

Basi di dati

Elisa Solinas

Corso dei proff. Ruggero Pensa (Teoria)

Sara Capecchi (Progettazione)

Luca Anselma (SQL)

A.A. 2016/2017

Updated on 25 giugno 2017

Indice

I	Teoria	7
1	Modello Relazionale	8
1.1	Relazione	8
1.2	Attributo	8
1.2.1	Tipologia	8
1.2.2	Dominio	8
1.2.3	Valore nullo	9
1.3	Schema di una relazione	9
1.4	Grado di una relazione	9
1.5	Istanza di una relazione	9
1.6	Cardinalità di una relazione	10
1.7	Notazioni ibride	10
1.8	Tupla	11
1.8.1	Valori di una tupla	11
1.9	Vincoli locali	12
1.9.1	Vincolo locale di dominio	12
1.9.2	Vincolo sui valori nulli	12
1.9.3	Altri vincoli	12
1.9.4	Vincolo di identificazione	13
1.9.5	Definizione di superchiave	13
1.9.6	Chiave candidata	13
1.9.7	Chiave principale	14
1.10	Correttezza di una relazione	14
1.11	Schema di una base dati	14
1.12	Vincoli globali	14
1.12.1	Vincolo di integrità referenziale	14
1.12.2	Vincoli globali generali	15
1.13	Base dati	15
1.13.1	Definizione di schema di una base dati	15
1.13.2	Stato di una base di dati	15
1.14	Considerazione finale sul modello relazionale	15

2	Algebra relazionale	16
2.1	Operatore selezione	16
2.1.1	Sintassi del predicato di selezione	17
2.1.2	Cardinalità della selezione	17
2.2	Operatore proiezione	17
2.2.1	Cardinalità della proiezione	18
2.3	Composizione di operatori	19
2.4	Operatori insiemistici: unione e differenza	19
2.4.1	Cardinalità dell'unione	20
2.4.2	Cardinalità della differenza	20
2.5	Operatore intersezione	21
2.5.1	Cardinalità dell'intersezione	21
2.6	Operatore ridenominazione	21
2.7	Uso corretto degli operatori insiemistici	21
2.8	Ridenominazione schema	22
2.9	Prodotto cartesiano	22
2.9.1	Cardinalità del prodotto cartesiano	23
2.10	Operatore \bowtie_{θ}	23
2.10.1	Cardinalità del \bowtie_{θ}	24
2.10.2	Join con schemi non disgiunti	25
2.10.3	Semplificazione della ridenominazione	26
2.11	Operatore Natural-Join	26
2.12	Operatore Semi-Join	26
2.12.1	Cardinalità del semi-join	27
2.13	Join esterni	27
2.13.1	Left Join	27
2.13.2	Right Join	27
2.13.3	Full Join	27
3	Tipologie di interrogazioni	28
3.1	Interrogazioni con negazione	28
3.1.1	Negazione essenziale	28
3.1.2	Esempi	29
3.2	Interrogazioni con quantificazione universale	29
3.2.1	Operatore derivato quoziente	29
3.3	Semantica di Codd del valore nullo	31
3.4	Logica a tre valori	31
3.4.1	Operatori e logica a tre valori	32
3.4.2	Espressioni e logica a tre valori	32
3.4.3	Limiti della logica a tre valori	33
3.4.4	Perché la logica a tre valori	33
3.5	Proprietà degli operatori algebrici	34
3.5.1	Proprietà commutativa del prodotto cartesiano	34

3.5.2	Proprietà commutativa del θ -join	34
3.5.3	Proprietà associativa del prodotto cartesiano	34
3.5.4	Proprietà associativa del θ -join	34
3.5.5	Proprietà della selezione multipla	35
3.5.6	Proprietà della sostituzione di operatori	35
3.5.7	Proprietà distributiva della selezione rispetto alla proiezione	35
3.5.8	Proprietà distributiva della selezione rispetto al prodotto cartesiano	36
3.5.9	Proprietà distributiva della selezione rispetto al join	36
3.5.10	Proprietà distributiva della selezione rispetto all'unione e differenza	36
3.5.11	Proprietà della proiezione multipla	36
3.5.12	Proprietà distributiva della proiezione rispetto al prodotto cartesiano	37
3.5.13	Proprietà distributiva della proiezione rispetto al join	37
3.5.14	Proprietà distributiva della proiezione rispetto all'unione	37
3.5.15	Altre proprietà	37
4	Calcolo relazionale	38
4.1	Calcolo su tuple con dichiarazione di range	38
4.1.1	Join nel calcolo relazionale	39
4.1.2	Quantificazione universale ed esistenziale	39
4.1.3	Negazione essenziale	40
4.1.4	Prodotto cartesiano	41
4.1.5	Unione	41
4.1.6	Limiti del calcolo relazionale	41
5	Normalizzazione	43
5.1	Ridondanze e anomalie	43
5.2	Dipendenze funzionali	44
5.3	Teoria di Armstrong	45
5.3.1	Assiomi della teoria di Armstrong	45
5.3.2	Vincolo di dipendenza funzionale	45
5.3.3	Attributi estranei	46
5.4	Forma normale di Boyce e Codd	46
5.4.1	Definizione di forma normale di Boyce e Codd	46
5.4.2	Decomposizione in forma normale di Boyce e Codd	47
5.5	Proprietà delle decomposizioni	47
5.5.1	Decomposizione senza perdita	47
5.5.2	Conservazione delle dipendenze	48
5.5.3	Qualità delle decomposizioni	48
5.6	Terza forma normale	48
5.6.1	Limitazioni della forma normale di Boyce e Codd	48
5.6.2	Definizione di terza forma normale	49
5.6.3	Decomposizione in terza forma normale	49
5.7	Teoria delle dipendenze e normalizzazione	49
5.7.1	Implicazione di dipendenze funzionali	50

<i>INDICE</i>	4
5.7.2 Coperture di insiemi di dipendenze funzionali	51
5.7.3 Sintesi di schemi in terza forma normale	52
6 Architettura dei DBMS	54
6.1 Memoria centrale	55
6.1.1 Le pagine	55
II SQL	58
7 Introduzione	59
7.1 PostgreSQL	59
7.2 Domini dei dati	60
7.2.1 Caratteri e stringhe	60
7.2.2 Tipi numerici esatti	60
7.2.3 Tipi numerici approssimati	61
7.2.4 Date	61
7.2.5 Intervalli	62
7.2.6 Altri domini	62
7.2.7 Domini personalizzati	62
7.3 Definizione di tabelle	63
7.3.1 Valori di default	64
7.4 Definizione dei vincoli	64
7.4.1 Vincoli intrarelazionali predefiniti	65
7.5 Modifiche alle definizioni	69
8 Query di Base	71
8.1 Clausola select	72
8.2 Clausola from	73
8.3 Clausola where	74
8.4 Gestione dei valori nulli	76
8.5 Duplicati	77
8.6 Uso di variabili	79
8.7 Ordinamento	80
9 Join	81
10 Aggregati	85
10.1 Operatori aggregati	85
10.1.1 Operatore count	85
10.1.2 Altri operatori aggregati	87
10.2 Interrogazioni con raggruppamento	88
10.2.1 Predicati sui gruppi	89
11 Operatori Insiemistici	92

12 Query Nidificate	96
12.1 Interrogazioni nidificate complesse	99
13 Modifica dei dati	103
13.1 Inserimento	103
13.2 Cancellazione	104
13.3 Modifica	104
 III Progettazione	 107
14 Metodologie e Modelli per il progetto	108
14.1 Introduzione alla progettazione	108
14.1.1 Il ciclo di vita dei sistemi informativi	108
14.1.2 Metodologie di progettazione e basi di dati	109
14.2 Il modello E-R	112
14.2.1 I costrutti principali del modello	112
14.3 Panoramica finale sul modello E-R	117
14.4 Documentazione di schemi E-R	118
14.4.1 Regole aziendali	118
14.4.2 Tecniche di documentazione	119
15 Progettazione Concettuale	121
15.1 Raccolta e analisi dei requisiti	121
15.1.1 Fonti dei requisiti	121
15.1.2 Specifica dei requisiti	122
15.1.3 Specifiche sulle operazioni	123
15.2 Criteri per la rappresentazione dei dati	123
15.2.1 Criteri generali di rappresentazione	123
15.2.2 Qualità di uno schema concettuale	123
16 Progettazione Logica	125
16.1 Fasi della progettazione logica	125
16.2 Analisi delle prestazioni su schemi E-R	126
16.3 Ristrutturazione di schemi E-R	128
16.3.1 Analisi delle ridondanze	128
16.3.2 Eliminazione delle generalizzazioni	129
16.3.3 Partizionamento/accorpamento di concetti	132
16.3.4 Scelta degli identificatori principali	134
16.4 Traduzione verso il modello relazionale	135
16.4.1 Entità e associazioni molti a molti	135
16.4.2 Associazioni uno a molti	138
16.4.3 Entità con identificatore esterno	139
16.4.4 Associazioni uno a uno	140

16.5 Documentazione di schemi logici	142
--	-----

Parte I

Teoria

Capitolo 1

Modello Relazionale

1.1 Relazione

Una relazione è definita da uno schema e dalle istanze della relazione (dette anche stato della relazione)

1.2 Attributo

L'attributo di una relazione (attributo relazionale) è definito come una coppia $A_i : T_i$ dove T_i è il tipo dell'attributo e A_i è il nome dell'attributo.

