro esempio, se indichiamo con S, I ed F rispettivamente gli attributi sovrano, inizio e fine, l'albero sintattico di *f* è quello rappresentato in figura.



In generale, per costruire l'albero sintattico di una formula *f* il compilatore comincia con l'identificare le sue *componenti* (*strutturali*): a partire dalla formula (*f*), vengono numerate le coppie di *parentesi connettive* (quelle che racchiudono formule secondo le specifiche sintattiche F2, F3, F4 ed F5). Per la formula *f* dell'esempio abbiamo:

$$(_0\ \exists y(\mathrm{S, I, F})\ \ (_1\ (_2\ (_3\ \mathtt{Regno}(y)\ _3) \wedge (_4\ y(\mathrm{S}) = x(\mathrm{S})\ _4)\ _2)\ \wedge$$
$$(_5\ \forall z(\mathrm{S, I, F})\ (_6\ (_7\ \neg(_8\ \mathtt{Regno}(z)\ _8)\ _7) \vee (_9\ y(\mathrm{I}) \leq z(\mathrm{I})\ _9)\ _6)\ _5)\ _1)\ _0)$$

Una volta numerate le coppie di parentesi connettive, le componenti di *f* sono le sottoformule di *f* che sono delimitate da una coppia di parentesi connettive che hanno lo stesso indice. Se con f_i indichiamo la componente racchiusa dalla coppia di parentesi $(_i$ e $_i)$, abbiamo nel nostro caso

f_0 $\quad \exists y(\mathrm{S, I, F})\ \ (_1\ (_2\ (_3\ \mathtt{Regno}(y)\ _3) \wedge (_4\ y(\mathrm{S}) = x(\mathrm{S})\ _4)\ _2)\ \wedge$

$$(_5\ \forall\ z(\mathrm{S},\ \mathrm{I},\ \mathrm{F})\ (_6\ (_7\ \neg(_8\ \mathtt{Regno}(z)\ _8)\ _7)\ \vee\ (_9\ y(\mathrm{I}) \leq z(\mathrm{I})\ _9)\ _6)\ _5)\ _1)$$

f_1 $(_2\ (_3\ \mathtt{Regno}(y)\ _3)\ \wedge\ (_4\ y(\mathrm{S}) = x(\mathrm{S})\ _4)\ _2)\ \wedge$

$$(_5\ \forall\ z(\mathrm{S},\ \mathrm{I},\ \mathrm{F})\ (_6\ (_7\ \neg(_8\ \mathtt{Regno}(z)\ _8)\ _7)\ \vee\ (_9\ y(\mathrm{I}) \leq z(\mathrm{I})\ _9)\ _6)\ _5)$$

f_2 $(_3\ \mathtt{Regno}(y)\ _3)\ \wedge\ (_4\ y(\mathrm{S}) = x(\mathrm{S})\ _4)$

f_3 $\mathtt{Regno}(y)$

f_4 $y(\mathrm{S}) = x(\mathrm{S})$

f_5 $\forall\ z(\mathrm{S},\ \mathrm{I},\ \mathrm{F})\ (_6\ (_7\ \neg(_8\ \mathtt{Regno}(z)\ _8)\ _7)\ \vee\ (_9\ y(\mathrm{I}) \leq z(\mathrm{I})\ _9)\ _6)$

f_6 $(_7\ \neg(_8\ \mathtt{Regno}(z)\ _8)\ _7)\ \vee\ (_9\ y(\mathrm{I}) \leq z(\mathrm{I})\ _9)$

f_7 $(_7\ \neg(_8\ \mathtt{Regno}(z)\ _8)$

f_8 $\mathtt{Regno}(z)$

f_9 $y(\mathrm{I}) \leq z(\mathrm{I})$

In generale, l'albero sintattico di una componente $f_\mathbf{i}$ di f è un albero ordinato $\mathcal{A}_\mathbf{i}$ con radice il nodo **i** e con etichette sui nodi così definito:

1. Se $f_\mathbf{i}$ è una formula atomica, allora $\mathcal{A}_\mathbf{i}$ ha un unico nodo ed il nodo **i** è etichettato con $f_\mathbf{i}$.

2. Siano $\mathcal{A}_\mathbf{h}$ e $\mathcal{A}_\mathbf{j}$ gli alberi sintattici di due componenti $f_\mathbf{h}$ ed $f_\mathbf{j}$ di f.

   a) Se $f_\mathbf{i}$ è della forma $\neg(f_\mathbf{h})$ oppure $\exists\ y(R)\ (f_\mathbf{h})$ oppure $\forall\ y(R)\ (f_\mathbf{h})$, allora la radice **i** di $\mathcal{A}_\mathbf{i}$ è etichettata rispettivamente con il simbolo "$\neg$" oppure con il simbolo "$\exists\ y(R)$" oppure con il simbolo "$\forall\ y(R)$" ed ha un solo figlio: la radice **h** di $\mathcal{A}_\mathbf{h}$.

   b) Se $f_\mathbf{i}$ è della forma $(f_\mathbf{h}) \wedge (f_\mathbf{j})$ oppure $(f_\mathbf{h}) \vee (f_\mathbf{j})$, allora la radice **i** di $\mathcal{A}_\mathbf{i}$ è etichettata rispettivamente con il simbolo "$\wedge$" oppure con il simbolo "$\vee$" ed ha due figli: il "primo" è la radice **h** di $\mathcal{A}_\mathbf{h}$, ed il "secondo" è la radice **j** di $\mathcal{A}_\mathbf{j}$.

Una volta completata la costruzione dell'albero sintattico $\mathcal{A}$ di f (e ammesso che la cosa vada a buon fine pena il rigetto della domanda), il compilatore procede al riconoscimento di f come formula ben formata. A questo scopo, i nodi di $\mathcal{A}$ vengono esaminati ordinatamente, dalle foglie alla radice. In questo modo, si verifica che ogni singola componente $f_\mathbf{i}$ di f sia una formula ben formata, nel qual caso viene calcolato il tipo di ogni variabile reale in $f_\mathbf{i}$.
Infine, si controlla che

— x sia l'unica variabile reale in f_0,

— f sia una formula aperta e ben formata,

— se x è parzialmente strutturata in f_0, allora il tipo di x in f_0 sia un sottoinsieme di R, oppure

se x è ben strutturata in f_0, allora il tipo di x in f_0 coincida con R.

2.4 Valutazione delle espressioni del Calcolo Relazionale

Sia $E = \{x(R) \mid f\}$ un'espressione del Calcolo Relazionale sulla base di dati $\mathcal{D}$, e sia d uno stato di $\mathcal{D}$. Il *valore* dell'espressione E calcolato *su* d è l'insieme delle ennuple t su R che, una volta sostituite alle occorrenze libere di x in f, rendono la formula f vera. Resta da chiarire a questo punto come viene valutata la formula f quando le occorrenze libere della variabile x vengono sostituite con un'ennupla t su R.

In generale, parleremo di "sostituzione" di una variabile reale e di "interpretazione" di una formula chiusa.

SOSTITUZIONE. Sia f una qualsiasi formula aperta e ben formata, sia x una variabile reale di f. Se x è ben strutturata in f, allora potremo sostituire le occorrenze libere di x in f con una qualsiasi ennupla appartenente al dominio di $T_f(x)$. Se invece x è solo parzialmente strutturata in f, allora potremo sostituire le occorrenze libere di x in f con una qualsiasi ennupla appartenente al dominio di un qualsiasi sottoinsieme di U che contenga $T_f(x)$. Indichiamo con $\mathcal{X}$ l'insieme delle ennuple che possono essere sostituite ad x. La *sostituzione* di un'occorrenza libera di x in f con un'ennupla t appartenente ad $\mathcal{X}$ produce una formula, che indicheremo con $f(x)|_{x=t}$, la quale si ottiene modificando ogni formula atomica g in f che contenga un'occorrenza libera di x in f nella maniera seguente:

> Se g = `Tab`(x) allora g viene sostituito con il valore VERO se t appartiene alla relazione di nome `Tab` contenuta in d; altrimenti g viene sostituito con il valore FALSO.
>
> Se $g = x(A)\theta y(B)$ allora g viene sostituito con $t(A)\theta y(B)$. Lo stesso vale per $y(B)\theta x(A)$. Ne caso particolare che $g = x(A)\theta x(B)$, allora g viene sostituito con il valore VERO se $t(A)\theta t(B)$, e con il valore FALSO in caso contrario.
>
> Se $g = x(A)\theta a$ allora g viene sostituito con il valore VERO se $t(A)\theta a$, e con il valore FALSO in caso contrario; analogamente, se $g = a\theta x(A)$.

Dopo aver sostituito tutte le occorrenze libere di ogni variabile reale di f, si ottiene una formula chiusa che, se non fosse per la presenza delle costanti logiche VERO e FALSO, sarebbe ancora (e verrà trattata nel seguito alla stregua di) una formula ben formata.

Esempio 2.1 (seguito). Consideriamo la formula $f(x)$

$$\forall\, y(\texttt{A\#}, \texttt{parte\_di}, \texttt{nome})\ (g(x, y))$$

dove $g(x, y)$ è la formula

$$\texttt{Inventario}(y) \wedge y(\texttt{parte\_di}) = 211 \wedge\ y(\texttt{nome}) = x(\texttt{nome}).$$

Se t è il valore fodera per poltrona dell'attributo NOME, allora $f(x)|_{x=t}$ è

$$\forall\, y(\texttt{A\#}, \texttt{parte\_di}, \texttt{nome})\ (g(x, y)|_{x=t})$$

dove $g(x, y)|_{x=t}$ è

$$\texttt{Inventario}(y) \wedge y(\texttt{parte\_di}) = 211 \wedge\ y(\texttt{nome}) = \text{fodera per poltrona}.$$

Consideriamo poi la formula $g(x, y)|_{x=t}$. Questa contiene occorrenze libere solo della variabile y, che possiamo sostituire ad esempio con una terna t' appartenente al dominio dell'insieme degli attributi {A#, parte_di, nome}. Se prendiamo t' = (216, 211, cintura di sicurezza) allora $g(x, y)|_{x=t,y=t'}$ è

VERO ∧ VERO ∧ FALSO

INTERPRETAZIONE. Sia f una formula chiusa e ben formata in cui possono essere presenti le costanti logiche VERO e FALSO. L'*interpretazione* di f consiste nell'assegnare ad f un valore di verità VERO e FALSO; tale valore sarà indicato con $val[f]$ ed è definito ricorsivamente alla maniera seguente:

1. Se f è una costante logica, allora $val[f] = f$.

2. Se $f = (g)$, allora $val[f] = val[g]$.

3. Se $f = \neg(g)$ allora $val[f] = \neg\, val[g]$.

4. Se $f = (g) \wedge (h)$ oppure $f = (g) \vee (h)$, allora nel primo caso si ha

$$val[f] = val[g] \wedge val[h],$$

e nel secondo

$$val[f] = val[g] \vee val[h].$$

5. Se $f = \exists x(R)(g)$ oppure $f = \forall x(R)(g)$ (sicché x è l'unica variabile reale di g) allora nel primo caso si ha

$$val[f] = \vee_{t \in DOM(R)} \quad val[g(x)|_{x=t}]$$

e nel secondo

$$val[f] = \wedge_{t \in DOM(R)} \quad val[g(x)|_{x=t}].$$

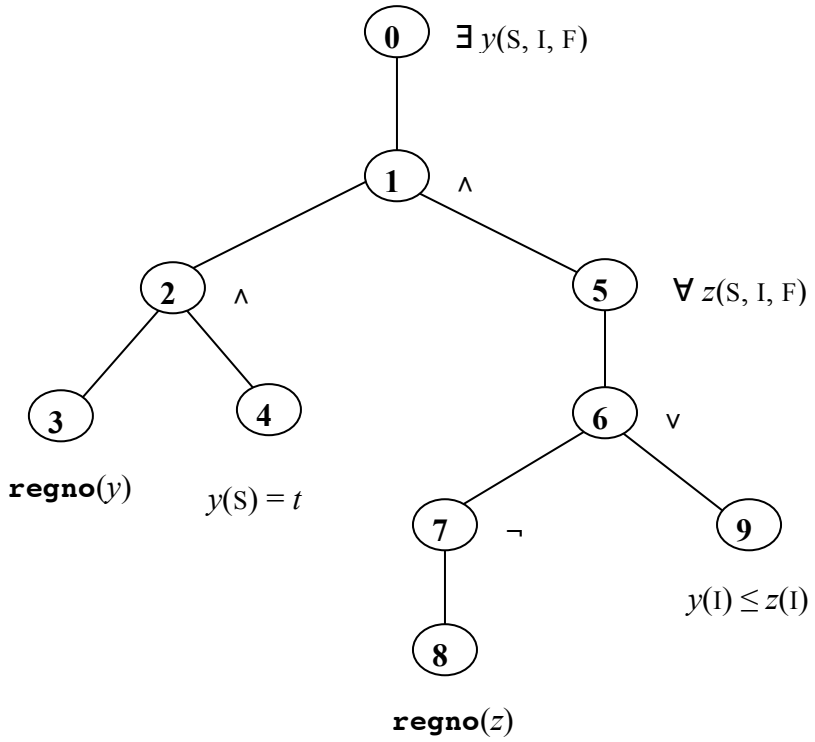
A questo punto sappiamo come valutare, dato uno stato d della base di dati $\mathcal{D}$ ed un'espressione $E = \{x(R)|f\}$, la formula $f(x)|_{x=t}$ dove t è una qualsiasi ennupla appartenente a $DOM(R)$. Chiamiamo *valore* di E su d, che indichiamo con $E(d)$, l'insieme (eventualmente infinito) delle ennuple t appartenenti a $DOM(R)$ per le quali $val[f(x)|_{x=t}] = \text{VERO}$, cioè

$$E(d) = \{t \in DOM(R) : val[f(x)|_{x=t}] = \text{VERO}\}.$$

Consideriamo ancora l'espressione $E = \{x(\text{SOVRANO}) \mid f\}$ dove f è *l*a formula

$$\exists\, y(\texttt{sovrano}, \texttt{inizio}, \texttt{fine})$$
$$(((\texttt{Regno}(y)) \wedge (y(\texttt{sovrano}) = x(\texttt{sovrano}))) \wedge$$
$$(\forall\, z(\texttt{sovrano}, \texttt{inizio}, \texttt{fine})$$
$$(((\neg(\texttt{Regno}(z))) \vee (y(\texttt{inizio}) \leq z(\texttt{inizio})))))$$

Assegnato uno stato d della base di dati, sia r la relazione di nome Regno contenuta in d. Per calcolare il valore $val[f(x)|_{x=t}]$ andremo ad eseguire la seguente procedura. Innanzitutto, il valore a viene sostituito a tutte le occorrenze libere della variabile x nelle etichette dei nodi dell'albero sintattico $\mathcal{A}$ di E.



Quindi, ad ogni nodo di $\mathcal{A}$ viene associata una o più espressioni con le regole d'interpretazione:

nodo i	lista di istruzioni associate al nodo i:	
$i = 0$	(0.1)	v_0 := FALSO.
	(0.2)	Per ogni ennupla t_1 su {S, I, F}
	(0.3)	$v_0 := v_0 \vee v_1(t_1)$
$i = 1$	(1.1)	$v_1(t_1) := v_2(t_1) \wedge v_5(t_1)$
$i = 2$	(2.1)	$v_2(t_1) := v_3(t_1) \wedge v_4(t_1)$
$i = 3$	(3.1)	$v_3(t_1) := (t_1 \in r)$
$i = 4$	(4.1)	$v_4(t_1) := (t_1(\mathrm{S}) = t)$
$i = 5$	(5.1)	$v_5(t_1)$:= VERO
	(5.2)	Per ogni ennupla t_2 su {S, I, F}
	(5.3)	$v_5(t_1) := v_5(t_1) \wedge v_6(t_1, t_2)$
$i = 6$	(6.1)	$v_6(t_1, t_2) := v_7(t_2) \vee v_9(t_1, t_2)$

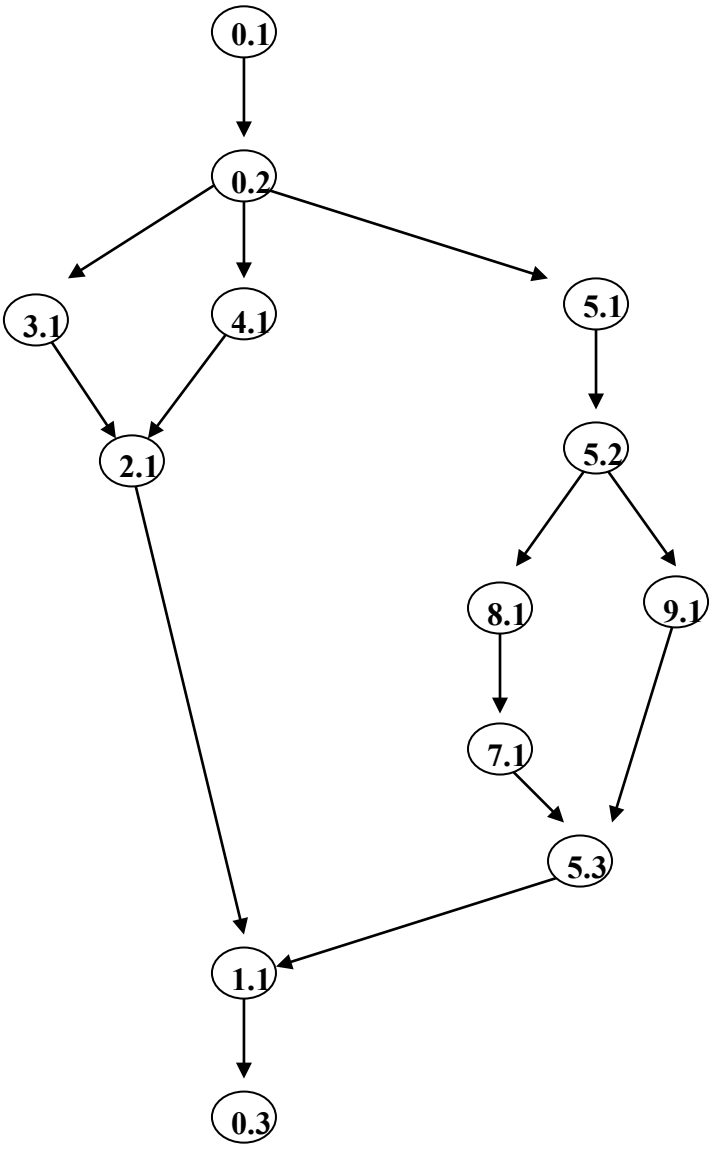
$i = 7$ (7.1) $v_7(t_2) := \neg\ v_8(t_2)$

$i = 8$ (8.1) $v_8(t_2) := (t_2 \in r)$

$i = 9$ (10.1) $v_9(t_1, t_2) := (t_1(\mathrm{I}) \leq t_2(\mathrm{I}))$

Come si vede, tra le diverse istruzioni esistono delle relazioni di precedenza; ad esempio, l'istruzione (0.2) deve seguire l'istruzione (0.1) ma deve precedere tutte le istruzioni che si avvalgono della sostituzione $y = t_1$; inoltre, l'istruzione (1.1) deve seguire l'istruzione (2.1) e le istruzioni (5.1), (5.2) e (5.3) perché richiede il calcolo preventivo di $v_2(t_1)$ e di $v_5(t_1)$ e così via.

Le relazioni di precedenza possono essere tradotte graficamente in un nuovo albero orientato.



Se elenchiamo le istruzioni rispettandone l'ordine di precedenza, otteniamo un programma la cui esecuzione fornirà il valore $val[f(x)|_{x=t}]$.

(1) v_0 := FALSO;

(2) per ogni ennupla t_1 su {S, I, F}

(2.1) $v_3(t_1) := (t_1 \in r)$;

(2.2) $v_4(t_1) := (t_1(\mathrm{S}) = t)$;

(2.3) $v_2(t_1) := v_3(t_1) \wedge v_4(t_1)$;

(2.4) $v_5(t_1)$:= VERO;

(2.5) per ogni ennupla t_2 su {S, I, F}

(2.5.1) $v_8(t_2) := (t_2 \in r)$;

(2.5.2) $v_7(t_2) := \neg\, v_8(t_2)$;

(2.5.3) $v_9(t_1, t_2) := (t_1(\mathrm{I}) \leq t_2(\mathrm{I}))$;

(2.5.4) $v_6(t_1, t_2) := v_7(t_2) \vee v_9(t_1, t_2)$;

(2.5.5) $v_5(t_1) := v_5(t_1) \wedge v_6(t_1, t_2)$;

(2.6) $v_1(t_1) := v_2(t_1) \wedge v_5(t_1)$;

(2.7) $v_0 := v_0 \vee v_1(t_1)$.

Alcune semplici euristiche possono applicarsi per ridurre la quantità di calcolo necessario. Innanzitutto, dopo l'ultima istruzione possiamo aggiungere

se v_0 = VERO allora Uscire dal ciclo (2)

Analogamente, dopo l'istruzione

$v_5(t_1) := v_5(t_1) \wedge v_6(t_1, t_2)$

possiamo aggiungere

se $v_5(t_1)$ = FALSO allora Uscire dal ciclo (2.5).

Un'ulteriore semplificazione si ottiene per una formula chiusa del tipo

$f = \exists y(S)\ (\mathtt{Tab}(s) \wedge g(s))$

allora, indicato con r la relazione di nome `Tab` contenuta nello stato d della base di dati, per il valore $val[f]$ abbiamo

$$val[f] = \vee_{t \in DOM(S)} \ (val[\texttt{Tab}(y)|_{y=t}] \wedge val[g(y)|_{y=t}])$$

$$= (\vee_{t \in r} \ (val[\texttt{Tab}(y)|_{y=t}] \wedge val[g(y)|_{y=t}])) \vee (\vee_{t \in DOM(S)-r} \ (val[\texttt{Tab}(y)|_{y=t}] \wedge \wedge val[g(y)|_{y=t}]))$$

$$= (\vee_{t \in r} \ (val[\text{VERO}] \wedge val[g(y)|_{y=t}])) \vee (\vee_{t \in DOM(S)-r} \ (val[\text{FALSO}] \wedge \wedge val[g(y)|_{y=t}]))$$

$$= (\vee_{t \in r} \ (\text{VERO} \wedge val[g(y)|_{y=t}])) \vee (\vee_{t \in DOM(S)-r} \ (\text{FALSO} \wedge val[g(y)|_{y=t}]))$$

$$= (\vee_{t \in r} \ val[g(y)|_{y=t}]) \vee (\vee_{t \in DOM(S)-r} \ \text{FALSO})$$

$$= (\vee_{t \in r} \ val[g(y)|_{y=t}]) \vee \text{FALSO}$$

$$= \vee_{t \in r} \ val[g(y)|_{y=t}]$$

Se invece la formula chiusa è del tipo

$$f = \forall y(S) \ (\neg \texttt{Tab}(y) \vee g(y))$$

allora, indicato con r la relazione di nome `Tab` contenuta nello stato d della base di dati, l'interpretazione di f si semplifica come segue

$$I[f] = \wedge_{t \in DOM(S)} \ (\neg \ val[\texttt{Tab}(y)|_{y=t}] \vee I[g(y)|_{y=t}]) =$$

$$= (\wedge_{t \in r} \ (\neg \ val[\texttt{Tab}(y)|_{y=t}] \vee val[g(y)|_{y=t}])) \wedge (\wedge_{t \in DOM(S)-r} \ (\neg \ val[\texttt{Tab}(y)|_{y=t}] \vee val[g(y)|_{y=t}]))$$

$$= (\wedge_{t \in r} \ (val[\text{FALSO}] \vee val[g(y)|_{y=t}])) \wedge (\wedge_{t \in DOM(S)-r} \ (val[\text{VERO}] \vee \vee val[g(y)|_{y=t}]))$$

$$= (\wedge_{t \in r} \ (\text{FALSO} \vee val[g(y)|_{y=t}])) \wedge (\wedge_{t \in DOM(R)-r} \ (\text{VERO} \vee val[g(y)|_{y=t}]))$$

$$= (\wedge_{t \in r} \ val[g(y)|_{y=t}]) \wedge (\wedge_{t \in DOM(S)-r} \ \text{VERO})$$

$$= (\wedge_{t \in r} \ val[g(y)|_{y=t}]) \wedge \text{VERO}$$

$$= \wedge_{t \in r} \ val[g(y)|_{y=t}]$$

Nel nostro esempio, la semplificazione porta al seguente programma:

(1) v_0 := FALSO;

(2) per ogni ennupla t_1 appartenente ad r

(2.1) $v_2(t_1) := (t_1(\text{S}) = t)$;

(2.2) $v_5(t_1) :=$ VERO;

(2.3) per ogni ennupla t_2 appartenente ad r

(2.3.1) $v_6(t_1, t_2) := (t_1(\text{I}) \leq t_2(\text{I}))$;

(2.3.2) $v_5(t_1) := v_5(t_1) \wedge v_6(t_1, t_2)$;

(2.4) $v_1(t_1) := v_2(t_1) \wedge v_5(t_1)$;

(2.5) $v_0 := v_0 \vee v_1(t_1)$.

2.5 Calcolo Relazionale ristretto

Due attributi A e B sono *simili* se hanno lo stesso dominio. Ovviamente, ogni attributo è simile a se stesso. Per semplicità, assumiamo che hanno intersezione vuota i domini di due attributi che non sono simili.

Sia f una formula del CR sulla base di dati $\mathcal{D}$, e sia d uno stato di $\mathcal{D}$. Il *dominio attivo* di un attributo A per f su d è l'insieme degli elementi a del dominio di A tali che

— per ogni nome `Tab` di tabella relazionale presente in f, esiste un'ennupla t nella relazione di nome `Tab` contenuta in d per cui $t(B) = a$ dove B è un qualsiasi attributo simile ad A, oppure

— f contiene una formula atomica del tipo

$x(B)\ \theta\ a$ $\qquad$ $a\ \theta\ x(B)$

dove B è un qualsiasi attributo simile ad A.

Il *dominio attivo* di un insieme R di attributi per f su d è l'insieme delle ennuple t su R tali che, per ogni $A \in R$, $t(A)$ appartiene al dominio attivo di A (per f su d).

Nella base di dati `STUART`, gli attributi

`sovrano` $\qquad$ `nome`

sono simili così come lo sono gli attributi

`inizio` $\qquad$ `fine` $\qquad$ `nascita` $\qquad$ `morte`,

sicché nella formula f

`Dinastia`$(x) \wedge x($`sesso`$) = \text{M} \wedge (\exists y($`sovrano, inizio, fine`$)$ (`Regno`$(y) \wedge y($`sovrano`$) = x($`nome`$))$

il dominio attivo dell'attributo `sovrano` per f su d contiene non solo tutti i nomi dei sovrani riportati nella relazione di nome `Regno` contenuta in d ma anche quelli di tutti i membri della dinastia, riportati nella relazione di nome `Dinastia` contenuta in d. Inoltre, $DOM^*(\{\texttt{sovrano}, \texttt{inizio}, \texttt{fine}) f, d)$ contiene sì le ennuple riportate nella relazione di nome `Regno` contenuta in d ma, più in generale, tutte le ennuple che si ottengono combinando gli elementi dei domini attivi di `sovrano`, `inizio` e `fine` per f su d.

Orbene tutte le sostituzioni di una variabile definita su R si ottengono considerando le sole ennuple appartenenti a $DOM^*(R; f, d)$. In questo modo, ad una formula chiusa f viene assegnato ancora un valore di verità, che indichiamo con $val^*[f]$, che si ottiene in maniera analoga a $val[f]$. Chiameremo questa l'*interpretazione restrittiva* di f. Infine, il *valore in senso stretto* di un'espressione $E = \{x(R) \mid f\}$ del CR sullo stato d di $\mathcal{D}$, che indichiamo con $E^*(d)$, è così definito

$$E^*(d) = \{t \in DOM^*(R; f, d): val^*[f(x)|_{x=t}] = \text{VERO}\}.$$

Si osservi che $E^*(d)$ è sempre un sottoinsieme di $E(d)$ ed è sempre una relazione su R (cioè contiene un numero finito di ennuple). L'insieme delle espressioni E del CR su $\mathcal{D}$ tali che $E^*(d) = E(d)$ per ogni stato d di $\mathcal{D}$ definisce il cosiddetto *calcolo relazionale ristretto* (su $\mathcal{D}$).

CAPITOLO 3

LINGUAGGIO D'INTERROGAZIONE PROCEDURALE

L'interrogazione di una base di dati avviene formulando le domande mediante *espressioni algebriche*. A tale scopo, vengono usati degli operatori, detti *operatori relazionali*, ognuno dei quali ha come argomento una o più relazioni e calcola una relazione.

3.1 Algebra relazionale

I primi operatori relazionali che consideriamo scaturiscono dalla natura insiemistica delle relazioni e sono gli operatori insiemistici di *unione* ($\cup$), *intersezione* ($\cap$) e *differenza* (–). Oltre agli operatori insiemistici, esistono specifici operatori relazionali che ora introduciamo.

SELEZIONE. L'operatore di selezione si applica ad una relazione e ne genera un sottoinsieme. Sia r una relazione con schema R; una "formula proposizionale" p su R è definita ricorsivamente a partire da "proposizioni atomiche" e dai connettivi logici. Come al solito, indichiamo con Θ l'insieme delle operazioni di confronto.

Sia θ una operazione in Θ, siano A e B due attributi in R che siano θ-confrontabili, e sia a un elemento del domino di A; le espressioni $A\ \theta\ B$ e $A\ \theta\ a$ sono *proposizioni atomiche su R*.

Ogni proposizione atomica è una *formula proposizionale*. Se p e q sono formule proposizionali su R, allora lo sono anche $\neg(p)$, $(p) \wedge (q)$ e $(p) \vee (q)$.

Sia t un'ennupla appartenente a $DOM(R)$ e sia p una formula proposizionale su R; l'interpretazione di p relativa a t è una delle due costanti logiche che indichiamo con $p(t)$ e definiamo alla maniera seguente. Se $p = A\ \theta\ B$, allora $p(t) =$ VERO se e solo se la condizione $t(A)\ \theta\ t(B)$ è vera; analogamente, se $p = A\ \theta\ a$, allora $p(t) =$ VERO se e solo se la condizione $t(A)\ \theta\ a$ è vera.
La *selezione* su r sotto la condizione p dà come risultato ancora una relazione con schema R, che indicheremo con $\sigma_p(r)$, così definita

$$\sigma_p(r) = \{t \in r : p(t) = \text{VERO}\}.$$

È facile dimostrare che la selezione è

— commutativa rispetto alla composizione, cioè $\sigma_p(\sigma_q(r)) = \sigma_q(\sigma_p(r))$,

— distributiva rispetto all'unione ed all'intersezione:

$$\sigma_p(r \cup s) = \sigma_p(r) \cup \sigma_p(s) \qquad \sigma_p(r \cap s) = \sigma_p(r) \cap \sigma_p(s).$$

Nel seguito, faremo spesso uso della abbreviazione $\sigma_{p,q}(r)$ per $\sigma_p(\sigma_q(r))$.

PROIEZIONE. L'operatore di proiezione si applica ad una relazione e ne genera un'altra che ha per schema un sottoinsieme dello schema della relazione originaria. Sia r una relazione con schema R e sia X un sottoinsieme proprio di R. La *proiezione* di r su X è la relazione con schema X, che indicheremo con $\pi_X(r)$, formata dalle restrizioni ad X delle ennuple in r, cioè

$$\pi_X(r) = \{t \in DOM(X) : \exists\, t' \in r \text{ tale che } t'[X] = t\}.$$

PRODOTTO CARTESIANO. A differenza della proiezione, il prodotto cartesiano si applica ad una coppia di relazioni ed il risultato è ancora una relazione. Siano r_1 ed r_2 due relazioni rispettivamente con schemi R_1 ed R_2. Sia $C = R_1 \cap R_2$; per ogni attributo A in C, introduciamo due copie A' e A'' di A e sia $C' = \{A' : A \in C\}$ e $C'' = \{A'' : A \in C\}$. Il *prodotto cartesiano* di r_1 ed r_2 è una relazione con schema $R = (R_1 - C) \cup (R_2 - C) \cup C' \cup C''$ che indicheremo con $r_1 \times r_2$, così definita

$$r_1 \times r_2 = \{t \in DOM(R) : \exists\, t_1 \in r_1 \text{ ed } \exists\, t_2 \in r_2 \text{ tali che}$$

$$t[R_1 - C] = t_1[R_1 - C],\ t[R_2 - C] = t_2[R_2 - C] \text{ e}$$

$$\text{per ogni } A \in C$$

$$t[A'] = t_1[A] \text{ e } t[A''] = t_2[A]\}.$$

Il prodotto cartesiano gode della proprietà commutativa e di quella associativa. Quest'ultima proprietà permette di definire senza ambiguità il prodotto cartesiano di tre o più relazioni.

RIDENOMINAZIONE. La ridenominazione di un attributo A prevede l'introduzione di un nuovo attributo B che non è presente nell'universo del discorso; per effetto della ridenominazione di A, il dominio di B è uguale al dominio di A. Siano ora r una relazione con schema R e A un attributo in R. Allora, r con A *ridenominato* B è una relazione con schema $R - \{A\} \cup \{B\}$, che indicheremo con $\rho_{B:=A}(r)$, così definita

$$\rho_{B:=A}(r) = \{t \in DOM(R - \{A\} \cup \{B\}) : \exists\, t' \in r \text{ tale che}$$
$$t'[R - \{A\}] = t[R - \{B\}] \wedge t'(A) = t(B)\}.$$

Più in generale, la ridenominazione assume la forma

$$\rho_{B_1:=A_1,\ \ldots,\ B_k:=A_k}(r)$$

che sta per

$$\rho_{B_1:=A_1}(\ldots(\rho_{B_k:=A_k}(r)\ \ldots)\ .$$

Utilizzando gli operatori relazionali già introdotti, se ne derivano altri che sono molto utili nella formulazione di espressioni algebriche. Questi sono l'operatore *theta-join*, l'operatore *join naturale* e la *divisione*.

THETA-JOIN. Siano r_1 ed r_2 due relazioni rispettivamente con schemi R_1 ed R_2, e sia

$$p = A_1\theta_1B_1,\ \ldots,\ A_k\theta_kB_k$$

dove

$A_1, \ldots, A_k$ sono attributi in R_1,

$B_1, \ldots, B_k$ sono attributi in R_2 e

per ogni h ($1 \leq h \leq k$), gli attributi A_h e B_h sono θ_h-confrontabili, dove θ_h è un'operazione di confronto nell'insieme Θ.

Il *prodotto theta-join* (θ-*join*) di r_1 ed r_2 è una relazione con lo stesso schema della relazione $r_1 \times r_2$, che indicheremo con $r_1 \bowtie_p r_2$ ed è così definita

$$r_1 \bowtie_p r_2 = \sigma_p(r_1 \times r_2)\ .$$

Un caso particolare del prodotto theta-join sia ha quando ogni θ_h ($1 \leq h \leq k$) è l'eguaglianza; allora, il prodotto theta-join prende il nome di *prodotto equi-join*.

JOIN NATURALE. Una variante dell'operatore theta-join è l'operatore *join* (*naturale*). Siano r_1 ed r_2 due relazioni rispettivamente con schemi R_1 ed R_2; il *prodotto join* di r_1 ed r_2 è una relazione con schema $R = R_1 \cup R_2$, che indicheremo con $r_1 \bowtie r_2$ ed è così definita

$$r_1 \bowtie r_2 = \{t \in DOM(R) : \exists\ t_1 \in r_1 \text{ ed } \exists\ t_2 \in r_2 \text{ tali che } t_1 = t[R_1] \text{ e } t_2 = t[R_2]\}.$$

Si osservi che il prodotto join di due relazioni si può esprimere utilizzando gli operatori di prodotto equi-join, di ridenominazione e di proiezione.