1.2.1 Tipologia

Un tipo T_i è caratterizzato da:

- T_i : nome **identificativo** del tipo
- D_i : **dominio** di valori
- Collezione di **operazioni** abilitate ad agire su D_i
- Operazioni di **confronto** su D_i

Tipi standard: Integer, Real, String, Char, Date, Boolean

Tipi utente: tipo enumerativo.

1.2.2 Dominio

Un tipo è sempre associato ad un dominio, ovvero a un insieme (potenziale infinito) di valori.

Possiamo immaginare una funzione

$$\text{dom}(A_i) = D_i \quad (1.1)$$

1.2.3 Valore nullo

Per i casi in cui si riveli necessario inserire delle tuple con attributi mancanti in una relazione, è previsto un **valore nullo** (NULL), un vero e proprio valore, con la seguente proprietà generale:

$$\forall T_i, \quad \text{NULL} \in D_i \quad (1.2)$$

Cioé, il valore NULL appartiene al dominio D_i di qualsiasi tipo T_i .

1.3 Schema di una relazione

Lo **schema di una relazione** è un insieme di attributi, espressi con la seguente notazione:

$$\begin{aligned} &\{A_1 : T_1, \dots, A_n : T_n\} \\ &R(A_1 : T_1, \dots, A_n : T_n) \end{aligned} \quad (1.3)$$

Spesso, per semplificare, si sottintende il tipo (che, però, deve sempre essere specificato nei DBMS):

$$\begin{aligned} &\{A_1 : T_1, \dots, A_n : T_n\} \rightarrow \{A_1, \dots, A_n\} \\ &R(A_1 : T_1, \dots, A_n : T_n) \rightarrow R(A_1, \dots, A_n) \end{aligned} \quad (1.4)$$

Definiamo l'insieme di attributi A come:

$$A : \{A_1 : T_1, \dots, A_n : T_n\} \quad (1.5)$$

Possiamo utilizzare la seguente notazione per indicare lo schema R :

$$R(A) \quad (1.6)$$

1.4 Grado di una relazione

La **cardinalità** $|A|$, anche detta **grado** della relazione, di uno schema di relazione è data dal numero di attributi dello schema.

L'insieme di attributi di uno schema di relazione è sempre un insieme non vuoto. Ergo:

$$|A| \geq 1 \quad (1.7)$$

1.5 Istanza di una relazione

Dato uno schema:

$$A : \{A_1 : T_1, \dots, A_n : T_n\} \quad (1.8)$$

Per semplicità, supponiamo che l'insieme A sia ordinato.

L'**istanza**, o **stato** r di una relazione è definito come un insieme di tuple per cui vale la proprietà:

$$\begin{aligned} & \langle v_1, \dots, v_n \rangle : \\ & \forall i \quad v_i \in \text{dom}(A_i) \end{aligned} \quad (1.9)$$

1.6 Cardinalità di una relazione

La **cardinalità** $|r|$ dell'istanza di una relazione r è data dal numero di tuple appartenenti alla relazione r ed è detta semplicemente **cardinalità della relazione**.

Contrariamente alla relazione, l'istanza di una relazione può essere vuota:

$$|r| \geq 0 \quad (1.10)$$

1.7 Notazioni ibride

Per indicare le relazioni utilizziamo lettere maiuscole:

$$R \text{ oppure } PAZIENTI \quad (1.11)$$

Per indicare l'istanza delle relazioni utilizziamo lettere minuscole:

$$r \text{ oppure } pazienti \quad (1.12)$$

Specifichiamo lo schema anche per le istanze:

- $r(A_1, \dots, A_n)$ o $pazienti(COD, Cognome, Nome, Residenza, AnnoNascita)$
- $r(R)$ o $pazienti(PAZIENTI)$: poco usate.
- r o $pazienti$: se lo schema è ben chiaro

Possiamo usare anche la notazione tabellare: Ciascuna relazione è definita da una coppia:

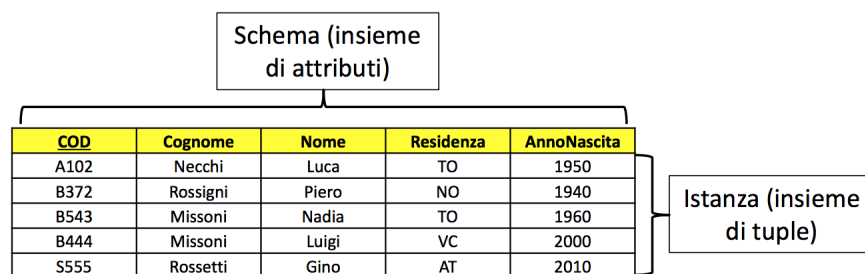


Figura 1.1: Notazione tabellare

$$\langle R, r \rangle \quad (1.13)$$

dove R è lo schema della relazione e r è l'istanza della relazione.

1.8 Tupla

Data la relazione $r(A_1, \dots, A_n)$, per indicare una singola **tupla** si usa la notazione

$$t = \langle v_1, \dots, v_n \rangle \quad (1.14)$$

Poiché t prende valori all'interno della relazione R :

$$t \in r(A_1, \dots, A_n) \quad (1.15)$$

1.8.1 Valori di una tupla

Per indicare il valore di una tupla t in corrispondenza dell'attributo A_i o di un insieme di attributi A_i, \dots, A_k si utilizzano le seguenti notazioni:

$$\begin{aligned} t[A_i] &= v_i \\ t.A_i &= v_i \\ t[A_i, \dots, A_k] &= \langle v_i, \dots, v_k \rangle \\ t.(A_i, \dots, A_k) &= \langle v_i, \dots, v_k \rangle \end{aligned} \quad (1.16)$$

Per richiamare la tupla su tutti gli attributi si utilizzano le seguenti notazioni:

$$t, t[A], t.A \quad (1.17)$$

Non essendo importante l'ordine degli attributi, la tupla non è altro che una **rappresentazione tabellare** che fa corrispondere a ogni attributo un valore del suo dominio:

$$t : A \rightarrow D \quad (1.18)$$

Esempio

A	D
COD	A102
Cognome	Necchi
Nome	Luca
Residenza	TO
AnnoNascita	1950

Possiamo adesso definire l'istanza r di una relazione è dunque un insieme di funzioni

$$\{t_1, \dots, t_n\} \quad (1.19)$$

Ciascuna funzione t_i definisce una particolare corrispondenza tra attributi e valori. Le funzioni t_i sono tutte distinte tra loro: ciò significa che in una relazione non può esistere una tupla identica a un'altra.

La tupla t è **compatibile** con lo schema A di una relazione se:

- t è una **funzione totale** su A .
- Ogni valore prodotto da t appartiene a D .
- Vale il vincolo:

$$\forall A_i, \quad t.A_i \in \text{dom}(A_i) \quad (1.20)$$

1.9 Vincoli locali

I **vincoli locali** sono caratterizzazioni dei valori possibili delle tuple in modo che tali valori rispettino (non siano in contrasto con) la realtà che vogliamo rappresentare.

1.9.1 Vincolo locale di dominio

Per ogni attributo abbiamo un insieme ben definito (anche infinito) di valori possibili. Inoltre:

- Qualsiasi valore v del dominio D_i è atomico
- Un attributo il cui valore è una relazione non è atomico (solo alcuni DBMS, tra cui Oracle, lo permettono).

Una relazione è **in prima forma normale** quando tutti gli attributi sono abbinati a domini di **valori atomici**.

1.9.2 Vincolo sui valori nulli

Un vincolo NOT NULL su un attributo A_i è soddisfatto se, per ogni $t \in r$, il valore $t.A_i$ non è nullo.