Andiamo a calcolare il numero di ennuple contenute nella relazione $r_1 \bowtie r_2$. Se $R_1 \cap R_2 = \varnothing$, allora $r_1 \bowtie r_2$ coincide con il prodotto cartesiano di r_1 ed r_2 e quindi $|r_1 \bowtie r_2| = |r_1| \cdot |r_2|$. Consideriamo ora il caso che $R_1 \cap R_2 \neq \varnothing$. Siano $X = R_1 \cap R_2$, $Y = R_1 - R_2$ e $Z = R_2 - R_1$. Allora,

$$r_1 \bowtie r_2 = \{(x, y, z) : x \in \pi_X(r_1) \cap \pi_X(r_2), y \in \pi_Y(\sigma_{X=x}(r_1)), z \in \pi_Z(\sigma_{X=x}(r_2))\}$$

e, siccome $|\pi_Y(\sigma_{X=x}(r_1))| = |\sigma_{X=x}(r_1)|$ e $|\pi_Z(\sigma_{X=x}(r_2))| = |\sigma_{X=x}(r_2)|$, abbiamo

$$|r_1 \bowtie r_2| = \sum_{x \in \pi_X(r_1) \cap \pi_X(r_2)} |\sigma_{X=x}(r_1)| \cdot |\sigma_{X=x}(r_2)| .$$

Si osservi che, nel caso che $R_1 \subseteq R_2$, si ha che

$$r_1 \bowtie r_2 = \{t \in r_2 : t[X] \in r_1\}$$

e quindi

$$|r_1 \bowtie r_2| = \sum_{x \in r_1} |\sigma_{X = x}(r_2)| .$$

Come il prodotto cartesiano, anche il prodotto join gode della proprietà commutativa e di quella associativa. Quest'ultima proprietà permette di definire senza ambiguità il prodotto join di tre o più relazioni $r_1, r_2, \ldots, r_n$. Dal punto di vista computazionale, possiamo calcolare il join $r_1 \bowtie r_2 \bowtie \ldots \bowtie r_n$ secondo l'*procedura join sequenziale*

$$((r_{i_1} \bowtie r_{i_2}) \bowtie \ldots) \bowtie r_{i_n}$$

dove $(r_{i_1}, r_{i_2}, \ldots, r_{i_n})$ è una qualsiasi permutazione di $(r_1, r_2, \ldots, r_n)$; allora, le relazioni

$$r_{i_1} \bowtie r_{i_2} \qquad (r_{i_1} \bowtie r_{i_2}) \bowtie r_{i_3} \qquad \ldots \qquad ((r_{i_1} \bowtie r_{i_2}) \bowtie \ldots) \bowtie r_{i_{n-1}}$$

sono chiamate relazioni "intermedie" della procedura join sequenziale. Per evitare che le dimensioni delle relazioni intermedie siano esageratamente maggiori della dimensione del risultato del join, è buona regola scegliere la permutazione $(r_{i_1}, r_{i_2}, \ldots, r_{i_n})$ in maniera tale che

— la relazione r_{i_1} sia tra quelle che hanno il minimo numero di ennuple, e

— per ogni $h > 1$, r_{i_h} sia tra quelle per cui l'intersezione dello schema di r_{i_h} con l'unione degli schemi $r_{i_1}, \ldots, r_{i_{h-1}}$ sia massima.

Il vantaggio di questa euristica è che occorre tenere traccia di una singola relazione intermedia per volta; inoltre, talora si riesce a ridurre al minimo le dimensioni delle relazioni temporanee. Quello che non è garantito che le relazioni temporanee abbiano dimensioni non superiori alla dimensione del risultato, cioè della relazione $r_1 \bowtie r_2 \bowtie \ldots \bowtie r_n$. Ad esempio, consideriamo il join $r_1 \bowtie r_2 \bowtie r_3$, dove le relazioni r_1, r_2 ed r_3 hanno rispettivamente schema $R_1 = \{A, B\}$, $R_2 = \{A, C\}$ e $R_3 = \{B, C\}$. Assumiamo che A, B e C siano attributi binari e che $r_i = \{(0, 1), (1, 0)\}(i = 1, 2, 3)$. Con la procedura join sequenziale

$(r_1 \bowtie r_2) \bowtie r_3$

la relazione intermedia $r_1 \bowtie r_2$ (che ha schema $\{A, B, C\}$) ha lo stesso numero di ennuple di r_1; esplicitamente, si ha $r_1 \bowtie r_2 = \{(0, 1, 1), (1, 0, 0)\}$. Comunque, il join delle tre relazioni è una relazione vuota (con schema $\{A, B, C\}$).

DIVISIONE. Siano r ed s due relazioni rispettivamente con schemi R ed S, dove S è un sottoinsieme proprio di R. La *divisione* di r per s dà come risultato la relazione con schema $R–S$, che indicheremo con $r \div s$, la quale è così definita:

$$r \div s = \{t \in DOM(R–S) : \{t\} \times s \subseteq r\} =$$

$$= \{t \in DOM(R–S) : \forall\, t' \in s\ \exists\, t'' \in r\ (t''[R–S] = t\ \&\ t''[S] = t')\}.$$

La relazione $r \div s$ è chiamata il *quoziente* della divisione di r per s. Essa contiene tutte e sole le ennuple t su $R–S$ tali che t appartiene a $\pi_{R–S}(r)$ e t non appartiene alla relazione $\pi_{R–S}((\pi_{R–S}(r) \bowtie s) – r)$. Pertanto, la relazione $r \div s$ può essere anche definita in termini degli operatori già visti nei due modi seguenti

$$r \div s = \pi_{R–S}(r) – \pi_{R–S}((\pi_{R–S}(r) \bowtie s) – r)$$

$$r \div s = \bigcap_{v \in s}\ \pi_{R–S}(\sigma_{S=v}(r)).$$

3.2 Linguaggio Algebrico

Data una base di dati relazionale $\mathcal{D}$, per definire un'espressione algebrica abbiamo bisogno di

- Θ un insieme di operazioni di confronto tra valori di attributi che comprende almeno l'eguaglianza e la diseguaglianza, le quali sono definite per il dominio di ogni attributo in $\mathcal{D}$,
- O l'insieme dei seguenti operatori relazionali

  unione,
  intersezione,
  differenza,
  proiezione,
  prodotto join,
  divisione,
  ridenominazione,
  selezione con operazioni in Θ,
  connettivi logici,

prodotto theta-join con operazioni in Θ.

Daremo ora la definizione di un'*espressione algebrica* e del suo *tipo* che è di natura ricorsiva.

E1. Se `Tab` è il nome di una tabella relazionale presente in $\mathcal{D}$, allora `Tab` è un'espressione algebrica e il suo tipo è lo schema della tabella relazionale.

E2. Se r è una costante relazionale con schema R, allora r è un'espressione algebrica ed il suo tipo è R.

E3. Se E è un'espressione algebrica con schema R e p è una formula proposizionale su R, allora $\sigma_p(E)$ è un'espressione di tipo R.

E4. Se E_1 ed E_2 sono espressioni algebriche entrambe di tipo R, allora

$(E_1) \cup (E_2)$ è un'espressione algebrica di tipo R,

$(E_1) \cap (E_2)$ è un'espressione algebrica di tipo R,

$(E_1) - (E_2)$ è un'espressione algebrica di tipo R.

E5. Se E è un'espressione algebrica di tipo R ed X è una lista di attributi distinti appartenenti ad R, allora

$\pi_X (E)$

è un'espressione algebrica di tipo X.

E6. Se E_1 ed E_2 sono espressioni algebriche di tipo R_1 ed R_2 e se R_2 è un sottoinsieme proprio di R_1, allora

$(E_1) \div (E_2)$

è un'espressione algebrica di tipo $R_1 - R_2$.

E7. Se E_1 ed E_2 sono espressioni algebriche di tipo R_1 ed R_2 allora

$(E_1) \bowtie (E_2)$

è un'espressione algebrica di tipo $R_1 \cup R_2$.
Se poi $R_1 \cap R_2 = \varnothing$ e se $A_1, \ldots, A_k$ sono attributi in R_1 e $B_1, \ldots, B_k$ sono attributi in R_2 tali che gli attributi A_h e B_h sono θ_h-confrontabili, $1 \leq h \leq k$, allora

$(E_1) \bowtie_p (E_2)$,

dove $p = A_1\ \theta_1\ B_1, \ldots, A_k\ \theta_k\ B_k$, è un'espressione algebrica di tipo $R_1 \cup R_2$.

E8. Se E è un'espressione algebrica di tipo R e se l'insieme $\{A_1, \ldots, A_k\}$ è un sottoinsieme di R e l'insieme $\{B_1, \ldots, B_k\}$ ha intersezione vuota con R, allora

$$\rho_{B_1:=A_1, \ldots, B_k:=A_k}(E)$$

è un'espressione algebrica di tipo $R - \{A_1, \ldots, A_k\} \cup \{B_1, \ldots, B_k\}$.

Data la base di dati $\mathcal{D}$ ed una qualsiasi stringa E, sta al compilatore decidere se E è o meno un'espressione dell'Algebra Relazionale (come nel caso del Calcolo Relazionale). A tale scopo, il compilatore prova a costruire l'*albero sintattico* di E. Di nuovo, vengono numerate le coppie di *parentesi connettive* (quelle che racchiudono espressioni secondo le specifiche sintattiche E3 - E8) nella stringa (E). Una volta numerate tali coppie di parentesi, le *componenti* (*strutturali*) di E sono le sottostringhe di E che sono delimitate da una coppia di parentesi che hanno lo stesso indice.

Esempio 3.1 Consideriamo la base di dati STUART e la domanda: "Chi fu il primo sovrano?". Per comodità, indichiamo con S ed I rispettivamente gli attributi sovrano e inizio, un'espressione algebrica E per questa domanda è

$$\pi_S ((\sigma_{I \le A} ((\pi_{S,I} (\text{Regno})) \times (\rho_{R:=S,A:=I} (\pi_{S,I} (\text{Regno}))))) \div (\rho_{R:=S,A:=I} (\pi_{S,I} (\text{Regno}))))$$

La numerazione delle coppie di parentesi connettive dà come risultato

$$(_0\ \pi_S\ (_1\ (_2\ \sigma_{I \le A}\ (_3\ (_4\ \pi_{S,I}\ (_5\ \text{Regno}\ )_5)\ )_4) \times (_6\ \rho_{R:=S,A:=I}\ (_7\ \pi_{S,I}\ (_8\ \text{Regno}\ )_8)\ )_7)\ )_6)\ )_3)\ )_2) \div (_9\ \rho_{R:=S,A:=I}\ (_{10}\ \pi_{S,I}\ (_{11}\ \text{Regno}\ )_{11})\ )_{10})\ )_1)\ )_0)$$

Così, se con E_i indichiamo l'i-esima componente di E, abbiamo

E_0 $\pi_S\ (_1\ (_2\ \sigma_{I \le A}\ (_3\ (_4\ \pi_{S,I}\ (_5\ \text{Regno}\ )_5)\ )_4) \times (_6\ \rho_{R:=S,A:=I}\ (_7\ \pi_{S,I}\ (_8\ \text{Regno}\ )_8)\ )_7)\ )_6)\ )_3)\ )_2) \div (_9\ \rho_{R:=S,A:=I}\ (_{10}\ \pi_{S,I}\ (_{11}\ \text{Regno}\ )_{11})\ )_{10})\ )_9)\ )_1)$

E_1 $(_2\ \sigma_{I \le A}\ (_3\ (_4\ \pi_{S,I}\ (_5\ \text{Regno}\ )_5)\ )_4) \times (_6\ \rho_{R:=S,A:=I}\ (_7\ \pi_{S,I}\ (_8\ \text{Regno}\ )_8)\ )_7)\ )_6)\ )_3)\ )_2) \div (_9\ \rho_{R:=S,A:=I}\ (_{10}\ \pi_{S,I}\ (_{11}\ \text{Regno}\ )_{11})\ )_{10})$

E_2 $\sigma_{I \le A}\ (_3\ (_4\ \pi_{S,I}\ (_5\ \text{Regno}\ )_5)\ )_4) \times (_6\ \rho_{R:=S,A:=I}\ (_7\ \pi_{S,I}\ (_8\ \text{Regno}\ )_8)\ )_7)\ )_6)\ )_3)$

E_3 $(_4\ \pi_{S,I}\ (_5\ \text{Regno}\ )_5)\ )_4) \times (_6\ \rho_{R:=S,A:=I}\ (_7\ \pi_{S,I}\ (_8\ \text{Regno}\ )_8)\ )_7)\ )_6)$

E_4 $\pi_{S,I}\ (_5\ \text{Regno}\ )_5)$

E_5 Regno

E_6 $\rho_{R:=S,A:=I}\ (_7\ \pi_{S,I}\ (_8\ \text{Regno}\ )_8)\ )_7)$

E_7 $\pi_{S,I}$ (**8** Regno **8**)

E_8 Regno

E_9 $\rho_{R:=S,A:=I}$ (**10** $\pi_{S,I}$ (**11** Regno **11**) **10**)

E_{10} $\pi_{S,I}$ (**11** Regno **11**)

E_{11} Regno

Sia $E_\mathbf{i}$ la generica componente di E. L'albero sintattico di $E_\mathbf{i}$ è un albero ordinato $\mathcal{A}_\mathbf{i}$ con radice **i** e con etichette sui vertici così definito:

1. Se $E_\mathbf{i}$ è il nome di una tabella relazionale presente in $\mathcal{D}$ oppure una costante relazionale, allora la radice **i** di $\mathcal{A}_\mathbf{i}$ è etichettata con $E_\mathbf{i}$.

2. Siano $\mathcal{A}_\mathbf{h}$ e $\mathcal{A}_\mathbf{j}$ gli alberi sintattici di due componenti $E_\mathbf{h}$ ed $E_\mathbf{j}$ di E.

   a) Se $E_\mathbf{i}$ è della forma

   $$\sigma_p\,(E_\mathbf{h}) \qquad \pi_X\,(E_\mathbf{h}) \qquad \rho_{B_1:=A_1,\,\ldots,\,B_k:=A_k}\,(E_\mathbf{h})$$

   allora la radice **i** di $\mathcal{A}_\mathbf{i}$ è etichettata rispettivamente con i simboli

   $$\sigma_p \qquad \pi_X \qquad \rho_{B_1:=A_1,\,\ldots,\,B_k:=A_k}$$

   ed ha un solo figlio: la radice **h** di $\mathcal{A}_\mathbf{h}$.

   b) Se $E_\mathbf{i}$ è della forma

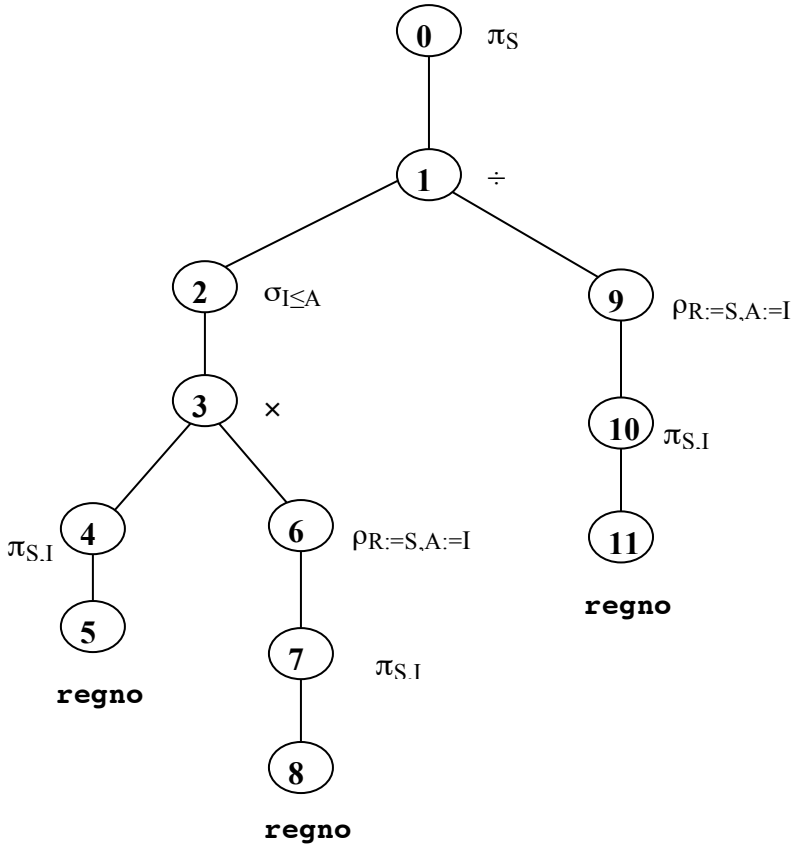
   $$(E_\mathbf{h}) \cup (E_\mathbf{j}) \qquad (E_\mathbf{h}) \cap (E_\mathbf{j}) \qquad (E_\mathbf{h}) - (E_\mathbf{j})$$
   $$(E_\mathbf{h}) \div (E_\mathbf{j}) \qquad (E_\mathbf{h}) \bowtie (E_\mathbf{j}) \qquad (E_\mathbf{h}) \bowtie_p (E_\mathbf{j})$$

   allora la radice **i** di $\mathcal{A}_\mathbf{i}$ è etichettata rispettivamente con i simboli

   $$\cup \qquad \cap \qquad -$$
   $$\div \qquad \bowtie \qquad \bowtie_p$$

   ed ha due figli: il "primo" è la radice **h** di $\mathcal{A}_\mathbf{h}$, ed il "secondo" è la radice **j** di $\mathcal{A}_\mathbf{j}$.

Per l'espressione algebrica dell'Esempio 3.1, l'albero sintattico è quello mostrato in figura.



Se la costruzione dell'albero sintattico $\mathcal{A}$ di E va a compimento, la stringa E è riconosciuta come un'espressione dell'Algebra Relazionale. Altrimenti, la domanda viene rigettata.

3.3 Valutazione di un'espressione algebrica

Sia $d = \{r_1, \ldots, r_n\}$ uno stato della base di dati $\mathcal{D}$, dove r_i è una relazione con schema R_i, e sia E un'espressione algebrica. Il *valore* di E su d è la relazione che ha per schema il tipo di E e che si ottiene applicando gli operatori in E alle relazioni contenute in d i cui nomi figurano in E. Esplicitamente, andremo a valutare le espressioni associate ai vertici dell'albero sintattico $\mathcal{A}$ procedendo dalle foglie alla radice.

Esempio 3.1 (seguito) La relazione che viene associata alle foglie 5, 8 e 11 di $\mathcal{A}$ è la relazione di nome `Regno` contenuta nello stato d della base di dati (v. Tabella 1.6). Così, la relazione associata ai vertici 4, 7 e 10 di $\mathcal{A}$ è la Tabella 3.1.

S	I
Giacomo I	1603
Carlo I	1625
Carlo II	1660
Giacomo II	1685
Maria II	1688
Anna	1702

Tabella 3.1

La relazione associata ai vertici 6 e 9 di $\mathcal{A}'$ è la Tabella 3.2.

R	A
Giacomo I	1603
Carlo I	1625
Carlo II	1660
Giacomo II	1685
Maria II	1688
Anna	1702

Tabella 3.2

Esamineremo infine i restanti vertici di $\mathcal{A}$ nell'ordine seguente: 3, 2, 1 e 0.
La relazione associata al vertice 3 di $\mathcal{A}$ è la Tabella 3.3.

S	I	R	A
Giacomo I	1603	Giacomo I	1603
Giacomo I	1603	Carlo I	1625
Giacomo I	1603	Carlo II	1660
Giacomo I	1603	Giacomo II	1685
Giacomo I	1603	Maria II	1688
Giacomo I	1603	Anna	1702
Carlo I	1625	Giacomo I	1603
Carlo I	1625	Carlo I	1625
Carlo I	1625	Carlo II	1660
Carlo I	1625	Giacomo II	1685
Carlo I	1625	Maria II	1688
Carlo I	1625	Anna	1702
Carlo II	1660	Giacomo I	1603
Carlo II	1660	Carlo I	1625
Carlo II	1660	Carlo II	1660
Carlo II	1660	Giacomo II	1685
Carlo II	1660	Maria II	1688
Carlo II	1660	Anna	1702
Giacomo II	1685	Giacomo I	1603
Giacomo II	1685	Carlo I	1625
Giacomo II	1685	Carlo II	1660
Giacomo II	1685	Giacomo II	1685
Giacomo II	1685	Maria II	1688
Giacomo II	1685	Anna	1702
Maria II	1688	Giacomo I	1603
Maria II	1688	Carlo I	1625

Maria II	1688	Carlo II	1660
Maria II	1688	Giacomo II	1685
Maria II	1688	Maria II	1688
Maria II	1688	Anna	1702
Anna	1702	Giacomo I	1603
Anna	1702	Carlo I	1625
Anna	1702	Carlo II	1660
Anna	1702	Giacomo II	1685
Anna	1702	Maria II	1688
Anna	1702	Anna	1702

Tabella 3.3

Il valore su *d* dell'espressione associata al vertice 2 di $\mathcal{A}$ è la Tabella 3.4.

S	I	R	A
Giacomo I	1603	Giacomo I	1603
Giacomo I	1603	Carlo I	1625
Giacomo I	1603	Carlo II	1660
Giacomo I	1603	Giacomo II	1685
Giacomo I	1603	Maria II	1688
Giacomo I	1603	Anna	1702
Carlo I	1625	Carlo I	1625
Carlo I	1625	Carlo II	1660
Carlo I	1625	Giacomo II	1685
Carlo I	1625	Maria II	1688
Carlo I	1625	Anna	1702
Carlo II	1660	Carlo II	1660
Carlo II	1660	Giacomo II	1685
Carlo II	1660	Maria II	1688
Carlo II	1660	Anna	1702
Giacomo II	1685	Giacomo II	1685
Giacomo II	1685	Maria II	1688
Giacomo II	1685	Anna	1702
Maria II	1688	Maria II	1688
Maria II	1688	Anna	1702
Anna	1702	Anna	1702

Tabella 3.4

Finalmente, possiamo valutare i valori su *d* delle espressioni associate ai vertici 1 e 0 di $\mathcal{A}$. Questi sono rispettivamente la Tabella 3.5 e la Tabella 3.6.

S	I
Giacomo I	1603

Tabella 3.5

S
Giacomo I

Tabella 3.6

Alcune semplici euristiche possono applicarsi per ridurre la quantità di calcolo necessario. Come mostrato nell'Esempio 3.1, possiamo avere componenti uguali, vale a dire che a due distinti vertici i e j di $\mathcal{A}$ può essere associata un'identica sottoespressione di E ($E_{\mathbf{i}} = E_{\mathbf{j}}$), che prende in nome di *sottoespressione comune*. Un'espressione E' è un *sottoespressione comune massima* se esistono due distinti vertici i e j tali che $E' = E_{\mathbf{i}} = E_{\mathbf{j}}$ ed ai padri di i e j sono associate due distinte componenti di E. Così, è conveniente rappresentare la struttura sintattica di E mediante un *albero principale* $\mathcal{A}$ e tanti *alberi ausiliari* quante sono le sottoespressioni comuni massime di E. Ogni albero ausiliario ha la precedenza su $\mathcal{A}$ e, dati due alberi ausiliari $\mathcal{A}'$ e $\mathcal{A}''$, $\mathcal{A}''$ ha la precedenza su $\mathcal{A}'$ se la sottoespressione comune massima associata $\mathcal{A}''$ è una sottoespressione della sottoespressione comune massima associata $\mathcal{A}'$. Finalmente, andremo a valutare le espressioni associate ai vertici degli alberi ausiliari e poi le espressioni associate ai vertici dell'albero principale. In ognuno di questi alberi, la valutazione procede dalle foglie alla radice; inoltre, un albero ausiliario $\mathcal{A}'$ sarà valutato dopo un altro albero ausiliario $\mathcal{A}''$ se $\mathcal{A}''$ ha precedenza su $\mathcal{A}'$.

CAPITOLO 4

AGGIORNAMENTO DI UNA BASE DI DATI

Nella maggior parte dei casi una base di dati è soggetta a continue operazioni di aggiornamento affinché le informazioni in essa contenute siano sempre rispondenti allo stato attuale delle cose. Ogni operazione di aggiornamento ha per oggetto una singola relazione contenuta nello stato corrente della base di dati, sia esso *d*, e richiede l'aggiunta di una singola ennupla, oppure l'eliminazione o la modifica di più ennuple.

Vediamo ora come si esprimono le singole operazioni di aggiornamento che abbia per oggetto una tabella relazionale `Tab` con schema *R*. Il risultato dell'operazione sarà quello di modificare la relazione di nome `Tab` contenuta in *d*.
Iniziamo con l'operazione di aggiunta.

AGGIUNTA

Alla relazione di nome `Tab` viene aggiunta una singola ennupla *t*. L'operazione è sintatticamente corretta se *t* è un'ennupla su *R*, cioè $t \in DOM(R)$. L'operazione poi è efficace se l'ennupla *t* non appartiene già alla relazione di nome `Tab`.
Ad esempio, per una tabella relazionale di nome `Nomi` con schema {`nome`, `cognome`} l'aggiunta dell'ennupla (`Andrea`', '`Rossi`')) si esprime in *SQL* con il comando

```
INSERT [INTO] Nomi
VALUES      (nome = 'Andrea', cognome = 'Rossi');
```

CANCELLAZIONE

Dalla relazione di nome `Tab` vengono eliminate tutte le ennuple che soddisfano una certa condizione *C*. L'operazione è sintatticamente corretta se gli attributi coinvolti nella condizione *C* appartengono ad *R*. L'operazione poi è efficace se la relazione di nome `Tab` contiene almeno un'ennupla che soddisfa la condizione *C*.
Un caso tipico si ha quando la condizione *C* assume la forma $X = x$, in cui *X* è un sottoinsieme di *R* ed *x* è un'ennupla su *X*, cioè $x \in DOM(X)$. Se poi *X* è una sovrachiave del modello per le relazioni di nome `Tab`, allora il risultato della cancellazione sarà una relazione con un'ennupla in meno (sempreché l'operazione sia efficace).

Ad esempio, per una tabella relazionale di nome Indirizzi il cui schema contiene l'attributo provincia i cui valori sono le sigle delle province, la cancellazione degli indirizzi in provincia di Torino si esprime in *SQL* con il comando

```
DELETE [FROM]   Indirizzi
WHERE           provincia = 'TO';
```

MODIFICA

Ogni ennupla t della relazione di nome Tab che soddisfa una certa condizione C viene modificata in maniera tale che per la sua versione aggiornata t' si abbia $t'[Y] = y$ e $t'[R–Y] = t[R–Y]$. L'operazione è sintatticamente corretta se gli attributi coinvolti nella condizione C appartengono ad R e se Y è un sottoinsieme di R ed y è un'ennupla su Y, cioè $y \in DOM(Y)$. L'operazione poi è efficace se la relazione di nome Tab contiene almeno un'ennupla che soddisfa la condizione C.

Ad esempio, per una tabella relazionale di nome Nomi il cui schema contiene l'attributo nome, la modifica dei nomi in cui il nome è stato erroneamente scritto Tilde invece di Matilde si esprime in *SQL* con il comando

```
UPDATE      Nomi
SET         nome = 'Matilde'
WHERE       nome = 'Tilde';
```

Dopo aver verificato la correttezza sintattica di un'operazione di aggiornamento, si passa finalmente alla sua *esecuzione*, il cui risultato è una nuova relazione r' che potrebbe essere inconsistente sotto il profilo semantico, cioè potrebbe non essere conforme al modello delle relazioni di nome Tab. Ad esempio l'operazione

```
INSERT      Regno
VALUES      ('Napoleone', 1740, 1600);
```

nella base di dati STUART, sebbene sia sintatticamente corretta, è manifestamente erronea perché il vincolo che stabilisce che valore dell'attributo fine deve sempre risultare maggiore o uguale al valore dell'attributo inizio.

Quando un'operazione di aggiornamento viene eseguita sullo stato corrente $d = \{\ldots, r, \ldots\}$ di $\mathcal{D}$ e genera un nuovo stato $d' = \{\ldots, r', \ldots\}$ di $\mathcal{D}$, il sistema di gestione della base di dati controlla che d' sia uno stato valido; se così non fosse, l'aggiornamento viene rifiutato ripristinando lo stato d di $\mathcal{D}$.

Esempio 4.1 Supponiamo di voler eseguire l'aggiornamento della relazione di nome Partenze

volo	giorno	pilota	ora
112	lunedì	Rossi	10:30
112	mercoledì	Verdi	10:30
200	giovedì	Verdi	12:00
112	venerdì	Rossi	10:30

con l'operazione di modifica

```
UPDATE    Partenze
SET       ora = 9:30
WHERE     volo = 112 AND giorno = 'lunedì';
```

Qui si osservi che la condizione

volo = 112 AND giorno = 'lunedì'

seleziona un'unica ennupla visto che {volo, giorno} è verosimilmente la chiave primaria del modello. L'operazione è sintatticamente corretta ma il risultato sarebbe la relazione

volo	giorno	pilota	ora
112	lunedì	Rossi	9:30
112	mercoledì	Verdi	10:30
200	giovedì	Verdi	12:00
112	venerdì	Rossi	10:30

che non è conforme al modello delle relazioni di nome Partenze perché vìola la dipendenza funzionale volo → ora.

Per ottenere il risultato desiderato, andrebbe eseguita l'operazione

```
UPDATE    Partenze
SET       ora = 9:30
WHERE     volo = 112;
```

che cambia da 10:30 a 9:30 il valore di ora in tutte le ennuple con volo = 112.

La relazione che ne risulterebbe

volo	giorno	pilota	ora
112	lunedì	Rossi	9:30
112	mercoledì	Verdi	9:30
200	giovedì	Verdi	12:00
112	venerdì	Rossi	9:30

è questa volta sì conforme al modello delle relazioni di nome Partenze.

CAPITOLO 5

MODELLI REGOLARI

5.1 *Ridondanza dei dati*

Uno dei principi ispiratori della tecnologia delle basi di dati fu quello di ridurre al minimo la *ridondanza dei dati* utilizzati dalle applicazioni: anziché un *data set* per ciascuna applicazione, una "banca centrale" a cui le diverse applicazioni potevano attingere per ottenere i dati di proprio interesse. E con un'unica banca di dati, non ci sarebbero stati i problemi legati alla possibile incoerenza tra i *data set* su informazioni comuni.

Ora supponiamo di avere una relazione r che soddisfa la dipendenza funzionale $X \rightarrow Y$, vale a dire che, per ogni valore x di X, la relazione $\pi_Y(\sigma_{X=x}(r))$ contiene al più un'ennupla. Se r contiene due (o più) ennuple distinte t e t' che concordano su X allora, una volta che si conosca il valore di Y in t, sia esso y (cioè $t[Y] = y$), in forza della dipendenza funzionale $X \rightarrow Y$ il valore di Y in t' non può che essere y ed è, dunque, ridondante in palese violazione del principio ispiratore delle basi di dati.

Esempio 5.1 Riprendiamo la tabella relazionale `Partenze` della base di dati `AEROPORTO` e consideriamo il modello per le relazioni di nome `Partenze` definito dalla coppia $M = [R, F]$ dove R = {volo, giorno, pilota, ora} ed F è l'insieme delle tre dipendenze funzionali

volo → ora {giorno, pilota, ora} → volo {volo, giorno} → pilota

Supponiamo che la relazione di nome `Partenze` contenuta nello stato attuale d della base di dati `AEROPORTO` sia la relazione (conforme ad M) riportata nella Tabella 5.1.

volo	giorno	pilota	ora
112	lunedì	Rossi	10:30
112	mercoledì	Verdi	10:30
200	giovedì	Verdi	12:00
112	venerdì	Rossi	10:30

Tabella 5.1 Una relazione di nome `Partenze`

Salta subito all'occhio la ridondanza dei dati (dovuta alla dipendenza funzionale volo → ora) visto che il valore 10:30 dell'attributo ora per il volo 112 si ripete tre volte.

Esempio 5.2 Supponiamo che per la tabella relazionale di nome `Esame` abbiamo il modello $M = [R, F]$ dove

$R = \{\texttt{studente}, \texttt{materia}, \texttt{docente}, \texttt{voto}\}$
$F = \{\texttt{materia} \rightarrow \texttt{docente}, \{\texttt{studente}, \texttt{materia}\} \rightarrow \texttt{voto}\}\}$

e supponiamo che la coppia {studente, materia} sia stata designata come chiave primaria. Assumiamo che la relazione di nome `Esame` contenuta nello stato attuale della base di dati sia la relazione (conforme ad *M*) riportata nella Tabella 5.2

studente	materia	docente	voto
Gödel	logica	Hilbert	30
Gödel	algebra	Fermat	27
Wiles	algebra	Fermat	30

Tabella 5.2 Una relazione di nome `Esame`

Anche in questa relazione la ridondanza dei dati (dovuta alla dipendenza funzionale materia → docente) salta subito all'occhio visto che il valore Fermat dell'attributo docente per la materia Algebra si ripete due volte.

Vorremmo pertanto che, per ogni tabella relazionale `Tab` della base di dati, il modello delle relazioni di nome `Tab` non sia tale da generare ridondanza di dati. A questo scopo introdurremo il requisito di "regolarità" per un modello volto a garantire che in nessuna relazione conforme al modello potrà mai esservi ridondanza dei dati.
Dato un modello $M = [R, F]$, diciamo che una dipendenza funzionale *f* applicabile ad *R* è *implicata* da *F* se ogni relazione conforme ad *M* soddisfa anche la dipendenza funzionale *f*. (Ovviamente, ogni dipendenza funzionale in *F* è implicata da *F*.) Ne segue che un sottoinsieme *X* di *R* è una sovrachiave di *M* se la dipendenza funzionale $X \rightarrow R$ è implicata da *F*.

Un sottoinsieme *X* di *R* è una *criticità* del modello $M = [R, F]$ se *X* non è una sovrachiave di *M* e se esiste un attributo *A* appartenente ad $R–X$ tale che la dipendenza funzionale $X \rightarrow A$ sia implicata da *F*.

Nell'Esempio 5.1 è una criticità l'insieme elementare {volo}, mentre nell'Esempio 5.2 è l'insieme elementare {materia} ad essere una criticità.

In maniera equivalente, possiamo definire le criticità alla maniera seguente.

Un sottoinsieme X di R è una *criticità* del modello $M = [R, F]$ se esistono due attributi A e B appartenenti ad $R–X$, tali che la dipendenza funzionale $X \rightarrow A$ è implicata da F mentre la dipendenza funzionale $X \rightarrow B$ non è implicata da F.

Si osservi che l'esistenza dell'attributo B ci assicura che X non è una sovrachiave e, quindi, ci permette di costruire una relazione conforme ad M che contiene due ennuple (distinte) t e t' che concordano su X e discordano su B, cioè con $t[X] = t'[X]$ e $t(B) \neq t'(B)$. L'esistenza poi dell'attributo A comporta che il valore di A in t' è ridondante noto il valore di A in t.

Un modello è *regolare* se non ha criticità, cioè se in nessuna relazione conforme al modello v'è ridondanza dei dati. Nel prossimo capitolo, svilupperemo una teoria dell'implicazione per dipendenze funzionali, che ci fornirà un algoritmo (*test di regolarità*) per decidere se un modello è o meno regolare.

Resta comunque aperto il quesito «che fare ?» se un modello non è regolare. Rimandando l'argomento al Capitolo 10, ci limitiamo per ora ad accennare ad una possibile soluzione che verrà illustrata con qualche esempio.

Esempio 5.1 (seguito) Le cose sarebbero andate meglio se, al momento della progettazione della base di dati AEROPORTO, invece della tabella relazionale Partenze(volo, giorno, pilota, ora) avessimo introdotto le due tabelle relazionali

Giornale	con schema	{volo, giorno, pilota}
Imbarco	con schema	{volo, ora}

Allora, lo stato della nuova base di dati prima dell'esecuzione dell'operazione di modifica dell'ora del volo 112 conterrebbe, al posto della Tabella 5.1, le due relazioni riportate nelle Tabelle 5.3 e 5.4:

volo	giorno	pilota
112	lunedì	Rossi
112	mercoledì	Verdi
200	giovedì	Verdi
112	venerdì	Rossi

Tabella 5.3 Una relazione di nome Giornale

volo	ora
112	10:30
200	12:00

Tabella 5.4 Una relazione di nome Imbarco

e in nessuna delle due v'è ridondanza di dati.

Esempio 5.2 (seguito) Le cose sarebbero andate meglio se invece della tabella relazionale Esame avessimo introdotto le due tabelle relazionali

Verbale con schema { materia, studente, voto}

Corso con schema {materia, docente}

Allora, lo stato della nuova base di dati conterrebbe, al posto della Tabella 5.2, le due relazioni riportate nelle Tabelle 5.5 e 5.6:

materia	studente	voto
Logica	Gödel	30
Algebra	Gödel	27
Algebra	Wiles	30

Tabella 5.5 Una relazione di nome Verbale

materia	docente
Logica	Hilbert
Algebra	Fermat

Tabella 5.6 Una relazione di nome Corso

e in nessuna delle due v'è ridondanza di dati.

5.2 *Anomalie*

L'altra faccia della ridondanza dei dati è data dagli "aggiornamenti anomali". Consideriamo le operazioni di aggiornamento che hanno per oggetto una tabella relazionale Tab e una singola ennupla *t* che si vuole aggiungere, cancellare o modificare nella relazione *r* di nome Tab contenuta nello stato corrente della base di dati. L'aggiunta o la modifica di *t* potrebbe dare come risultato una relazione che non è conforme al modello e questo avviene se in *r* v'è ridondanza di dati.

Cominciamo con le operazioni di aggiunta e di modifica.