Il funzionamento **pratico** è il seguente: al momento del caricamento dei dati in una tabella, il DBMS verifica che la tupla sia compatibile e che i valori degli attributi per cui è attivo il vincolo NOT NULL siano non nulli.

1.9.3 Altri vincoli

I DBMS forniscono la sintassi necessaria per la verifica dei seguenti vincoli:

- **Restrizioni sui domini:** specificano i valori che non possono essere assunti da un particolare attributo.
ES: una persona non può avere più di 120 anni, e non può essere nata nel futuro.
ES: la temperatura corporea di un paziente non può superare i 42 gradi ed essere inferiore ai 34.
- **Confronti sui domini:** esprimono legami che devono essere rispettati da due o più attributi.
ES: la data d'inizio ricovero dev'essere antecedente alla data di fine ricovero.

1.9.4 Vincolo di identificazione

Il vincolo di identificazione ci permette di identificare univocamente una tupla all'interno di uno schema, ed è espresso attraverso il vincolo di **chiave relazionale**.

Le chiavi relazionali possono essere composte da più valori, come nella tabella *RICOVERI*.

La nozione di vincolo di chiave relazionale è solitamente definita dal contesto applicativo (es. *COD* dei *pazienti* o *MATR* dei medici).

Nella relazione di Codd gli elementi (tuple) dell'insieme (istanza) r devono essere tutti distinti tra di loro.

1.9.5 Definizione di superchiave

Per poter definire in maniera formale il vincolo di chiave relazionale dobbiamo introdurre prima la nozione di superchiave.

Data una relazione $r(A)$, un sottoinsieme di attributi $sk \subseteq A$ è una **superchiave**

$$\forall i, j \quad (t_i[sk] = t_j[sk]) \rightarrow (t_i[A] = t_j[A]) \quad (1.21)$$

ES: nella tabella *PAZIENTI*, $(COD, Nome)$ è una superchiave.

1.9.6 Chiave candidata

$k \subseteq A$ è **chiave candidata** se:

- k è superchiave di R
- k è superchiave **minimale** di R (ogni sottoinsieme proprio di k non deve soddisfare le condizioni di superchiave).

ES: *COD* è superchiave, quindi $\{COD; Cognome\}$ non è chiave candidata.

Se sk è una superchiave, allora $sk \subseteq w \subseteq A$ è una superchiave, quindi sono valide le seguenti **proprietà**:

- A è necessariamente una superchiave (per la definizione stessa di istanza, ogni tupla è distinta).
- A è superchiave, quindi A deve contenere sempre almeno una chiave candidata.

1.9.7 Chiave principale

La chiave principale $pk \subseteq A$ è una chiave scelta dal progettista tra tutte le possibili chiavi candidate.

Tutti gli attributi di una chiave principale devono rispettare in vincolo NOT NULL: una chiave candidata, invece, può ammettere valori nulli.

1.10 Correttezza di una relazione

Un'istanza di relazione $r(A)$ è corretta se ogni tupla $t \in r$ è compatibile con lo schema A e sono soddisfatti tutti i vincoli locali (o intrarelazionali).

1.11 Schema di una base dati

Lo schema di una base dati è un insieme di schemi di relazione tutti con nome:

$$S = \{R_1, \dots, R_n\} \quad (1.22)$$

S rispetta le seguenti proprietà:

- I nomi devono essere tutti distinti
- Ad ogni schema si abbinano i rispettivi vincoli locali
- Ogni relazione R_i esibisce una chiave principale

1.12 Vincoli globali

Così come è possibile definire dei vincoli locali (**intrarelazionali**) su ogni relazione R è altresì possibile definire vincoli globali (**interrelazionali**) su uno schema di basi di dati S .

1.12.1 Vincolo di integrità referenziale

Consideriamo una relazione $R_h(\underline{PK}, \dots)$ dove PK è l'insieme di attributi della chiave principale, e una relazione $R_s(\dots, B, \dots)$ dove B è un insieme di attributi qualsiasi.

Esiste un **vincolo di integrità referenziale** degli attributi B rispetto alla tavola R_h se:

$$(t_i \in r(R_s)) \rightarrow \forall t_i \exists t_j : [(t_j \in r(R_h)) \wedge (t_i[B] = t_j[PK])] \quad (1.23)$$

Per poter creare il vincolo è necessario che i domini degli attributi in PK siano compatibili con i domini degli attributi in B .

1.12.2 Vincoli globali generali

I vincoli globali generali sono suggeriti dalle regole imposte dal sistema informativo e sono detti **regole di business**.

ES: in un sistema bancario con relazione *CONTOCORRENTE* e relazione *MUTUO*, tutti i titolari di mutuo devono essere ten

1.13 Base dati

1.13.1 Definizione di schema di una base dati

La definizione dello schema S di una base dati è completata dai vincoli globali.

1.13.2 Stato di una base di dati

Dato uno schema di base dati $S = \{R_1, \dots, R_n\}$, lo **stato della base di dati** è definito come:

$$DB = \{ \langle R_1, r_1 \rangle, \dots, \langle R_n, r_n \rangle \} \quad (1.24)$$

Dove ogni r_i deve essere corretta e devono essere soddisfatti tutti i vincoli globali.

1.14 Considerazione finale sul modello relazionale

Il modello relazionale è un modello **orientato ai valori**.

Quando costruisco informazioni che coinvolgono tuple di relazioni diverse non posso che lavorare sui valori contenuti nelle tuple.

L'identificazione (chiave) di una relazione è basata sui valori presenti nella chiave.

In altri modelli (gerarchico, reticolare) le corrispondenze tra dati avvengono attraverso riferimenti (puntatori).

Capitolo 2

Algebra relazionale

Il modello relazionale descrive il modello del **DDL** (Data Definition Language), mentre l'algebra relazionale si occupa del **DML** (Data Manipulation Language), che manipola le tuple presenti nella base dati.

Il cuore del DML è il linguaggio di interrogazione.

Secondo il modello dell'interrogazione di Codd, ciascuna interrogazione ha come input la base dati e come output una relazione.

ES: elencare i pazienti ricoverati nel reparto il cui primario è il Dott. Neri; visualizzare cognome, nome e nome del reparto.

Codd formalizza l'interrogazione utilizzando il paradigma algebrico, cioè una costruzione **procedurale** dell'interrogazione: ciascuna interrogazione elenca i passi per eseguire l'operazione.

La presentazione è però più astratta, in quanto utilizza degli operatori algebrici.

Gli operatori dell'algebra relazionale ricevono, come argomenti, relazioni e producono in uscita altre relazioni (dette anche **relazioni virtuali**).

2.1 Operatore selezione

Data una relazione r su uno schema A , l'operatore di selezione viene indicato con

$$\sigma_p(r(A)) \quad (2.1)$$

Dove p è un predicato e $r(A)$ è l'argomento dell'operatore.

La selezione produce una relazione senza nome che ha l'identico schema A della relazione argomento, ma solo le tuple che soddisfano il predicato p .

2.1.1 Sintassi del predicato di selezione

Il predicato p è un'espressione booleana di predicati atomici, che possono essere solo di due tipi: confronto tra due attributi, oppure confronto tra un attributo e una costante.

2.1.2 Cardinalità della selezione

La cardinalità della relazione virtuale prodotta dall'operazione di selezione $\sigma_p(r(A))$ è

$$0 \leq |\sigma_p(r(A))| \leq |r(A)| \quad (2.2)$$

Esempi

ES1: selezione dei pazienti con residenza a Torino

$$\sigma_{Residenza='TO'}(pazienti) \quad (2.3)$$

COD	Cognome	Nome	Residenza	AnnoNascita
A102	Necchi	Luca	TO	1950
B543	Missoni	Nadia	TO	1950

ES2: selezione dei pazienti residenti a Torino o a Vercelli

$$\sigma_{Residenza='TO' \wedge Residenza='VC'}(pazienti) \quad (2.4)$$

COD	Cognome	Nome	Residenza	AnnoNascita
A102	Necchi	Luca	TO	1950
B543	Missoni	Nadia	TO	1950
B444	Missoni	Luigi	VC	2000

ES3: selezione dei pazienti non residenti a Torino

$$\sigma_{\neg Residenza='TO'}(pazienti) \quad (2.5)$$

COD	Cognome	Nome	Residenza	AnnoNascita
B372	Rossigni	Piero	NO	1940
B444	Missoni	Luigi	VC	2000
S555	Rossetti	Gino	AT	2010

2.2 Operatore proiezione

Data una relazione $r(A)$ e un insieme di attributi A_i, \dots, A_k tutti appartenenti ad A , l'operatore di proiezione

$$\Pi_{A_i, \dots, A_k}(r(A)) \quad (2.6)$$

produce come risultato una relazione avente:

- Schema: $\{A_i, \dots, A_k\}$
- Istanza: tutte le tuple della relazione argomento, ma solo rispetto ai campi A_i, \dots, A_k

2.2.1 Cardinalità della proiezione

A prima vista sembrerebbe che la cardinalità della proiezione sia uguale alla cardinalità della relazione argomento, ovvero

$$|\Pi_{A_i, \dots, A_k}(r(A))| = |r(A)| \quad (2.7)$$

Invece la cardinalità della proiezione è data da:

$$|\Pi_{A_i, \dots, A_k}(r(A))| \leq |r(A)| \quad (2.8)$$

Per capire il perché, vedere l'ES2.