Esempio 5.1 (seguito) Riprendiamo la relazione di nome Partenze riportata nella Tabella 5.1 e, come già visto al capitolo precedente, supponiamo ora di voler modificare lo stato *d* della base di dati con l'operazione

```
UPDATE    Partenze
SET       ora = 9:30
WHERE     volo = 112 AND giorno = 'lunedì';
```

Nel nuovo stato *d′* della base di dati, avremo una relazione di nome Partenze, la quale però vìola la dipendenza funzionale volo → ora e pertanto non è conforme al modello. Sappiamo già che per ottenere una relazione conforme al modello avremmo dovuto eseguire una modifica che consiste nell'assegnare all'attributo ora il valore 9:30 in tutte le ennuple in cui volo = 112:

```
UPDATE    Partenze
SET       ora = 9:30
WHERE     volo = 112;
```

In questo modo, avremmo ottenuto la relazione riportata in Tabella 5.7 che è conforme ad *M*.

volo	giorno	pilota	ora
112	lunedì	Rossi	9:30
112	mercoledì	Verdi	9:30
200	giovedì	Verdi	12:00
112	venerdì	Rossi	9:30

Tabella 5.7 La relazione di nome Partenze aggiornata

Avremmo potuto evitare di ricorrere all'operazione di modifica multipla se, al momento della progettazione della base di dati AEROPORTO, avessimo introdotto le due tabelle relazionali

Giornale con schema {volo, giorno, pilota}

Imbarco con schema {volo, ora}

Allora, lo stato della base di dati prima dell'esecuzione dell'operazione di modifica dell'ora del volo 112 conterrebbe, al posto della Tabella 5.1, le due relazioni riportate nelle Tabelle 5.3 e 5.4 cosicché la modifica potrebbe essere ora eseguita con l'operazione

```
UPDATE    Imbarco
SET       ora = 9:30
WHERE     volo = 112;
```

e il risultato sarebbe quello di modificare il valore dell'attributo ora nell'unica ennupla della relazione di nome Imbarco con volo = 112:

volo	ora
112	9:30
200	12:00

Tabella 5.8 La relazione di nome Imbarco aggiornata

Un altro tipo di anomalia si ha con l'operazione di cancellazione. Supponiamo di voler cancellare dalla relazione r di nome Tab tutte le ennuple con $X = x$, dove X è una criticità per il modello delle relazioni di nome Tab. Siccome X è una criticità, esiste un attributo $A \notin X$ tale che $X \rightarrow A$ è una dipendenza funzionale implicata da F. Ora, se r contiene solo un'ennupla t con $t[X] = x$, allora avremmo perso l'informazione che associa ad x il valore $t(A)$.

Esempio 5.2 (seguito) Riprendiamo la relazione di nome Esame riportata nella Tabella 5.2. Supponiamo che essa venga aggiornata con l'operazione di cancellazione

```
DELETE    Esame
WHERE     materia = 'logica';
```

La nuova relazione di nome Esame sarà quella riportata in Tabella 5.9

studente	materia	docente	voto
Gödel	algebra	Fermat	27
Wiles	algebra	Fermat	30

Tabella 5.9 La relazione di nome Esame aggiornata

ma così avremo perso l'informazione che Hilbert è il docente di Logica.

Le cose sarebbero invece andate bene se, al momento della progettazione della base Se, invece, avessimo usato le due tabelle relazionali

Verbale con schema { materia, studente, voto}

Corso con schema {materia, docente}

Allora, lo stato della base di dati, prima dell'esecuzione dell'operazione di cancellazione dell'ennupla

(studente = Gödel, materia = logica, docente = Hilbert, voto = 30)

conterrebbe, al posto della Tabella 5.2, le due relazioni riportate nelle Tabelle 5.5 e 5.6 cosicché l'operazione di cancellazione avrebbe coinvolto solo la relazione di nome Verbale:

```
DELETE    Verbale
WHERE     studente = 'Gödel' AND materia = 'logica';
```

ed avrebbe potuto essere eseguita semplicemente eliminando quell'ennupla dalla relazione di nome Verbale. Il risultato dell'esecuzione sarebbe allora la relazione riportata nella Tabella 5.10.

materia	studente	voto
algebra	Gödel	27
algebra	Wiles	30

Tabella 5.10 La relazione di nome Verbale aggiornata

CAPITOLO 6

TEORIA DELLE DIPENDENZE FUNZIONALI

Dato un modello $M = [R, F]$, affrontiamo ora lo studio dell'insieme delle dipendenze funzionali implicate da F al fine di ottenere un algoritmo per decidere se esiste o meno una qualche criticità di M e, quindi, per decidere se M è un modello regolare.
Data una dipendenza funzionale $X \rightarrow Y$, chiamiamo X e Y rispettivamente il *determinante* e il *dipendente* della dipendenza funzionale. Diremo che una dipendenza funzionale è *banale* se il suo il dipendente è un sottoinsieme del suo determinante.

***Teorema* 6.1** Una dipendenza funzionale f è soddisfatta da ogni relazione al cui schema f è applicabile se e solo se f è banale.
Dimostrazione. (*se*) Sia f una dipendenza funzionale applicabile all'insieme R, e sia r un'arbitraria relazione con schema R. Se r è una relazione vuota, allora f è banalmente soddisfatta da r. Altrimenti, sia f la dipendenza funzionale $X \rightarrow Y$. Siccome $t[Y] = (t[X])[Y]$ per ogni ennupla t in r, prese arbitrariamente due ennuple t_1 e t_2 in r tali che $t_1[X] = t_2[X]$, abbiamo $t_1[Y] = (t_1[X])[Y] = (t_2[X])[Y] = t_2[Y]$, di qui la tesi.

(*solo se*) Sia $X \rightarrow Y$ una dipendenza funzionale che è soddisfatta da ogni relazione. Supponiamo per assurdo che Y contenga un attributo che non appartiene ad X. Sia $X = \{A_1, \ldots, A_k\}$ ed $X \cup Y = \{A_1, \ldots, A_n\}$, $n \geq k + 1$. Allora, ogni relazione con schema $R = X \cup Y$ del tipo

A_1 …	A_k	A_{k+1}	…	A_n
a_1 …	a_k	a_{k+1}	…	a_n
a_1 …	a_k	b_{k+1}	…	b_n

dove $a_i \neq b_i$ per ogni i, $k + 1 \leq i \leq n$, vìola la dipendenza funzionale $X \rightarrow Y$ (contraddizione). □

Mostriamo ora che il numero delle dipendenze funzionali banali applicabili ad uno insieme R di n attributi cresce in maniera esponenziale rispetto ad n. Qui e nel seguito, indicheremo il determinante ed il dipendente di una dipendenza funzionale f rispettivamente con $det(f)$ e $dip(f)$. Dunque, per ogni dipendenza funzionale banale f abbiamo $\varnothing \neq dip(f) \subseteq det(f)$. Così, fissato un sottoinsieme X di R con k attributi, $1 \leq k \leq n$, il numero di dipendenze funzionali banali f con $det(f) = X$ è pari a $2^k - 1$, e lo stesso vale per qualsiasi altro sottoinsieme di R formato da k attributi. Ora, siccome i

sottoinsiemi di R di cardinalità k sono in numero pari a $\binom{n}{k}$, abbiamo che il numero di dipendenze funzionali banali f con $|det(f)| = k$ è uguale a

$$\binom{n}{k}\left(2^k - 1\right)$$

In definitiva, il numero delle dipendenze funzionali banali applicabili ad R è pari a

$$\sum_{k=1}^{n}\binom{n}{k}(2^k-1) = \sum_{k=0}^{n}\binom{n}{k}(2^k-1) = \sum_{k=0}^{n}\binom{n}{k}2^k - \sum_{k=0}^{n}\binom{n}{k} = 3^n - 2^n .$$

Ciononostante, siccome il numero delle dipendenze funzionali applicabili ad R è pari a $(2^n–1)^2$, le dipendenze funzionali banali costituiscono solo una frazione infinitesima:

$$\lim_{n\to\infty}\frac{3^n - 2^n}{(2^n-1)^2} = 0$$

6.1 Chiusura logica di un insieme di dipendenze funzionali

Dato un modello $M = [R, F]$, chiamiamo *chiusura* (*logica*) di F *rispetto* ad R l'insieme delle dipendenze funzionali che sono applicabili ad R e sono implicate da F. Indichiamo con $\langle F\rangle_R$ tale insieme. Così, $\langle F\rangle_R$ è l'insieme di tutte le dipendenze funzionali f applicabili da R che sono soddisfatte da ogni relazione conforme ad M, cioè

$$\langle F\rangle_R = \{f \mid f \text{ è applicabile ad } R \text{ e } \mathrm{SAT}_R(F) \subseteq \mathrm{SAT}_R(f)\}.$$

Dalla stessa definizione di $\langle F\rangle_R$ segue che:

(1) $\mathrm{SAT}_R(F) = \mathrm{SAT}_R(\langle F\rangle_R)$ e

(2) per ogni dipendenza funzionale f che sia applicabile ad R e non sia in $\langle F\rangle_R$, l'insieme $\mathrm{SAT}_R(\langle F\rangle_R \cup \{f\})$ è strettamente contenuto in $\mathrm{SAT}_R(F)$.

Dimostriamo che l'insieme $\langle F\rangle_R$ gode delle seguenti tre proprietà:

(a) $F \subseteq \langle F\rangle_R$,

(b) se $F \subseteq G$, allora $\langle F\rangle_R \subseteq \langle G\rangle_R$,

(c) $\langle\langle F\rangle_R\rangle_R = \langle F\rangle_R$.

(a) Ovviamente, ogni dipendenza funzionale in F appartiene a $\langle F\rangle_R$.

(*b*) Se f appartiene a $\langle F\rangle_R$, allora f è soddisfatta da ogni relazione in $SAT_R(F)$. Siccome $F \subseteq G$, allora $SAT_R(F) \supseteq SAT_R(G)$ e quindi f è soddisfatta da ogni relazione in $SAT_R(G)$ cosicché f appartiene a $\langle G\rangle_R$.

(*c*) In forza della proprietà (*a*) e (*b*), abbiamo $\langle F\rangle_R \subseteq \langle\langle F\rangle_R\rangle_R$. Inoltre, per la (1) abbiamo sia $SAT_R(F) = SAT_R(\langle F\rangle_R)$ che $SAT_R(\langle F\rangle_R) = SAT_R(\langle\langle F\rangle_R\rangle_R)$ cosicché $SAT_R(F) = SAT_R(\langle\langle F\rangle_R\rangle_R)$. Ora, se $\langle F\rangle_R$ fosse un sottoinsieme proprio di $\langle\langle F\rangle_R\rangle_R$, allora per la (2) avremmo che $SAT_R(F)$ sarebbe strettamente contenuto in $SAT_R(\langle\langle F\rangle_R\rangle_R)$ e ne deriverebbe una contraddizione. Pertanto, deve aversi $\langle\langle F\rangle_R\rangle_R = \langle F\rangle_R$.

In genere, $\langle F\rangle_R$ contiene molte altre dipendenze funzionali oltre a quelle presenti in F. Per esempio, in forza del Teorema 6.1, qualunque sia F, $\langle F\rangle_R$ contiene sempre tutte le dipendenze funzionali banali applicabili ad R e, quindi, la dimensione di $\langle F\rangle_R$ è sempre esponenziale nel numero di attributi contenuti in R.
Proviamo ora due proprietà fondamentali di $\langle F\rangle_R$.

***Teorema* 6.2** (*Proprietà Transitiva Generalizzata*). Se $\langle F\rangle_R$ contiene le dipendenze funzionali $X \rightarrow Y$ e $T \rightarrow Z$ e se T è un sottoinsieme di Y, allora $\langle F\rangle_R$ contiene anche la dipendenza funzionale $X \rightarrow Y \cup Z$.
Dimostrazione. Sia r una qualsiasi relazione con schema R che soddisfi tutte le dipendenze funzionali in F. Per ipotesi, F implica le dipendenze funzionali $X \rightarrow Y$ e $T \rightarrow Z$, cosicché per ogni coppia di ennuple t_1 e t_2 in r, si ha

(1) se $t_1[X] = t_2[X]$ allora $t_1[Y] = t_2[Y]$ e
(2) se $t_1[T] = t_2[T]$ allora $t_1[Z] = t_2[Z]$.

Allora, per ogni coppia di ennuple t_1 e t_2 in r con $t_1[X] = t_2[X]$ si ha per la condizione (1) che $t_1[Y] = t_2[Y]$ e quindi, siccome T è un sottoinsieme di Y, $t_1[T] = t_2[T]$, la qual cosa per la condizione (2) implica che $t_1[Z] = t_2[Z]$. Siccome $t_1[Y] = t_2[Y]$ e $t_1[Z] = t_2[Z]$, si ha $t_1[Y \cup Z] = t_2[Y \cup Z]$, di qui la tesi. □

***Teorema* 6.3** (*Proprietà Proiettiva*). Se $\langle F\rangle_R$ contiene la dipendenza funzionale $X \rightarrow Y$ e Z è un sottoinsieme di Y, allora $\langle F\rangle_R$ contiene anche la dipendenza funzionale $X \rightarrow Z$.
Dimostrazione. Sia r una qualsiasi relazione con schema R che soddisfi tutte le dipendenze funzionali in F. Si osservi che, siccome Z è un sottoinsieme di Y, allora per ogni ennupla su R si ha $t[Z] = (t[Y])[Z]$. Ora, per ipotesi, per ogni coppia di ennuple t_1 e t_2 in r, se $t_1[X] = t_2[X]$ allora $t_1[Y] = t_2[Y]$. Ma se $t_1[Y] = t_2[Y]$ allora si ha pure che $t_1[Z] = (t_1[Y])[Z] = (t_2[Y])[Z] = t_2[Z]$, di qui la tesi. □

Oltre a queste due proprietà, $\langle F \rangle_R$ possiede altre proprietà che possono essere dimostrate direttamente oppure, come vedremo, possono essere "derivate" dalle Proprietà Transitiva Generalizzata e Proiettiva.

***Teorema* 6.4** (*Proprietà Accrescitiva*). Se $\langle F \rangle_R$ contiene la dipendenza funzionale $T \rightarrow Z$ ed X è un sovrainsieme di T, allora $\langle F \rangle_R$ contiene anche la dipendenza funzionale $X \rightarrow Z$.
Dimostrazione. Infatti, $\langle F \rangle_R$ certamente contiene la dipendenza funzionale banale $X \rightarrow X$. Inoltre, per ipotesi, $\langle F \rangle_R$ contiene la dipendenza funzionale $T \rightarrow Z$, allora per la Proprietà Transitiva Generalizzata $\langle F \rangle_R$ contiene anche la dipendenza funzionale $X \rightarrow X \cup Z$. Finalmente, per la Proprietà Proiettiva $\langle F \rangle_R$ contiene la dipendenza funzionale $X \rightarrow Z$. □

***Teorema* 6.5** (*Proprietà Additiva*). Se $\langle F \rangle_R$ contiene le dipendenze funzionali $X \rightarrow Y$ ed $X \rightarrow Z$, allora $\langle F \rangle_R$ contiene anche la dipendenza funzionale $X \rightarrow Y \cup Z$.
Dimostrazione. Infatti, $\langle F \rangle_R$ contiene sempre la dipendenza funzionale banale $X \rightarrow X$. Inoltre, per ipotesi, $\langle F \rangle_R$ contiene la dipendenza funzionale $X \rightarrow Y$, allora per la Proprietà Transitiva Generalizzata $\langle F \rangle_R$ contiene anche la dipendenza funzionale $X \rightarrow X \cup Y$. Sempre per ipotesi, $\langle F \rangle_R$ contiene la dipendenza funzionale $X \rightarrow Z$. Ancora per la Proprietà Transitiva Generalizzata $\langle F \rangle_R$ contiene anche la dipendenza funzionale $X \rightarrow X \cup Y \cup Z$. Finalmente, per la Proprietà Proiettiva $\langle F \rangle_R$ contiene la dipendenza funzionale $X \rightarrow Y \cup Z$. □

***Teorema* 6.6** (*Proprietà Transitiva*). Se $\langle F \rangle_R$ contiene le dipendenze funzionali $X \rightarrow Y$ ed $Y \rightarrow Z$, allora $\langle F \rangle_R$ contiene anche la dipendenza funzionale $X \rightarrow Z$.
Dimostrazione. Per la Proprietà Transitiva Generalizzata, $\langle F \rangle_R$ contiene la dipendenza funzionale $X \rightarrow Y \cup Z$ e, per la Proprietà Proiettiva, la dipendenza funzionale $X \rightarrow Z$.
□

6.2 Chiusura logica di un insieme di attributi

Dato un modello $M = [R, F]$, consideriamo ora un qualsiasi sottoinsieme X di R e indichiamo con $\langle X \rangle_F$ l'insieme degli attributi A in R tali che la dipendenza $X \rightarrow A$ sia implicata da F. Per convenzione, per il sottoinsieme vuoto di R poniamo $\langle \varnothing \rangle_F = \varnothing$. Chiamiamo l'insieme $\langle X \rangle_F$ la *chiusura* (*logica*) di X *rispetto* ad F.

***Teorema* 6.7** Dato un modello $M = [R, F]$, una dipendenza funzionale $X \rightarrow Y$ è implicata da F se e solo se Y è un sottoinsieme di $\langle X \rangle_F$.
Dimostrazione. (*se*) Assumiamo che la dipendenza funzionale $X \rightarrow A$ sia implicata da F per ogni attributo A in Y. Per la proprietà additiva, la dipendenza funzionale $X \rightarrow Y$ è implicata da F.

(*solo se*) Assumiamo che la dipendenza funzionale $X \rightarrow Y$ sia implicata da F. Per la proprietà proiettiva, per ogni attributo A in Y la dipendenza funzionale $X \rightarrow A$ è implicata da F e, quindi, appartiene a $\langle X \rangle_F$. Dunque, siccome ogni attributo A in Y appartiene a $\langle X \rangle_F$, l'insieme Y è un sottoinsieme di $\langle X \rangle_F$. □

Vedremo nel prossimo paragrafo che esiste un semplice algoritmo per calcolare $\langle X \rangle_F$. Per il momento, dimostriamo alcune utili proprietà dell'insieme $\langle X \rangle_F$.

***Lemma* 6.1** La chiusura logica $\langle X \rangle_F$ gode delle seguenti tre proprietà:

— $X \subseteq \langle X \rangle_F$
— se $X \subseteq Y$ allora $\langle X \rangle_F \subseteq \langle Y \rangle_F$
— $\langle \langle X \rangle_F \rangle_F = \langle X \rangle_F$.

Dimostrazione. La proprietà estensiva deriva dal fatto che la dipendenza funzionale $X \rightarrow X$ banalmente appartiene ad $\langle F \rangle_R$.
Dimostriamo ora la proprietà monotona. Siccome $X \rightarrow \langle X \rangle_F$ è implicata da F ed $X \subseteq Y$, per la proprietà accrescitiva si ha che la dipendenza funzionale $Y \rightarrow \langle X \rangle_F$ è implicata da F. Ne segue allora che $\langle X \rangle_F \subseteq \langle Y \rangle_F$.
Dimostriamo infine l'idempotenza. Per la proprietà estensiva, si ha $\langle X \rangle_F \subseteq \langle \langle X \rangle_F \rangle_F$. D'altra parte, siccome le dipendenze funzionali $X \rightarrow \langle X \rangle_F$ e $\langle X \rangle_F \rightarrow \langle \langle X \rangle_F \rangle_F$ sono entrambe implicate da F, per la proprietà transitiva si ha che la dipendenza funzionale $X \rightarrow \langle \langle X \rangle_F \rangle_F$ è implicata da F, cosicché $\langle \langle X \rangle_F \rangle_F \subseteq \langle X \rangle_F$. Riassumendo, abbiamo che $\langle X \rangle_F \subseteq \langle \langle X \rangle_F \rangle_F \subseteq \langle X \rangle_F$, la qual cosa prova l'assunto. □

Le tre proprietà testé dimostrate della chiusura $\langle X \rangle_F$ di un insieme di attributi sono molto simili alle tre proprietà della chiusura $\langle F \rangle_R$ di un insieme di dipendenze funzionali. In realtà, possiamo pensare ad $\langle X \rangle_F$ come al risultato dell'applicazione all'insieme X di un operatore che possiamo indicare con $\langle\ \rangle_F$; parimenti, $\langle F \rangle_R$ può essere visto come il risultato dell'applicazione all'insieme delle dipendenze funzionali F di un operatore che possiamo indicare con $\langle\ \rangle_R$. Dal punto di vista algebrico, gli operatori $\langle\ \rangle_F$ e $\langle\ \rangle_R$ sono entrambi esempi del concetto astratto di "operatore di chiusura", la cui definizione vogliamo qui ricordare.

Sia E un insieme (finito) e sia $\wp(E)$ l'insieme delle parti di E; una funzione σ con dominio $\wp(E)$ e codominio $\wp(E)$ è un *operatore di chiusura* se gode delle seguenti tre proprietà (che sono le versioni astratte delle tre proprietà di cui godono $\langle X \rangle_F$ e $\langle F \rangle_R$):

C1. (*estensività*) $X \subseteq \sigma(X)$;
C2. (*monotonicità*) se $X \subseteq Y$ allora $\sigma(X) \subseteq \sigma(Y)$;

C3. (*idempotenza*) $\sigma(\sigma(X)) = \sigma(X)$.

Dato un operatore di chiusura σ, in algebra si dice che un sottoinsieme X di E è un insieme *convesso* (o *chiuso*) se $\sigma(X) = X$. Ora gli insiemi convessi hanno la bella proprietà di formare una famiglia che è chiusa rispetto all'intersezione insiemistica nel senso che l'intersezione di due insiemi convessi è ancora un insieme convesso:

se $\sigma(X) = X$ e $\sigma(Y) = Y$ allora $\sigma(X \cap Y) = X \cap Y$.

Per provarlo, prendiamo due insiemi convessi X e Y e consideriamo la loro intersezione $X \cap Y$. Siccome $X \cap Y \subseteq X$ e $X \cap Y \subseteq Y$, per la proprietà C2 abbiamo

$\sigma(X \cap Y) \subseteq \sigma(X)$ $\sigma(X \cap Y) \subseteq \sigma(Y)$

e quindi

$\sigma(X \cap Y) \subseteq \sigma(X) \cap \sigma(Y)$.

Ora, siccome sia X che Y sono convessi, abbiamo $\sigma(X) = X$ e $\sigma(Y) = Y$ e quindi

$\sigma(X \cap Y) \subseteq \sigma(X) \cap \sigma(Y) = X \cap Y$.

Infine, per la proprietà C1 abbiamo

$X \cap Y \subseteq \sigma(X \cap Y) \subseteq \sigma(X) \cap \sigma(Y) = X \cap Y$

da cui $X \cap Y = \sigma(X \cap Y)$. Dunque, $X \cap Y$ è un insieme convesso.

Tornando ora al nostro modello $M = [R, F]$, visto che $\langle\ \rangle_F$ è un operatore di chiusura, possiamo applicare la nozione di convessità chiamando un sottoinsieme X di R *convesso* nel modello M se $\langle X \rangle_F = X$.

Un sottoinsieme X di R è una criticità di M se e solo se X non è né una sovrachiave di M né un insieme convesso in M.

Ne segue che condizione necessaria e sufficiente perché M sia regolare è che ogni sottoinsieme di R sia una sovrachiave di M o un insieme convesso.

6.3 Algoritmo di chiusura di un insieme di attributi

Per calcolare la chiusura di un insieme di attributi X rispetto ad un insieme di dipendenze funzionali F, utilizzeremo il seguente algoritmo che, come si vedrà, sfrutta solo la proprietà estensiva dell'operatore di chiusura $\langle\ \bullet\ \rangle_F$ e la proprietà Transitiva Generalizzata.

— *Algoritmo di Bernstein* (`Closure`), 1976 —

Dati: un insieme F di dipendenze funzionali ed un insieme X di attributi.
Risultato: un sovrainsieme V di X.

Procedura

(1) $V' := \varnothing$; $V := X$;

(2) Fintantoché $V' \neq V$ ripetere:

(2.1) $V' := V$;
(2.2) per ogni dipendenza funzionale $T \rightarrow Z$ in F

se $T \subseteq V$ allora $V := V \cup Z$.

Dunque l'algoritmo termina quando, per ogni dipendenza funzionale $T \rightarrow Z$ in F, o T non è un sottoinsieme di V oppure $T \cup Z$ è un sottoinsieme di V.

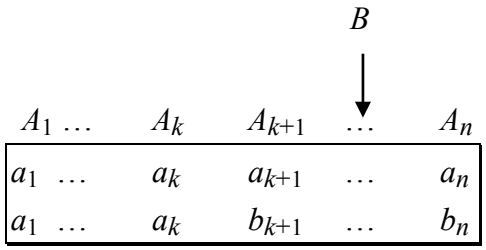
***Teorema* 6.8** L'applicazione dell'algoritmo di Bernstein ad un insieme X di attributi e ad un insieme F di dipendenze funzionali calcola correttamente la chiusura di X rispetto ad F.
Dimostrazione. Siano V_0, V_1, …, V_i, …, V_m i valori distinti assunti dalla variabile V nel corso dell'applicazione dell'algoritmo. Cominciamo con il dimostrare che $V_m \subseteq \langle X \rangle_F$. A questo scopo, grazie al Teorema 6.7, basta provare che la dipendenza funzionale $X \rightarrow V_m$ appartiene a $\langle F \rangle_R$. Dimostriamo per induzione che $X \rightarrow V_i$ appartiene a $\langle F \rangle_R$, per ogni i.

Passo base. Inizialmente si ha $V_0 = X$ e, siccome banalmente $X \rightarrow X$ appartiene a $\langle F \rangle_R$, è vero che $V_0 \subseteq \langle X \rangle_F$.

Passo induttivo. Assumiamo che la dipendenza funzionale $X \rightarrow V_i$ appartenga a $\langle F \rangle_R$. Sia poi $T \rightarrow Z$ la dipendenza funzionale in F che ha determinato l'incremento della variabile V dal valore V_i al valore V_{i+1}. Allora, siccome le dipendenze funzionali $X \rightarrow V_i$ e $T \rightarrow Z$ appartengono entrambe a $\langle F \rangle_R$ e $T \subseteq V_i$, per la proprietà transitiva generalizzata anche la dipendenza funzionale $X \rightarrow V_{i+1}$ appartiene a $\langle F \rangle_R$.

Dunque, $V_m \subseteq \langle X \rangle_F$. Per dimostrare che vale l'eguaglianza, è sufficiente a questo punto provare che l'insieme $U = \langle X \rangle_F - V_m$ è vuoto. Supponiamo per assurdo che U non sia vuoto e sia B un attributo in U. Allora, siccome B appartiene a $\langle X \rangle_F$, la

dipendenza funzionale $X \rightarrow B$ deve appartenere a $\langle F \rangle_R$, cioè deve essere soddisfatta da ogni relazione con schema R che soddisfi tutte le dipendenze in F. L'assurdo verrà provato dall'esistenza di una relazione r che soddisfa tutte le dipendenze in F e che vìola la dipendenza funzionale $X \rightarrow B$. Sia $V_m = \{A_1, \ldots, A_k\}$ ed $R = \{A_1, \ldots, A_n\}$. Consideriamo una relazione r con schema R del tipo



dove $a_i \neq b_i$ per ogni i, $k + 1 \leq i \leq n$. Ovviamente, siccome X è un sottoinsieme di V_m e B non appartiene a V_m, la relazione r vìola la dipendenza funzionale $X \rightarrow B$. D'altra parte, è facile provare che r soddisfa tutte le dipendenze funzionali in F. La cosa è ovvia se $F = \varnothing$. Se $F \neq \varnothing$, prendiamo in F una generica dipendenza funzionale $T \rightarrow Z$. Distinguiamo due casi a seconda che T sia o meno un sottoinsieme di V_m.

> (Caso 1) Se $T \subseteq V_m$ allora, quando la dipendenza funzionale $T \rightarrow Z$ è stata esaminata nel corso dell'ultima iterazione dall'algoritmo, abbiamo che $Z \subseteq V_m$. Dunque, le due ennuple di r concordano tanto su T quanto su Z e, quindi, la relazione r soddisfa la dipendenza funzionale $T \rightarrow Z$.
>
> (Caso 2) Se $T - V_m \neq \varnothing$, allora le due ennuple di r non concordano su T e, quindi, banalmente la relazione r soddisfa la dipendenza funzionale $T \rightarrow Z$.

Dunque, la relazione r soddisfa tutte le dipendenze funzionali in F. Tuttavia, abbiamo visto che r vìola la dipendenza funzionale $X \rightarrow B$ che, quindi, non può appartenere a $\langle F \rangle_R$ (contraddizione). La contraddizione deriva dall'aver supposto che l'insieme U non fosse vuoto. Pertanto, non può che essere $\langle X \rangle_F = V_m$. □

Esempio 6.1 Siano $F = \{\text{A} \rightarrow \text{D}, \text{AB} \rightarrow \text{E}, \text{BF} \rightarrow \text{E}, \text{CD} \rightarrow \text{F}, \text{E} \rightarrow \text{C}\}$ ed $X = \{\text{A}, \text{E}\}$. Inizialmente, $V = \{\text{A}, \text{E}\}$. Durante la prima scansione di F, la dipendenza funzionale $\text{A} \rightarrow \text{D}$ è usata per aggiungere D a V, e la dipendenza funzionale $\text{E} \rightarrow \text{C}$ per aggiungere C a V; così, al termine della prima esecuzione del ciclo (2) si ha $V = \{\text{A}, \text{C}, \text{D}, \text{E}\}$. Durante la seconda scansione di F, la dipendenza funzionale $\text{CD} \rightarrow \text{F}$ è usata per aggiungere F a V; così, al termine della seconda esecuzione del ciclo (2), troviamo $V = \{\text{A}, \text{C}, \text{D}, \text{E}, \text{F}\}$. La terza scansione di F lascia V invariato e, dunque, $\{\text{A}, \text{C}, \text{D}, \text{E}, \text{F}\}$ è il valore calcolato dall'Algoritmo di Chiusura. ■

Dal punto di vista computazionale, si vede facilmente che certe operazioni previste dall'Algoritmo di Chiusura possono essere eliminate. Al paragrafo A.1 dell'Appendice viene data un'implementazione efficiente dell'algoritmo di Bernstein che richiede un numero di operazioni che è lineare nella *dimensione* di F, definita come

$$||F|| = \sum_{i=1,\ldots,p} |det(f_i)| + |dip(f_i)|.$$

A partire dall'algoritmo di Bernstein, possiamo facilmente costruire un algoritmo per decidere se una dipendenza funzionale è o meno implicata da F.

Test di Implicazione

Dati:	Un modello $M = [R, F]$ ed una dipendenza funzionale $X \rightarrow Y$ applicabile ad R.
Risultato:	Un valore della variabile logica *test*.

Procedura

(1) *test* := FALSO.
(2) Calcolare $\langle X \rangle_F$ con l'algoritmo di Bernstein.
(3) Se Y è un sottoinsieme di $\langle X \rangle_F$, allora *test* := VERO.

Diremo che il Test di Implicazione dà *esito positivo* (rispettivamente, *negativo*) se il valore finale della variabile *test* è VERO (rispettivamente, FALSO).

***Teorema* 6.9**. Il Test di Implicazione dà esito positivo se e solo se la dipendenza funzionale $X \rightarrow Y$ è implicata da F.
Dimostrazione. Per i Teoremi 6.7 e 6.8. □

Si osservi infine che dal Teorema 6.9 segue che, per generare l'insieme delle dipendenze funzionali appartenenti ad $\langle F \rangle_R$, sono sufficienti le tre proprietà (assiomi):

***Assioma* I** Per ogni sottoinsieme non-vuoto X di R, $\langle F \rangle_R$ contiene la dipendenza funzionale $X \rightarrow X$.

***Assioma* II** (*Proprietà Transitiva Generalizzata*). Se $\langle F \rangle_R$ contiene le dipendenze funzionali $X \rightarrow Y$ e $T \rightarrow Z$ e se T è un sottoinsieme di Y, allora $\langle F \rangle_R$ contiene anche la dipendenza funzionale $X \rightarrow Y \cup Z$.

***Assioma* III** (*Proprietà Proiettiva*). Se $\langle F \rangle_R$ contiene la dipendenza funzionale $X \rightarrow Y$ e Z è un sottoinsieme di Y, allora $\langle F \rangle_R$ contiene anche la dipendenza funzionale $X \rightarrow Z$.

Ogni altra proprietà di $\langle F \rangle_R$ può essere *derivata* dai tre assiomi.

Ad esempio, la proprietà accrescitiva

"Se $\langle F \rangle_R$ contiene la dipendenza funzionale $T \rightarrow Z$ ed X è un sovrainsieme di T, allora $\langle F \rangle_R$ contiene anche la dipendenza funzionale $X \rightarrow Z$. "

ammette la seguente derivazione:

(1)	$X \rightarrow X$	in $\langle F \rangle_R$ per l'Assioma I
(2)	$T \rightarrow Z$	in $\langle F \rangle_R$ per ipotesi
(3)	$X \rightarrow T \cup Z$	in $\langle F \rangle_R$ per l'Assioma II applicato alle dipendenze funzionali (1) e (2)
(4)	$X \rightarrow Z$	in $\langle F \rangle_R$ per l'Assioma III applicato alla dipendenza funzionale (3).

Così pure, la proprietà additiva

"Se $\langle F \rangle_R$ contiene le dipendenze funzionali $X \rightarrow Y$ ed $X \rightarrow Z$, allora $\langle F \rangle_R$ contiene anche la dipendenza funzionale $X \rightarrow Y \cup Z$."

ammette la seguente derivazione:

(1)	$X \rightarrow X$	in $\langle F \rangle_R$ per l'Assioma I
(2)	$X \rightarrow Y$	in $\langle F \rangle_R$ per ipotesi
(3)	$X \rightarrow X \cup Y$	in $\langle F \rangle_R$ per l'Assioma II applicato alle dipendenze funzionali (1) e (2)
(4)	$X \rightarrow Z$	in $\langle F \rangle_R$ per ipotesi
(5)	$X \rightarrow X \cup Y \cup Z$	in $\langle F \rangle_R$ per l'Assioma II applicato alle dipendenze funzionali (3) e (4)
(6)	$X \rightarrow Y \cup Z$	in $\langle F \rangle_R$ per l'Assioma III applicato alla dipendenza (5)

6.4 Test di Regolarità

Finalmente siamo in grado di produrre un algoritmo per decidere se un modello sia o meno regolare.

***Lemma* 6.2** Dato un modello $M = [R, F]$, condizione necessaria e sufficiente perché M sia regolare è che non esista nessuna dipendenza funzionale in F il cui determinante sia una criticità di M.

Dimostrazione. La condizione è ovviamente necessaria. Viceversa, assumiamo che il determinante di nessuna dipendenza funzionale in F sia una criticità di M, cioè che, per ogni dipendenza funzionale $T \rightarrow Z$ in F, se $\langle T \rangle_F \neq T$ allora $\langle T \rangle_F = R$. Vogliamo dimostrare che M non può avere criticità, cioè che per ogni sottoinsieme X di R se $\langle X \rangle_F \neq X$ allora $\langle X \rangle_F = R$. Infatti, supponiamo che $\langle X \rangle_F \neq X$; allora, per come opera l'algoritmo di Bernstein, deve esistere almeno una dipendenza funzionale nonbanale $T \rightarrow Z$ tale che $T \subseteq X$ e Z non è un sottoinsieme di X. Ora, siccome $T \rightarrow Z$ non è una dipendenza funzionale banale, abbiamo $\langle T \rangle_F \neq T$ e, quindi, visto che, per ipotesi, T non può essere una criticità di M, abbiamo anche $\langle T \rangle_F = R$. D'altra parte, essendo $T \subseteq X$, si ha pure $\langle T \rangle_F \subseteq \langle X \rangle_F$ e, siccome $\langle T \rangle_F = R$, deve risultare $\langle X \rangle_F = R$. □

Test di Regolarità

Dati: Un modello $M = [R, F]$.
Risultato: Un valore della variabile logica *test*.

Procedura

(1) *test* := VERO.

(2) Per ogni dipendenza funzionale $X \rightarrow Y$ in F, se $Y–X \neq \varnothing$ allora

calcolare $\langle X \rangle_F$ con l'algoritmo di Bernstein;
se $\langle X \rangle_F \neq R$, allora *test* := FALSO ed Uscire.

Diremo che il Test di Regolarità dà *esito positivo* (rispettivamente, *negativo*) se il valore finale della variabile *test* è VERO (rispettivamente, FALSO)

***Teorema* 6.10** Un modello è regolare se e solo il Test di Regolarità dà esito positivo.

Dimostrazione. Sia $M = [R, F]$ un modello. Se M è regolare, allora nessun sottoinsieme di R è una criticità e, in particolare, il determinante di nessuna dipendenza funzionale in F lo è cosicché il Test di Regolarità darà esito positivo. Viceversa, se il Test di Regolarità dà esito positivo, allora il determinante di nessuna

dipendenza funzionale in F è una criticità e, per il Lemma 6.2, nessun sottoinsieme di R lo è. □

Esempio 6.2 Consideriamo il modello $M = [R, F]$ dove $R = \{\texttt{città}, \texttt{via}, \texttt{CAP}\}$ ed F contiene le due sole dipendenze funzionali:

$$\{\texttt{città}, \texttt{via}\} \rightarrow \texttt{CAP} \qquad\qquad \texttt{CAP} \rightarrow \texttt{città}$$

Siccome $\langle\{\texttt{CAP}\}\rangle_F = \{\texttt{città}, \texttt{CAP}\} \neq \{\texttt{città}, \texttt{via}, \texttt{CAP}\}$, il Test di Regolarità dà esito negativo. Pertanto, M non è regolare e l'insieme elementare $\{\texttt{CAP}\}$ è una criticità di M.

■

CAPITOLO 7

CHIAVI DI UN MODELLO

7.1 Calcolo di una chiave

Cominciamo con un algoritmo che permette di trovare una chiave di un modello.

Dati: Un modello $M = [R, F]$.

Risultato: Un sottoinsieme X di R.

Procedura

(1) $X := R$.

(2) Per ogni A in X

$Z := X–\{A\}$;
calcolare $\langle Z \rangle_F$ con l'algoritmo di Bernstein;
se $\langle Z \rangle_F = R$, allora $X := Z$.