Se, però, gli attributi proiettati A_i, \dots, A_k formano una **superchiave** della relazione argomento, allora:

$$|\Pi_{A_i, \dots, A_k}(r(A))| = |r(A)| \quad (2.9)$$

Esempi

ES1: proiezione della relazione *pazienti* sull'attributo *COD* e sull'attributo *Cognome*:

$$\Pi_{COD, Cognome}(pazienti) \quad (2.10)$$

COD	Cognome
A102	Necchi
B372	Rossigni
B543	Missoni
B444	Missoni
S555	Rossetti

ES2: proiezione della relazione *pazienti* sull'attributo *Cognome*:

Cognome
Necchi
Rossigni
Missoni
Missoni
Rossetti

Questa non è più una relazione valida nel modello relazionale teorico di Codd (non è più un insieme).

Dunque, il risultato è un'istanza di relazione senza ripetizioni:

Cognome
Necchi
Rossigni
Missoni
Rossetti

2.3 Composizione di operatori

L'algebra relazionale è **composizionale**, ovvero si possono costruire espressioni di algebra relazionale componendo insieme operatori.

Bisogna fare attenzione a che l'espressione algebrica sia sintatticamente corretta, verificando che gli operatori siano consistenti e coerenti con gli argomenti

Esempio

ES1: Elencare codice e nome dei pazienti residenti a Torino o a Vercelli.

1. Selezione dei pazienti residenti a Torino o Vercelli.

$$\sigma_{Residenza='To' \vee Residenza='VC'}(pazienti) \quad (2.11)$$

2. La relazione virtuale prodotta dalla selezione può essere usata come argomento per la proiezione.

$$\Pi_{COD, Nome}(\sigma_{Residenza='To' \vee Residenza='VC'}(pazienti)) \quad (2.12)$$

Il risultato è una relazione che ha come schema quello della proiezione, e come istanza solo le tuple che soddisfano il predicato della selezione.

ES2: La relazione

$$\sigma_{Residenza='To' \vee Residenza='VC'}(\Pi_{COD, Nome}(pazienti)) \quad (2.13)$$

diventa sintatticamente scorretta perché la relazione virtuale prodotta da $\Pi_{COD, Nome}(pazienti)$ non contiene l'attributo *Residenza*.

2.4 Operatori insiemistici: unione e differenza

Per definizione, gli operatori insiemistici richiedono che gli schemi delle relazioni argomento siano identici.

Il risultato dell'operatore insiemistico sulle relazioni argomento $r_1(A)$ e $r_2(A)$ è una relazione che ha come schema lo stesso schema A delle relazioni argomento e come istanza:

- Unione $r_1 \cup r_2$: l'unione delle tuple di r_1 e r_2
- Differenza $r_1 - r_2$: tutte le tuple di r_1 che non sono contenute in r_2

2.4.1 Cardinalità dell'unione

La cardinalità dell'unione $r_1(A) \cup r_2(A)$ è data da:

$$0 \leq |r_1(A) \cup r_2(A)| \leq |r_1(A)| + |r_2(A)| \quad (2.14)$$

La cardinalità può essere minore della somma delle cardinalità delle due relazioni se ci sono delle tuple ripetute.

2.4.2 Cardinalità della differenza

La cardinalità della differenza $r_1(A) - r_2(A)$ è:

$$0 \leq |r_1(A) - r_2(A)| \leq |r_1(A)| \quad (2.15)$$

Esempi

ES1: elencare cognomi e nomi di tutte le persone coinvolte nel database "Ricoveri Ospedalieri" (quindi, medici e pazienti):

$$\Pi_{Cognome, Nome}(pazienti) \cup \Pi_{Cognome, Nome}(medici) \quad (2.16)$$

Cognome	Nome
Necchi	Lucca
Rossigni	Piero
Missoni	Nadia
Missoni	Luigi
Rossetti	Gino
Neri	Piero
Bisi	Mario
Bargio	Sergio
Belli	Nicola
Mizzi	Nicola
Monti	Mario

ES2: elencare le province di residenza dei medici in cui non risiede alcun paziente

$$\Pi_{Residenza}(medici) - \Pi_{Residenza}(pazienti) \quad (2.17)$$

Residenza
NO

2.5 Operatore intersezione

Esattamente come gli altri due operatori insiemistici, l'operatore intersezione è definito su relazioni aventi lo stesso schema.

Il risultato dell'intersezione $r_1(A) \cap r_2(A)$ è una relazione che ha come schema lo stesso schema A delle relazioni argomento e istanza tutte le tuple di r_1 contenute anche in r_2 .

L'operatore di intersezione insiemistica può essere reso con l'operatore di differenza insiemistica:

$$r_1(A) \cap r_2(A) = r_1(A) - (r_1(A) - r_2(A)) \quad (2.18)$$

2.5.1 Cardinalità dell'intersezione

La cardinalità dell'intersezione soddisfa la seguente relazione:

$$0 \leq |r_1(A) \cap r_2(A)| \leq \min |r_1(A)|, |r_2(A)| \quad (2.19)$$

2.6 Operatore ridenominazione

Data una relazione $r(A)$ con $A = \{A_1, \dots, A_i, \dots, A_k, \dots, A_n\}$ il risultato della ridenominazione

$$\rho_{B_i, \dots, B_k \leftarrow A_i, \dots, A_k}(r) \quad (2.20)$$

è una relazione virtuale, che non modifica la relazione della base dati $r'(A')$ avente:

- Schema:

$$A' = \{A_1, \dots, A_i, \dots, A_k, \dots, A_n\} \quad (2.21)$$

- Istanza:

$$r' = r(\text{stesse tuple}) \quad (2.22)$$

Esempio

ES1: Elencare i medici che non sono primari:

$$\Pi_{MATR}(medici) - \rho_{MATR \leftarrow Primario}(\Pi_{Primario}(reparti)) \quad (2.23)$$

2.7 Uso corretto degli operatori insiemistici

Nell'utilizzo degli operatori insiemistici abbinati all'operatore di ridenominazione, bisogna assicurarsi sempre che gli attributi delle relazioni siano definiti su domini omogenei (coerenti).

ES1: i seguenti sono insiemi con domini omogenei:

$$\Pi_{MATR}(medici) - \rho_{MATR \leftarrow \text{Primario}}(\Pi_{\text{Primario}}(reparti)) \quad (2.24)$$

ES2: i seguenti sono insiemi con domini non omogenei (anche se gli attributi sono dello stesso tipo):

$$\Pi_{Nome}(medici) - \rho_{Nome \leftarrow Nome-Rep}(\Pi_{Nome-Rep}(pazienti)) \quad (2.25)$$

2.8 Ridenominazione schema

Data una relazione r con schema:

$$R(A) \quad A = \{A_1, \dots, A_i, \dots, A_k, \dots, A_n\} \quad (2.26)$$

Il risultato della ridenominazione dello schema

$$\rho_{R'(B_i, \dots, B_k) \leftarrow R(A_i, \dots, A_k)}(r) \quad (2.27)$$

è una relazione virtuale (che non modifica la relazione della base dati) con schema:

$$R'(A_1, \dots, B_i, \dots, B_k, \dots, A_n) \quad (2.28)$$

ES:

$$\begin{aligned} & \rho_{PUTENTI(CF, Provincia) \leftarrow PAZIENTI(COD, Residenza)}(pazienti) \\ & UTENTI(CF, Cognome, Nome, Provincia, AnnoNascita) \end{aligned} \quad (2.29)$$

2.9 Prodotto cartesiano

Date due relazioni $r_1(A)$ e $r_2(B)$, con $A \cap B = \emptyset$ (i due schemi non hanno attributi in comune), il prodotto cartesiano:

$$r_1(A) \times r_2(B) \quad (2.30)$$

Produce come risultato una relazione r' avente come risultato:

- Schema: R' composto dall'unione degli schemi $A \cup B$
- Istanza: giustapposizione (combinazione) di tutte le tuple di $r_1(A)$ con tutte le tuple di $r_2(B)$

A differenza della sua controparte matematica, nel modello di Codd il prodotto cartesiano gode della proprietà commutativa:

$$r_1(A) \times r_2(B) = r_2(A) \times r_1(B) \quad (2.31)$$

Infatti, nella relazione di Codd:

- Nello schema di una relazione non è importante l'ordine degli attributi.
- Nell'istanza di una relazione non è importante l'ordine delle tuple.