Il seguente esempio mostra un'applicazione dell'algoritmo ed evidenzia la dipendenza del risultato che se ne ottiene (cioè della chiave trovata) dall'ordine in cui gli attributi sono esaminati.

Esempio 7.1 Sia $M = [R, F]$, dove

$R = \{A, B, C, D, E\}$

$F = \{\{A, B\} \rightarrow C, \{A, C\} \rightarrow B, D \rightarrow E\}$.

Se l'attributo C è esaminato prima dell'attributo B, allora la chiave che si trova è {A, B, D}; altrimenti, è {A, C, D}.

■

In pratica, per trovare una chiave si applica l'algoritmo suddetto assegnando ad X il valore $R–(dip(f)–det(f))$ dove f è una dipendenza funzionale nonbanale appartenente ad F.

Esempio 7.1 (seguito). A seconda che scegliamo come f la dipendenza funzionale $\{A, B\} \rightarrow C$ oppure $\{A, C\} \rightarrow B$ oppure $D \rightarrow E$ porremo inizialmente $X := \{A, B, D, E\}$ oppure $X := \{A, C, D, E\}$ oppure $X := \{A, B, C, D\}$.

■

7.2 Unicità della chiave

Ma quante chiavi può avere un modello? Come dimostrato dal seguente esempio, il loro numero può crescere in maniera esponenziale rispetto al numero degli attributi del modello, così che pensare di generarle tutte è un problema provatamente intrattabile.

Esempio 7.2 Sia $M = [R, F]$, dove

$$R = \{A_1, B_1, A_2, B_2, \ldots, A_n, B_n\}$$

$$F = \{A_1 \rightarrow B_1, B_1 \rightarrow A_1, A_2 \rightarrow B_2, B_2 \rightarrow A_2, \ldots, A_n \rightarrow B_n, B_n \rightarrow A_n\}.$$

È chiaro che un sottoinsieme X di R è una chiave se e solo se contiene n attributi e per ogni h, $1 \leq h \leq n$, X contiene o A_h oppure B_h. Siccome di siffatti insiemi ne esistono 2^n, abbiamo che il numero di chiavi è 2^n.

■

In certi casi però un modello ha un'unica chiave. Daremo ora una caratterizzazione di tali modelli. A questo scopo introduciamo la nozione di *sovrachiave esplicita* di un modello $M = [R, F]$. Se F è un insieme (eventualmente vuoto) di dipendenze funzionali banali, allora M ha un'unica sovrachiave esplicita data dallo stesso schema R; altrimenti, le sovrachiavi esplicite di M sono gli insiemi

$$R - (dip(f) - det(f))$$

dove f è una dipendenza funzionale nonbanale in F. Indichiamo con Y l'intersezione delle sovrachiavi esplicite di M, cioè $Y = R$ se F è un insieme (eventualmente vuoto) di dipendenze funzionali banali; altrimenti

$$Y = \cap_{f \in F} (R - (dip(f) - det(f))).$$

***Lemma* 7.1** Se un modello M ha un'unica chiave, allora l'intersezione delle sovrachiavi esplicite di M è una sovrachiave di M.

Dimostrazione. Sia X l'unica chiave di M. Allora X è contenuta in tutte le sovrachiavi di M; in particolare, essa è contenuta in tutte le sovrachiavi esplicite di M e, quindi, nella sua intersezione. Ne segue che l'intersezione delle sovrachiavi esplicite di M è una sovrachiave di M. □

***Lemma* 7.2** Dato un modello $M = [R, F]$, sia Y l'intersezione delle sovrachiavi esplicite di M. Se Y è una sovrachiave di M, allora Y è l'unica chiave di M.
Dimostrazione. La tesi è ovvia se F è un insieme (eventualmente vuoto) di dipendenze funzionali banali. Assumiamo che F contenga almeno una dipendenza funzionale nonbanale. È sufficiente allora provare che l'insieme Y è contenuto in ogni sovrachiave di M. Supponiamo, per assurdo, che esista una sovrachiave X di M che non contenga Y. Sia allora A un attributo in $Y–X$. Supponiamo ora di calcolare $\langle X \rangle_F$ con l'algoritmo di Bernstein. Siano $V_0, V_1, \ldots, V_m$ i valori distinti assunti dalla variabile V dell'algoritmo. Siccome X è una sovrachiave, dovremmo alla fine ottenere $V_m = R$. Pertanto, siccome $V_0 = X$ e X non contiene A, per un certo valore di $i \geq 0$ dovremmo avere $A \in V_{i+1}–V_i$. Sia $f^* \rightarrow Z$ la dipendenza funzionale che ha determinato l'incremento di V da V_i a V_{i+1}. Allora, siccome $det(f^*) \subseteq V_i$ ed $A \notin V_i$, avremmo che $A \notin det(f^*)$; d'altra parte, siccome $V_{i+1} = V_i \cup dip(f^*)$ ed $A \notin V_i$, dovremmo avere che $A \in dip(f^*)–det(f^*)$. D'altra parte, siccome

$$Y = \cap_{f \in F} (R – (dip(f)–det(f)) = R – \cup_{f \in F} (dip(f)–det(f)),$$

avremmo che A non appartiene ad Y (contraddizione). □

Come conseguenza dei Lemmi 7.1 e 7.2 abbiamo

***Teorema* 7.1** (*Teorema sull'unicità della chiave*) Un modello M ha un'unica chiave se e solo se l'intersezione delle sovrachiavi esplicite di M è una sovrachiave di M, nel qual caso l'intersezione delle sovrachiavi esplicite di M è l'unica chiave di M.
Dimostrazione. Sia Y l'intersezione delle sovrachiavi esplicite di M.
(*se*) Se Y è una sovrachiave di M allora, per il Lemma 7.2, Y è l'unica chiave di M.
(*solo se*) Se esiste un'unica chiave di M allora, per il Lemma 7.1, Y è una sovrachiave di M ed allora, per il Lemma 7.2, Y è l'unica chiave di M. □

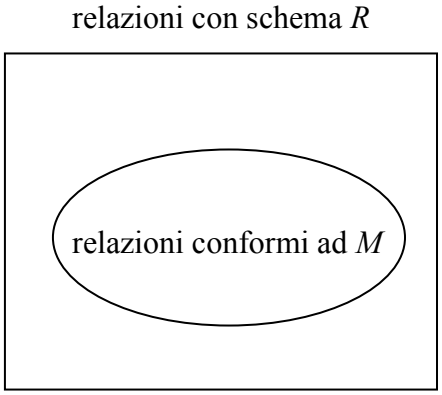
7.3 Chiavi minime

Una *chiave minima* di un modello M è una chiave di M col minor numero di attributi. Al paragrafo A.2 dell'Appendice verrà dimostrato che il problema di trovare una chiave minima si dimostra essere "intrattabile" dal punto di vista della teoria della complessità. Questo non toglie che, in casi particolari (come nel caso di una singola chiave), il problema possa essere di facile soluzione.

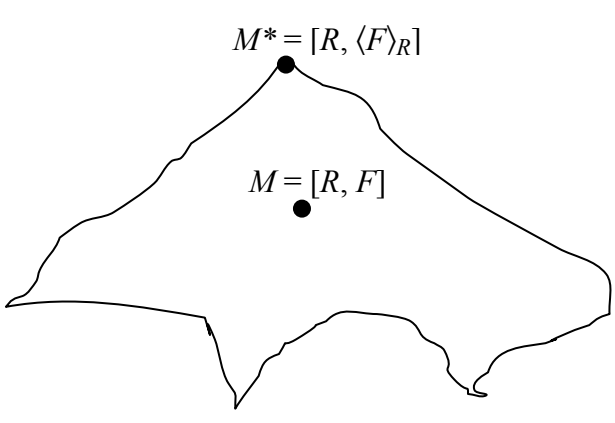
CAPITOLO 8

MODELLI EQUIVALENTI

Abbiamo visto che assegnare un modello $M = [R, F]$ ad una tabella relazionale `Tab` restringe le possibili relazioni di nome `Tab` alle sole relazioni conformi ad M.



Ora, dati due modelli con identico schema, siano essi $M = [R, F]$ ed $N = [R, G]$, diciamo che N è *restrittivo almeno quanto* M se $\mathrm{SAT}_R(G) \subseteq \mathrm{SAT}_R(F)$, cioè se ogni relazione conforme ad N è anche conforme ad M. (Si ricordi l'analogia con l'insieme delle soluzioni di un sistema di equazioni.) In forza dell'inclusione, ogni relazione in $\mathrm{SAT}_R(G)$ soddisfa tutte le dipendenze funzionali in F cosicché ogni dipendenza funzionale in F è implicata da G, cioè $F \subseteq \langle G \rangle_R$. Ne segue che $\langle F \rangle_R \subseteq \langle \langle G \rangle_R \rangle_R = \langle G \rangle_R$. Diremo anche che due modelli $M = [R, F]$ ed $N = [R, G]$ sono *equivalenti* se $\mathrm{SAT}_R(F) = \mathrm{SAT}_R(G)$. In altri termini, M e N sono equivalenti se N è restrittivo almeno quanto M ed M è restrittivo almeno quanto N. Ne segue che $\langle F \rangle_R \subseteq \langle G \rangle_R \subseteq \langle F \rangle_R$, cosicché M ed N sono equivalenti se e solo se $\langle F \rangle_R = \langle G \rangle_R$. Ovviamente, il modello $M^* = [R, \langle F \rangle_R]$ è equivalente ad M; inoltre se $N = [R, G]$ è equivalente ad M, allora $G \subseteq \langle F \rangle_R$.



Se $M = [R, F]$ ed $N = [R, G]$ sono modelli equivalenti, allora tutte le proprietà logiche dell'uno si estendono all'altro; così, ad esempio

- per ogni insieme X, si ha che $\langle X \rangle_F = \langle X \rangle_G$;
- un sottoinsieme di R è una sovrachiave (rispettivamente, una chiave) per M se e solo se è una sovrachiave (rispettivamente, una chiave) per N;
- M è regolare se e solo se N è regolare.

Un'altra proprietà dell'equivalenza è legata alle "proiezioni" di un modello. La *proiezione* di un modello $M = [R, F]$ su un sottoinsieme non-vuoto R' di R, è il modello $M' = [R', F']$ dove F' è l'insieme delle dipendenze funzionali che sono implicate da F e che sono applicabili ad R', cioè $F' = \{f \in \langle F \rangle_R : det(f) \cup dip(f) \subseteq R'\}$.

Esempio 8.1 Consideriamo un modello $M = [R, F]$ dove $R = \{A, B, C\}$ ed F contiene le due sole dipendenze funzionali: $A \rightarrow B$ e $B \rightarrow C$. La proiezione di M su $R' = \{A, C\}$ è il modello $M' = [R', F']$ in cui F' contiene le due dipendenze funzionali nonbanali $A \rightarrow C$ e $A \rightarrow \{A, C\}$ e le cinque dipendenze funzionali banali:

$$A \rightarrow A \qquad C \rightarrow C \qquad \{A, C\} \rightarrow A \qquad \{A, C\} \rightarrow C \qquad \{A, C\} \rightarrow \{A, C\}.$$

■

Si osservi che, se $M = [R, F]$ ed $N = [R, G]$ sono modelli equivalenti allora, siccome $\langle F \rangle_R = \langle G \rangle_R$, si ha che le proiezioni di M ed N su ogni sottoinsieme di R sono uguali.

Vogliamo ora risolvere il problema di decidere se due modelli dati $M = [R, F]$ ed $N = [R, G]$ sono o meno equivalenti. Va da sé che è questo il caso se e solo se

ogni dipendenza funzionale g in G è implicata da F
ogni dipendenza funzionale f in F è implicata da G

Esempio 8.3 I due modelli $M = [R, F]$ ed $N = [R, G]$, dove $R = \{A, B, C, D, E\}$, $F = \{A \rightarrow B,\ A \rightarrow C,\ \{C, D\} \rightarrow E\}$ e $G = \{A \rightarrow \{B, C\},\ \{C, D\} \rightarrow E\}$, sono equivalenti. Infatti, per la proprietà additiva F implica la dipendenza funzionale $A \rightarrow \{B, C\}$ presente in G–F. D'altro canto, per la proprietà proiettiva G implica le dipendenze funzionali $A \rightarrow B$ e $A \rightarrow C$ presenti in F–G.

■

Test di Equivalenza

Dati: Due modelli $M = [R, F]$ e $N = [R, G]$.

Risultato: Il valore della variabile logica *test*

Procedura

(1) *test* := VERO.

(2) Per ogni g in $G–F$

Applicare il Test di Implicazione per decidere se g è implicata da F; se il Test di Implicazione dà esito negativo, allora *test* := FALSO ed Uscire.

(3) Per ogni f in $F–G$

Applicare il Test di Implicazione per decidere se f è implicata da G; se il Test di Implicazione dà esito negativo, allora *test* := FALSO ed Uscire.

Diremo che il Test di Equivalenza dà *esito positivo* (rispettivamente, *negativo*) se il valore finale della variabile *test* è VERO (rispettivamente, FALSO). Va da sè che M ed N sono equivalenti se e solo se il Test di Equivalenza dà esito positivo.

Dati due modelli equivalenti $M = [R, F]$ ed $N = [R, G]$, dal punto di vista computazionale non è indifferente applicare ad F o a G il Test di Regolarità, perché la dimensione dell'insieme delle dipendenze funzionali (F o G) è un fattore determinante della complessità dell'algoritmo. È naturale allora cercare, dato un modello M, un modello equivalente ad M di dimensione minima. Questo problema si dimostra essere "intrattabile" dal punto di vista della teoria della complessità (v. paragrafo A.3 dell'Appendice). Presentiamo ora due tecniche per ottenere una soluzione buona, anche se non ottima.

La prima tecnica (*tecnica della riduzione*) consiste nell'eliminazione di certe dipendenze funzionali che risultano essere "ridondanti". La seconda tecnica (*tecnica della contrazione*) tende a ridurre le dimensioni delle dipendenze funzionali nonridondanti.

(*Tecnica della riduzione*) Una dipendenza funzionale f in F è *ridondante* se $F-\{f\}$ è equivalente ad F. Ad esempio, ogni dipendenza funzionale banale in F è ridondante. Diremo poi che un insieme F di dipendenze funzionali è *minimale* se nessuna dipendenza funzionale in F è ridondante.

Per decidere se una dipendenza funzionale f in F è ridondante basta applicare il seguente *Test di Ridondanza*.

Test di Ridondanza

Dati:	Un insieme di dipendenze funzionali F ed una dipendenza funzionale f appartenente ad F.
Risultato:	Un valore della variabile logica *test*.

Procedura

(1) *test* := FALSO.

(2) $G := F-\{f\}$.

(3) Se l'applicazione del Test di Implicazione ad f e G dà esito positivo, allora *test* := VERO.

Diremo che il Test di Ridondanza dà *esito positivo* (rispettivamente, *negativo*) se il valore finale della variabile *test* è VERO (rispettivamente, FALSO).
Il seguente algoritmo consente di trovare un sottoinsieme minimale G di F cosicché il modello $N = [R, G]$ è equivalente ad M e di dimensione minore o uguale a quella di M.

Algoritmo di Riduzione

Dati:	Un insieme F di dipendenze funzionali.
Risultato:	Un sottoinsieme G di F.

Procedura

(1) $G := F$.

(2) Per ogni dipendenza funzionale f in F

Applicare il Test di Ridondanza ad f e G;
se il Test di Ridondanza dà esito positivo allora $G := G-\{f\}$.

Vogliamo avvertire che il risultato dell'Algoritmo di Riduzione può dipendere dall'ordine in cui le dipendenze funzionali in F vengono esaminate. Così, con due diversi ordinamenti si possono ottenere due distinti modelli minimali.

Esempio 8.4 Consideriamo l'insieme $F = \{A \rightarrow B,\ A \rightarrow C,\ B \rightarrow A,\ B \rightarrow C\}$. Supponiamo di applicare l'Algoritmo di Riduzione e di esaminare le dipendenze funzionali in F nello stesso ordine in cui le abbiamo elencate. Allora, andremo ad eliminare la dipendenza funzionale $A \rightarrow C$. Elimineremo invece la dipendenza funzionale $B \rightarrow C$ se, ad esempio, esaminiamo le dipendenze funzionali nell'ordine $A \rightarrow B$, $B \rightarrow C$, $B \rightarrow A$, $A \rightarrow C$. ■

(*Tecnica della contrazione*) Supponiamo che F sia già minimale. Consideriamo una generica dipendenza funzionale $X \rightarrow Y$ in F. Per ridurne la dimensione, che è data da $|X| + |Y|$, tenteremo di restringere prima Y e poi X, ammesso che né Y né X siano formati da un singolo attributo. Per restringere Y, basta prendere $Z = Y–X$. Per restringere X con $|X| > 1$, possiamo procedere alla maniera seguente. Se A è un generico attributo in X, calcoliamo $\langle Z \rangle_F$ dove $Z = X–\{A\}$. Ora se $Y \subseteq \langle Z \rangle_F$ allora possiamo sostituire $X \rightarrow Y$ con $Z \rightarrow Y$. Infatti, sia

$$G = (F - \{X \rightarrow Y\}) \cup \{Z \rightarrow Y\}.$$

Per la proprietà accrescitiva, $X \rightarrow Y$ è implicata da $Z \rightarrow Y$ e, quindi, da G sicché $\langle F \rangle_R \subseteq \langle G \rangle_R$. D'altra parte, siccome $Y \subseteq \langle Z \rangle_F$, la dipendenza funzionale $Z \rightarrow Y$ è implicata da F sicché $\langle G \rangle_R \subseteq \langle F \rangle_R$. Dunque, i due modelli $M = [R, F]$ e $N = [R, G]$ sono equivalenti. Ora, se ripetiamo questa procedura per ogni attributo in X, avremo ridotto X al minimo.

Algoritmo di Contrazione

Dati: Un insieme F di dipendenze funzionali.
Risultato: Un insieme G di dipendenze funzionali.

Procedura

(1) $G := F$.

(2) Per ogni dipendenza funzionale $X \rightarrow Y$ in G

se $|X| \neq 1$ allora

$Z := X$;

per ogni attributo A in Z

calcolare $\langle Z - \{A\} \rangle_G$ con l'algoritmo di Bernstein;

se $Y \subseteq \langle Z - \{A\} \rangle_G$ allora $Z := Z - \{A\}$;

$G := (G - \{X \rightarrow Y\}) \cup \{Z \rightarrow Y\}$.

Utilizzando gli Algoritmi di Riduzione prima e di Contrazione poi, si ottiene sempre un modello equivalente ad *M* e di dimensione non superiore.

CAPITOLO 9

Forme Normali

9.1 Prima Forma Normale

Consideriamo una tabella relazionale di nome `Tab` a cui sia stato assegnato il modello $M = [R, F]$. Abbiamo visto che un'operazione di aggiornamento che abbia per oggetto `Tab` potrebbe dare come risultato una relazione che non è conforme al modello M. Vedremo quando e come questo si potrà evitare. Prima però, va sgomberato il campo da un'altra questione: siamo sicuri che il modello M assegnato a `Tab` sia stato ben definito sotto il profilo semantico? Nell'esempio che ora discutiamo, vedremo che, se il progettista non ha avuto abbastanza cura già solo nella scelta degli attributi dello schema R, allora una relazione conforme ad M potrebbe essere inaccettabile sotto il profilo semantico. Tipicamente questo problema si presenta con gli attributi "composti" che sono "sovradimensionati".

Un attributo A in R è *composto* se il suo dominio può essere visto come un insieme di ennuple con due o più componenti; cioè se esiste un insieme Z di due o più attributi (che chiamiamo una *scomposizione* di A) tale che $Z \cap R = \varnothing$ e il dominio di A è uguale a $DOM(Z)$. Un tipico esempio è l'attributo `data`, i cui valori sono ennuple; nella fattispecie, Z può essere l'insieme {`giorno`, `mese`, `anno`} oppure l'insieme {`giorno_mese`, `anno`}. Supponiamo che R contenga un attributo composto A; diremo che A è *sovradimensionato* nel modello M se

— esiste un attributo B in R tale che F contenga la dipendenza funzionale $A \rightarrow B$, ed

— esiste una scomposizione $Z = \{C, D\}$ di A che permetta di definire un modello $M' = [R', F']$ dove

$$R' = R - \{A\} \cup Z \qquad F' = F - \{A \rightarrow B\} \cup \{C \rightarrow B\}$$

In altre parole, l'attributo composto A è sovradimensionato in M se la dipendenza funzionale di B da A è solo dovuta all'attributo C.

Esempio 9.1 Supponiamo che per le relazioni di nome `Compleanno` abbiamo specificato il modello $M = [R, F]$ in cui R = {`nome`, `data`, `segno`} ed F contenga le due dipendenze funzionali

`nome` → {`data`, `segno`} `data` → `segno`.

La Tabella 9.1 riporta una relazione conforme ad $M = [R, F]$.

nome	data	segno
Alfredo	7 giugno 1949	Gemelli
Maria	14 Luglio 1963	Cancro
Vincenzo	30 Gennaio 1959	Acquario
Giuseppe	30 Gennaio 1960	Acquario

Tabella 9.1 Una relazione conforme ad $M = [R, F]$.

Supponiamo ora di aggiungere l'ennupla

Carla	7 giugno 1968	Leone

Si osservi che la nuova relazione è conforme ad M. Tuttavia, Alfredo e Carla, pur festeggiando il compleanno lo stesso giorno dello stesso mese, hanno distinti segni zodiacali. L'incongruenza sta nel fatto che, nella base di dati, l'attributo data è un attributo composto sovradimensionato e che solo una sua parte (giorno e mese) determina segno. La soluzione sta nel definire come modello per le relazioni di nome Compleanno il modello $M = [R, F]$, dove R = {nome, giorno_mese, anno, segno} ed F = {nome → giorno_mese, nome → anno, giorno_mese → segno}.

■

Gli attributi composti sovradimensionati rientrano nella categoria degli attributi *non-atomici*. Ad esempio, se si avesse una tabella relazionale di nome Persona con schema {nome, residenza} in cui i valori di residenza riportino le città di residenza e gli indirizzi delle persone, allora non avremmo modo di conoscere le persone che risiedono in una data città. Ora, se questa fosse un'informazione "interessante" per i potenziali utenti della base di dati, l'attributo non-atomico residenza andrebbe evitato e il progettista farebbe meglio ad assegnare a Persona lo schema {nome, città, indirizzo} anziché {nome, residenza}.

Diremo che un modello è in *prima forma normale* se contiene solo attributi *atomici*. Una base di dati è in prima forma normale se il modello assegnato ad ogni tabella relazionale presente nella base di dati è in prima forma normale.

9.2 Forma Normale di Boyce-Codd

Un modello è in *forma normale* di *Boyce-Codd* se è in prima forma normale ed è regolare. Una base di dati è in forma normale di Boyce-Codd se il modello assegnato ad ogni tabella relazionale presente nella base di dati è in forma normale di Boyce-Codd.

Esempio 9.2 Consideriamo il modello $M = [R, F]$ in cui

$$R = \{\texttt{codice\_fiscale}, \texttt{nominativo}, \texttt{anno}, \texttt{sesso}, \texttt{segno}\}$$

ed F contenga l'unica dipendenza funzionale

$$\texttt{codice\_fiscale} \rightarrow \{ \texttt{nominativo}, \texttt{anno}, \texttt{sesso}, \texttt{segno} \}.$$

Qui gli attributi

nominativo anno segno

indicano rispettivamente

il nome e cognome l'anno di nascita il segno zodiacale.

Si rammenti che il codice fiscale è così composto:

— i primi sei caratteri sono desunti dal nome e cognome

— i due caratteri seguenti sono le ultime due cifre dell'anno di nascita

— il carattere seguente è una lettera identificativa del mese di nascita

— i due caratteri seguenti indicano il giorno di nascita, aumentato di 40 se la persona è di sesso femminile

— i quattro caratteri seguenti formano un codice identificativo del comune di nascita

— l'ultimo carattere è un codice di controllo che serve anche a distinguere i casi di omonimia.

Ovviamente, il modello M è regolare ma non è in prima forma normale; dunque non è in forma normale di Boyce-Codd. Per avere un modello in prima forma normale dobbiamo scomporre l'attributo codice_fiscale che è sovradimensionato. Una maniera è la seguente. Indichiamo con

A l'attributo che contenga l'informazione data dai primi sei caratteri di codice_fiscale (le sei lettere desunte dal nominativo)

B l'attributo che contenga l'informazione data dai due caratteri seguenti di codice_fiscale (le ultime due cifre dell'anno di nascita)

C l'attributo che contenga l'informazione data dal carattere seguente di codice_fiscale (lettera identificativa del mese di nascita)

D l'attributo che contenga l'informazione data dai due caratteri seguenti di codice_fiscale (il giorno di nascita, aumentato di 40 se la persona è di sesso femminile)

E l'attributo che contenga l'informazione data dagli ultimi cinque caratteri di codice_fiscale

La semantica di questi attributi impone le seguenti dipendenze funzionali

$$\text{nominativo} \to A \qquad \text{anno} \to B \qquad \{C, D\} \to \text{segno} \qquad D \to \text{sesso}$$

a cui va aggiunta la seguente traduzione della dipendenza funzionale originaria, cioè

$$\{A, B, C, D, E\} \to \{\text{nominativo}, \text{anno}, \text{sesso}, \text{segno}\}.$$

Se indichiamo con F' l'insieme di queste cinque dipendenze funzionali e con R' l'insieme dei nove attributi $\{A, B, C, D, E, \text{nominativo}, \text{anno}, \text{sesso}, \text{segno}\}$, otteniamo un modello $M' = [R', F']$ che è in prima forma normale ed ha la stessa semantica del modello M. Se poi vogliamo che il dipendente di ogni dipendenza funzionale sia formato da un singolo attributo e che nessuna dipendenza funzionale sia ridondante, otteniamo il modello $M'' = [R', F'']$ equivalente ad M', dove F'' contiene le sei dipendenze funzionali

$$\text{nominativo} \to A \qquad \text{anno} \to B \qquad \{C, D\} \to \text{segno} \qquad D \to \text{sesso}$$

$$\{A, B, C, D, E\} \to \text{nominativo} \qquad \{A, B, C, D, E\} \to \text{anno}.$$

Si osservi infine che il modello M'' non è regolare perché contiene delle criticità; per esempio, sono criticità i quattro insiemi:

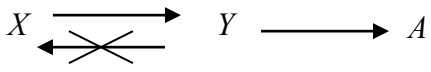
$$\{\text{nominativo}\} \qquad \{\text{anno}\} \qquad \{C, D\} \qquad \{D\}.$$

Vedremo in seguito come "normalizzare" il modello M''. ■

Daremo ora una caratterizzazione dei modelli regolari e, quindi, dei modelli in forma normale di Boyce-Codd. A tale scopo abbiamo bisogno di qualche nozione aggiuntiva.

Dato un modello $M = [R, F]$, diremo che un attributo A *dipende transitivamente* da un sottoinsieme X di $R–\{A\}$ se esiste un sottoinsieme Y di $R–\{A\}$ tali che:

> le dipendenze funzionali $X \to Y$ ed $Y \to A$ appartengono ad $\langle F \rangle_R$, e
> la dipendenza funzionale $Y \to X$ non appartiene ad $\langle F \rangle_R$.



Supponiamo ora che X sia una chiave di M. Allora, un attributo A dipende transitivamente da X se $A \notin X$ ed esiste un sottoinsieme Y di $R-\{A\}$ tale che la dipendenza funzionale $Y \rightarrow A$ appartiene ad $\langle F \rangle_R$ ed Y non è una sovrachiave di M.

***Lemma* 9.1** Dato un modello $M = [R, F]$, le tre condizioni seguenti sono equivalenti:

(1) Il modello M non è regolare.

(2) Esistono un attributo A ed una chiave X di M che non contiene A tali che l'attributo A dipende transitivamente da X.

(3) Esiste un attributo A che dipende transitivamente da ogni chiave di M che non contenga A.

Dimostrazione. (1) $\Rightarrow$ (2). Sia Y è una criticità di M, cioè Y non è né un insieme convesso né una sovrachiave. Siccome Y non è un insieme convesso, esiste almeno un attributo in $\langle Y \rangle_F - Y$. Sia esso A. L'insieme $R-\{A\}$ è ovviamente una sovrachiave di M. Allora esiste una chiave di M contenuta in $R-\{A\}$; sia essa X. Ora, siccome X non contiene A e, siccome Y non è una sovrachiave, la dipendenza funzionale $Y \rightarrow X$ non è implicata da F. Dunque, l'attributo A dipende transitivamente da X.

(2) $\Rightarrow$ (1). Siccome A dipende transitivamente da X, deve esistere un sottoinsieme Y di $R-\{A\}$ tale che:

la dipendenza funzionale $Y \rightarrow A$ è implicata da F, e
la dipendenza funzionale $Y \rightarrow X$ non è implicata da F.

Pertanto, Y non è un insieme convesso e non è una sovrachiave; dunque, Y è una criticità di M.

(2) $\Rightarrow$ (3). Sia X una chiave di M da cui A dipenda transitivamente e sia X' un'altra chiave di M che non contenga A. Ora, siccome A dipende transitivamente da X, deve esistere un sottoinsieme Y di $R-\{A\}$ tale che:

la dipendenza funzionale $Y \rightarrow A$ è implicata da F, e
la dipendenza funzionale $Y \rightarrow X$ non è implicata da F.

Visto che anche X' è una chiave di M e la dipendenza funzionale $Y \rightarrow X$ non è implicata da F, non lo è neppure la dipendenza funzionale $Y \rightarrow X'$. Ne segue che l'attributo A dipende transitivamente da X'.

(3) $\Rightarrow$ (2). Ovvio. □

Per il Lemma 9.1 abbiamo la seguente caratterizzazione dei modelli in forma normale di Boyce-Codd.

***Teorema* 9.1** Un modello è in forma normale di Boyce-Codd se e solo è in prima forma normale e non esiste nessun attributo che dipenda transitivamente da una chiave del modello.

9.3 Terza Forma Normale

Introduciamo ora una forma normale più debole della forma normale di Boyce-Codd. Un attributo si dice *primario* per un modello M se appartiene ad almeno una chiave di M, e *secondario* altrimenti. Si osservi che se A è un attributo secondario di M che dipende transitivamente da una chiave, allora A dipende transitivamente da ogni chiave di M (visto che nessuna chiave contiene A).

Un modello è in *terza forma normale* se è in prima forma normale e nessun attributo secondario dipende transitivamente da una chiave del modello (o, equivalentemente, nessun attributo secondario dipende transitivamente da una qualsiasi chiave del modello). Si osservi che un modello $M = [R, F]$ in terza forma normale può avere criticità ma, se Y è una criticità di M, allora l'insieme $\langle Y \rangle_F - Y$ contiene solo attributi primari per M.

Ovviamente, ogni modello in forma normale di Boyce-Codd è in terza forma normale. Il seguente esempio mostra un modello che è in terza forma normale ma non in forma normale di Boyce-Codd.

Esempio 9.3 Consideriamo il modello con schema {città, via, CAP} in cui siano definite le due dipendenze funzionali: {città, via} → CAP e CAP → città. Non essendoci attributi sovradimensionati, il modello è in prima forma normale. Le uniche chiavi del modello sono {città, via} e {via, CAP} e, quindi, i tre attributi sono tutti primari. Dunque, il modello è in terza forma normale. D'altra parte, il modello non è però regolare (e quindi non è in forma normale di Boyce-Codd) perché l'attributo (primario) città dipende transitivamente dalla chiave {via, CAP} per via della dipendenza funzionale CAP → città il cui determinante non è una chiave del modello.

■

Abbiamo visto che, dato un modello in prima forma normale, il problema di decidere se esso è o meno in forma normale di Boyce-Codd può essere risolto in maniera efficiente (cioè in tempo polinomiale) con il Test di Regolarità. Non così per il problema di decidere se un modello in prima forma normale è o meno in terza forma normale. Infatti, il Test di Regolarità ci consente di affermare che, se troviamo in F una dipendenza funzionale nonbanale $T \rightarrow Z$ tale che T non sia una sovrachiave,

allora ogni attributo *A* appartenente a *Z–T* dipende transitivamente (tramite *T*) da ogni chiave del modello che non contenga *A*. Ma, se ci proponiamo di trovare un attributo secondario che dipenda transitivamente da una chiave del modello, allora veniamo a trovarci di fronte al problema preliminare di saper riconoscere se un dato attributo è primario o secondario e questo è un problema che non può essere risolto in maniera efficiente, cosa di cui ci si convince facilmente ricordando che il numero delle chiavi del modello può essere esponenziale.

CAPITOLO 10

Normalizzazione

Consideriamo ancora una volta la tabella relazionale di nome Partenze nella base di dati AEROPORTO con il suo modello $M = [R, F]$ dove

R = {volo, giorno, pilota, ora}

F = {volo → ora, {volo, giorno} → pilota}.

Abbiamo già visto che M non è regolare e, quindi, pur essendo in prima forma normale non è in forma normale di Boyce-Codd e, quindi, non immune da anomalìe. Vediamo allora come sarebbero andate le cose se avessimo progettato la base di dati AEROPORTO facendo uso non della tabella relazionale Partenze ma di due tabelle relazionali:

— Giornale con schema R_1 = {volo, giorno, pilota}

— Imbarco con schema R_2 = {volo, ora}.

Quanto ai modelli $M_1 = [R_1, F_1]$ di Giornale e $M_2 = [R_2, F_2]$ di Imbarco, avremmo dovuto prendere per F_1 l'insieme delle dipendenze funzionali che sono implicate da F e sono applicabili ad R_1, e per F_2 l'insieme delle dipendenze funzionali che sono implicate da F e sono applicabili ad R_2. Dunque, a meno delle dipendenze funzionali banali che rappresentiamo con puntini di sospensione, avremmo

F_1 = {{volo, giorno} → pilota, {volo, giorno} → {volo, giorno, pilota}, …}

F_2 = {volo → ora, volo → {volo, ora}, …}.

È facile vedere che ora sia M_1 che M_2 sono entrambi modelli regolari e, quindi, in forma normale di Boyce-Codd. Dunque la base di dati che si ottiene dalla base di dati AEROPORTO sostituendo Partenze con Giornale e Imbarco è in forma normale di Boyce-Codd. Ma siamo sicuri che ora è tutto a posto? Quello che dobbiamo verificare è che la nuova base di dati abbia lo stesso "contenuto informativo" della base di dati AEROPORTO. Analizziamo per prima cosa il contenuto informativo delle due basi di dati dal punto di vista dell'utente e supponiamo che egli interroghi la base di dati AEROPORTO ponendo le tre domande che nell'Algebra Relazionale si esprimono come:

E_1 = Partenze
$E_2 = \pi_{\{\text{volo, giorno, pilota}\}}$ (Partenze)
$E_3 = \pi_{\{\text{volo, ora}\}}$ (Partenze)

Per fare le stesse domande alla nuova base di dati egli dovrà utilizzare le tre espressioni dell'Algebra Relazionale:

E'_1 = (Giornale) ⋈ (Imbarco)
E'_2 = (Giornale)
E'_3 = (Imbarco)

Ora, se la nuova base di dati ha lo stesso contenuto informativo di AEROPORTO, allora l'utente dovrà ricevere le stesse risposte qualunque sia il momento in cui pone le tre domande, vale a dire, se in ogni istante, detti d e d' rispettivamente gli stati della base di dati AEROPORTO e della nuova base di dati, i valori dell'espressione di E_i su d e dell'espressione E'_i su d' ($1 \leq i \leq 3$) coincidono. Indichiamo con r la relazione di nome Partenze nello stato d di AEROPORTO. Allora, per i valori delle espressioni E_i su d avremo:

$E_1(d) = r$
$E_2(d) = \pi_{\{\text{volo, giorno, pilota}\}}(r)$
$E_3(d) = \pi_{\{\text{volo, ora}\}}(r)$

Indichiamo, poi, con r_1 ed r_2 le relazioni di nome Giornale e di nome Imbarco nello stato d' della nuova base di dati. Per i valori delle espressioni E'_i su d' avremo:

$E'_1(d') = r_1 \bowtie r_2$
$E'_2(d') = r_1$
$E'_3(d') = r_2$

Dunque, dovremo avere

$E_1(d) = E'_1(d')$	se e solo se	$r = r_1 \bowtie r_2$
$E_2(d) = E'_2(d')$	se e solo se	$r_1 = \pi_{\{\text{volo, giorno, pilota}\}}(r)$
$E_3(d) = E'_3(d')$	se e solo se	$r_1 = \pi_{\{\text{volo, ora}\}}(r)$

che possiamo così riassumere:

$r = \pi_{R_1}(r) \bowtie \pi_{R_2}(r)$ $\qquad$ $r_1 = \pi_{R_1}(r)$ $\qquad$ $r_2 = \pi_{R_2}(r)$.

D'altra parte, visto che deve aversi

$$r = r_1 \bowtie r_2$$

ci aspettiamo che, se r_1 ed r_2 sono relazioni conformi rispettivamente ad M_1 e M_2, il prodotto join $r_1 \bowtie r_2$ sia una relazione conforme ad M.

Abbiamo, così, il seguente criterio di scomponibilità:

— ogni relazione r conforme ad M deve essere uguale al prodotto join delle sue proiezioni sugli schemi di M_1 ed M_2:

$$r = \pi_{R_1}(r) \bowtie \pi_{R_2}(r)$$

— per ogni coppia di relazioni r_1 ed r_2 conformi rispettivamente ad M_1 e M_2, il prodotto join $r_1 \bowtie r_2$ deve essere una relazione conforme ad M, cioè deve soddisfare tutte le dipendenze funzionali in F.

Se entrambe le condizioni sono rispettate, la scomposizione di M nella coppia M_1 ed M_2 viene detta "conservativa"; altrimenti, la scomposizione di M porta ad una perdita di informazione. Nei prossimi paragrafi daremo degli algoritmi per decidere se una scomposizione è o meno conservativa; dopodiché, daremo un algoritmo per trovare una scomposizione conservativa di un modello.

10.1 Scomposizioni conservative

Un *ricoprimento* di un insieme di attributi R è una famiglia di sottoinsiemi di R che non sono fra loro confrontabili rispetto all'inclusione e la cui unione coincide con R. Un particolare ricoprimento di R è la cosiddetta *partizione banale* di R, cioè $\{R\}$.
Sia $\boldsymbol{S} = \{R_1, \ldots, R_k\}$ un ricoprimento di R. Indichiamo con $M_h = [R_h, F_h]$ la proiezione di M su R_h ($1 \leq h \leq k$). L'insieme $\{M_1, \ldots, M_k\}$ prende il nome di *scomposizione* di M *indotta* da $\boldsymbol{S}$. Diremo che la scomposizione di M indotta da $\boldsymbol{S}$

— *conserva i dati* (gode della lossless join property, per dirla all'inglese) se ogni relazione conforme ad M è il prodotto join delle sue proiezioni su R_1, …, R_k;

— *conserva le dipendenze* se, per ogni relazione r_1 conforme ad M_1, …, e per ogni relazione r_k conforme ad M_k, il prodotto join delle relazioni r_1, …, r_k è una relazione conforme ad M;

— è *conservativa* se conserva sia i dati che le dipendenze.

Dunque, se una scomposizione non conserva i dati (ha un lossy join, per dirla all'inglese) allora, data una relazione r conforme ad M, in genere non abbiamo modo di ricostruire r a partire dalle sue proiezioni su R_1, …, R_k. Inoltre, se una scomposizione conserva i dati allora, date k relazioni r_1, …, r_k conformi rispettivamente ai modelli M_1, …, M_k, non è detto che il prodotto join di r_1, …, r_k sia una relazione conforme ad M. Ovviamente, la relazione

$$r_1 \bowtie \ldots \bowtie r_k$$

soddisfa tutte le dipendenze funzionali nel sottoinsieme $\langle G \rangle_R$ di $\langle F \rangle_R$, dove l'insieme G è così definito:

$$G = \bigcup_{h=1, \ldots, k} F_h$$

Tuttavia, potrebbe esserci una qualche dipendenza funzionale in $F - \langle G \rangle_R$ che sia violata dalla relazione $r_1 \bowtie \ldots \bowtie r_k$ e, in tal caso, la relazione non sarebbe conforme ad M. Dunque, perché si abbia la conservazione delle dipendenze, il modello $N = [R, G]$ deve essere equivalente ad M, cioè $\langle G \rangle_R = \langle F \rangle_R$. Ora, siccome $\langle G \rangle_R \subseteq \langle F \rangle_R$, N è equivalente ad M se e solo se $F \subseteq \langle G \rangle_R$. Dunque

La scomposizione di un modello $M = [R, F]$ indotta da un ricoprimento $\boldsymbol{S}$ di R conserva le dipendenze se e solo se ogni dipendenza funzionale in F è implicata da G.

Il modello $N = [R, G]$ prende il nome di *sottomodello* di M *generato* da $\boldsymbol{S}$.

Esempio 10.1 Sia R = {città, via, CAP} e sia F dato dalle due dipendenze funzionali:

{città, via} → CAP CAP → città

Avevamo già visto che il modello $M = [R, F]$ non è regolare. Ora, la scomposizione generata dal ricoprimento $\boldsymbol{S}$ = {{città, via}, {città, CAP}, {via, CAP}} di R non conserva le dipendenze per via della dipendenza funzionale {città, via} → CAP la quale appartiene ad F ma non è implicata dal sottomodello di M generato da $\boldsymbol{S}$.

■

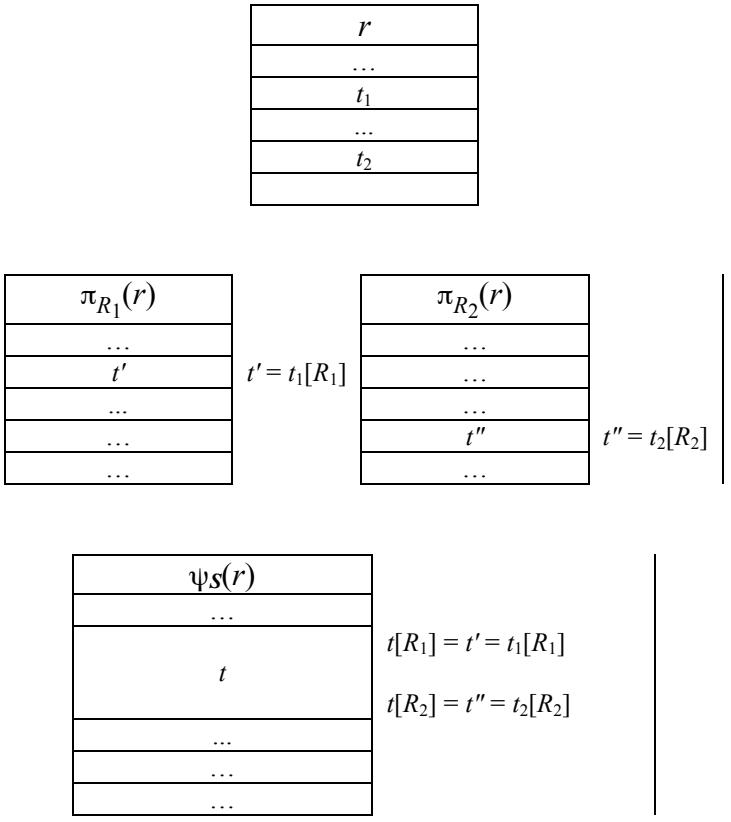
10.2 L'operatore di *proiezione-join*

Sia $M = [R, F]$ un modello e sia $\boldsymbol{S} = \{R_1, \ldots, R_k\}$ un ricoprimento di R. Si ricordi che la scomposizione di M indotta da $\boldsymbol{S}$ conserva i dati se ogni relazione r conforme ad M ogni relazione conforme ad M è il prodotto join delle sue proiezioni su R_1, …, R_k. Conviene esprimere tale requisito in termini di un nuovo operatore relazionale che chiamiamo operatore di *proiezione-join* associato ad $\boldsymbol{S}$ ed indichiamo con $\psi_{\boldsymbol{S}}$. Se r è

una qualsiasi relazione con schema R, $\psi_{\boldsymbol{S}}(r)$ è la relazione con schema R data dal prodotto join delle proiezioni di r su $R_1, \ldots, R_k$; esplicitamente, si ha:

$$\psi_{\boldsymbol{S}}(r) = \{t \in DOM(R): \exists\, t_1, \ldots, t_k \in r \quad t_1[R_1] = t[R_1], \ldots, t_k[R_k] = t[R_k]\}.$$

Nel caso k =2, la cosa può essere illustrata come segue.



Una relazione r con schema R si dice essere un *punto fisso* di $\psi_{\boldsymbol{S}}$ se $\psi_{\boldsymbol{S}}(r) = r$. Così, il requisito di conservazione dei dati può esprimersi alla maniera seguente:

> La scomposizione di un modello $M = [R, F]$ indotta da un ricoprimento $\boldsymbol{S}$ di R conserva i dati se e solo se ogni relazione conforme ad M è un punto fisso dell'operatore di proiezione-join $\psi_{\boldsymbol{S}}$.

Diamo ora alcune proprietà dell'operatore $\psi_{\boldsymbol{S}}$. Dimostriamo innanzitutto che $\psi_{\boldsymbol{S}}$ è un operatore di chiusura definito su $DOM(R)$.

(*estensività*) $r \subseteq \psi_{\boldsymbol{S}}(r)$
(*monotonicità*) se $r \subseteq r'$ allora $\psi_{\boldsymbol{S}}(r) \subseteq \psi_{\boldsymbol{S}}(r')$
(*idempotenza*) $\psi_{\boldsymbol{S}}(\psi_{\boldsymbol{S}}(r)) = \psi_{\boldsymbol{S}}(r)$

(*estensività*). Se $t \in r$, allora $t \in \psi_S(r)$ (basta prendere $t_1 = \ldots = t_k = t$).

(*monotonicità*). Se $r \subseteq r'$ e $t \in \psi_S(r)$, allora

$$\exists\, t_1, \ldots, t_k \in r \quad t_1[R_1] = t[R_1], \ldots, t_k[R_k] = t[R_k]$$

ma, siccome $r \subseteq r'$, le ennuple $t_1, \ldots, t_k$ appartengono anche ad r' e quindi $t \in \psi_S(r')$. Dunque $\psi_S(r) \subseteq \psi_S(r')$.

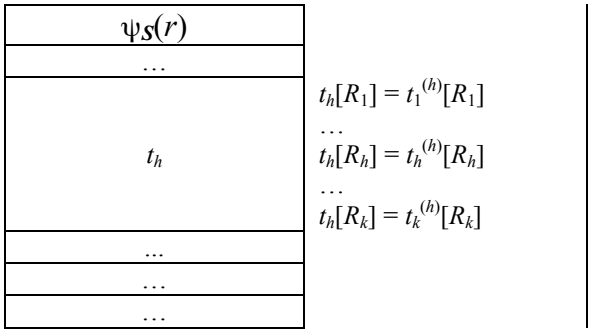
(*idempotenza*). Per la proprietà estensiva si ha $\psi_S(r) \subseteq \psi_S(\psi_S(r))$. Dunque, basta dimostrare che $\psi_S(\psi_S(r)) \subseteq \psi_S(r)$. Sia t una generica ennupla in $\psi_S(\psi_S(r))$. Vogliamo dimostrare che t appartiene anche alla relazione $\psi_S(r)$. Ora siccome t appartiene $\psi_S(\psi_S(r))$, per definizione esistono k ennuple $t_1, \ldots, t_k$ in $\psi_S(r)$ tali che $t_1[R_1] = t[R_1]$, $\ldots$, $t_k[R_k] = t[R_k]$.

$\psi_S(r)$
…
t_1
...
t_h
...
t_k
...

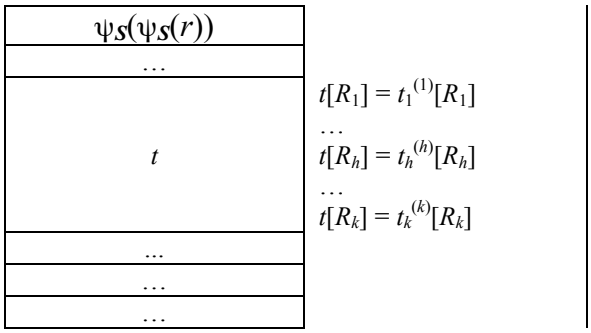
$\psi_S(\psi_S(r))$	
…	
t	$t[R_1] = t_1[R_1]$ … $t[R_h] = t_h[R_h]$ … $t[R_k] = t_k[R_k]$
...	
…	
…	

Consideriamo l'ennupla t_h, $1 \leq h \leq k$. Siccome t_h appartiene a $\psi_S(r)$, esistono k ennuple $t_1^{(h)}, \ldots, t_k^{(h)}$ in r tali che $t_1^{(h)}[R_1] = t_h[R_1], \ldots, t_k^{(h)}[R_k] = t_h[R_k]$.

r
…
$t_1^{(h)}$
...
$t_h^{(h)}$
...
$t_k^{(h)}$
...



Ne segue che esistono k ennuple $t_1^{(1)}$, …, $t_k^{(k)}$ in r tali che $t_1^{(1)}[R_1] = t[R_1]$, …, $t_k^{(k)}[R_k] = t[R_k]$, il che prova che t appartiene a $\psi_S(r)$.



Daremo ora un'altra utile proprietà dell'operatore ψ_S. Introduciamo la seguente relazione di equivalenza tra relazioni con schema R. Due relazioni r ed r', entrambe con schema R, si dicono *equivalenti modulo* $\boldsymbol{S}$ se

$$\pi_{R_1}(r) = \pi_{R_1}(r'),\ \ \ldots, \pi_{R_k}(r) = \pi_{R_k}(r').$$

Per ogni relazione r con schema R, indichiamo con $[r]_S$ la classe delle relazioni che sono equivalenti ad r modulo $\boldsymbol{S}$. Si osservi che, se due relazioni r' ed r'' appartengono ad $[r]_S$ cioè se

$$\pi_{R_h}(r) = \pi_{R_h}(r') = \pi_{R_h}(r''), \qquad (1 \le h \le k)$$

allora si ha anche $r' \cup r'' \in [r]_S$ poiché

$$\pi_{R_h}(r' \cup r'') = \pi_{R_h}(r') \cup \pi_{R_h}(r'') = \pi_{R_h}(r). \qquad (1 \le h \le k)$$

Dunque, la classe $[r]_S$ è chiusa rispetto all'unione. Pertanto, l'unione di tutte le relazioni in $[r]_S$ è ancora una relazione che appartiene a $[r]$. La relazione data dall'unione di tutte le relazioni in $[r]_S$ è detta l'*elemento massimale* di $[r]_S$.

Esempio 10.2 Siano A, B e C tre attributi binari. Si consideri la relazione r con schema {A, B, C} riportata in Tabella 10.1.

ABC
000
010
101

Tabella 10.1

Se $\boldsymbol{S} = \{\{A, B\}, \{B, C\}\}$, allora $[r]_{\boldsymbol{S}}$ contiene le sette relazioni

ABC	ABC	ABC	ABC	ABC	ABC	ABC
000	001	000	000	000	001	000
010	010	001	001	010	010	001
101	100	010	010	100	100	010
		100	101	101	101	100
						101

di cui la prima è quella riportata nella Tabella 10.1 e l'ultima è l'elemento massimale di $[r]_{\boldsymbol{S}}$.

■

Dimostriamo ora che non solo $\psi_{\boldsymbol{S}}(r) \in [r]_{\boldsymbol{S}}$ ma anche che $\psi_{\boldsymbol{S}}(r)$ è l'elemento massimale (rispetto all'inclusione insiemistica) della classe $[r]_{\boldsymbol{S}}$.

Per la proprietà estensiva di $\psi_{\boldsymbol{S}}$, si ha che $r \subseteq \psi_{\boldsymbol{S}}(r)$ e quindi

$$\pi_{R_1}(r) \subseteq \pi_{R_1}(\psi_{\boldsymbol{S}}(r)) \quad \ldots \quad \pi_{R_k}(r) \subseteq \pi_{R_k}(\psi_{\boldsymbol{S}}(r)).$$

Sarà allora sufficiente provare che

$$\pi_{R_h}(\psi_{\boldsymbol{S}}(r)) \subseteq \pi_{R_h}(r) \qquad (h = 1, \ldots, k).$$

Sia t' una generica ennupla in $\pi_{R_h}(\psi_{\boldsymbol{S}}(r))$. Allora esiste un'ennupla t in $\psi_{\boldsymbol{S}}(r)$ tale che $t[R_h] = t'$. D'altra parte, siccome $t \in \psi_{\boldsymbol{S}}(r)$, r contiene k ennuple $t_1, \ldots, t_k$ tali che

$$t[R_1] = t_1[R_1] \quad \ldots \quad t[R_k] = t_k[R_k].$$

Ora, siccome $t_h \in r$, si ha anche che $t_h[R_h] \in \pi_{R_h}(r)$, ma $t_h[R_h] = t[R_h] = t'$, cioè $t' \in \pi_{R_h}(r)$ e, quindi, $\pi_{R_h}(\psi_{\boldsymbol{S}}(r)) \subseteq \pi_{R_h}(r)$.

r
…
t_1
...
t_h
...
t_k
...

$\pi_{R_h}(r)$
…
$t_h[R_h]$
...
…
…

$\psi_{\boldsymbol{S}}(r)$	
…	
t	$t[R_1] = t_1[R_1]$ … $t[R_h] = t_h[R_h]$ … $t[R_k] = t_k[R_k]$
...	
…	
…	

$\pi_{R_h}(\psi_{\boldsymbol{S}}(r))$	
…	
t'	$t' = t[R_h] = t_h[R_h]$
...	
…	
…	

Dunque $\psi_{\boldsymbol{S}}(r)$ appartiene alla classe $[r]_{\boldsymbol{S}}$. Sia ora r^* l'elemento massimale della classe $[r]_{\boldsymbol{S}}$. Siccome $[r]_{\boldsymbol{S}} = [r^*]_{\boldsymbol{S}}$, si ha che le due relazioni r^* e $\psi_{\boldsymbol{S}}(r^*)$ sono equivalenti modulo $\boldsymbol{S}$ e, siccome r^* è l'elemento massimale di $[r]_{\boldsymbol{S}}$, si ha che $\psi_{\boldsymbol{S}}(r^*) \subseteq r^*$. D'altra parte, per la proprietà estensiva di $\psi_{\boldsymbol{S}}$, si ha che r^* è un sottoinsieme di $\psi_{\boldsymbol{S}}(r^*)$. Dunque, $\psi_{\boldsymbol{S}}(r^*) = r^*$. Infine, siccome le due relazioni r ed r^* sono equivalenti modulo $\boldsymbol{S}$, si ha che $\psi_{\boldsymbol{S}}(r^*) = \psi_{\boldsymbol{S}}(r)$ e, quindi, $\psi_{\boldsymbol{S}}(r) = r^*$.

10.3 Test di conservazione dei dati

Daremo un algoritmo per decidere se la scomposizione di un modello $M = [R, F]$ indotta da un ricoprimento di R conserva i dati. L'algoritmo si applica però solo a modelli "semplici", che ora introduciamo.

Un modello $M = [R, F]$ è un *modello semplice* se

— M è minimale (nessuna dipendenza funzionale in F è ridondante),
— il determinante di ogni dipendenza funzionale in F è ridotto ai minimi termini,
— il dipendente di ogni dipendenza funzionale in F è formato da un singolo attributo.

Si osservi che, dato un qualsiasi modello M, è sempre possibile trovare con gli Algoritmi di Riduzione e di Contrazione un modello semplice M' che sia equivalente ad M.

Dunque, sia $M = [R, F]$ un modello, che senza perdere in generalità assumiamo sia semplice, e sia $\boldsymbol{S}$ un ricoprimento di R. Possiamo rappresentare il ricoprimento $\boldsymbol{S}$ mediante una matrice di simboli, che chiamiamo il *tableau* associato ad $\boldsymbol{S}$. Siano $A_1, \ldots, A_n$ ed $R_1, \ldots, R_k$ due ordinamenti arbitrari rispettivamente degli attributi in R e degli insiemi in $\boldsymbol{S}$, e sia poi $Q = \{1, 2, \ldots, q\}$, dove

$$q = n\,(k+1) - \sum_{h=1,\,\ldots k} |R_h|.$$

Il tableau associato ad $\boldsymbol{S}$ è la matrice $T = ((T_{h,i}))$, $1 \le h \le k$ ed $1 \le i \le n$, che si ottiene con la seguente procedura:

(1) $p := n$;
(2) per $h = 1, \ldots, k$
(3) per $i = 1, \ldots, n$, se $A_i \in R_h$ allora $T_{h,i} := i$ altrimenti $p := p+1$ e $T_{h,i} := p$.

Indichiamo con T_h l'h.ma riga di T, $1 \le h \le n$. Se X è un sottoinsieme di R, con $T_h[X]$ indicheremo la stringa di numeri che nella riga T_h di T corrispondono agli attributi presenti in X. Dato un modello semplice $M = [R, F]$ e il tableau T associato ad un ricoprimento $\boldsymbol{S}$ di R, il seguente algoritmo consente di decidere se la scomposizione di M indotta da $\boldsymbol{S}$ conserva o meno i dati.

Test di Conservazione dei Dati

(Algoritmo di Aho-Beeri-Ullman, 1979)

Dati: Un modello semplice $M = [R, F]$ ed un ricoprimento $\boldsymbol{S} = \{R_1, \ldots, R_k\}$ di R.

Risultato: Un valore della variabile logica *test*.

(1) *test* := FALSO.

(2) Costruire il tableau T associato ad $\boldsymbol{S}$.

(3) Finché il tableau T non possa essere ulteriormente modificato, ripetere:

Per ogni dipendenza funzionale $X \rightarrow A_i$ in F,
se T contiene due righe T_h e $T_{h'}$ tali che
$T_h[X] = T_{h'}[X]$ e $T_{h,i} < T_{h',i}$, allora, $T_{h',i} := T_{h,i}$.

(4) Se T contiene la riga $(1, \ldots, n)$, allora *test* := VERO.

Si osservi che possiamo rendere più veloce l'esecuzione del Test sostituendo al passo (3) l'istruzione

$$T_{h',i} := T_{h,i}$$

con "sostituire nella colonna i-esima ogni occorrenza di $T_{h',i}$ con $T_{h,i}$." Il motivo è che, se esiste un'altra riga $T_{h''}$ tale che $T_{h'',i} = T_{h',i}$, allora in un precedente tableau si aveva $T_{h'',i} > T_{h',i}$ e l'eguaglianza è stata ottenuta applicando una dipendenza funzionale $Y \rightarrow A_i$ in F alle due righe $T_{h''}$ e $T_{h'}$ che allora si trovavano nella situazione $T_{h'}[Y] = T_{h''}[Y]$. Ora supponiamo di aver eseguito l'istruzione $T_{h',i} := T_{h,i}$. A questo punto, riapplicando la dipendenza funzionale $Y \rightarrow A_i$ alle due righe $T_{h''}$ e $T_{h'}$ dovremo porre $T_{h'',i} := T_{h',i}$ $(= T_{h,i})$.

Diremo che il Test di Conservazione dei Dati *dà esito positivo* (rispettivamente, *negativo*) se il valore finale della variabile *test* è VERO (rispettivamente, FALSO).

Esempio 10.3 Consideriamo il modello semplice $M = [R, F]$, dove $R = \{A, B, C, D\}$ ed $F = \{A \rightarrow B, \{A, C\} \rightarrow D\}$. Il tableau associato al ricoprimento $\boldsymbol{S} = \{\{A, B\}, \{A, C, D\}\}$ di R è riportato in Tabella 10.2.

A	*B*	*C*	*D*
1	2	5	6
1	7	3	4

Tabella 10.2

La dipendenza funzionale $A \rightarrow B$ porta a sostituire (nella seconda riga) 7 con 2, ed il nuovo tableau è riportato in Tabella 10.3.

A	*B*	*C*	*D*
1	2	5	6
1	2	3	4

Tabella 10.3

La dipendenza funzionale $\{A, C\} \rightarrow D$ non modifica il contenuto di questo tableau e, siccome questo contiene la riga (1, 2, 3, 4), possiamo concludere che il Test di Conservazione dei Dati dà esito positivo. ■

Esempio 10.4 Consideriamo il modello semplice $M = [R, F]$, dove $R = \{A, B, C, D\}$ ed $F = \{A \rightarrow B, C \rightarrow D\}$. Il tableau associato al ricoprimento $\boldsymbol{S} = \{\{A, B\}, \{C, D\}\}$ di R è riportato in Tabella 10.4.

A	*B*	*C*	*D*
1	2	5	6
7	8	3	4

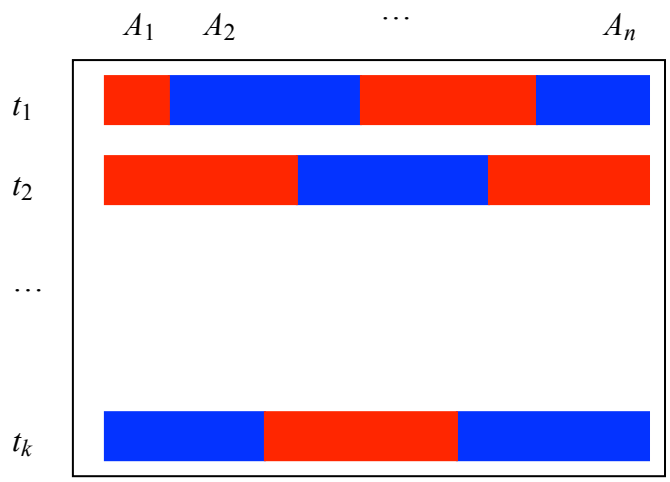
Tabella 10.4

Esso non viene modificato né da $A \rightarrow B$ né da $C \rightarrow D$. Pertanto, il Test di Conservazione dei Dati dà esito negativo. ■

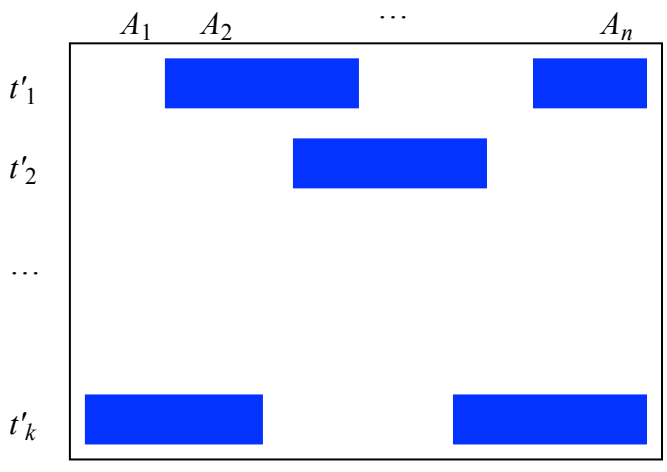
***Lemma* 10.1** Sia $M = [R, F]$ un modello semplice e sia $\boldsymbol{S}$ un ricoprimento di R. Se la scomposizione di M indotta da $\boldsymbol{S}$ conserva i dati, allora il Test di Conservazione dei Dati dà esito positivo.

Dimostrazione. Siano $R = \{A_1, \ldots, A_n\}$ ed $\boldsymbol{S} = \{R_1, \ldots, R_k\}$. Siano poi $T^{(0)}$ il tableau associato ad $\boldsymbol{S}$ e T il tableau finale prodotto dal Test di Conservazione dei Dati. Associamo ai numeri 1, …, q altrettanti elementi $a_1, \ldots, a_q$ dell'unione dei domini di $A_1, \ldots, A_n$ in maniera tale che, se il numero p, $1 \leq p \leq q$, compare nella colonna i.esima di $T^{(0)}$, allora a_p appartiene al dominio di A_i. In questo modo, a $T^{(0)}$ e a T restano associate due relazioni con schema R; indichiamole con $r^{(0)}$ ed r. Si osservi che, mentre non è detto che la relazione $r^{(0)}$ sia conforme ad M, è sempre vero che la relazione r è conforme ad M perché soddisfa tutte le dipendenze funzionali in F (ogni volta che si è incontrata una dipendenza funzionale f in F che era violata dal tableau corrente, il tableau è stato modificato in maniera tale da soddisfare f). Ora, siccome

per ipotesi la scomposizione di M indotta da $\boldsymbol{S}$ conserva i dati, r è un punto fisso di $\psi_{\boldsymbol{S}}$ e, quindi, $\psi_{\boldsymbol{S}}(r) = r$.
Indichiamo ora con t^* l'ennupla $(a_1, \ldots, a_n)$. L'assunto sarà provato dimostrando che r contiene t^*. Siano $t_1, t_2, \ldots, t_k$ le ennuple in r che corrispondono alle k righe di $T^{(0)}$.



Si osservi innanzitutto che, siccome $T^{(0)}_{h,i} \leq n$ se e solo se $A_i \in R_h$, si ha sempre $t_h[R_h] = t^*[R_h]$; inoltre, siccome $\boldsymbol{S}$ è un ricoprimento di R, l'ennupla t^* può essere ottenuta prendendo il prodotto join delle k ennuple $t_1[R_1]$, $t_2[R_2]$, …, $t_k[R_k]$ e, quindi, $t^* \in \psi_{\boldsymbol{S}}(r^{(0)})$. Siano ora $t'_1, \ldots, t'_k$ le ennuple in r. Per come opera il Test di Conservazione dei dati, si ha $t_h[R_h] = t'_h[R_h]$ per ogni h, $1 \leq h \leq k$:



Pertanto, l'ennupla t^* appartiene anche alla relazione $\psi_{\boldsymbol{S}}(r) = r$. Dunque, r contiene l'ennupla t^* e, quindi, T contiene la riga $(1, \ldots, n)$. □

***Teorema* 10.1** Dati un modello semplice $M = [R, F]$ ed un ricoprimento $\boldsymbol{S}$ di R, la scomposizione di M indotta da $\boldsymbol{S}$ conserva i dati se e solo se il Test di Conservazione dei Dati dà esito positivo.

Dimostrazione. Per il Lemma 10.1 è sufficiente dimostrare che, se il Test di Conservazione dei Dati dà esito positivo, allora la scomposizione di M indotta da $\boldsymbol{S}$ conserva i dati.

Sia $R = \{A_1, \ldots, A_n\}$ e sia $T^{(0)}$ il tableau associato ad $\boldsymbol{S}$. Una *valutazione* di $T^{(0)}$ è una funzione α che associa ad ogni p, $1 \le p \le q$, un elemento di $dom(A_i)$ se p compare nella colonna i-esima. Per ogni riga $T^{(0)}_h = (p_{h1}, \ldots, p_{hn})$, $1 \le h \le k$, indichiamo con $\alpha(T^{(0)}_h)$ il risultato della valutazione α di $T^{(0)}$, cioè

$$\alpha(T^{(0)}_h) = (\alpha(p_{h1}), \ldots, \alpha\, p_{hn})).$$

Così, ogni $\alpha(T^{(0)}_h)$ è un'ennupla su R.

Sia ora r una qualsiasi relazione con schema R. Una valutazione α di $T^{(0)}$ è *compatibile* con r se per ogni h, $1 \le h \le k$, $\alpha(T^{(0)}_h)$ è un'ennupla appartenente ad r. Indichiamo poi con $T^{(0)}(r)$ la relazione con schema R così definita:

$$T^{(0)}(r) = \{\alpha(1), \ldots, \alpha(n): \alpha \text{ è una valutazione di } T^{(0)} \text{ compatibile con } r\}.$$

È allora facile dimostrare che $T^{(0)}(r) = \psi_{\boldsymbol{S}}(r)$.

Ora, quando applichiamo l'algoritmo di Aho-Beeri-Ullman, il tableau viene modificato e siano $T^{(0)}$, $T^{(1)}$, …, $T^{(m)}$ i valori distinti del tableau. Consideriamo la generica trasformazione da $T^{(l)}$ a $T^{(l+1)}$. Questa è il risultato dell'applicazione a $T^{(l)}$ di una dipendenza f in F. Ora, se f è soddisfatta da r, allora $T^{(l)}(r) = T^{(l+1)}(r)$; altrimenti, $T^{(l+1)}(r)$ è un sottoinsieme proprio di $T^{(l)}(r)$. Ne segue che, se $r \in SAT(F)$ (cioè se r è conforme ad M), allora $T^{(0)}(r) = T^{(1)}(r) = \ldots = T^{(m)}(r)$; altrimenti, $T^{(0)}(r) \supset T^{(m)}(r)$.

Supponiamo ora che $T^{(m)}$ contenga la riga $(1, 2, \ldots, n)$. Allora $T^{(m)}(r) = r$. D'altra parte se r è conforme ad M, siccome $T^{(0)}(r) = \psi_{\boldsymbol{S}}(r)$, abbiamo che $T^{(m)}(r) = \psi_{\boldsymbol{S}}(r)$. In definitiva, per ogni relazione conforme ad M si ha sia $T^{(m)}(r) = \psi_{\boldsymbol{S}}(r)$ sia $T^{(m)}(r) = r$, cioè $\psi_{\boldsymbol{S}}(r) = r$ e, dunque, la scomposizione di M indotta da $\boldsymbol{S}$ conserva i dati. □

Il Teorema 10.1 ha come conseguenza il seguente risultato di cui omettiamo la dimostrazione.

***Corollario* 10.1** Siano $M = [R, F]$ un modello semplice ed $\boldsymbol{S} = \{R_1, R_2\}$ un ricoprimento di R. La scomposizione di M indotta da $\boldsymbol{S}$ conserva i dati se e solo se $R_1 \cap R_2 \neq \varnothing$ e almeno una delle due dipendenze funzionali $R_1 \cap R_2 \rightarrow R_1 - R_2$ ed $R_1 \cap R_2 \rightarrow R_2 - R_1$ appartiene ad $\langle F \rangle_R$.

Il teorema che segue consente di estendere il risultato del Test di Conservazione dei Dati che si ottiene per un certo ricoprimento a tutta una classe di ricoprimenti.

***Teorema* 10.2** Dato un modello $M = [R, F]$, siano $\boldsymbol{S}$ e $\boldsymbol{S}'$ due ricoprimenti di R tali che ogni insieme in $\boldsymbol{S}$ è contenuto in almeno un insieme in $\boldsymbol{S}'$. Se la scomposizione di M indotta da $\boldsymbol{S}$ conserva i dati, allora anche la scomposizione di M indotta da $\boldsymbol{S}'$ conserva i dati. Se la scomposizione di M indotta da $\boldsymbol{S}'$ non conserva i dati, allora neppure la scomposizione di M indotta da $\boldsymbol{S}$ conserva i dati.

Dimostrazione. Siano $\boldsymbol{S} = \{R_1, \ldots, R_k\}$ ed $\boldsymbol{S}' = \{R'_1, \ldots, R'_m\}$. Per ipotesi, per ogni h, $1 \leq h \leq k$, esiste un insieme in $\boldsymbol{S}'$ che contiene R_h, sia $R'_{j(h)}$ uno qualsiasi di questi insiemi. Consideriamo ora una qualsiasi relazione r con schema R. Si ricordi che le due relazioni $\psi_{\boldsymbol{S}}(r)$ e $\psi_{\boldsymbol{S}'}(r)$ sono così definite:

$$\psi_{\boldsymbol{S}}(r) = \{t \in DOM(R): \exists\, t_1, \ldots, t_k \in r \quad t_1[R_1] = t[R_1], \ldots, t_k[R_k] = t[R_k]\}$$

$$\psi_{\boldsymbol{S}'}(r) = \{t \in DOM(R): \exists\, t'_1, \ldots, t'_m \in r \quad t'_1[R'_1] = t[R'_1], \ldots, t'_m[R'_m] = t[R'_m]\}$$

Vogliamo dimostrare che $\psi_{\boldsymbol{S}'}(r) \subseteq \psi_{\boldsymbol{S}}(r)$, cioè che ogni ennupla t in $\psi_{\boldsymbol{S}'}(r)$ appartiene anche a $\psi_{\boldsymbol{S}}(r)$. Siccome t appartiene a $\psi_{\boldsymbol{S}'}(r)$, esistono m ennuple $t'_1, \ldots, t'_m$ nella relazione r tali che

$$t[R'_1] = t'_1[R'_1], \ldots, t[R'_m] = t'_m[R'_m].$$

Ora, siccome $R_h \subseteq R'_{j(h)}$, abbiamo anche che

$$t[R_1] = t'_{j(1)}[R_1], \ldots, t[R_k] = t'_{j(k)}[R_k].$$

Siano allora $t_1 = t'_{j(1)}, \ldots, t_k = t'_{j(k)}$. Abbiamo allora che

$$t[R_1] = t_1[R_1], \ldots, t[R_k] = t_k[R_k]$$

la qual cosa prova che t appartiene anche a $\psi_{\boldsymbol{S}}(r)$. Dunque, abbiamo che

$$r \subseteq \psi_{\boldsymbol{S}'}(r) \subseteq \psi_{\boldsymbol{S}}(r).$$

Pertanto, se r è un punto fisso di $\psi_{\boldsymbol{S}}$, allora r è anche un punto fisso di $\psi_{\boldsymbol{S}'}$; mentre, se r non è un punto fisso di $\psi_{\boldsymbol{S}'}$, allora r non è un punto fisso neppure di $\psi_{\boldsymbol{S}}$. □

10.4 Test di conservazione delle dipendenze

Siano $M = [R, F]$ un modello semplice, $\boldsymbol{S}$ un ricoprimento di R ed $N = [R, G]$ il sottomodello di M generato da $\boldsymbol{S}$. Daremo un algoritmo per decidere se la scomposizione di M indotta da $\boldsymbol{S}$ conserva le dipendenze, cioè se ogni dipendenza funzionale in F è implicata da G. L'algoritmo è basato sul fatto che una dipendenza funzionale $X \rightarrow A$ in F è implicata da G se e solo se l'attributo A appartiene ad $\langle X \rangle_G$.

Ora, per calcolare $\langle X \rangle_G$ non conviene applicare l'algoritmo di Bernstein, perché questo richiederebbe il calcolo preliminare di G che è un insieme esageratamente grande. Possiamo invece applicare il seguente algoritmo.

***Algoritmo di Beeri-Honeyman* (1982)**

Dati: Un modello semplice $M = [R, F]$, un ricoprimento $\boldsymbol{S} = \{R_1, \ldots, R_k\}$ di R ed un sottoinsieme non-vuoto X di R.