Esempi**ES1:**

$$ABC \times DE \quad (2.32)$$

A	B	C	D	E
a1	b1	c1	d1	e1
a1	b1	c1	d2	e2
a1	b1	c1	d3	e3
a2	b2	c2	d1	e1
a2	b2	c2	d2	e2
a2	b2	c2	d3	e3

ES2:

$$DE \times ABC \quad (2.33)$$

D	E	A	B	C
d1	e1	a1	b1	c1
d1	e1	a2	b2	c2
d2	e2	a1	b1	c1
d2	e2	a2	b2	c2
d3	e3	a1	b1	c1
d3	e3	a2	b2	c2

2.9.1 Cardinalità del prodotto cartesiano

La cardinalità del prodotto cartesiano $r_1(A) \times r_2(B)$ rispetta la seguente relazione:

$$0 \leq |r_1(A) \times r_2(B)| = |r_1(A)| \cdot |r_2(B)| \quad (2.34)$$

Infatti:

- r_1 e r_2 sono insiemi di tuple, quindi le tuple sono tutte disgiunte
- giustapponendo tuple disgiunte con le altre tuple disgiunte ottengo una relazione di tuple disgiunte

2.10 Operatore \bowtie_θ

L'operatore di \bowtie_θ serve a costruire informazioni estratte da più relazioni e a mettere in correlazione informazioni di una relazione con informazioni di un'altra relazione.

Prendendo in input:

- Due relazioni $r_1(A)$ e $r_2(B)$ con $A \cap B = \emptyset$

- Una condizione (predicato) di join θ da intendersi come una congiunzione di confronti tra attributi del tipo $A_i \phi B_j$, dove ϕ è un operatore di confronto

Il θ -join è definito come:

$$r_1(A) \bowtie_{\theta} r_2(B) = \sigma_{\theta}(r_1(A) \times r_2(B)) \quad (2.35)$$

Esempi

ES1: Elencare tutte le informazioni sui pazienti ricoverati.

$$\begin{aligned} &ricoveri \bowtie_{PAZ=COD} pazienti \\ &\sigma_{PAZ=COD}(ricoverati \times pazienti) \end{aligned} \quad (2.36)$$

Il prodotto cartesiano produce tutte le combinazioni di tutte le tuple di *ricoveri* con tutte le tuple di *pazienti*.

A questo punto selezioniamo soltanto quelle che soddisfano la condizione di join: $PAZ = COD$.

ES2: Elencare le informazioni dei primari di ogni reparto.

$$reparti \bowtie_{Primario=MATR} medici \quad (2.37)$$

ES3: Elencare i reparti con le informazioni del primario solo nel caso in cui il primario afferisce al reparto che dirige.

$$reparti \bowtie_{(Primario=MATR) \wedge (COD=Reparto)} medici \quad (2.38)$$

ES4: Elenco dei reparti abbinati ai dati dei rispettivi primari solo nel caso in cui il primario non vi afferisca.

$$reparti \bowtie_{(Primario=MATR) \wedge (COD \neq Reparto)} medici \quad (2.39)$$

2.10.1 Cardinalità del \bowtie_{θ}

La cardinalità di del \bowtie_{θ} si calcola sfruttando la cardinalità della selezione, e soddisfa la seguente relazione:

$$0 \leq |\sigma_{\theta}(r_1(A) \times r_2(B))| \leq |r_1(A) \times r_2(B)| \quad (2.40)$$

Sappiamo, però, che $|r_1(A) \times r_2(B)| = |r_1(A)| \cdot |r_2(B)|$, quindi:

$$0 \leq |r_1(A) \times r_2(B)| \leq |r_1(A)| \cdot |r_2(B)| \quad (2.41)$$

La cardinalità del θ -join ha un range molto ampio.

2.10.2 Join con schemi non disgiunti

Quando effettuiamo il join tra schemi non disgiunti (che hanno quindi degli attributi in comune), dobbiamo rinominare gli attributi che causano ambiguità almeno in uno dei due schemi.

ES1: elencare i pazienti abbinati ai medici che risiedono nella stessa città.

$$\begin{aligned} \rho_{CognomeM, NomeM, ResidenzaM} \leftarrow & Cognome, Nome, Residenza(pazienti) \\ & \bowtie_{ResidenzaP=ResidenzaM} \\ \rho_{CognomeP, NomeP, ResidenzaP} \leftarrow & Cognome, Nome, Residenza(medici) \end{aligned} \quad (2.42)$$

Esempio

Quali medici hanno avuto in cura il paziente Missoni Luigi?

1. Per sapere dov'è stato ricoverato ho bisogno di combinare i dati della relazione pazienti con la relazione ricoveri

$$\sigma_{Cognome='Missoni' \wedge Nome='Luigi'}(pazienti) \bowtie_{COD=PAZ} (ricoveri) \quad (2.43)$$

2. Per conoscere i medici afferenti al reparto ho bisogno della relazione medici, che però, ha molti attributi omonimi ad attributi della relazione virtuale prodotta al punto precedente
3. Ridenomino gli attributi di medici

$$\rho_{CM, NM, RM, Rep} \leftarrow Cognome, Nome, Residenza, Reparto(medici) \quad (2.44)$$

4. Metto in join le due espressioni:

$$\begin{aligned} & \bowtie_{Reparto=Rep} \\ \rho_{CM, NM, RM, Rep} \leftarrow & Cognome, Nome, Residenza, Reparto(medici) \end{aligned} \quad (2.45)$$

5. Infine proietto sui dati che mi interessano:

$$\begin{aligned} & \Pi_{COD, AnnoNascita, CM, NM} \\ & [\sigma_{Cognome='Missoni' \wedge Nome='Luigi'}(pazienti) \bowtie_{COD=PAZ} (ricoveri) \\ & \quad \bowtie_{Reparto=Rep} \\ & \quad \rho_{CM, NM, RM, Rep} \leftarrow Cognome, Nome, Residenza, Reparto(medici)] \end{aligned} \quad (2.46)$$

2.10.3 Semplificazione della ridenominazione

L'operatore di ridenominazione appesantisce la lettura delle espressioni algebriche. Quando ho attributi con lo stesso nome su relazioni diverse, posso considerare il nome dell'attributo specificando la tavola a cui appartiene, usando la **dot-notation**.

Facendo riferimento all'esempio precedente, l'interrogazione diventa:

$$\begin{aligned} & \Pi_{COD, AnnoNascita, medici.Cognome, medici.Nome} \\ & [\sigma_{Cognome='Missoni' \wedge Nome='Luigi'}(pazienti) \bowtie_{COD=PAZ} (ricoveri)] \\ & \bowtie_{Reparto=Rep} (medici) \end{aligned} \quad (2.47)$$

2.11 Operatore Natural-Join

Date due relazioni $r_1(A)$ e $r_2(B)$, con $A = \{\mathbf{X}, Y\}$ e $B = \{\mathbf{X}, Z\}$ (X , Y e Z sono insiemi disgiunti di attributi) e $Y \cap Z = \emptyset$.

Il natural-join è definito come

$$\begin{aligned} r_1(A) \bowtie r_2(B) = \\ \Pi_{X,Y,Z}(r_1 \bowtie_{\theta} \rho_{X' \leftarrow X}(r_2)) \end{aligned} \quad (2.48)$$

dove $\theta = (X_i = X'_i) \wedge \dots \wedge (X_k = X'_k)$

Informalmente: *il natural join tra due relazioni che hanno almeno un attributo in comune ha come condizione (sottintesa) l'uguaglianza tra gli attributi che hanno lo stesso nome.*

2.12 Operatore Semi-Join

Date due relazioni $r_1(A)$ e $r_2(B)$ definiamo il semi-join come:

$$\begin{aligned} r_1(A) \ltimes_{\theta} r_2(B) = \\ \Pi_A(r_1(A) \bowtie_{\theta} r_2(B)) \end{aligned} \quad (2.49)$$

L'operatore di semi-join funziona da selettore sulla prima tabella sfruttando la seconda.

Esempi

ES1: elencare tutti i dati dei primari.

Per risolvere l'interrogazione selezioniamo solo i medici che sono anche primari:

$$medici \ltimes_{MATR=Primario} reparti \quad (2.50)$$

ES2: elencare tutti i dati dei pazienti ricoverati in chirurgia:

$$\begin{aligned} & \text{pazienti} \bowtie_{\text{COD}=\text{PAZ}} \\ & (\text{ricoveri} \bowtie_{\text{Reparto}=\text{reparti.COD}} \\ & (\sigma_{\text{Nome-Rep}=\text{'Chirurgia'}}(\text{reparti}))) \end{aligned} \quad (2.51)$$

2.12.1 Cardinalità del semi-join

La cardinalità del semi-join è uguale alla cardinalità della proiezione sullo schema A , quindi al massimo alla cardinalità della relazione costruita su A

$$\begin{aligned} |r_1(A) \bowtie_{\theta} r_2(B)| &= |\Pi_A((r_1(A) \bowtie_{\theta} r_2(B)))| \\ 0 &\leq |\Pi_A((r_1(A) \bowtie_{\theta} r_2(B)))| \leq |r_1(A)| \\ 0 &\leq |r_1(A) \bowtie_{\theta} r_2(B)| \leq |r_1(A)| \end{aligned} \quad (2.52)$$

2.13 Join esterni

Chiamiamo L la tabella a sinistra del join e R la tabella a destra.

2.13.1 Left Join

Il left join viene indicato con $L \bowtie R$, ed consiste nel seguente procedimento:

1. Vengono prese in considerazione tutte le tuple della tavola L , nessuna escluse.
2. Si verifica se le tuple della tabella L sono in join con tuple della tabella R .
3. Qualora una tupla di L non faccia join con nessuna tupla della tavola R , si inseriscono valori nulli in corrispondenza degli attributi della seconda tavola.

2.13.2 Right Join

Il right join viene indicato con $L \bowtie R$, ed consiste nel seguente procedimento:

1. Vengono prese in considerazione tutte le tuple della tavola R , nessuna escluse.
2. Si verifica se le tuple della tabella R sono in join con tuple della tabella L .
3. Qualora una tupla di agenzie non faccia join con nessuna tupla della tavola L , si inseriscono valori nulli in corrispondenza degli attributi della prima tavola.

2.13.3 Full Join

Il full join è l'unione del left join e del right join:

$$L \bowtie R = (L \bowtie R) \cup (L \bowtie R) \quad (2.53)$$

Capitolo 3

Tipologie di interrogazioni

3.1 Interrogazioni con negazione

La negazione all'interno di un'interrogazione può essere di due tipi: essenziale o non essenziale, a seconda della possibilità o meno di riscrivere l'interrogazione senza usare la negazione.

ES1: l'interrogazione *"elencare i pazienti non residenti a Torino"* può essere riformulata come *"elencare i pazienti con residenza diversa da Torino"*. Questa è interrogazione con negazione non essenziale.

ES2: l'interrogazione *"elencare i medici non primari"* non può essere riformulata senza utilizzare la negazione. Quindi, questa interrogazione contiene una negazione non essenziale.

3.1.1 Negazione essenziale

Lo schema generale di risoluzione di un'interrogazione con negazione essenziale consta dei passi seguenti:

1. Definire l'universo del discorso U
2. Rispondere alla domanda in forma positiva P
3. Trovo la risposta R all'interrogazione usando l'operatore differenza: $R = U - P$

Ovviamente, per poter utilizzare la differenza, gli schemi di U e P devono essere uguali (e, ovviamente, compatibili).

3.1.2 Esempi

ES1: elencare i medici non primari

$$\begin{aligned}
 U &= \text{medici} \\
 P &= \Pi_{MATR, Cognome, Nome, Residenza, Reparto}(\text{medici} \bowtie_{MATR=Primario} \text{reparti}) \\
 R &= \text{medici} - \Pi_{MATR, Cognome, Nome, Residenza, Reparto}(\text{medici} \bowtie_{MATR=Primario} \text{reparti})
 \end{aligned}
 \tag{3.1}$$

ES2:

$$\begin{aligned}
 U &= \Pi_{COD, Cognome, Nome, Residenza}(\text{pazienti}) \\
 P &= \Pi_{COD, \text{pazienti.Cognome}, \text{pazienti.Nome}, \text{pazienti.Residenza}} \\
 &\quad (\text{pazienti} \bowtie_{\text{pazienti.Residenza}=\text{medici.Residenza}} \text{medici}) \\
 R &= \Pi_{COD, Cognome, Nome, Residenza}(\text{pazienti}) - \\
 &\quad \Pi_{COD, \text{pazienti.Cognome}, \text{pazienti.Nome}, \text{pazienti.Residenza}} \\
 &\quad (\text{pazienti} \bowtie_{\text{pazienti.Residenza}=\text{medici.Residenza}} \text{medici})
 \end{aligned}
 \tag{3.2}$$

3.2 Interrogazioni con quantificazione universale

Tutti gli esempi visti finora ricadono nella casistica della quantificazione esistenziale.

Esiste un medico non primario

Esiste un paziente non residente a Torino

Quando, invece troviamo le parole **"tutti"**, **"ogni"**, **"sempre"** nell'interrogazione, ci troviamo davanti a casi di quantificazione universale.

Possiamo risolvere questo tipo di interrogazioni riconducendoci alla quantificazione esistenziale, ricordandoci che la quantificazione universale è la negazione di una quantificazione esistenziale.

3.2.1 Operatore derivato quoziente

Dati due insiemi disgiunti di attributi A e B , l'operatore derivato quoziente si definisce come:

$$r(A, B) \div s(B) = \Pi_A(r) - \Pi_A((\Pi_A(r) \times s) - r) \tag{3.3}$$

Esempio1

Consideriamo le seguenti tabelle E = esami, e P = piano di studi.

MATR	Corso
2	Programmazione
3	Algebra
2	Basi di dati
3	Programmazione
2	Algebra

Corso
Programmazione
Basi di dati
Algebra

Elencare le matricole abbinate a tutti i corsi previsti dal piano di studi.

1: Individuo l'universo di riferimento

$$U = \Pi_{MATR}(e) \quad (3.4)$$

2: Analizzo tutte le combinazioni possibili di matricola con i corsi previsti:

$$\Pi_{MATR}(e) \times p \quad (3.5)$$

MATR	Corso
2	Programmazione
2	Basi di dati
2	Algebra
3	Programmazione
3	Basi di dati
3	Algebra

3: Trovo la differenza con e (hanno lo stesso schema)

$$(\Pi_{MATR}(e) \times p) - e \quad (3.6)$$

MATR	Corso
3	Basi di dati

4: Proiettando questo risultato su $MATR$, otteniamo l'elenco delle matricole che non hanno ancora superato qualche corso.

$$\Pi_{MATR}((\Pi_{MATR}(e) \times p) - e) \quad (3.7)$$

5: A questo punto possiamo riconsiderare l'universo di partenza, e sottrarre ad esso il risultato ottenuto:

$$\Pi_{MATR}(e) - \Pi_{MATR}((\Pi_{MATR}(e) \times p) - e) \quad (3.8)$$

Esempio 2

Elencare i pazienti che hanno subito ricovero in ogni reparto.

Espandiamo l'operatore quoziente:

$$\begin{aligned} \Pi_{PAZ, Reparto}(ricoveri) \div \Pi_{Reparto}(ricoveri) = \\ \frac{\Pi_{PAZ}(\Pi_{PAZ, Reparto}(ricoveri)) - \Pi_{PAZ}((\Pi_{PAZ}(\Pi_{PAZ, Reparto}(ricoveri)) \times \Pi_{Reparto}(ricoveri)) - \Pi_{PAZ, Reparto}(ricoveri))}{\Pi_{PAZ, Reparto}(ricoveri)} \end{aligned} \quad (3.9)$$

3.3 Semantica di Codd del valore nullo

Fin qui abbiamo ipotizzato l'assenza di valori nulli.

In realtà, però, il modello di Codd ne prevede l'esistenza. Il valore nullo è un valore vero e proprio, che appartiene a ogni dominio.

Il valore nullo può avere differenti significati:

- Informazione esistente, ma mancante.
- Informazione inesistente
- Assenza di informazione

La semantica di Codd del valore nullo è il comportamento frontale di una interrogazione a fronte di tuple che contengono valori nulli.

In particolare, ci riferiamo alla selezione σ_p con predicato p , che è una combinazione di confronti.