Risultato: Un sovrainsieme V di X.

Procedura

(1) $V := X$;

(2) Finché V non possa essere ulteriormente ampliato ripetere:

Per $h = 1, \ldots, k$,

calcolare $\langle V \cap R_h \rangle_F$ con l'algoritmo di Bernstein; aggiungere a V tutti gli attributi che appartengono sia ad R_h che a $\langle V \cap R_h \rangle_F$, cioè

$$V := V \cup (R_h \cap \langle V \cap R_h \rangle_F).$$

Si osservi che l'algoritmo termina quando, per ogni h, $1 \leq h \leq k$, tutti gli attributi in

$$R_h \cap \langle V \cap R_h \rangle_F$$

fanno già parte di V.

***Lemma* 10.2** L'algoritmo di Beeri-Honeyman calcola correttamente $\langle X \rangle_G$.

Dimostrazione. Siano $V_0, V_1, \ldots, V_m$ i valori distinti che sono man mano assunti dalla variabile insiemistica V durante l'esecuzione dell'algoritmo. Così,

$$V_0 = X \subset V_1 \subset \ldots \subset V_m.$$

Per provare che $V_m = \langle X \rangle_G$, dimostreremo che:

(*a*) V_m è un sottoinsieme di $\langle X \rangle_G$,

(*b*) $\langle X \rangle_G$ è un sottoinsieme di V_m.

(*a*) Dimostreremo per induzione che per ogni i, $0 \leq i \leq m$, $V_i \subseteq \langle X \rangle_G$.

PASSO BASE. Per $i = 0$, l'inclusione $V_i \subseteq \langle X \rangle_G$ è ovvia perché $V_0 = X$.

PASSO INDUTTIVO. Assumiamo, per ipotesi, che l'inclusione $V_i \subseteq \langle X \rangle_G$ valga per un certo i, $0 \leq i < m$. Vogliamo dimostrare che vale anche l'inclusione $V_{i+1} \subseteq \langle X \rangle_G$. A tale scopo, basterà dimostrare che l'insieme

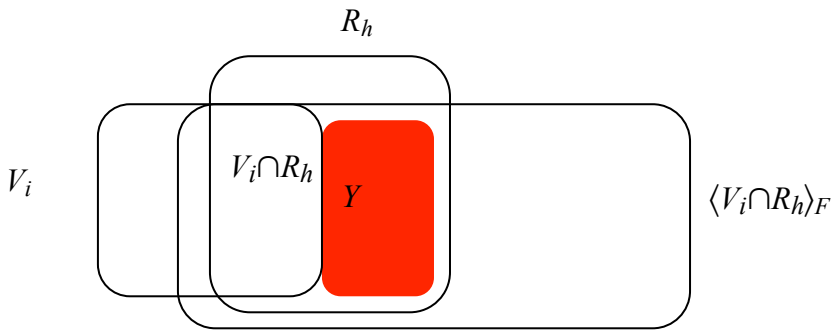
$$Y = V_{i+1} - V_i$$

è un sottoinsieme di $\langle V_i \rangle_G$ perché allora, per l'ipotesi induttiva, avremo

$$V_{i+1} = V_i \cup Y \subseteq V_i \cup \langle V_i \rangle_G = \langle V_i \rangle_G \subseteq \langle \langle X \rangle_G \rangle_G = \langle X \rangle_G.$$

Ora, supponiamo che V_{i+1} sia stato ottenuto da V_i quando, nella scansione di $\boldsymbol{S}$, abbiamo esaminato l'insieme R_h così che si ha

$$V_{i+1} = V_i \cup (R_h \cap \langle V_i \cap R_h \rangle_F) \qquad Y \subseteq R_h \cap \langle V_i \cap R_h \rangle_F$$



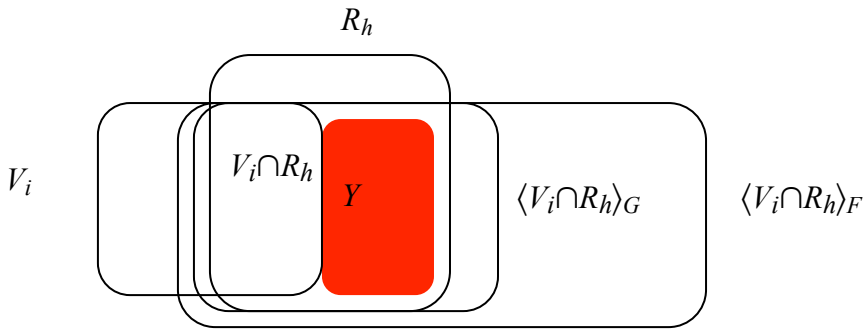
e quindi

$$Y \subseteq R_h \qquad \text{e} \qquad Y \subseteq \langle V_i \cap R_h \rangle_F .$$

Ora, siccome Y è un sottoinsieme di $\langle V_i \cap R_h \rangle_F$, la dipendenza funzionale

$$f : V_i \cap R_h \rightarrow Y$$

è implicata da F; inoltre, siccome sia il determinante (cioè, l'insieme $V_i \cap R_h$) della dipendenza funzionale f sia il suo dipendente (cioè Y) sono sottoinsiemi di R_h, f è applicabile ad R_h. Ne viene che f appartiene ad F_h e, quindi, a G. Pertanto, Y deve essere un sottoinsieme di $\langle V_i \cap R_h \rangle_G$, cioè $Y \subseteq \langle V_i \cap R_h \rangle_G$.



Ne segue che

$$Y \subseteq \langle V_i \cap R_h \rangle_G \subseteq \langle V_i \rangle_G$$

che, come si è detto, è quanto basta per concludere che $V_{i+1} \subseteq \langle X \rangle_G$.
Dunque possiamo concludere che per ogni i, $1 \leq i \leq m$, vale l'inclusione $V_i \subseteq \langle X \rangle_G$. In particolare, per $i = m$, abbiamo che il valore finale V_m di V calcolato dall'algoritmo è un sottoinsieme di $\langle X \rangle_G$.

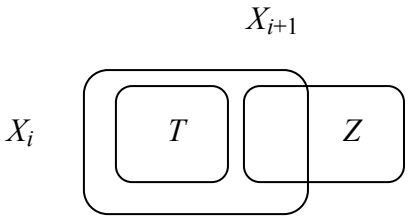
(*b*) Supponiamo, in via teorica, di conoscere G e di applicare l'algoritmo di Bernstein per calcolare la chiusura di X rispetto a G. Siano $X_0, X_1, \ldots, X_n$ i valori tra loro distinti man mano trovati durante l'esecuzione dell'algoritmo; così

$$X_0 = X \subset X_1 \subset \ldots \subset X_n = \langle X \rangle_G.$$

Dimostriamo ora per induzione che ogni X_i, $0 \leq i \leq n$, è un sottoinsieme di V_m.

PASSO BASE. Per $i = 0$, l'inclusione $X_i \subseteq V_m$ è ovvia perché $X \subseteq V_m$.

PASSO INDUTTIVO. Assumiamo, per ipotesi, che l'inclusione $X_i \subseteq V_m$ valga per un certo i, $0 \leq i < n$; dimostriamo che anche X_{i+1} è un sottoinsieme di V_m.
Supponiamo che X_{i+1} sia stato ottenuto a partire da X_i quando, nella scansione di G, abbiamo esaminato la dipendenza funzionale $f : T \rightarrow Z$. Così, abbiamo che $T \subseteq X_i$ e $X_{i+1} = X_i \cup Z$.



Siccome T è un sottoinsieme di X_i, per l'ipotesi induttiva abbiamo che

$$T \subseteq V_m.$$

Basterà allora dimostrare che anche l'insieme Z è un sottoinsieme di V_m perché allora

$$X_{i+1} = X_i \cup Z \subseteq V_m .$$

Si osservi ora che, siccome f è in G, la dipendenza funzionale f è implicata da F ed è applicabile ad un qualche insieme R_h in $\boldsymbol{S}$, cioè $T \subseteq R_h$ e $Z \subseteq R_h$. Ora, siccome $T \subseteq R_h$ e $T \subseteq V_m$, abbiamo che

$$T \subseteq V_m \cap R_h .$$

Inoltre, siccome f è implicata da F**,** abbiamo che

$$\langle T \rangle_F \subseteq \langle V_m \cap R_h \rangle_F \ .$$

D'altra parte, essendo f è implicata da F, abbiamo $Z \subseteq \langle T \rangle_F$ e, siccome $Z \subseteq R_h$,

$$Z \subseteq R_h \cap \langle T \rangle_F \subseteq R_h \cap \langle V_m \cap R_h \rangle_F .$$

Infine, siccome V_m è il valore finale della variabile insiemistica V, nell'ultima iterazione dell'algoritmo, tutti gli attributi in $R_h \cap \langle V_m \cap R_h \rangle_F$ fanno già parte di V_m, cioè

$$R_h \cap \langle V_m \cap R_h \rangle_F \subseteq V_m ,$$

e quindi

$$Z \subseteq R_h \cap \langle T \rangle_F \subseteq R_h \cap \langle V_m \cap R_h \rangle_F \subseteq V_m$$

che, come si è detto, è quanto basta per concludere che $X_{i+1} \subseteq V_m$.

Dunque possiamo concludere che per ogni i, $1 \leq i \leq n$, vale l'inclusione $X_i \subseteq V_m$. In particolare, per $i = n$, abbiamo che $X_n = \langle X \rangle_G$ è un sottoinsieme di V_m. □

Quanto alla complessità dell'Algoritmo di Beeri-Honeyman, si osservi che il numero m delle fasi (vedi punto (1) della Dimostrazione del Lemma 10.2) è minore o uguale a $|R|-1$, ed ogni fase richiede l'esecuzione di k operazioni in ciascuna delle quali è dominante il calcolo di $\langle V \cap R_h \rangle_F$ che richiede un tempo lineare nella dimensione di M. Se questa è q, allora la complessità dell'algoritmo è $O(|R| \times q \times |\boldsymbol{S}|)$.

Test di Conservazione delle Dipendenze

Dati:	Un modello semplice $M = [R, F]$ ed un ricoprimento $\boldsymbol{S} = \{R_1, \ldots, R_k\}$ di R.
Risultato:	Un valore della variabile logica *test*.

Procedura

(1) *test* := VERO

(2) Per ogni dipendenza funzionale $X \rightarrow A$ in F

calcolare $\langle X \rangle_G$ con l'Algoritmo di Beeri-Honeyman;

se $A \notin \langle X \rangle_G$ allora *test* := FALSO ed Uscire.

Diremo che il Test di Conservazione delle Dipendenze *dà esito positivo* (rispettivamente, *negativo*) se il valore finale della variabile *test* è VERO (rispettivamente, FALSO).

Come diretta conseguenza al Lemma 10.2 abbiamo che

***Teorema* 10.3** Sia M un modello semplice $M = [R, F]$. La scomposizione di M indotta da un ricoprimento $\boldsymbol{S}$ di R conserva le dipendenze se e solo se il Test di Conservazione delle Dipendenze dà esito positivo.

Esempio 10.5 Consideriamo il modello semplice $M = [R, F]$

$$R = \{A, B, C, D\} \qquad \text{ed} \qquad F = \{A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow A\}$$

e la scomposizione di M indotta dal ricoprimento $\boldsymbol{S} = \{R_1 = \{A, B\}, R_2 = \{B, C\}, R_3 = \{C, D\}\}$ di R. Le proiezioni di M su $\boldsymbol{S}$ sono

$$M_1 = [R_1, F_1] \qquad M_2 = [R_2, F_2] \qquad M_3 = [R_3, F_3]$$

dove F_h, $1 \leq h \leq 3$, contiene tutte e nove le dipendenze funzionali applicabili ad R_h. L'insieme

$$G = F_1 \cup F_2 \cup F_3$$

contiene $3 \times 9 = 27$ dipendenze funzionali, tra cui le dipendenze funzionali $A \rightarrow B$, $B \rightarrow C$ e $C \rightarrow D$. Pertanto, il modello $N = [R, G]$ è equivalente ad M se e solo se la dipendenza funzionale $D \rightarrow A$ è implicata da G, cioè se e solo se l'attributo A appartiene alla chiusura dell'insieme $\{D\}$ rispetto a G. Per calcolare $\langle \{D\} \rangle_G$

applichiamo l'Algoritmo di Beeri-Honeyman. Il valore iniziale della variabile insiemistica V è $\{D\}$. Dopo la prima scansione di $\boldsymbol{S}$, il valore di V è $\{C, D\}$. Dopo la seconda scansione di $\boldsymbol{S}$, il valore di V è $\{B, C, D\}$. Dopo la terza scansione di $\boldsymbol{S}$, il valore di V è $\{A, B, C, D\}$, cioè $V = R$. L'Algoritmo di Beeri-Honeyman termina dopo la quarta scansione di $\boldsymbol{S}$ che lascia il valore di V inalterato. Dunque, anche la dipendenza funzionale $D \rightarrow A$ è implicata da G e, dunque, la scomposizione di M indotta da $\boldsymbol{S}$ conserva le dipendenze.

■

Il teorema che segue consente di estendere il risultato del Test di Conservazione delle Dipendenze che si ottiene per un certo ricoprimento a tutta una classe di ricoprimenti.

***Teorema* 10.4** Dato un modello $M = [R, F]$, siano $\boldsymbol{S}$ e $\boldsymbol{S}'$ due ricoprimenti di R tali che ogni insieme in $\boldsymbol{S}$ è contenuto in almeno un insieme in $\boldsymbol{S}'$. Se la scomposizione di M indotta da $\boldsymbol{S}$ conserva le dipendenze, allora anche la scomposizione di M indotta da $\boldsymbol{S}'$ conserva le dipendenze. Se la scomposizione di M indotta da $\boldsymbol{S}'$ non conserva le dipendenze, allora neppure la scomposizione di M indotta da $\boldsymbol{S}$ conserva le dipendenze.

Dimostrazione. Siano $\boldsymbol{S} = \{R_1, \ldots, R_k\}$ ed $\boldsymbol{S}' = \{R'_1, \ldots, R'_m\}$. Per ipotesi, per ogni h, $1 \leq h \leq k$, esiste un insieme in $\boldsymbol{S}'$ che contiene R_h, sia $R'_{j(h)}$ uno qualsiasi di questi insiemi. Siano $N = [R, G]$ ed $N' = [R, G']$ i sottomodelli di M generati rispettivamente da $\boldsymbol{S}$ e $\boldsymbol{S}'$. Vogliamo dimostrare che $G \subseteq G'$. Consideriamo ora una qualsiasi dipendenza funzionale f in G. Siccome f appartiene a G, f appartiene ad $\langle F \rangle_R$ ed è applicabile ad un insieme R_h per qualche valore di h, $1 \leq h \leq k$. Inoltre, siccome $R_h \subseteq R'_{j(h)}$, f è applicabile anche a $R'_{j(h)}$ e, quindi, appartiene anche a G'. Ne viene che

$$\langle G \rangle_R \subseteq \langle G' \rangle_R \subseteq \langle F \rangle_R.$$

Ora, se la scomposizione di M indotta da $\boldsymbol{S}$ conserva le dipendenze, si ha che $\langle G \rangle_R = \langle F \rangle_R$ e, quindi, anche che $\langle G' \rangle_R = \langle F \rangle_R$. Se, invece, la scomposizione di M indotta da $\boldsymbol{S}'$ non conserva le dipendenze, allora si ha che $\langle G' \rangle_R \neq \langle F \rangle_R$ e quindi, anche che $\langle G \rangle_R \neq \langle F \rangle_R$. □

Esempio 10.1 (seguito) Abbiamo già visto che la scomposizione di M indotta da $\boldsymbol{S} = \{\{\text{COMUNE, VIA}\}, \{\text{COMUNE, CAP}\}, \{\text{VIA, CAP}\}\}$ non conserva le dipendenze. Per il Teorema 10.6 non esiste nessuna scomposizione di M che conservi le dipendenze fatta eccezione per quella indotta dalla partizione banale di R. Si osservi infine che i modelli prodotti dalla scomposizione di M indotta da $\boldsymbol{S}$ sono tutti regolari.

■

10.5 Scomposizione conservativa in Terza Forma Normale

Dato un modello $M = [R, F]$ in prima forma normale, ci domandiamo se è possibile trovare un ricoprimento $\boldsymbol{S}$ di R tale che la scomposizione di M indotta da $\boldsymbol{S}$ sia conservativa e che tutte le proiezioni di M su $\boldsymbol{S}$ siano modelli regolari (e quindi in forma normale di Boyce-Codd). La risposta a questo quesito è negativa nel caso generale dal momento che esistono casi in cui un tale ricoprimento non esiste proprio. Ad esempio, per il modello $M = [R, F]$ dell'Esempio 10.1 (v. sopra), l'unico ricoprimento di R che induce una scomposizione conservativa di M è la partizione banale $\boldsymbol{S} = \{R\}$, cosicché il risultato della scomposizione di M è un unico modello (M stesso), che non è in forma normale di Boyce-Codd perché non è regolare. Inoltre, per quei casi in cui pure esiste, costruirlo è in generale un problema arduo. Ci domandiamo allora se è possibile trovare un ricoprimento $\boldsymbol{S}$ di R tale che la scomposizione di M indotta da $\boldsymbol{S}$ sia conservativa e che tutte le proiezioni di M su $\boldsymbol{S}$ siano modelli in terza forma normale. Questa volta, la risposta è positiva e il seguente algoritmo costruisce un ricoprimento siffatto, a patto che si faccia uso di un modello semplice equivalente ad M.

Scomposizione in Terza Forma Normale

Input: Un modello semplice $M = [R, F]$ in prima forma normale.
Output: Un ricoprimento $\boldsymbol{S}$ di R.

Procedura

(1) $\boldsymbol{S} := \varnothing$.

(2) Scegliere un ordinamento delle dipendenze funzionali in F; quindi, per ogni dipendenza funzionale $X \rightarrow A$ in F,

> se l'insieme $X \cup \{A\}$ non è contenuto in nessun insieme in $\boldsymbol{S}$, allora aggiungere l'insieme $X \cup \{A\}$ ad $\boldsymbol{S}$ ed eliminare da $\boldsymbol{S}$ i sottoinsiemi propri di $X \cup \{A\}$ eventualmente presenti.

(3) Trovare una chiave Z di M e, se Z non è contenuto in nessun insieme in $\boldsymbol{S}$, allora aggiungere Z ad $\boldsymbol{S}$.

***Teorema* 10.5** Dato un modello semplice $M = [R, F]$ che sia in Prima Forma Normale, l'algoritmo di Scomposizione in Terza Forma Normale genera un ricoprimento di R

che induce una scomposizione conservativa di M tale che le proiezioni di M sono modelli in Terza Forma Normale.

Dimostrazione. Sia $\{R_1, \ldots, R_k\}$ la famiglia di insiemi che si ottiene dopo aver eseguito le istruzioni (1) e (2) dell'algoritmo di Scomposizione in Terza Forma Normale, e sia R_0 l'insieme degli attributi in R che non sono presenti in nessuna delle dipendenze funzionali contenute in F. Senza perdere in generalità, d'ora in poi assumiamo che $R_0 \neq \emptyset$. Qualora R_0 fosse vuoto, la dimostrazione che segue andrà modificata con aggiustamenti più o meno ovvi. Consideriamo il ricoprimento $\boldsymbol{S}_0 = \{R_0, R_1, \ldots, R_k\}$ di R. Dimostriamo innanzitutto che:

(*a*) la scomposizione di M indotta da $\boldsymbol{S}_0$ conserva le dipendenze,

(*b*) le proiezioni $M_0, M_1, \ldots, M_k$ di M su $R_0, R_1, \ldots, R_k$ sono tutti modelli in terza forma normale.

(*a*) Siccome ogni dipendenza funzionale in F è applicabile ad uno degli insiemi $R_1, \ldots, R_k$, banalmente si ha che $F \subseteq F_1 \cup \ldots \cup F_k \subset F_0 \cup F_1 \cup \ldots \cup F_k = G$ la qual cosa implica la scomposizione di M indotta da $\boldsymbol{S}_0$ conserva le dipendenze in F.

(*b*) Cominciamo con il modello M_0. L'unica chiave di M_0 è R_0 e, quindi, M_0 è banalmente in terza forma normale perché non contiene attributi secondari.

Consideriamo ora il generico modello M_h per $h > 0$. Assumiamo che per un dato ordinamento delle dipendenze funzionali in F (vedi Istruzione 2), sia $X \rightarrow A$ la dipendenza funzionale in F che ha generato R_h cosicché $R_h = X \cup \{A\}$. Ora, siccome la dipendenza funzionale $X \rightarrow R_h$ appartiene ad F_h ed il determinante della dipendenza funzionale $X \rightarrow A$ non è riducibile (visto che M è un modello semplice), X è una chiave di M_h. Dunque, ogni attributo in X è un attributo primario per M_h, mentre l'attributo A può essere sia primario che secondario. Distinguiamo i due casi.

Caso 1: A è un attributo primario per M_h. Allora R_h non contiene attributi secondari per M_h e, banalmente, M_h è in terza forma normale.

Caso 2: A è un attributo secondario per M_h. Si osservi innanzitutto che, siccome X è una chiave di M_h, A è l'unico attributo secondario per M_h, cosicché M_h è in terza forma normale se e solo se A non dipende transitivamente da nessuna chiave di M_h. A tale scopo, dimostriamo prima che X è l'unica chiave di M_h. Se, infatti, esistesse un'altra chiave X' di M_h allora, siccome X' non può essere un sottoinsieme di X, si avrebbe che $X' - X \neq \emptyset$; d'altra parte, siccome $X' \subseteq R_h = X \cup \{A\}$, si avrebbe che $X' - X = \{A\}$. Dunque, A apparterebbe ad una chiave (X') di M_h, la qual cosa contrasta con l'assunzione che A sia un attributo secondario per M_h. Dal fatto che X è l'unica chiave di M_h segue che M_h è in terza forma normale se e solo se A non dipende transitivamente da X. Ora, se per assurdo A dipendesse transitivamente da X, allora,

siccome $R_h = X \cup \{A\}$, dovrebbe esistere un sottoinsieme proprio Y di X tale che la dipendenza funzionale $Y \rightarrow A$ sia implicata da F; ma questo è in contrasto con il fatto che il determinante della dipendenza funzionale $X \rightarrow A$ è irriducibile (visto che M è un modello semplice). Dunque, A non dipende transitivamente dall'unica chiave di M_h e, quindi, M_h è in terza forma normale.

Resta così provato che anche ogni modello M_h ($0 \leq h \leq k$) è in terza forma normale. Riassumendo, la scomposizione di M indotta da $\boldsymbol{S}_0$ conserva le dipendenze e ciascuna delle proiezioni di M su $\boldsymbol{S}_0$ è un modello in terza forma normale.
Sia Z una chiave di M calcolata dall'Istruzione 3. Si osservi innanzitutto che R_0 è sempre un sottoinsieme di Z e, siccome $R_0 \neq \emptyset$, Z non è incluso in nessun R_h, $1 \leq h \leq k$. Dunque, $\boldsymbol{S} = \{Z, R_1, \ldots, R_k\}$ è un ricoprimento di R e, siccome la scomposizione di M indotta da $\boldsymbol{S}_0$ conserva le dipendenze, per il Teorema 10.4 abbiamo che anche $\boldsymbol{S}$ induce una scomposizione di M che conserva le dipendenze. Inoltre, come già provato, ogni modello M_h ($1 \leq h \leq k$) è in terza forma normale. Proviamo ora che anche la proiezione di M su Z è in terza forma normale. Sia M' la proiezione di M su Z. Siccome Z è l'unica chiave di M', nessun attributo in M' è secondario e questo implica banalmente che M' è un modello in terza forma normale. Dunque, la scomposizione di M indotta da $\boldsymbol{S}$ conserva le dipendenze e ciascuna delle proiezioni di M su $\boldsymbol{S}$ è un modello in terza forma normale. A questo punto, resta solo da provare che la scomposizione di M indotta da $\boldsymbol{S}$ conserva i dati, cioè che il Test di Conservazione dei Dati dà esito positivo. Siano $R = \{A_1, \ldots, A_n\}$ e $Z = \{A_1, \ldots, A_m\}$, $m < n$. Supponiamo di applicare l'algoritmo di Bernstein per calcolare l'insieme $\langle Z \rangle_F$ che, essendo Z una chiave di M, coincide con R. Sia

$$A_{m+1}, \ldots, A_n$$

l'ordine in cui gli attributi in $R–Z$ vengono aggiunti a $\langle Z \rangle_F$ (= R) e siano

$$X_1 \rightarrow A_{m+1}, \ldots, X_{n-m} \rightarrow A_n$$

le dipendenze funzionali in F utilizzate a questo scopo. Si osservi che $X_1 \subseteq Z$ e

$$X_j \subseteq Z \cup \{A_{m+1}, \ldots, A_{m+j-1}\}$$

per ogni j, $2 \leq j \leq n–m$. Inoltre, per ogni j esiste un insieme $R_{h(j)}$ in $\boldsymbol{S}$ che contiene $X_j \cup \{A_{m+j}\}$ (vedi istruzione 2).
Consideriamo ora il tableau T associato ad $\boldsymbol{S}$ con il seguente ordinamento delle colonne e delle righe. Ordiniamo le colonne del tableau T associato ad $\boldsymbol{S}$ prendendo prima gli attributi in Z (cioè $A_1, \ldots, A_m$) e poi gli attributi in $R–Z$ (cioè $A_{m+1}, \ldots, A_n$).

Ordiniamo poi le righe di T prendendo per ultima la riga (T_{k+1}) che corrisponde a Z. In questo modo

$$T_{k+1,i} = i \qquad \text{se } i \leq m$$
$$T_{k+1,i} > n \qquad \text{se } m + 1 \leq i \leq n\ .$$

A questo punto, applichiamo ora il Test di Conservazione dei Dati con l'accortezza di scandire F esaminando prima le dipendenze funzionali $X_1 \rightarrow A_{m+1}, \ldots, X_{n-m} \rightarrow A_n$. Così, quando viene esaminata la dipendenza funzionale $X_j \rightarrow A_{m+j}$, $1 \leq j \leq n-m$, nel tableau troviamo che

> la riga $T_{h(j)}$ (corrispondente a $R_{h(j)}$) contiene valori minori o uguali ad n solo in corrispondenza delle colonne intestate agli attributi in $R_{h(j)}$ e, in particolare, agli attributi in $X_j \cup \{A_{m+j}\}$, e
>
> la riga T_{k+1} contiene valori minori o uguali ad n solo in corrispondenza delle prime $m+j-1$ colonne.

Siccome $X_j \subseteq Z \cup \{A_{m+1}, \ldots, A_{m+j-1}\}$, abbiamo che

$$T_{h(j)}[X_j] = T_{k+1}[X_j]$$

mentre

$$T_{h(j),m+j} = m+j \qquad \text{e} \qquad T_{k+1,m+j} > n.$$

Allora, $T_{k+1,m+j}$ verrà posto uguale ad $m+j$ ($\leq n$). Pertanto, dopo aver esaminato tutte le dipendenze funzionali $X_1 \rightarrow A_{m+1}, \ldots, X_{n-m} \rightarrow A_n$ in F, la riga T_{k+1} verrà a contenere solo valori minori o uguali ad n cosicché il Test di Conservazione dei Dati dà esito positivo e questo, per il Teorema 10.1, dimostra che la scomposizione di M indotta da $\boldsymbol{S}$ conserva i dati. □

Esempio 10.6 Consideriamo il modello M con schema $R = \{A, B, C, D, E, F, G\}$ e dipendenze funzionali

$$A \rightarrow B,\ B \rightarrow A,\ \{A, B\} \rightarrow C,\ \{C, D, E\} \rightarrow F,\ F \rightarrow D,\ F \rightarrow E.$$

Il modello M non è semplice perché il determinante della dipendenza funzionale $\{A, B\} \rightarrow C$ è riducibile. Un modello semplice equivalente ad M è dato dal modello M' con schema R e dipendenze funzionali

$$A \rightarrow B,\ B \rightarrow A,\ A \rightarrow C,\ \{C, D, E\} \rightarrow F,\ F \rightarrow D,\ F \rightarrow E.$$

Una chiave di M' è $\{A, F, G\}$. L'applicazione ad M' dell'algoritmo di Scomposizione in Terza Forma Normale genera il ricoprimento

$$\boldsymbol{S} = \{\{A, B\}, \{A, C\}, \{A, F, G\}, \{C, D, E, F\}\}$$

di R. Per il Teorema 10.7, la scomposizione di M' indotta da $\boldsymbol{S}$ è conservativa e le proiezioni di M' su $\boldsymbol{S}$ sono tutti modelli in terza forma normale. Inoltre, si osservi che, la proiezione di M' (ovvero di M) su $\{C, D, E, F\}$ contiene le dipendenze funzionali

$$\{C, D, E\} \rightarrow F, F \rightarrow D, F \rightarrow E$$

cosicché la proiezione di M' (ovvero di M) su $\{C, D, E, F\}$ non è un modello regolare (visto che $\{F\}$ è una criticità ovvero l'attributo primario D (o E) dipende transitivamente (parzialmente, nella fattispecie) dalla chiave $\{C, F\}$ per il tramite di $\{F\}$). Mentre tutte le altre proiezioni di M' su $\boldsymbol{S}$ sono modelli regolari e, quindi, sono modelli in forma normale di Boyce-Codd. Infine, siccome M ed M' sono equivalenti, anche la scomposizione di M indotta da $\boldsymbol{S}$ è conservativa e le proiezioni di M su $\boldsymbol{S}$ sono tutti modelli in terza forma normale e, con la sola eccezione della proiezione di M su $\{C, D, E, F\}$, sono modelli in forma normale di Boyce-Codd.

Esempio 10.7 Consideriamo il modello $M = [R, F]$ dove R contiene i seguenti attributi:

M	materia	D	docente	O	ora
A	aula	S	studente	V	voto

ed F contiene le seguente dipendenze funzionali:

$$M \rightarrow D \qquad \{O, A\} \rightarrow M \qquad \{O, D\} \rightarrow A$$
$$\{M, S\} \rightarrow V \qquad \{O, S\} \rightarrow A$$

Il modello M è semplice e $Z = \{S, O\}$ è una chiave di M. L'applicazione dell'algoritmo di scomposizione in terza forma normale genera il ricoprimento $\boldsymbol{S} = \{\{M, D\}, \{M, O, A\}, \{D, O, A\}, \{M, S, V\}, \{S, O, A\}\}$. Si osservi che $\boldsymbol{S}$ non contiene Z perché Z è contenuto nell'insieme $\{S, O, A\}$ presente in $\boldsymbol{S}$.

10.6 Chiavi esterne

Consideriamo infine un modello M che sia in terza forma normale. Se il test di regolarità dà esito positivo, allora M è in forma normale di Boyce-Codd ed allora si è certi di non incorrere in anomalìe di aggiornamento. Se il test di regolarità dà esito negativo, allora M non è in forma normale di Boyce-Codd ed allora bisogna mettere in conto di incorrere in anomalìe di aggiornamento. In tal caso, l'aggiunta o la modifica di un'ennupla in una relazione conforme ad M potrebbe richiedere una revisione dell'intera relazione per rimuovere inconsistenze dovute alle criticità di M. Il seguente esempio mostra una tecnica per gestire le anomalìe.

Esempio 10.8 Consideriamo una base di dati $\mathcal{D}$ che contiene una tabella relazionale di nome `Strade` a cui sia associato il modello $M = [R, F]$ dove $R = \{$`via, città, cap`$\}$ ed $F = \{\{$`via, città`$\} \rightarrow$ `cap`, `cap` $\rightarrow$ `città`$\}$. Il modello M ha due chiavi,

$$\{\texttt{via, città}\} \quad \text{e} \quad \{\texttt{via, cap}\},$$

cosicché tutti e tre gli attributi sono primari; inoltre, per via della dipendenza funzionale cap → città, l'insieme elementare {cap} è una criticità di M. Dunque, M è in terza forma normale ma non in forma normale di Boyce-Codd. Una soluzione è data da una base di dati $\mathcal{D}'$ che si ottiene da $\mathcal{D}$ sostituendo alla tabella relazionale Strade due tabelle relazionali denominate Strade_1 e Strade_2

a cui vengono associati rispettivamente i modelli

$$M_1 = [R_1 = \{\text{via, città, cap}\}, F_1 = \{\{\text{via, città}\} \rightarrow \text{cap}\}]$$

$$M_2 = [R_2 = \{\text{città, cap}\}, F_2 = \{\text{cap} \rightarrow \text{città}\}]$$

e su cui imponiamo il vincolo d'inclusione

$$\text{Strade}_1\ (\text{città, cap}) \triangleright \text{Strade}_2\ (\text{città, cap})$$

che richiede che, in ogni possibile stato di $\mathcal{D}'$, se r_1 e r_2 sono rispettivamente le relazioni di nome Strade_1 e Strade_2, allora la proiezione di r_1 su R_2 sia un sottoinsieme di r_2.

Ad esempio, siano

via	città	cap
Verdi	A	1000
Verdi	B	2000
Leonardo	A	1000
Fermi	B	2000
Dante	A	1000

r_1

città	cap
A	1000
B	2000

r_2

Consideriamo l'operazione

modificare Strade_1 (Dante, A, 1000; Dante, AA, 1000).

Sebbene il risultato dell'operazione soddisfi l'unica dipendenza funzionale {via, città} → cap presente in F_1, tuttavia viene violato il vincolo d'inclusione perché (AA, 1000) non è presente in r_2. Dunque, l'operazione viene rifiutata.

Consideriamo invece l'operazione

modificare Strade_2 (A, 1000; AA, 1000).

Il risultato dell'operazione consiste nell'aggiornare la relazione r_2 di nome $\texttt{Strade}_2$ (visto che soddisfa l'unica dipendenza funzionale cap → città presente in F_2) e nel propagare l'aggiornamento ("a cascata") su tutta la relazione r_1 con il risultato seguente:

via	città	cap
Verdi	AA	1000
Verdi	B	2000
Leonardo	AA	1000
Fermi	B	2000
Dante	AA	1000

r'_1

città	cap
AA	1000
B	2000

r'_2

CAPITOLO 11

ORGANIZZAZIONE E GESTIONE DEI DATI

Nella maggior parte dei casi, le informazioni contenute in una base di dati sono registrate su supporti magnetici (dischi, cilindri, tamburi, …) della *memoria secondaria* (o "memoria di massa"). In questo capitolo riassumiamo per grandi linee le *tecniche di organizzazione fisica* comunemente usate per memorizzare il contenuto informativo (relazioni) di una base di dati, per rendere spedite le operazioni di aggiornamento e per ridurre quanto più possibile i tempi di attesa per le risposte alle interrogazioni dei suoi utenti. Dunque, nella progettazione fisica di una base di dati occorre tener conto di diversi fattori: la quantità di memoria occupata, il tipo e la frequenza degli aggiornamenti, il tipo di domande degli utenti che ricorrono più frequentemente, etc.
Cominciamo dalla memoria secondaria, di cui diamo una rappresentazione schematica, che è tutto quello che serve per illustrare le varie tecniche di organizzazione.

11.1 Memoria secondaria

Per *memoria secondaria* (o "memoria di massa") si intende il complesso di dispositivi magnetici (dischi, cilindri, tamburi, …) dove sono fisicamente registrate le relazioni di una base di dati. Possiamo rappresentare la memoria come una serie di *celle*, capaci ciascuna di contenere un'informazione di 1 *byte* (1 B = 8 *bit*); le celle sono numerate progressivamente da 0 ad M–1, dove M varia tipicamente tra 2^{24} byte e 2^{30} byte, e per ogni i, $0 \le i \le M$–1, diremo che la cella i-esima ha *indirizzo* i. Diremo anche che la *dimensione* della memoria secondaria è pari ad M byte.
La memoria secondaria è fisicamente suddivisa in *blocchi*. Un blocco è un insieme di celle fisicamente contigue e l'indirizzo della sua prima cella è anche il suo indirizzo. La dimensione D di un blocco varia tra 2^9 byte e 2^{12} byte. Pertanto, la memoria secondaria sarà composta da $b = M/D$ blocchi; così, se $M = 2^{24}$ byte e $D = 2^9$ byte, allora $b = 2^{15}$, mentre se $M = 2^{30}$ byte e $D = 2^{12}$ byte, allora $b = 2^{18}$. Si osservi che l'indirizzo di un blocco è sempre un multiplo di D (0, D, 2D, …) e un puntatore al blocco che ha indirizzo I è la codifica binaria di I. Tipicamente, al puntatore ad un blocco si riservano 4 byte che è quanto basta per coprire tutti i numeri da 0 a 2^{32}–1 e, quindi, per fornire l'indirizzo di uno dei b blocchi di cui si compone la memoria secondaria.

Le due tabelle che seguono riportano i multipli del byte come potenze di 10 e come potenze di 2.