Il problema che si pone è: che valore di verità assume il confronto quando una tupla ha valore nullo in corrispondenza dell'attributo oggetto del confronto?

Quando il valore della tupla in corrispondenza dell'attributo A_i è nullo, il valore di verità del confronto di un attributo con la costante è **sconosciuto**.

3.4 Logica a tre valori

Codd propone di collocarsi in una logica a tre valori: *Falso*(F), *Vero*(V), *Sconosciuto*(U). Quindi, se $t.A_i$ è nullo, il valore di verità del confronto

$$t.A_i \theta \text{costante} \quad (3.10)$$

sarà U .

3.4.1 Operatori e logica a tre valori

Poiché i predicati θ sono spesso formati da una composizione di AND, OR, NOT, vediamo le **tavole di verità** dei tre operatori nella logica a tre valori:

AND	F	U	T	OR	F	U	T	NOT	
F	F	F	F	F	F	U	T	F	T
U	F	U	U	U	U	U	T	U	U
T	F	U	T	T	T	T	T	T	F

3.4.2 Espressioni e logica a tre valori

Date le tabelle di verità a tre valori, ogni tupla presa in considerazione genera un valore di verità dell'intera espressione che può essere F, T o U.

Ma quando eseguo una selezione e trovo una tupla che ha valore di verità sconosciuto, seleziono o no la tupla?

Considerando una tupla t , per sapere se questa è selezionata o meno, usiamo la **funzione di collasso** del predicato $p(t)$:

$p(t)$	$c(p(t))$
F	F
U	F
T	T

La tupla viene considerata solo se la funzione di collasso è T.

Tutte le volte che una tupla presenta valore nullo, in corrispondenza della selezione dà sempre valore di verità sconosciuto.

Per poter cercare una tupla che ha (o non ha) un valore nullo in corrispondenza dell'attributo A_i scrivo un predicato con le clausole:

$$\begin{aligned}
 A_i \text{ IS NULL} & \quad \text{se mi interessa il valore nullo} \\
 A_i \text{ IS NOT NULL} & \quad \text{se mi interessano i valori non nulli}
 \end{aligned}
 \tag{3.11}$$

ES1:

$$\sigma_{Residenza \text{ IS NULL}}(clienti) \tag{3.12}$$

COD	Residenza
200	NULL
400	NULL

ES2:

$$\sigma_{Residenza \text{ IS NOT NULL}}(clienti) \tag{3.13}$$

COD	Residenza
100	TO
300	TO

3.4.3 Limiti della logica a tre valori

La logica a tre valori viola il **principio di non contraddizione** per cui vale:

$$(p \wedge \neg p) = F \quad (3.14)$$

Se però, $p = U$, abbiamo

$$(p \wedge \neg p) = U \quad (3.15)$$

La logica a tre valori, inoltre, viola il **principio del terzo escluso**:

$$(p \vee \neg p) = T \quad (3.16)$$

Se però, $p = U$, abbiamo

$$(p \vee \neg p) = U \quad (3.17)$$

Esempio

Ipotizziamo che *pazienti* ammetta valore nullo in corrispondenza dell'attributo *AnnoNascita*, la selezione

$$\sigma_{AnnoNascita < 1980 \vee AnnoNascita \geq 1980}(pazienti) \quad (3.18)$$

contiene un predicato della forma $(p \vee \neg p)$, ma la funzione collasso dà falso, per cui:

$$|\sigma_{AnnoNascita < 1980 \vee AnnoNascita \geq 1980}(pazienti)| \leq |pazienti| \quad (3.19)$$

L'interrogazione

$$\sigma_{AnnoNascita < 1980 \vee AnnoNascita \geq 1980}(pazienti) \quad (3.20)$$

è equivalente a

$$\sigma_{AnnoNascita \text{ IS NOT NULL}}(pazienti) \quad (3.21)$$

3.4.4 Perché la logica a tre valori

Immaginiamo un confronto $\neg(A_i = \text{costante})$:

- Quando arriva una tupla che ha valore nullo in corrispondenza di A_i l'uguaglianza con la costante genera sconosciuto
- La funzione NOT applicata ad A_i genera sconosciuto
- La funzione collasso non fa selezionare la tupla

Se il confronto tra un valore nullo e una costante avesse assunto il valore F, la negazione avrebbe resituito il valore T!

3.5 Proprietà degli operatori algebrici

Le proprietà degli operatori ci permettono di semplificare espressioni complesse o per determinare l'equivalenza tra due espressioni.

Inoltre, le proprietà sono effettivamente utilizzate dai DBMS per ottimizzare le interrogazioni.

3.5.1 Proprietà commutativa del prodotto cartesiano

Ricordando che lo schema risultato del prodotto cartesiano è l'unione dello schema, e che l'ordine degli attributi in una relazione di Codd non è rilevante, possiamo concludere che il prodotto cartesiano è commutativo, cioè:

$$r(A) \times s(B) = s(B) \times r(A) \quad (3.22)$$

3.5.2 Proprietà commutativa del θ -join

Ricordando che il prodotto cartesiano è commutativo e che possiamo esprimere il θ -join come una selezione sul prodotto cartesiano, concludiamo che il θ -join è commutativo:

$$r(A) \bowtie_{\theta} s(B) = s(B) \bowtie_{\theta} r(A) \quad (3.23)$$

3.5.3 Proprietà associativa del prodotto cartesiano

Consideriamo le due espressioni :

$$\begin{aligned} (r(A) \times s(B)) \times u(C) \\ r(A) \times (s(B) \times u(C)) \end{aligned} \quad (3.24)$$

Ricordando che:

- Lo schema di $r(A) \times s(B)$ è $A \cup B$
- Lo schema di $(r(A) \times s(B)) \times u(C)$ è $(A \cup B) \cup C$
- Lo schema di $(A \cup B) \cup C$ è uguale a $A \cup (B \cup C)$

Possiamo concludere, di conseguenza, che i due schemi coincidono, e che, quindi il prodotto cartesiano è associativo:

$$(r(A) \times s(B)) \times u(C) = r(A) \times (s(B) \times u(C)) \quad (3.25)$$

3.5.4 Proprietà associativa del θ -join

Ricordando che il prodotto cartesiano è associativo e che possiamo esprimere il θ -join come una selezione sul prodotto cartesiano, concludiamo che il θ -join è associativo:

$$(r(A) \bowtie_{\theta_1} s(B)) \bowtie_{\theta_2} u(C) = r(A) \bowtie_{\theta_1} (s(B) \bowtie_{\theta_2} u(C)) \quad (3.26)$$

3.5.5 Proprietà della selezione multipla

$$\sigma_{p \wedge q}(r(A)) = \sigma_p(\sigma_q(r(A))) \quad (3.27)$$

La selezione $p \wedge q$ sceglie le tuple che soddisfano sia il predicato p che il predicato q .

La selezione di p applicata alle tuple selezionate da q , seleziona le tuple che soddisfano contemporaneamente p e q .

3.5.6 Proprietà della sostituzione di operatori

$$\sigma_{p \wedge q}(r(A))(r(A)) = \sigma_p(r(A)) \cap \sigma_q(r(A)) \quad (3.28)$$

$$\sigma_{p \vee q}(r(A)) = \sigma_p(r(A)) \cup \sigma_q(r(A)) \quad (3.29)$$

$$\sigma_{p \wedge \neg q}(r(A)) = \sigma_p(r(A)) - \sigma_q(r(A)) \quad (3.30)$$

3.5.7 Proprietà distributiva della selezione rispetto alla proiezione

$$\sigma_p \Pi_X(r(A)) = \Pi_X \sigma_p(r(A)) \quad (3.31)$$

Vale la condizione per cui il predicato p è definito solo su attributi di X .

Notiamo che, se troviamo $\sigma_p \Pi_X(r(A))$ possiamo sempre riscrivere $\Pi_X \sigma_p(r(A))$, mentre il viceversa non è sempre valido.

Se troviamo $\Pi_X \sigma_p(r(A))$, per poter scrivere $\sigma_p \Pi_X(r(A))$ è necessario che il predicato p sia definito solo su attributi X .