Potenze del byte in base 10

1 *kilobyte*	1 kB	=	10^3 B
1 *megabyte*	1 MB	=	10^6 B
1 *gigabyte*	1 GB	=	10^9 B
1 *terabyte*	1 TB	=	10^{12} B
1 *petabyte*	1 PB	=	10^{15} B
1 *exabyte*	1 EB	=	10^{18} B
1 *zettabyte*	1 ZB	=	10^{21} B
1 *yottabyte*	1 YB	=	10^{24} B

Potenze del byte in base 2

1 *kibibyte*	1 KiB	=	2^{10} B	= 1,024 kB
1 *mebibyte*	1 MiB	=	2^{20} B	= 1,049 MB
1 *gibibyte*	1 GiB	=	2^{30} B	= 1,074 GB
1 *tebibyte*	1 TiB	=	2^{40} B	= 1,100 TB
1 *pebibyte*	1 PiB	=	2^{50} B	= 1,126 PB
1 *exbibyte*	1 EiB	=	2^{60} B	= 1,153 EB
1 *zebibyte*	1 ZiB	=	2^{70} B	= 1,181 ZB
1 *yobibyte*	1 YiB	=	2^{80} B	= 1,209 YB

Dunque, la dimensione M della memoria secondaria varia tra 2^{24} byte = 4 MiB (≈ 4 MB) e 2^{30} byte = 1 GiB (≈ 1 GB), mentre la dimensione D di un blocco varia tra 2^9 byte = 0,5 KiB (≈ 0,5 kB) e 2^{12} byte = 4 KiB (≈ 4 kB).

È compito del *sistema di controllo* del dispositivo di memoria tradurre l'indirizzo logico in un indirizzo fisico. Con 2^{18} blocchi abbiamo 2^{18} indirizzi; vale a dire che, per specificare l'indirizzo (logico) di un blocco servono almeno 18 bit; usualmente, l'indirizzo di un blocco occupa 4 celle, cioè ha una lunghezza di 4 byte e, per motivi di efficienza, l'indirizzo di ogni blocco è sempre specificato da un multiplo di 4.

L'*accesso* ai blocchi della memoria secondaria è regolato dal *sistema di controllo* della memoria secondaria: con un comando di *lettura* del blocco di indirizzo I, il contenuto del blocco viene integralmente copiato in un'area riservata (*buffer*) di D

celle contigue della memoria centrale (RAM), mentre con un comando di *scrittura* nel blocco di indirizzo I, il contenuto del buffer viene copiato nel blocco di indirizzo I. In entrambi i casi, l'operazione prende un tempo che è dell'ordine di decine o più di millisecondi dovuto al movimento di parti elettromeccaniche del dispositivo di memoria. D'altra parte, una tipica operazione di elaborazione nella memoria centrale (RAM) è di natura elettronica ed è dell'ordine di nanosecondi (dovuta alla velocità c di propagazione di segnali elettromagnetici).

Sottomultipli del *secondo* (s)

1 *millisecondo*	1 *ms*	$= 10^{-3}$ s
1 *microsecondo*	1 μs	$= 10^{-6}$ s
1 *nanosecondo*	1 *ns*	$= 10^{-9}$ s
1 *picosecondo*	1 *ps*	$= 10^{-12}$ s

Pertanto, l'*accesso* ad un blocco (scrittura o lettura) richiede molto ma molto più tempo di un'operazione di elaborazione nell'unità centrale. Per dare un'idea, è come se, per fare una breve colazione al banco del bar (che mettiamo richieda qualche minuto), si dovesse fare un viaggio della durata di qualche anno (!). Per questo enorme divario tra i tempi di accesso alla memoria secondaria e i tempi di elaborazione nell'unità centrale, da una parte si tenta di ridurre al minimo gli accessi ai blocchi e dall'altra si utilizzano due o più buffer cosicché, mentre si copia il contenuto di un blocco in un buffer, l'unità centrale può elaborare le informazioni contenute in un altro blocco il cui contenuto sia già stato copiato in un altro buffer.

11.2 *File*

Le relazioni contenute in (uno stato di) una base di dati vengono memorizzate in strutture di dati (*file*) residenti nella memoria secondaria.

Un *file* è un insieme di *record* che hanno tutti lo stesso *formato* e questo è definito da una sequenza di *campi*, ciascuno con un proprio *tipo di dato* (stringa, reale, intero, booleano, puntatore, …). Supponiamo che il formato dei record del file sia definito dalla sequenza $(C_1, \ldots, C_k)$ di campi. Così un record sarà formato da una sequenza $(c_1, \ldots, c_k)$, dove è un valore del campo compatibile con il tipo di dato di C_h, $1 \leq h \leq k$. Per esempio, se C_h è di tipo intero, c_h sarà un numero intero (nonnegativo) ed occuperà $l_h = 4$ byte; se C_h è di tipo puntatore, c_h di nuovo occuperà $l_h = 4$ byte; se invece C_h è di tipo stringa, c_h sarà una stringa ed occuperà un numero di byte l_h minore o uguale al massimo consentito dal tipo di C_h. Chiamiamo *lunghezza* del record $(c_1, \ldots, c_k)$, la somma $L = l_1 + \ldots + l_k$. Ne viene che due distinti record $(c_1, \ldots,$

c_k) e (c'_1, …, c'_k), possono avere lunghezze diverse, nel qual caso i record si dicono a *lunghezza variabile*. D'ora in poi, per semplicità, assumiamo che i record del file abbiano tutti la stessa lunghezza.

Come per una relazione contenuta in uno stato della base di dati, possiamo aggiornare un file

aggiungendo un record,

cancellando o modificando uno o più record

oppure cercare dei particolari record di un file.

La selezione dei record da cancellare, da modificare o da cercare si basa su una qualche condizione su un dato insieme X di campi; per esempio, la condizione che X assuma un dato valore oppure un valore appartenente ad un dato insieme. L'insieme X è chiamata la *chiave della ricerca*.

Nel valutare il costo computazionale di ognuna delle operazioni terremo conto soltanto delle operazioni di trasferimento del contenuto di uno o più blocchi nel buffer della memoria centrale e viceversa, cioè del numero di accessi alla memoria secondaria.

11.4 *Configurazione dei blocchi*

Il file è memorizzato in un certo numero di blocchi della memoria secondaria ed è compito del *file system* riservarne un numero sufficiente e individuarli tra quelli non impegnati da altri file. Supponiamo che il file sia memorizzati in n blocchi, B_1, …, B_n. Un'apposita struttura dati (ad esempio una lista) tiene traccia degli indirizzi dei blocchi B_1, …, B_n (block directory del file), un'altra contiene la descrizione del file (data dictionary); entrambe, se non occupano troppo spazio, sono tenute permanentemente nella memoria centrale per tutto il tempo che il file viene usato. Inoltre, usualmente i blocchi B_1, …, B_n sono *concatenati* nel senso che in ogni blocco B_i c'è una zona di celle (LINK) che contiene l'indirizzo del blocco B_{i+1} se $i < n$, altrimenti un valore convenzionale (*valore nullo*) che sta a segnalare che B_n è l'ultimo blocco.

Supponiamo di voler memorizzare un certo numero di record in uno dei blocchi che il *file system* abbia assegnato al file. In linea teorica, il blocco potrà ospitare non più di $P = \lfloor D/l \rfloor$ record, dove D è la dimensione del blocco ed L è la lunghezza di ciascun

record. Chiameremo P la *capacità* del blocco in termini di record del file. Così, se $D = 2^{12}$ byte e $L = 20$ byte, allora $P = 204$.
A tale scopo, i blocchi che il *file system* ha riservato al file vengono tutti suddivisi in zone (sub-blocks), che per comodità chiamiamo *pagine*, tutte della stessa lunghezza l dei record del file. La *posizione di una pagina* all'interno di un blocco è data dal numero delle pagine di quel blocco che la precedono. Pertanto, una pagina che occupa la posizione p, $0 \leq p \leq P-1$, in un blocco che ha indirizzo I si estende dalla cella di indirizzo $I+pL$ alla cella di indirizzo $I+(p+1)L-1$. Inoltre, ogni pagina è suddivisa in segmenti, che per comodità chiamiamo *linee*, della stessa dimensione dei campi dei record del file. La *posizione di una linea* di una certa pagina all'interno di un blocco è data da $pL + l$ dove p è il numero delle pagine di quel blocco che precedono quella pagina e l è la somma delle lunghezze delle linee che precedono quella linea è data dal numero delle linee di quella pagina che la precedono. Infine, una zona (LINK) del blocco è riservata all'indirizzo dell'eventuale blocco successivo; in sua assenza, LINK conterrà un valore nullo.

11.4 *Formattazione delle pagine*

Ogni pagina è suddivisa in segmenti, uno per ogni campo del record. Per semplicità, anche questi segmenti li chiamiamo *campi*.
Consideriamo ora un'operazione di cancellazione. Supponiamo di voler cancellare un record r memorizzato in una certa pagina. La maniera più ovvia sarebbe quella di avere un ulteriore campo info (1 byte) che riporta lo *stato* di quella pagina, cioè se va considerata *libera* o *occupata*. Così per cancellare r basterebbe modificare lo stato della pagina da occupata a libera, così che essa possa essere riutilizzata per registrarvi un nuovo record quando se ne presenti l'occasione. Ma cosa avviene se un altro record (dello stesso file o di un altro file) contiene un puntatore che rimanda proprio al record r. In tal caso, se si andasse a registrare un nuovo record r' del file nella stessa pagina che ospitava il record r, tutti i puntatori ad r verrebbero allora erroneamente imputati ad r'. Per ovviare a questo inconveniente, si considera il record r "appuntato" (pinned) alla pagina, cosicché la pagina resta impegnata dal record r anche dopo che questo sia stato cancellato ed allora tutti i puntatori ad r restano per così dire "in sospeso" (dangling). Vanno dunque distinte le pagine occupate da record appuntati cancellati da quelle occupate da record del file "in uso". A questo scopo, il campo info deve contenere anche un *bit di cancellazione* con valore "1" se il record va considerato "in uso" (cioè facente parte del file) e "0" se va considerato "non in uso". Allora per cancellare il record r, basterà modificare il bit di cancellazione nel campo info da 1 a 0. Poi, si modifica lo stato della pagina da occupata a libera solo se non esistono puntatori al record r.

Infine, il campo info contiene anche un bit "1" oppure "0" a seconda che il blocco contenga o meno nelle pagine successive altri record del file.

Supponiamo di voler memorizzare una relazione che contiene i primi *N* numeri di Fibonacci

(1 1 2 3 5 8 13 21 34 55 …)

Assumiamo che la relazione abbia schema {progressivo, numero e cifre}, e che progressivo sia la chiave primaria. Ovviamente, il file, che per comodità chiamiamo Fibonacci, sarà fatto di record il cui formato contiene i tre "campi/attributi" progressivo, numero e cifre:

progressivo	4 B	il numero progressivo del numero di Fibonacci
numero	4 B	un numero di Fibonacci
cifre	4 B	numero di cifre decimali

con l'aggiunta del campo info (1 byte). Così, fino a questo punto, la lunghezza di un record è di 12 byte. Comunque, per motivi di efficienza nella gestione del file, si fa in modo che

- le pagine siano "allineate" nel senso che la posizione della prima linea di ogni pagina sia un multiplo di 4,
- i campi che contengono dati numerici o puntatori siano "allineati" nel senso che la posizione di ogni campo sia un multiplo di 4.

A questo scopo, non solo va scelta una conveniente sequenza dei campi ma va anche previsto di aggiungere *campi non significativi* (padding). Un possibile formato dei record del file Fibonacci (e, quindi, di una pagina di un blocco riservato al file) è dunque il seguente

Formato di un record del file Fibonacci

campo	*lunghezza*
info	1 B
spazio (campo non significativo)	3 B
numero	4 B
progressivo	4 B
cifre	4 B

cosicché la lunghezza del record è $L = 16$ byte.

11.5 *Configurazione dei blocchi*

Supponiamo che un blocco di dimensione *D* sia riservato ad un file i cui record hanno lunghezza *L*. Come si è detto, ogni blocco sarà suddiviso in $P = \lfloor D/L \rfloor$ pagine, a cui seguirà la zona LINK (di quattro byte).

Nel caso del file Fibonacci, se $D = 2^9$ (= 512) byte, allora ogni blocco sarà suddiviso in P = 32 pagine e la sua configurazione sarà la seguente

Configurazione di un blocco

PRIMA PAGINA

campo	*lunghezza*	*posizione*
info	1 B	0
spazio	3 B	1
numero	4 B	4
progressivo	4 B	8
cifre	4 B	12

SECONDA PAGINA

campo	*lunghezza*	*posizione*
info	1 B	16
spazio	3 B	17
numero	4 B	20
progressivo	4 B	24
cifre	4 B	28

…

ULTIMA PAGINA

campo	*lunghezza*	*posizione*
info	1 B	496
spazio	3 B	497
numero	4 B	500
progressivo	4 B	504
cifre	4 B	508

LINK	4 B	512

Si osservi che la posizione della prima linea di ogni pagina, così come le posizioni dei campi numero, progressivo e cifre sono sempre multipli di 4, e lo stesso dicasi per la posizione di LINK.
Infine, il numero n di blocchi necessari a memorizzare gli N record del file Fibonacci non è inferiore ad $n = \lceil N/P \rceil$ dove P è il numero di pagine di cui si compone un blocco. Così per N = 10000 e P = 32, abbiamo n = 313.
Questa configurazione non è molto efficiente se si vuole individuare una pagina libera dove memorizzare un nuovo record perché occorre scorrere una alla volta le pagine fino a trovarne una in cui il campo info contenga lo stato della pagina uguale ad "1". Si può fare di meglio riservando all'inizio del blocco una zona di uno o più byte che contenga un array dove siano riportati gli stati delle 32 pagine del blocco.

Per riferirsi ad un record dovremo specificare l'indirizzo I del blocco che contiene la pagina dove è memorizzato nonché il valore x che in quel record ha la chiave primaria del file. In alternativa, possiamo riferirci al record specificando $I + p$ dove p è la posizione della prima linea della pagina che contiene il record.

11.6 Organizzazione non-ordinata (heap)

Con questa tecnica i record vengono ammassati l'uno dopo l'altro. Supponiamo che $B_1, \ldots, B_n$ sia la catena dei blocchi impegnati dal file. Il blocco di testa B_0 del file conterrà sia l'indirizzo del primo blocco (B_1) che quello dell'ultimo (B_n). In "condizioni normali" per ogni $i < n$ il blocco B_i è pieno nel senso che non contiene pagine libere.

Aggiunta. Il record da aggiungere viene memorizzato nella prima pagina libera dell'ultimo blocco B_n, ammesso che ve ne sia una. Se questo è il caso, sono necessari solo 2 accessi alla memoria secondaria: uno per trasferire il contenuto di B_n in memoria centrale, l'altro per copiare il contenuto aggiornato di B_n nella memoria secondaria. Nel caso però che tutte le pagine di B_n siano occupate, viene richiesto al *file system* un nuovo blocco B e il record viene memorizzato nella prima pagina di B. Va da sé che, se tutte le pagine di B_n erano occupate, allora nel blocco di testa B_0 l'indirizzo del blocco B prenderà il posto dell'indirizzo del blocco B_n e la zona LINK di B_n conterrà anche l'indirizzo del blocco B.

Ricerca. Consideriamo una selezione di record con chiave di ricerca X. Per selezionare i record, viene effettuata una *ricerca lineare* dei blocchi: si esamina il primo blocco B_1 del file (il cui indirizzo è contenuto in B_0) e poi, via via, i blocchi successivi $B_2, \ldots, B_n$ tenendo a mente che l'indirizzo del blocco B_i è contenuto nella zona LINK del blocco B_{i-1} per $i > 1$. Noto l'indirizzo di un blocco, si procede alla maniera seguente. Inizialmente, l'intero contenuto del blocco viene copiato nel buffer della memoria centrale e una variabile intera i viene inizialmente posta uguale ad 1. Il contenuto della copia della pagina i.esima del blocco, cioè il contenuto del buffer dalla posizione $(i-1)l$ alla posizione $il-1$, viene copiato in una variabile v di tipo record (in un'area di lavoro della memoria centrale). Quindi, prima si controlla se la pagina è libera e, se così, se il record sia in uso (esaminando il bit di cancellazione del campo info); solo in tal caso, si procede a verificare se il contenuto dei campi in X della variabile v soddisfa o meno il criterio di selezione. Se sì, solo allora il record viene "selezionato". Quindi, si esamina il contenuto del campo info della variabile v

per sapere se il blocco contenga o meno altri record del file nelle pagine successive. Se non ve ne sono, allora la ricerca in quel blocco termina; altrimenti, si incrementa il valore di *i* di un'unità e si passa ad esaminare la pagina successiva. Per quanto detto, siccome i record che soddisfano il criterio di selezione possono essere in uno qualsiasi degli *n* blocchi, la ricerca richiede

n accessi

alla memoria secondaria.

A titolo d'esempio, consideriamo il file `Fibonacci` con $N = 10000$ record e supponiamo che i record siano stati memorizzati senza nessun ordine in 313 blocchi. Allora, la ricerca dei record che contengono i numeri di Fibonacci dal *p*.esimo al *q*.esimo, cioè dei record che soddisfano la condizione

$$p \leq \texttt{progressivo} \leq q$$

richiede sempre 313 accessi. Analogamente, la ricerca dei record che contengono i numeri di Fibonacci compresi in un dato intervallo [*a*, *b*], cioè che soddisfano la condizione

$$a \leq \texttt{numero} \leq b$$

richiede ancora 313 accessi.

Un caso a sé si ha quando la chiave di ricerca *X* è una chiave identificativa del file e il criterio di selezione è della forma $X = x$ perché, allora, la ricerca si può considerare conclusa non appena si sia trovato un record che soddisfi il criterio di selezione. Nel *caso peggiore*, il record si trova nell'ultimo blocco B_n oppure il file non contiene nessun record che soddisfa il criterio di selezione; in tal caso, il numero degli accessi alla memoria secondaria è pari ad *n*. Per valutare il numero di accessi nel *caso medio*, si ragiona come segue. Se il record si trovasse nell'*i*.esimo blocco B_i ($1 \leq i \leq n$), allora si dovrebbero esaminare i blocchi B_1, …, B_i e, quindi, sarebbero necessari *i* accessi. Sia π_i la probabilità che il record sia nel blocco B_i; allora, il numero medio *a* di accessi è dato da

$$a = \sum_{i=1,\ldots,n} i\, \pi_i \,.$$

Ora, se *N* è il numero dei record (in uso) del file ed N_i è il numero di record (in uso) che sono memorizzati nel blocco B_i, allora $\pi_i = N_i/N$ cosicché il numero medio di accessi alla memoria secondaria è

$$\sum_{i=1,\ldots,n} i\, (N_i/N) \,.$$

In condizioni normali, i blocchi $B_1, \ldots, B_{n-1}$ sono tutti pieni (non hanno pagine libere) cosicché $N_1 = \ldots = N_{n-1}$. Supponiamo ora, per puro esercizio, che anche il blocco B_n sia pieno e, quindi, che i record del file siano distribuiti in modo uniforme tra gli n blocchi; dunque, $N_1 = \ldots = N_n = N/n$. Allora, si ha che $\pi_i = N_i/N = 1/n$ e, quindi, il numero medio di accessi alla memoria secondaria è

$$a = (1/n) \sum_{i=1,\ldots,n} i = (n+1) / 2 \approx n/2.$$

Consideriamo, di nuovo, il file `Fibonacci` e supponiamo di voler cercare il p.esimo numero di Fibonacci, cioè quello che soddisfa la condizione

$$\texttt{progressivo} = p$$

Siccome `progressivo` è la chiave primaria del file, il numero medio di accessi è pari a circa $n/2 = 156$.

Cancellazione. La cancellazione dei record che soddisfano una condizione selettiva, viene effettuata con una ricerca lineare. Una volta individuati in un blocco i record da cancellare, basterà modificarne il bit di cancellazione e copiare in memoria secondaria il contenuto aggiornato del blocco. Pertanto, nel caso peggiore l'operazione richiede $2n$ accessi alla memoria secondaria.
Si osservi che se, dopo la cancellazione, un blocco si trova a contenere solo pagine libere, allora il blocco viene restituito al *file system* con la conseguente modifica della catena dei blocchi (modifica del campo LINK nel blocco precedente). Inoltre, periodicamente, il file viene compattato: in ogni blocco, fatta eccezione dell'ultimo, che contenga pagine libere (ma non tutte), si copiano altrettanti record presenti nell'ultimo blocco. Va da sé che, se l'ultimo blocco viene svuotato, allora il blocco viene restituito al *file system*. Dunque, come si diceva all'inizio, in *condizioni normali* tutti i blocchi della catena, ad eccezione dell'ultimo, sono pieni.

Qualora la chiave di ricerca X sia una chiave identificativa del file e il criterio di selezione sia della forma $X = x$ allora, come si è visto, la ricerca del record da cancellare richiede nel caso peggiore un numero degli accessi alla memoria secondaria pari ad n, e nel caso medio ne serviranno $\approx (n/2)$. Dunque, la cancellazione richiede nel caso peggiore un numero degli accessi alla memoria secondaria pari ad $n+1$, e nel caso medio $\approx (n/2) +1$.

Modifica. Come per la cancellazione, il costo dell'operazione in termini di numero medio di accessi alla memoria secondaria è pari a $2n$. Se poi la chiave di ricerca X è una chiave identificativa del file e il criterio di selezione è della forma $X = x$, allora la modifica richiede nel caso peggiore un numero degli accessi alla memoria secondaria pari ad n+1, e nel caso medio $\approx (n/2) + 1$.

11.4 Organizzazione modulare (hash)

Con questo tipo di organizzazione i record del file sono ripartiti in parti o *moduli* (bucket) in base al valore di un insieme di campi X, che chiamiamo *base della ripartizione*. A tale scopo provvede una *funzione di ripartizione* H (hashing function) che associa ad ogni possibile valore x di X un numero $h = H(x)$ compreso tra 0 ed m–1, dove m è un intero prefissato. Così, il modulo h.esimo conterrà tutti e solo i record del file (ammesso che ve ne siano) in cui X assume il valore

$$x = H^{-1}(h).$$

Ogni modulo viene fisicamente memorizzato come fosse un file non-ordinato e un *indice dei moduli* (bucket directory) riporta per ogni h ($0 \le h \le m$–1), gli indirizzi (eventualmente lo stesso valore nullo dei campi LINK se il modulo h.esimo è vuoto) del primo e dell'ultimo della catena dei blocchi impegnati dal modulo h.esimo. Ora se m non è eccessivamente grande, l'intero indice dei moduli (che ha dimensioni proporzionali ad m) può risiedere permanentemente in memoria centrale durante l'utilizzo del file.

Se indichiamo con N_h (≥ 0) il numero dei record nel modulo h.esimo, $0 \le h \le m$–1, allora serviranno $n_h = \lceil N_h/P \rceil$ blocchi, dove P è il numero delle pagine di un blocco, per un totale di $n = n_0 + \ldots + n_{m-1}$ blocchi. Particolare attenzione si pone alla scelta della funzione di ripartizione f, che nel caso ottimale distribuisce gli N record del file negli m moduli in maniera quanto più possibile uniforme. Nel caso ottimale $N_0 = \ldots = N_{m-1} = N/m$ cosicché per ogni modulo serviranno $\lceil N/mP \rceil$ blocchi.

Per ottenere una equi-distribuzione, si usa spesso come funzione di ripartizione il resto della divisione (della codifica binaria) del valore della base della ripartizione X per un numero primo.

A titolo di esempio, consideriamo il file `Fibonacci` con $N = 10000$ record e supponiamo di prendere come base della ripartizione il campo `numero`. Prendiamo poi tre possibili funzioni di ripartizione $H(x)$:

$H_2(x)$ = resto della divisione di x per 2

$H_3(x)$ = resto della divisione di x per 3

$H_5(x)$ = resto della divisione di x per 5.

Con $H_2(x)$ avremo due moduli: il modulo 0 per i numeri di Fibonacci pari, e il modulo 1 per i numeri di Fibonacci dispari. È un utile esercizio calcolare la ripartizione dei record tra i due moduli. Se prendiamo la successione dei resti modulo 2 dei numeri di Fibonacci otteniamo cicli di lunghezza 3, tutti del tipo

$$(1, 1, 0).$$

Dunque, il modulo 0 conterrà $N_0 = N/3$ record del file e impegnerà $\lceil N_0/P \rceil = 105$ blocchi, mentre il modulo 1 ne conterrà $N_1 = 2N/3$ record del file e impegnerà $\lceil N_1/P \rceil = 210$ blocchi.

Con $H_3(x)$ avremo tre moduli. Ora nella successione dei resti modulo 3 otteniamo cicli di lunghezza 8, tutti del tipo

$$(1, 1, 2, 0, 2, 2, 1, 0).$$

Dunque il modulo 0 conterrà $N_0 = N/4$ record del file e impegnerà $\lceil N_0/P \rceil = 79$ blocchi, mentre i moduli 1 e 2 ne conterranno ciascuno $3N/8$ e impegneranno ciascuno $\lceil N/4P \rceil = 118$ blocchi.

Con $H_5(x)$ avremo cinque moduli. Ora nella successione dei resti modulo 5 otteniamo cicli di lunghezza 20 tutti del tipo

$$(1, 1, 2, 3, 0, 3, 3, 1, 4, 0, 4, 4, 3, 2, 0, 2, 2, 4, 1, 0).$$

Dunque, ciascuno dei cinque moduli conterrà $N/5$ record del file (equipartizione) e impegnerà $\lceil N/5P \rceil = 63$ blocchi.

Per finire, è chiaro che se prendiamo come base della ripartizione il campo `progressivo` (che è la chiave del file `Fibonacci`), allora avremo sempre una equipartizione cosicché ogni modulo impegnerà $\lceil N/mP \rceil$ blocchi.

Aggiunta. Per aggiungere un record che abbia il valore x della base della ripartizione X, si calcola prima $h = H(x)$. A questo punto, l'indice dei moduli fornisce l'indirizzo dell'ultimo della catena dei blocchi impegnati dal modulo h.esimo. Finalmente, il record viene memorizzato nella prima pagina libera di quel blocco (ammesso che ve ne sia una). Dunque, solitamente sono necessari 2 accessi alla memoria secondaria come nell'organizzazione non-ordinata.

Ricerca. Vanno distinti due casi a seconda che la chiave di ricerca coincida o meno con la base della ripartizione. Se sono diverse, allora la ricerca è lineare su tutti gli $n = N/P$ blocchi impegnati del file. Se, invece, la chiave di ricerca coincide con la base della ripartizione e la condizione selettiva coinvolge k degli m moduli, allora basta una ricerca lineare sulla catena dei blocchi dei k moduli interessati dalla ricerca, per un totale di $\lceil kN/mP \rceil$ blocchi.

Supponiamo che il file `Fibonacci` sia organizzato in maniera modulare con base di ripartizione `progressivo` e funzione di ripartizione $H_5(x)$. Se si vogliono i record soddisfano la condizione

$$1000 \leq \texttt{numero} \leq 2000$$

oppure la condizione

$$1000 \leq \texttt{progressivo} \leq 2000$$

vanno esaminati i blocchi di tutti e cinque i moduli per un totale di $\lceil N/P \rceil = 313$ blocchi. Se invece si vuole il millesimo numero di Fibonacci, allora basterà esaminare gli $\lceil N/5P \rceil = 63$ blocchi del modulo 0.

Cancellazione. Rispetto alla ricerca, la cancellazione dei record che soddisfano una condizione selettiva richiede un accesso in più per ogni blocco contenente un record selezionato.

Modifica. La modifica ha lo stesso costo della cancellazione quando nessun campo da modificare appartiene alla base della ripartizione; altrimenti, i record modificati vanno ricollocati dopo aver calcolato i nuovi moduli a cui appartengono.

11.5 Organizzazione ordinata (ISAM)

Si è visto che l'organizzazione modulare su base X è il meglio che si può ottenere pensando ad una ricerca con condizione selettiva $X = x$. Comunque, per le operazioni di ricerca con diverse condizioni selettive, per esempio $a \leq X \leq b$ oppure $Y = y$ con $Y \neq X$, l'organizzazione modulare non ha molto più da offrire rispetto all'organizzazione non-ordinata. La tecnica di organizzazione che ora presentiamo mira a rendere efficiente l'esecuzione di quelle operazioni di ricerca in cui la condizione selettiva sia nella forma

$$a \leq X \leq b$$

dove X è la chiave primaria del file. A questo scopo, i record del file vengono ordinati (in maniera lessicografica) in base ai valori di X. Esplicitamente, sia $(B_1, \ldots, B_n)$ la catena dei blocchi impegnati dal file. In ogni blocco i record sono memorizzati per valori crescenti di X; inoltre, se u_i e v_i sono i valori di X nel primo record e nell'ultimo record del blocco B_i, allora

$$v_1 < u_2 \qquad v_2 < u_3 \qquad \ldots \qquad v_{n-1} < u_n$$

Infine, in ogni blocco si lascia volutamente un certo numero di pagine libere (tipicamente il 20 %) per l'eventuale inserimento di nuovi record del file. Oltre a questo, che viene chiamato *file principale*, viene creato un file ausiliario con n record, chiamato l'*indice* del file principale, anch'esso ordinato come il file principale sulla base di X. I record dell'indice, che per comodità chiamiamo *voci*, sono le coppie

$$(u_1, I_1) \qquad (u_2, I_2) \qquad \ldots \qquad (u_n, I_n),$$

dove I_i è l'indirizzo del blocco B_i. In realtà, il valore u_1 nella prima voce dell'indice è posto uguale ad un valore, convenzionalmente indicato con $-\infty$, che è più piccolo di ogni possibile valore di X. Ovviamente, X è una chiave identificativa dell'indice. cosicché un valore x di X determina al più una voce.

Indichiamo con $(b_1, \ldots, b_m)$ la lista dei blocchi necessari a memorizzare l'indice del file principale, e con $(k_1, \ldots, k_m)$ la *block directory* dell'indice, cioè la lista degli indirizzi dei blocchi $(b_1, \ldots, b_m)$. Si osservi che, siccome la lunghezza di una voce dell'indice è pari alla lunghezza di X più 4 byte (necessari per memorizzare l'indirizzo di un blocco), ogni blocco dell'indice può ospitare un numero di voci maggiore del numero di record del file principale e quindi $m < n$. Questo in molti casi fa sì che la *block directory* dell'indice possa essere mantenuta in memoria centrale durante l'intero utilizzo del file principale. D'ora in poi supponiamo che questo sia il caso.

file principale

blocco	*indirizzo*	*contenuto*
B_1	I_1	$(u_1, \ldots), \ldots, (v_1, \ldots)$
B_2	I_2	$(u_2, \ldots), \ldots, (v_2, \ldots)$
…	…	……
B_n	I_n	$(u_n, \ldots), \ldots, (v_n, \ldots)$

indice del file principale

blocco	*indirizzo*	*contenuto*
b_1	k_1	$(u_1, I_1), (u_2, I_2), \ldots$
…	…	……
b_m	k_m	$\ldots, (u_n, I_n)$

Ricerca sulla chiave primaria. Consideriamo il problema di localizzare il record del file principale che abbia $X = x$. Indipendentemente dall'esistenza di tale record nel file principale, si andrà a cercare la voce (v, J) che "copre" x nel senso che

— $v \leq x$ e

— se (v, J) non è l'ultima voce dell'indice (cioè $v \neq u_n$) e (w, K) è la voce successiva a (v, J), allora $x < w$.

È chiaro che, una volta trovata la voce (v, J) che copre x, la localizzazione del record cercato è nel blocco che è all'indirizzo J. Si osservi che se $v = x$ allora il record appartiene al file principale ed è proprio il primo record nel blocco che è all'indirizzo I. Dunque, ora la questione è come trovare la voce che copre x e questo è un problema di ricerca sull'indice che può essere risolto con una *ricerca binaria*.

(1) Porre $i := 1; f := m$.

(2) Fintantoché $i \neq f$ ripetere:

(2.1) Porre $h := \lceil (i+f)/2 \rceil$.

(2.2) Trovare l'indirizzo h.esimo nella lista degli indirizzi (k_1, …, k_m) presente nella *block directory* dell'indice.

(2.3) Caricare in memoria centrale il blocco (b_h) che è all'indirizzo k_h.

(2.4) Sia (u, I) la prima voce nel blocco b_h.

Caso 1: $x = u$. In tal caso, la voce cercata è (v, I) e la ricerca termina.

Caso 2: $x < u$. Porre $f := h-1$ e tornare all'istruzione (2.1).

Caso 3: $x > u$. Esaminare le voci nel blocco b_h alla ricerca di una voce (v, J) che non sia l'ultima in b_h e sia tale che, detta (w, K) la voce successiva a (v, J), si abbia $v \leq x < w$. Se la si trova, allora la voce cercata è (v, J) e la ricerca termina. Se invece la ricerca dà esito negativo allora, porre $i := h$ e tornare all'istruzione (2.1).

(3) Se $i = f$ allora esaminare le voci nel blocco b_i alla ricerca di una voce (v, J) che non sia l'ultima in b_i e sia tale che, detta (w, K) la voce successiva a (v, J), si abbia $v \leq x < w$. Se la si trova, allora la voce cercata è (v, J). Se invece la ricerca dà esito negativo allora la voce cercata è l'ultima voce di b_i.

Si osservi che, poiché ad ogni iterazione il numero dei blocchi dell'indice viene dimezzato, dopo al più $\lceil \log_2 m \rceil + 1$ passi la ricerca binaria termina.
Finalmente, una volta trovata la voce (v, J) che copre x, basterà caricare il blocco che è all'indirizzo J per localizzare il record del file principale con $X = x$.

Dato un file (principale) con N = 30.000 record tutti di lunghezza L = 100 byte. Assumiamo che la dimensione di un blocco sia D = 1024 byte, cosicché un blocco può contenere fino a $P = \lfloor D/L \rfloor$ = 10 record. Occorrono dunque $n = \lceil N/P \rceil$ = 3000 blocchi per memorizzare il file principale. Supponiamo che il file principale sia ordinato in base alla sua chiave X che abbia lunghezza 11 B. Dunque, la lunghezza di una voce dell'indice del file principale è 15 B, cosicché un blocco può contenere fino a 68 voci. Pertanto, l'indice impegnerà m = 45 blocchi. Una ricerca binaria sull'indice richiederà al più $\lceil \log_2 m \rceil$ = 6 accessi alla memoria secondaria e, dunque, la localizzazione di un record richiede 7 accessi alla memoria secondaria. Vediamolo con un esempio. Supponiamo di cercare il record con $X = x$ e che la voce che copre x è l'ultima del 35.esimo blocco (b_{35}) dell'indice. Dunque, se (u, I) è la prima voce nel blocco b_i, allora $u < x$ per $i \leq 35$ mentre $x < u$ per $i > 35$.

Prima Iterazione. Con $i = 1$ ed $f = 45$ si ha $h = \lceil (i+f)/2 \rceil = 23$. Siccome il valore di X nella prima voce nel blocco b_{23} è minore di x, si esamina l'intero contenuto di b_{23} ma senza successo. A questo punto si pone $i := 23$.

Seconda Iterazione. Con $i = 23$ ed $f = 45$ si ha $h = \lceil (i+f)/2 \rceil = 34$. Siccome il valore di X nella prima voce nel blocco b_{34} è minore di x, si esamina l'intero contenuto di b_{34} anche questa volta senza successo. A questo punto si pone $i := 34$.

Terza Iterazione. Con $i = 34$ ed $f = 45$ si ha $h = \lceil (i+f)/2 \rceil = 40$. Siccome il valore di X nella prima voce del blocco b_{40} è maggiore di x, si pone $f := 39$.

Quarta Iterazione. Con $i = 34$ ed $f = 39$ si ha $h = \lceil (i+f)/2 \rceil = 37$. Siccome il valore di X nella prima voce del blocco b_{37} è maggiore di x, si pone $f := 36$.

Quinta Iterazione. Con $i = 34$ ed $f = 36$ si ha $h = \lceil (i+f)/2 \rceil = 35$. Siccome il valore di X nella prima voce del blocco b_{35} è minore di x, si esamina l'intero contenuto di b_{35} di nuovo senza successo. A questo punto si pone $i := 35$.

Sesta Iterazione. Con $i = 35$ ed $f = 36$ si ha $h = \lceil (i+f)/2 \rceil = 36$. Siccome il valore di X nella prima voce del blocco b_{36} è maggiore di x, si pone $f := 35$. Essendo $i = f$, si riesamina l'intero contenuto del blocco b_{35} senza successo; allora, si conclude che la voce che copre x è l'ultima voce in b_{35}.

Nel discutere ora le operazioni di aggiornamento e di ricerca, assumeremo inizialmente che i record del file principale non siano appuntati alle pagine del blocco

Aggiunta. Per aggiungere un nuovo record, occorre individuare il blocco del file principale e la posizione dove va inserito. Se x è il valore della chiave del record, viene prima determinata la voce (v, J) dell'indice che copre x con una ricerca binaria sull'indice; quindi, si esamina il contenuto del blocco B che ha indirizzo J. Se B contiene pagine libere, allora il record viene inserito nella sua giusta posizione traslando i record i cui valori di X seguono x nell'ordinamento. Altrimenti (cioè se B non contiene pagine libere), si danno due casi. Siano u e v i valori di X nel primo e ultimo record presente in B.

Caso 1: il blocco che precede B ha pagine libere. In tal caso, si traferisce il primo record in B (con $X = u$) nel blocco che precede B (in ultima posizione); quindi, si memorizza il record da inserire in B e si modifica il valore di X nella voce dell'indice che si riferisce al blocco B.

Caso 2: il blocco che precede B non ha pagine libere oppure B è il primo blocco.

Caso 2*a*: il blocco che segue *B* ha pagine libere. In tal caso, se $x < v$ allora si traferisce l'ultimo record in *B* (con $X = v$) nel blocco che segue *B* (in prima posizione) e si memorizza il record da inserire in *B*; mentre, se $x > v$ allora si memorizza il record da inserire nel blocco che segue *B* (in prima posizione). In entrambi i casi, si modifica il valore di *X* nella voce dell'indice che si riferisce al blocco che segue *B*.

Caso 2*b*: il blocco che segue *B* non ha pagine libere oppure *B* è l'ultimo blocco. In tal caso, viene richiesto al *file system* un nuovo blocco B' e si ripartiscono i record in *B* e quello da inserire tra i blocchi *B* e B'. Infine, nell'indice viene inserita una nuova voce corrispondente al blocco B' dopo la voce corrispondente al blocco *B*.

Cancellazione. Per l'eliminazione di record che soddisfano una condizione selettiva che faccia uso della chiave primaria (ad es., $a \leq X \leq b$) si procede con una ricerca binaria per selezionare il primo record da cancellare ($X = a$) il cui bit di cancellazione viene posto a 0; quindi, si procede a cancellare tutti i record che seguono il primo fintantoché $X \leq b$.
Per l'eliminazione di record che soddisfano una condizione selettiva con chiave di ricerca diversa dalla chiave primaria, si procede con una ricerca lineare.
In entrambi i casi, può presentarsi la necessità di modificare una o più voci dell'indice. Infine, se dopo la cancellazione un blocco si trova ad avere solo pagine libere, verrà restituito al *file system*.

Modifica. Per la modifica di record che soddisfano una condizione selettiva che faccia uso della chiave primaria (ad es., $a \leq X \leq b$) si procede con una ricerca binaria per selezionare il primo record da cancellare ($X = a$) che viene aggiornato ed eventualmente ricollocato qualora la modifica interessa qualche campo in *X*. Quindi, si procede ad aggiornare tutti i record che seguono il primo fintantoché $X \leq b$.
Per l'eliminazione di record che soddisfano una condizione selettiva con chiave di ricerca diversa dalla chiave primaria, si procede con una ricerca lineare.
In entrambi i casi, può presentarsi la necessità di modificare una o più voci dell'indice.

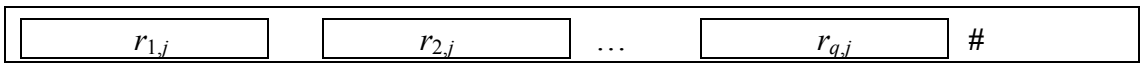
Indici

Gli indici sono *file* ausiliari che servono a rendere più spedita la ricerca dei record memorizzati nel *file* principale.

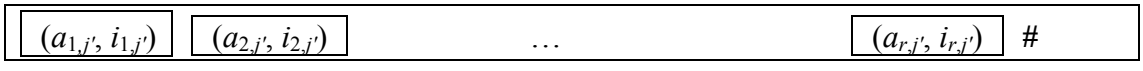
Indice. Un indice su un campo A, è un file ordinato su A in cui ogni record (*voce* dell'indice) contiene un valore a di A e la specifica dei blocchi che contengono record con $A = a$. Un indice è *denso* se contiene una voce per ogni record del file. Per semplicità ci limitiamo a considerare il caso in cui A sia una chiave. A seconda che A sia o meno la chiave primaria, l'indice è chiamato *indice primario* o *secondario*.

Indice primario: Dato un file principale ordinato in base alla chiave primaria A del file, è norma costruire anche l'indice primario basato su A, che è a sua volta un file ordinato. Ogni voce dell'indice contiene l'indirizzo di un blocco del file principale ed il valore a di A nel primo record del blocco, detto *record di ancoraggio*. Per la ricerca di un record possiamo impiegare una ricerca binaria con un miglioramento dato al fatto che il numero di blocchi necessari per memorizzare un indice è minore del numero dei blocchi necessari a memorizzare il file principale. Supponiamo che occorrano m' blocchi, $B'_1, \ldots, B'_{m'}$, per memorizzare l'indice primario.

blocco B_j del file principale



blocco $B'_{j'}$ dell'indice primario



Ricerca binaria nell'indice primario. Trovare l'ennupla t con $t(A) = a$, dove A è la chiave primaria della relazione.

test := Falso

$h := 1$; $k := m'$;

fintantoché $k \geq h$ ripetere

$j' := \lfloor (h+k)/2 \rfloor$;

copiare il blocco $B'_{j'}$ nel buffer;

siano $a_{1,j'}$ ed $a_{r,j'}$ i valori del campo A nel primo e nell'ultimo record di $B'_{j'}$ (v. Figura)

se $a < a_{1,j'}$ allora $k := j'-1$ altrimenti

se $a > a_{r,j'}$ allora $h := j'+1$ altrimenti (in tal caso $a_{1,j'} \leq a \leq a_{r,j'}$)

scorrere il buffer e prendere l'ultima voce (a', i_j) con $a \geq a'$;

copiare il blocco B_j nel buffer;

scorrere il buffer in cerca di un record con $A = a$;

se esiste è questo il record cercato

Esempio. Dato un file con n = 30.000 record di lunghezza l = 100 byte. Assumiamo che la dimensione di un blocco sia d = 1024 byte. Abbiamo visto che una ricerca binaria nel file principale basata sulla chiave primaria richiede all'incirca

$\log_2 3000 = 12$ *accessi*.

Supponiamo ora che la chiave abbia lunghezza 9 byte, e che un puntatore ad un blocco abbia lunghezza 6 byte. Allora una voce dell'indice primario avrà lunghezza 15 byte e quindi la capacità di un blocco è ora c' = 68 voci. Siccome l'indice primario ha una voce per ognuno dei 3000 blocchi del file principale, occorrono m' = 45 blocchi per memorizzare l'indice. Pertanto, una ricerca binaria nell'indice basata sulla chiave richiede circa 6 accessi; individuato il blocco $B_{j'}$ dell'indice occorre un altro accesso per il blocco B_j del file principale.

Indici secondari: L'indice è basato su di un attributo A che o è una chiave secondaria del file oppure non è una chiave. Il file principale può anche non essere ordinato, ma

l'indice è un file ordinato su A. Per semplicità esaminiamo solo il caso che A sia una chiave secondaria. Ogni voce riporta per ogni valore a di A nel file l'indirizzo del blocco che contiene il record con $A = a$; pertanto, l'indice è denso. Siccome l'indice è ordinato su A, è possibile una ricerca binaria, ma i record di ancoraggio non riguardano A.

L'indice è denso. Il miglioramento consiste nel fatto che se il file principale non è ordinato oppure è ordinato sulla chiave principale, allora la ricerca su A richiederebbe una ricerca lineare, mentre ora si può effettuare una ricerca binaria sull'indice.

Esempio. Dato un file con n = 30.000 record di lunghezza l = 100 byte. Assumiamo che la dimensione di un blocco sia d = 1024 byte. Una ricerca lineare nel file principale richiede in media circa $m/2$ = 1500 *accessi* e nel caso peggiore 3000 *accessi*. Supponiamo ora di aver costruito un file secondario con chiave analitica A che abbia lunghezza 9 byte, ed assumiamo che un puntatore ad un blocco abbia lunghezza 6 byte. Allora una voce dell'indice secondario avrà lunghezza 15 byte e quindi la capacità di un blocco è ora c' = 68 voci. Siccome l'indice secondario ha una voce per ogni valore di A nel file ed A è una chiave, allora avremmo 30000 voci che richiederanno m' = 442 blocchi. Pertanto, una ricerca binaria nell'indice basata su A richiede circa 9 accessi a blocchi dell'indice più uno ad un blocco del file principale.

Accanto al file principale F, utilizzeremo un altro file ordinato F', che chiamiamo *indice*, che contiene i record (x_1, I_1), (x_2, I_2), …, (x_n, I_n), dove I_i è l'indirizzo del blocco B_i $(1 \leq i \leq n)$. Per distinguerli dai record del file principale chiameremo *voci dell'indice* i record del file F'. Va notato che le voci dell'indice sono record che non sono appuntati alle pagine in cui sono scritti. Supponiamo di memorizzare l'indice F' nei blocchi $B'_0, B'_1, \ldots, B'_m$ della memoria secondaria e che in B'_0 sia disponibile una tabella T che riporti per ogni h $(1 \leq h \leq m)$ l'indirizzo I_h del blocco B'_h.

APPENDICE

A.1 L'algoritmo di chiusura di Beeri-Bernstein (`LinClosure`)

In questo paragrafo vogliamo valutare la "complessità" dell'algoritmo di Bernstein.

In generale, la *complessità* (*teorica*) di un algoritmo è il costo in termini di "operazioni elementari" che ci si aspetta di eseguire "nel caso peggiore" per un input di dimensione assegnata. Dunque, se x è la dimensione dell'input, la complessità dell'algoritmo è una funzione $T(x)$ che dà il massimo numero di operazioni elementari necessari perché l'algoritmo completi il suo lavoro. Ovviamente, $T(x)$ dipende da cosa si intenda per "operazione elementare". Faremo uso del modello di computazione basato una "macchina ad accesso casuale" (RAM). Anche così, esprimere la funzione $T(x)$ non è affatto semplice, ma spesso esiste una funzione $f(x)$ non troppo complicata che approssima il comportamento di $T(x)$ e scriveremo $T(x) = O(f(x))$ per dire che $T(x)$ è "dell'ordine" di $f(x)$ nel senso che esistono una costante $c > 0$ e una dimensione soglia $\xi > 0$ tali che $T(x) \leq c\, f(x)$ per ogni $x \geq \xi$. Nella maggior parte dei casi, la complessità $T(x)$ è almeno $O(x)$ visto che la maggior parte degli algoritmi leggono l'intero dato di input.

Tornando all'algoritmo di Bernstein, mostriamo che la sua complessità non è lineare. Successivamente, daremo un algoritmo meno semplice ma la cui complessità è lineare.

I dati di input dell'algoritmo di Bernstein sono F ed X. Sia p il numero delle dipendenze funzionali in F, cioè $|F| = p$ e sia n il numero di attributi presenti in F. Possiamo assumere senza perdere in generalità che X contenga solo attributi presenti in F cosicché $|X| \leq n$. Il caso peggiore si ha quando tutte le dipendenze funzionali in F vengono utilizzate per ottenere il valore finale della variabile V dell'Algoritmo di Chiusura e, ad ogni iterazione (esecuzione del ciclo (2.2)), sola una di loro viene effettivamente usata per incrementare il valore corrente della variabile V. Questo realmente capita ad esempio per $F = \{A_2 \rightarrow A_1,\ A_3 \rightarrow A_2,\ \ldots,\ A_n \rightarrow A_{n-1}\}$ ed $X = \{A_n\}$. In tal caso, abbiamo che la variabile V assume n valori distinti:

$$V_0 = \{A_n\},\ V_1 = \{A_{n-1}, A_n\},\ \ldots,\ V_{n-1} = \{A_1, \ldots, A_{n-1}, A_n\}.$$

Dunque, in generale con $p = |F|$ abbiamo che il numero delle iterazioni è al più uguale a p. Inoltre, ogni esecuzione del ciclo (2.2) richiede un numero $O(np)$ di operazioni elementari visto che occorrono $O(n)$ operazioni elementari per valutare l'inclusione tra due insiemi di attributi. Dunque, l'algoritmo di Bernstein ha complessità $O(np^2)$.

L'esempio precedente suggerisce come ridurre la complessità dell'algoritmo di Bernstein. L'implementazione può essere così ottenuta. Primo, se durante la sua esecuzione per qualche dipendenza funzionale $T \rightarrow Z$ si trova che T è incluso in V allora, quando Z viene aggiunto a V, la dipendenza funzionale $T \rightarrow Z$ non sarà più esaminata nelle esecuzioni successive del ciclo (2.2). Secondo, possiamo ulteriormente ridurre il numero di dipendenze funzionali da esaminare in un'esecuzione del ciclo (2.2) se sappiamo quali delle dipendenze funzionali in F hanno il determinante contenuto nel valore corrente di V. A tale scopo, abbiamo bisogno di una struttura dati che dinamicamente riporti quali dipendenze funzionali in F hanno il determinante contenuto nel valore corrente di V. Per far questo, enumeriamo le dipendenze funzionali da 1 a p, cioè $F = \{f_1, \ldots, f_p\}$ ed introduciamo

per ogni attributo A una lista $L(A) = \{i \leq p: A \in det(f_i)\}$

un array $k(i)$ di interi nonnegativi, $1 \leq i \leq p$, dove $k(i)$ fornisce il numero degli attributi in $det(f_i)$ che sono assenti dal valore corrente di V, cioè $k(i) = |det(f_i) - V|$.

Ovviamente la dipendenza funzionale f_i potrà essere usata per incrementare il valore di V solo quando $k(i) = 0$. Supponiamo che, ad un certo punto, venga esaminata la dipendenza funzionale $T \rightarrow Z$ e ci si trovi nella condizione $T \subseteq V$. Sia questa la j.esima dipendenza funzionale, cosicché $k(j) = 0$. Allora verrà eseguita l'istruzione $V := V \cup Z$. La sua esecuzione consiste nell'esaminare ogni attributo A in Z e, se A non era già presente in V, allora

A verrà aggiunto al valore di V;

la lista $L(A)$ verrà scandita e, per ogni $i \in L(A)$, l'intero $k(i)$ verrà decrementato di un'unità.

Terminata l'esecuzione dell'istruzione $V := V \cup Z$, si andrà in cerca di un valore di i tale che $k(i) = 0$. A tale scopo, basta tener traccia dell'insieme degli attributi A in V le cui liste non siano ancora state scandite. Se indichiamo tale insieme con U, abbiamo anche il vantaggio che la fine dell'algoritmo viene segnalata dalla condizione $U = \varnothing$.

Algoritmo di Beeri-Bernstein

Dati: una lista $(f_1, \ldots, f_p)$ di dipendenze funzionali ed un insieme X di attributi.

Risultato: un sovrainsieme V di X.

Procedura

(1) $V := X$; $U := X$.
Per ogni A creare una lista vuota $L(A)$.
Per ogni $i = 1, \ldots, p$
$k(i) := |det(f_i)|$;
per ogni attributo $A \in det(f_i)$, aggiungere i alla lista $L(A)$.

(2) Fintantoché $U \neq \varnothing$ ripetere:

(2.1) scegliere un attributo A in U ed eliminarlo da U;

(2.2) per ogni $i \in L(A)$

(2.3) $k(i) := k(i) - 1$;

(2.4) se $k(i) = 0$ allora

$W := dip(f_i) - V$; $V := V \cup W$; $U := U \cup W$.

Esempio A.1 Consideriamo la lista $F = (f_1, f_2, f_3, f_4, f_5)$ ed $X = \{A, E\}$ con

$f_1 = A \rightarrow D$, $f_2 = \{A, B\} \rightarrow E$, $f_3 = \{B, I\} \rightarrow E$,

$f_4 = \{C, D\} \rightarrow I$, $f_5 = E \rightarrow C$.

(Passo 1). $V = \{A, E\}$, $U = \{A, E\}$.

i	$k(i)$
1	1
2	2
3	2
4	2
5	1

A	$L(A)$
A	(1, 2)
B	(2, 3)
C	(4)
D	(4)
E	(5)
I	(3)

(Passo 2). Prima iterazione. Scelto l'attributo A in U, abbiamo U = {E}. Dalla scansione della lista L(A), otteniamo

i	$k(i)$
1	**0**
2	**1**
3	2
4	2
5	1

W = {D}, V = {A, D, E}, U = {E, D}.

Seconda iterazione. Scelto l'attributo E in U, abbiamo U = {D}. Dalla scansione della lista L(E), otteniamo

i	$k(i)$
1	0
2	1
3	2
4	2
5	**0**

W = {C}, V = {A, C, D, E}, U = {C, D}.

Terza iterazione. Scelto l'attributo C in U, abbiamo U = {D}. Dalla scansione della lista L(C), otteniamo

i	$k(i)$
1	0
2	1
3	2
4	**1**
5	0

Quarta iterazione. Scelto l'attributo D in U, abbiamo $U = \varnothing$. Dalla scansione della lista L(D), otteniamo

i	$k(i)$
1	0
2	1
3	2
4	**0**
5	0

$W = \{\text{I}\}$, $V = \{\text{A, C, D, E, I}\}$, $U = \{\text{I}\}$.

Quinta iterazione. Scelto l'attributo I in U, abbiamo $U = \varnothing$. Dalla scansione della lista L(I), otteniamo

i	$k(i)$
1	0
2	1
3	**1**
4	0
5	0

Siccome $U = \varnothing$, l'algoritmo termina.

■

Valutiamo la complessità dell'Algoritmo di Beeri-Bernstein nella dimensione $||F||$ dell'input.

Passo (1). L'inizializzazione delle variabili insiemistiche V ed U richiede $O(||F||)$ operazioni elementari. Inoltre, calcolare un singolo $k(i)$ richiede $|det(f_i)|$ operazioni elementari e calcolarli tutti richiede $O(||F||)$ operazioni elementari. Infine, un valore dell'indice i viene inserito in $|det(f_i)|$ liste e quindi la creazione delle liste $L(A)$ richiede $O(||F||)$ operazioni elementari.

Passo (2). Ogni attributo è aggiunto ad U al più una volta. Inoltre, quando un attributo A è aggiunto ad U, una singola operazione di decrementazione della variabile $k(i)$ è richiesta per ogni $i \in L(A)$. Siccome f_i compare in $|det(f_i)|$ liste, il numero di operazioni di decrementazioni sono al più $\sum_{i=1,\ldots,p} |det(f_i)|$ e, quindi, $O(||F||)$. Poi, il test della condizione $k(i) = 0$ sarà positivo al più una volta ed allora i non apparirà in nessuna delle liste che ancora attendono di essere esaminate. Infine, gli aggiornamenti

degli insiemi W ed U richiedono un numero $|dip(f_i)|$ di operazioni elementari se vengono rappresentati da sequenze di bit e, quindi, il numero totale di tali aggiornamenti è proporzionale a $||F||$.

Siccome entrambi i passi richiedono un numero di operazioni che è $O(||F||)$, la complessità dell'Algoritmo di Beeri-Bernstein è lineare nella dimensione di F e, quindi, migliore dell'algoritmo di Bernstein.

A.2 Il problema della chiave minima

Consideriamo il problema di trovare una chiave minima che, per quanto detto, può enunciarsi alla maniera seguente

> $\mathfrak{P}$. Dato un modello $M = [R, F]$, trovare una sovrachiave di M col minor numero di attributi.

Chiameremo le chiavi minime di M le *soluzioni* di $\mathfrak{P}$. La versione decisionale del problema $\mathfrak{P}$ è il problema che si enuncia alla maniera seguente:

> $\mathfrak{p}(k)$. Dati un modello $M = [R, F]$ ed un intero positivo $k \leq |R|$, esiste una sovrachiave di M con cardinalità non superiore a k ?

Chiameremo le sovrachiavi di M con k o meno attributi, se ve ne sono, le *soluzioni* di $\mathfrak{p}(k)$. Ora, è chiaro che, se conoscessimo una soluzione X del problema $\mathfrak{P}$, sapremmo che $\mathfrak{p}(k)$ non ammette soluzioni per $k < |X|$ e che X è una soluzione di $\mathfrak{p}(k)$ per $k \geq |X|$. Viceversa, se sapessimo che m è il più grande valore di k per cui il problema $\mathfrak{p}(k)$ non ammette soluzioni, sapremmo che le soluzioni di $\mathfrak{P}$ sono esattamente le soluzioni di $\mathfrak{p}(m+1)$. In altre parole, i problemi $\mathfrak{P}$ e $\mathfrak{p}(k)$ hanno la stessa complessità computazionale. La questione che ora si pone è la seguente: esiste un algoritmo efficiente (cioè polinomiale nella dimensione del problema) per risolvere il problema $\mathfrak{p}(k)$ qualunque sia k? Si *presume* che la risposta sia negativa perché $\mathfrak{p}(k)$ appartiene ad una classe di problemi decisionali (chiamata la classe "*NP*-completa") per i quali non esiste a tutt'oggi un algoritmo efficiente né, si congettura, mai se ne potrà trovare uno. Qualche parola sulla classe *NP*-completa.

Un problema decisionale $\mathfrak{d}$ appartiene alla *classe NP* se, qualora qualcuno (una sorta di *oracolo*) ci suggerisse una soluzione, diciamo X, al problema $\mathfrak{d}$, noi dovremmo poter verificare "facilmente" (cioè in tempo polinomiale) che effettivamente X è una soluzione del problema $\mathfrak{d}$. Un problema decisionale $\mathfrak{d}$ appartiene alla *classe NP-completa* se appartiene alla classe *NP* se esiste un problema decisionale $\mathfrak{d}^*$ la cui appartenenza alla classe *NP*-completa è già stata accertata, il quale sia "facilmente" (cioè in tempo polinomiale) riconducibile al problema $\mathfrak{d}$. Questo ci sta a dire che, se esistesse un algoritmo efficiente per risolvere $\mathfrak{d}$, allora *mutatis mutandis* potremmo utilizzare lo stesso algoritmo per risolvere in tempo polinomiale $\mathfrak{d}^*$.

L'appartenenza di $\mathfrak{p}(k)$ alla classe *NP* è presto verificata. Infatti, verificare che $|X| \leq k$ e che $\langle X \rangle_F = R$ può essere fatto in tempo polinomiale. Per provare l'appartenenza di $\mathfrak{p}(k)$ alla classe *NP* completa facciamo ricorso alla versione decisionale $\mathfrak{q}(k)$ del

problema $\mathfrak{Q}$ della "copertura minima" di un grafo dove per *copertura* si intende un insieme U di vertici del grafo tale che ogni arco del grafo abbia almeno un estremo in U:

> $\mathfrak{Q}$. Dato un grafo $G = (V, E)$, trovare una copertura di G col minor numero di vertici.

La versione decisionale del problema $\mathfrak{Q}$ è data dal seguente problema:

> $\mathfrak{q}(k)$. Dati un grafo $G = (V, E)$ ed un intero positivo $k < |V|$, esiste una copertura di G con non più di k vertici?

Dimostriamo per prima cosa che $\mathfrak{q}(k)$ è facilmente riconducibile a $\mathfrak{p}(k)$. Introduciamo per ogni vertice v di G un attributo che indichiamo con A_v, e per ogni arco e di G un attributo che indichiamo con A_e; i domini di questi attributi sono domini arbitrari. Siano

$$P = \{A_v: v \in V\} \qquad Q = \{A_e: e \in E\} \qquad R = P \cup Q.$$

Consideriamo il modello $M = [R, F]$ dove F è l'insieme delle $2|E| + 1$ dipendenze funzionali così definite:

$A_v \rightarrow A_e$ se v è un estremo di e (per ogni arco e di G)

$Q \rightarrow P$.

Così, la trasformazione di $\mathfrak{q}(k)$ in $\mathfrak{p}(k)$ richiede un tempo polinomiale.

Proviamo ora che esiste una copertura del grafo G di cardinalità non superiore a k se e solo se esiste una sovrachiave di M di cardinalità non superiore a k.

Cominciamo con la prova che l'esistenza di una copertura U di G con $|U| \leq k$ implica l'esistenza di una sovrachiave di M con non più di k attributi. Sia $X = \{A_v: v \in U\}$. Siccome U è una copertura di G, per ogni arco e di G, U contiene un vertice v che è un estremo di e. Pertanto, l'insieme X contiene, per ogni attributo A_e in Q, un attributo A_v tale che la dipendenza funzionale $A_v \rightarrow A_e$ è presente in F, cosicché la dipendenza funzionale $X \rightarrow A_e$ è implicata da F e, quindi, lo è anche la dipendenza funzionale $X \rightarrow Q$. Inoltre, grazie alla presenza in F della dipendenza funzionale $Q \rightarrow P$, abbiamo che anche la dipendenza funzionale $X \rightarrow P$ è implicata da F. Ne segue che $\langle X \rangle_F = R$, cioè X è una sovrachiave della stessa cardinalità di U e, dunque, di cardinalità non superiore a k.

Dimostriamo ora che l'esistenza di una sovrachiave X di M con $|X| \leq k$ implica l'esistenza di una copertura di G con non più di k vertici. Consideriamo gli insiemi

$$P' = P \cap X \qquad Q' = Q \cap X$$

$$V' = \{v \in V: A_v \in P'\} \qquad E' = \{e \in E: A_e \in Q'\}.$$

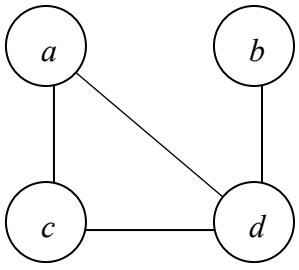
Per ogni arco e in E', scegliamo arbitrariamente uno dei due estremi di e, sia esso v_e; indichiamo poi con V^* l'insieme di tali vertici, cioè $V^* = \{v_e \in V: e \in E'\}$. Si osservi che $|V^*| \leq |E'|$. Siano poi

$$P^* = \{A_v: v \in V^*\} \qquad Y = P' \cup P^*.$$

Ovviamente, si ha che $|Y| \leq |P'| + |P^*| = |P'| + |V^*| \leq |P'| + |E'| = |P'| + |Q'| = |X| \leq k$. Ora, siccome $P' \subseteq Y$, la dipendenza funzionale $Y \rightarrow P'$ essendo banale è implicata da F. Inoltre, per ogni attributo A_e che appartiene a Q', l'arco e appartiene ad E' e, quindi, esiste un vertice in V^* che è estremo di e, cosicché esiste un attributo A_v in P^* e, quindi, in Y tale che la dipendenza funzionale $A_v \rightarrow A_e$ è presente in F; ne segue che la dipendenza funzionale $P^* \rightarrow Q'$ e, quindi, la dipendenza funzionale $Y \rightarrow Q'$ è implicata da F. Dunque, le due dipendenze funzionali $Y \rightarrow P'$ e $Y \rightarrow Q'$ sono implicate da F e, visto che $X = P' \cup Q'$, anche la dipendenza funzionale $Y \rightarrow X$ è implicata da F. Infine, siccome X è una sovrachiave di M (cioè $\langle X \rangle_F = R$), lo è pure Y (cioè $\langle Y \rangle_F = R$). Consideriamo ora l'insieme dei vertici $U = V' \cup V^*$. Ovviamente, si ha che $|U| = |Y| \leq k$. Ora supponiamo, per assurdo, che U non sia una copertura di G; allora, esisterebbe un arco e di G tale che nessuno dei suoi estremi appartenga ad U. Se così fosse, calcolando $\langle Y \rangle_F$ con l'algoritmo della chiusura, troveremmo che l'attributo A_e non appartiene a $\langle Y \rangle_F$ ed avremmo una contraddizione. Se ne conclude che U è una copertura di G (di cardinalità non superiore a k) ed U può essere calcolato in tempo polinomiale a partire da X.

Abbiamo così dimostrato che $\mathfrak{q}(k)$ appartiene alla classe NP ed è facilmente riconducibile a $\mathfrak{p}(k)$ e, siccome è già noto che $\mathfrak{q}(k)$ appartiene alla classe NP-completa, possiamo concludere che anche $\mathfrak{p}(k)$ appartiene alla classe NP-completa.

Esempio A.2 Consideriamo il grafo G mostrato in figura.



L'insieme $U = \{a, b\}$ non è una copertura di G perché nessuno dei due estremi dell'arco (c, d) appartiene ad U.

Introduciamo ora i due insiemi di attributi:

$$P = \{A_a, A_b, A_c, A_d\} \qquad Q = \{A_{(a,\,c)}, A_{(a,\,d)}, A_{(b,\,d)}, A_{(c,\,d)}\}$$

e consideriamo il modello $M = [R, F]$, dove $R = P \cup Q$ ed F contiene le nove dipendenze funzionali

$$A_a \rightarrow A_{(a,\,c)} \qquad A_a \rightarrow A_{(a,\,d)}$$

$$A_b \rightarrow A_{(b,\,d)}$$

$$A_c \rightarrow A_{(a,\,c)} \qquad A_c \rightarrow A_{(c,\,d)}$$

$$A_d \rightarrow A_{(a,\,d)} \qquad A_d \rightarrow A_{(b,\,d)} \qquad A_d \rightarrow A_{(c,\,d)}$$

$$Q \rightarrow P.$$

All'insieme $U = \{a, b\}$ corrisponde l'insieme di attributi $X = \{A_a, A_b\}$ e, siccome $\langle X \rangle_F = \{A_a, A_b, A_{(a,\,c)}, A_{(a,\,d)}, A_{(b,\,d)}\} \neq R$, X non è una sovrachiave del modello M.
D'altra parte, consideriamo l'insieme di attributi $X = \{A_d, A_{(a,\,c)}, A_{(b,\,d)}\}$. Allora, siccome la dipendenza funzionale $A_d \rightarrow \{A_{(a,\,d)}, A_{(b,\,d)}, A_{(c,\,d)}\}$ è implicata da F, anche la dipendenza funzionale $X \rightarrow Q$ e, quindi, la dipendenza funzionale $X \rightarrow R$ sono implicate da F. Dunque, X è una sovrachiave di M. Per costruire una copertura di G a partire da X, partiamo dall'insieme $E' = \{(a, c), (b, d)\}$. Abbiamo per V' quattro possibilità: $V' = \{a, b\}$, $V' = \{a, d\}$, $V' = \{b, c\}$ e $V' = \{c, d\}$. Prendiamone una; sia questa $V' = \{a, d\}$. Allora, l'insieme dei vertici $U = \{d\} \cup V' = \{a, d\}$ è una copertura di G ed ha un numero di vertici minore della cardinalità di X. ■

A.3 Il problema del modello di dimensione minima

Il risultato testé provato (cioè che, in generale, è un problema arduo quello di trovare una chiave minima) consente anche di provare che non è meno arduo il problema di trovare, dato un modello, un modello ad esso equivalente e di dimensione minima. Siano $\mathfrak{P}$ il problema della chiave minima e $\mathfrak{Q}$ il problema del modello equivalente di dimensione minima. Vogliamo dimostrare che risolvere $\mathfrak{Q}$ è *laborioso almeno quanto* risolvere $\mathfrak{P}$. Consideriamo il caso generico di $\mathfrak{P}$ in cui vogliamo trovare una chiave minima di un assegnato modello $M = [R, F]$. Siano ora A e B due attributi estranei ad R. Consideriamo il modello $N = [R \cup \{A, B\}, G]$ in cui

$$G = F \cup \{R \cup \{A\} \rightarrow B\}.$$

Il caso del problema $\mathfrak{Q}$ che vogliamo esaminare consiste nel trovare un modello $N^* = [R \cup \{A, B\}, G^*]$ equivalente ad N e di dimensione minima. Ovviamente, la trasformazione dal caso $\mathfrak{P}$ al caso $\mathfrak{Q}$ è polinomiale. Per dimostrare che, dato $N^* = [R \cup \{A, B\}, G^*]$, è possibile costruire in tempo polinomiale una chiave minima di M, abbiamo bisogno della seguente proprietà dei modelli $N^* = [R \cup \{A, B\}, G^*]$.

> Se $N^* = [R \cup \{A, B\}, G^*]$ è una modello equivalente ad N e di dimensione minima, allora
>
> (*i*) per ogni dipendenza funzionale $g \in G^*$, $det(g) \cap dip(g) = \emptyset$;
> (*ii*) G^* contiene una e solo una dipendenza funzionale della forma $Z \cup \{A\} \rightarrow B$ dove Z è una chiave minima di M.

(*i*) La proprietà deriva dal fatto che N è un modello di dimensione minima.
(*ii*) Supponiamo per assurdo che G^* non contenga nessuna dipendenza funzionale g tale che $A \in det(g)$ e $B \in dip(g)$. Allora, con l'algoritmo di Bernstein troveremmo che

$$\langle R \cup \{A\} \rangle_{G^*} = R \cup \{A\}.$$

D'altra parte,

$$\langle R \cup \{A\} \rangle_{G} = R \cup \{A, B\}$$

e quindi si avrebbe

$$\langle R \cup \{A\} \rangle_{G^*} \neq \langle R \cup \{A\} \rangle_{G}$$

cosa che contrasta con l'ipotesi che N^* è equivalente ad N.

Siano $g_1, \ldots, g_m$ le dipendenze funzionali in G^* tale che $A \in det(g_i)$ e $B \in dip(g_i)$ ($1 \leq i \leq m$); cioè, ogni g_i è della forma

$$X_i \cup \{A\} \rightarrow Y_i \cup B.$$

Supponiamo, per assurdo, che $m > 1$ oppure che, per qualche valore di i, l'insieme X_i non sia una chiave di M. Allora, è evidente che, scelta una qualsiasi chiave Z di M, il modello che si ottiene da N^* sostituendo le m dipendenze funzionali $g_1, \ldots, g_m$ con la sola dipendenza

$$Z \cup \{A\} \rightarrow B$$

sarebbe equivalente ad N^* ed avrebbe dimensione strettamente minore di N^*. Pertanto, siccome N^* ha dimensione minima, G^* contiene esattamente una dipendenza funzionale g con $A \in det(g)$ e $B \in dip(g)$ e g è della forma

$$Z \cup \{A\} \rightarrow B$$

dove Z è una chiave di M. Infine, Z deve essere una chiave minima di M perché N^* ha dimensione minima. Ne segue che, dato $N^* = [R \cup \{A, B\}, G^*]$, per trovare una chiave minima di M basterà scandire G^* fino ad incontrare quell'unica dipendenza funzionale g con la proprietà (*ii*) e, una volta trovata, concludere che $det(g) - \{A\}$ è una chiave minima di M. Siccome il tutto può essere fatto in tempo polinomiale, possiamo concludere che risolvere il problema del modello equivalente di dimensione minima risulta essere un compito laborioso almeno quanto risolvere il problema della chiave minima.

A.4 Le tredici regole di Codd

Rule 0: The *foundation rule*

A relational database management system must manage its stored data using only its relational capabilities. The system must qualify as *relational*, as a *database*, and as a *management system*. For a system to qualify as a relational database management system (RDBMS), that system must use its *relational* facilities (exclusively) to *manage* the *database*.

Rule 1: The *information rule*

All information in a relational database (including table and column names) is represented in only one way, namely as a value in a table.

Rule 2: The *guaranteed access rule*

All data must be accessible. This rule is essentially a restatement of the fundamental requirement for primary keys. It says that every individual scalar value in the database must be logically addressable by specifying the name of the containing table, the name of the containing column and the primary key value of the containing row.

Rule 3: *Systematic treatment of null values*

The DBMS must allow each field to remain null (or empty). Specifically, it must support a representation of "missing information and inapplicable information" that is systematic, distinct from all regular values (for example, "distinct from zero or any other number", in the case of numeric values), and independent of data type. It is also implied that such representations must be manipulated by the DBMS in a systematic way.

Rule 4: *Active online catalog based on the relational model*

The system must support an online, inline, relational catalog that is accessible to authorized users by means of their regular query language. That is, users must be able to access the database's structure (catalog) using the same query language that they use to access the database's data.

Rule 5: The *comprehensive data sublanguage rule*

The system must support at least one relational language that

1. **Has a linear syntax**
2. **Can be used both interactively and within application programs,**

3. **Supports data definition operations (including view definitions), data manipulation operations (update as well as retrieval), security and integrity constraints, and transaction management operations (begin, commit, and rollback).**

Rule 6: The *view* *updating rule*

All views that are theoretically updatable must be updatable by the system.

Rule 7: *High-level insert, update, and delete*

The system must support set-at-a-time *insert*, *update*, and *delete* operators. This means that data can be retrieved from a relational database in sets constructed of data from multiple rows and/or multiple tables. This rule states that insert, update, and delete operations should be supported for any retrievable set rather than just for a single row in a single table.

Rule 8: *Physical data independence*

Changes to the physical level (how the data is stored, whether in arrays or linked lists etc.) must not require a change to an application based on the structure.

Rule 9: *Logical data independence*

Changes to the logical level (tables, columns, rows, and so on) must not require a change to an application based on the structure. Logical data independence is more difficult to achieve than physical data independence.

Rule 10: *Integrity independence*

Integrity constraints must be specified separately from application programs and stored in the catalog. It must be possible to change such constraints as and when appropriate without unnecessarily affecting existing applications.

Rule 11: *Distribution independence*

The distribution of portions of the database to various locations should be invisible to users of the database. Existing applications should continue to operate successfully:

1. **when a distributed version of the DBMS is first introduced; and**
2. **when existing distributed data are redistributed around the system.**

Rule 12: The *nonsubversion rule*

If the system provides a low-level (record-at-a-time) interface, then that interface cannot be used to subvert the system, for example, bypassing a relational security or integrity constraint.

Alfabeto greco

α	Α	alfa
β	Β	beta
γ	Γ	gamma
δ	Δ	delta
ε	Ε	epsilon
ζ	Ζ	zeta
η	Η	eta
θ	Θ	theta
ι	Ι	iota
κ	Κ	kappa
λ	Λ	lambda
μ	Μ	mu (o mi)
ν	Ν	nu (o ni)
ξ	Ξ	csi
ο	Ο	omicron
π	Π	pi
ρ	Ρ	rho
σ	Σ	sigma
τ	Τ	tau
υ	Υ	ypsilon
φ	Φ	fi
χ	Χ	chi
ψ	Ψ	psi
ω	Ω	omega