ES1: l'espressione

$$\Pi_{COD, AnnoNascita} \sigma_{AnnoNascita > 1980}(pazienti) \quad (3.32)$$

Può essere riscritta come

$$\sigma_{AnnoNascita > 1980} \Pi_{COD, AnnoNascita}(pazienti) \quad (3.33)$$

ES2: l'espressione

$$\Pi_{COD} \sigma_{AnnoNascita > 1980}(pazienti) \quad (3.34)$$

Non può essere riscritta come

$$\sigma_{AnnoNascita > 1980} \Pi_{COD}(pazienti) \quad (3.35)$$

3.5.8 Proprietà distributiva della selezione rispetto al prodotto cartesiano

$$\sigma_p(r(A) \times s(B)) \quad (3.36)$$

Quando p coinvolge sia attributi di A che attributi di B non c'è nessuna possibilità di applicare proprietà distributive.

Se invece p coinvolge solo attributi contenuti nello schema di una delle due relazioni (ad esempio A) è possibile scrivere:

$$\sigma_p(r(A)) \times s(B) \quad (3.37)$$

Una volta applicato il prodotto cartesiano, applichiamo la selezione (per ipotesi) solo alle parti A delle giustapposizioni.

Selezionare le tuple giustapposte prendendo solo quelle che soddisfano p è equivalente a selezionare le tuple da r che soddisfano p e giustapparle in seguito alle tuple di s .

3.5.9 Proprietà distributiva della selezione rispetto al join

Ricordando che il prodotto cartesiano che possiamo esprimere il θ -join come una selezione sul prodotto cartesiano, concludiamo che possiamo utilizzare la proprietà distributiva della selezione rispetto al join.

$$\sigma_p(r(A) \bowtie_{\theta} s(B)) = \sigma_p(r(A)) \bowtie_{\theta} s(B) \quad (3.38)$$

3.5.10 Proprietà distributiva della selezione rispetto all'unione e differenza

$$\sigma_p(r(A) \cup s(A)) = \sigma_p(r(A)) \cup \sigma_p(s(A)) \quad (3.39)$$

$$\sigma_p(r(A) - s(A)) = \sigma_p(r(A)) - \sigma_p(s(A)) \quad (3.40)$$

$$\sigma_p(r(A) \cap s(A)) = \sigma_p(r(A)) \cap \sigma_p(s(A)) \quad (3.41)$$

3.5.11 Proprietà della proiezione multipla

Supponiamo X, Y sottoinsiemi di A :

$$\Pi_X \Pi_{X,Y} r(A) = \Pi_X r(A) \quad (3.42)$$

3.5.12 Proprietà distributiva della proiezione rispetto al prodotto cartesiano

Siano $X_A \subseteq A$ e $X_B \subseteq B$, ovvero:

$$\begin{aligned} X_A &= X \cap A \\ X_B &= X \cap B \end{aligned} \tag{3.43}$$

Vale la proprietà distributiva della proiezione rispetto al prodotto cartesiano, cioè:

$$\Pi_X(r(A) \times s(B)) = \Pi_{X_A}(r(A)) \times \Pi_{X_B}(r(B)) \tag{3.44}$$

3.5.13 Proprietà distributiva della proiezione rispetto al join

$$\Pi_X(r(A) \bowtie_{\theta} s(B)) = \Pi_{X_A}(r(A)) \bowtie_{\theta} \Pi_{X_B}(r(B)) \tag{3.45}$$

Siano $X_A \subseteq A$ e $X_B \subseteq B$, ovvero:

$$\begin{aligned} X_A &= X \cap A \\ X_B &= X \cap B \end{aligned} \tag{3.46}$$

Inoltre, è necessario che gli attributi coinvolti nel predicato θ siano tutti contenuti in $X_A \cup X_B$

3.5.14 Proprietà distributiva della proiezione rispetto all'unione

$$\Pi_X(r(A) \cup s(A)) = \Pi_X(r(A)) \cup \Pi_X(s(B)) \tag{3.47}$$

3.5.15 Altre proprietà

Le proprietà distributive della proiezione rispetto alla differenza e all'intersezione non esistono.

Capitolo 4

Calcolo relazionale

L'algebra relazionale è un linguaggio di tipo **procedurale**: l'utente indica le operazioni da compiere per arrivare al risultato e il DBMS sceglie poi la strategia ottimale.

Il calcolo relazionale, invece, segue un approccio **dichiarativo**, fondato sulla logica: esprime infatti le caratteristiche di risultato, senza specificare il procedimento utilizzato per ottenerlo.

4.1 Calcolo su tuple con dichiarazione di range

L'interrogazione è composta da tre parti:

$$\{T|L|F\} \quad (4.1)$$

- **T (Target)**: la lista di informazioni che voglio in uscita. Possiamo usare una delle seguenti sintassi:

$$\begin{aligned} & \text{variabile.Attributo1, variabile.Attributo2, ...} \\ & \text{variabile.(Attributo1, Attributo2)} \\ & \text{variabile.* (visualizza tutti gli attributi)} \\ & \text{Nome : variabile.Attributo} \end{aligned} \quad (4.2)$$

Tutte le variabili usate nella Target List devono essere dichiarate nella Range List.

- **L (Range List)**: introduzione di variabili abbinate a tavole di base, con la seguente sintassi:

$$\text{nomeVariabile(NomeTavola)} \quad (4.3)$$

- **F (Formula)**: predicato del primo ordine su predicati di base basati su confronti del tipo:

$$\begin{aligned} & x.A \theta \text{ costante} \\ & x.A_i \theta y.A_j \end{aligned} \quad (4.4)$$

Il predicato è costruito con i soliti operatori AND, OR, NOT, IMPLICAZIONE e in più, dai quantificatori \forall e \exists .

ES1: elencare i pazienti residenti a Torino

$$\begin{aligned} \mathbf{T} &: p.Nome, p.Cognome \\ \mathbf{L} &: p(PAZIENTI) \\ \mathbf{F} &: p.Residenza = 'TO' \end{aligned} \quad (4.5)$$

ES2: elencare i dati dei pazienti ricoverati nel reparto A (si vuole anche la data del ricovero)

$$\begin{aligned} \mathbf{T} &: p.Nome, p.Cognome, r.Inizio \\ \mathbf{L} &: p(PAZIENTI), r(RICOVERI) \\ \mathbf{F} &: p.COD = r.PAZ \wedge Reparto = 'A' \end{aligned} \quad (4.6)$$

4.1.1 Join nel calcolo relazionale

Nel calcolo relazionale, il join si costruisce semplicemente inserendo nella formula la condizione di confronto delle variabili (relazioni) coinvolte, preliminarmente dichiarate nella Range List.

4.1.2 Quantificazione universale ed esistenziale

Quando una variabile non è espressa nella Target List, possiamo riformulare l'interrogazione usando la quantificazione esistenziale o universale.

La sintassi utilizzata è la seguente:

$$\begin{aligned} &\exists \text{ variabile}(\text{Relazione})(\text{formula}) \\ &\forall \text{ variabile}(\text{Relazione})(\text{formula}) \end{aligned} \quad (4.7)$$

La formula è un predicato del primo ordine che può contenere sia variabili libere che quantificate, tuttavia le uniche variabili libere presenti nella formula devono essere dichiarate nella Range List.

ES1: elencare i dati dei pazienti ricoverati nel reparto A (non è richiesta la data di inizio ricovero):

$$\begin{aligned} \mathbf{T} &: p.Nome, p.Cognome \\ \mathbf{L} &: p(PAZIENTI) \\ \mathbf{F} &: \exists r(RICOVERI)(p.COD = r.PAZ \wedge Reparto = 'A') \end{aligned} \quad (4.8)$$

ES2: elencare i medici curanti del paziente $A102$.

$$\begin{aligned} \mathbf{T} &: m.Nome, m.Cognome \\ \mathbf{L} &: m(MEDICI) \\ \mathbf{F} &: \exists r(RICOVERI)(r.Reparto = m.Reparto \wedge r.PAZ = 'A102') \end{aligned} \quad (4.9)$$

ES3: elencare i medici curanti del paziente Rossini Piero.

$$\begin{aligned}
 \mathbf{T} &: m.Nome, m.Cognome \\
 \mathbf{L} &: m(MEDICI) \\
 \mathbf{F} &: \exists p(PAZIENTI) \\
 &(\exists r(RICOVERI)(p.COD = r.PAZ \wedge r.Reparto = m.Reparto) \\
 &\wedge p.Nome = 'Piero' \wedge p.Cognome = 'Rossini')
 \end{aligned} \tag{4.10}$$

ES4: elencare cognome e nome comune a pazienti e medici.

$$\begin{aligned}
 \mathbf{T} &: p.Nome, p.Cognome \\
 \mathbf{L} &: p(PAZIENTI) \\
 \mathbf{F} &: \exists m(MEDICI)(m.Nome = p.Nome \wedge m.Cognome = p.Cognome)
 \end{aligned} \tag{4.11}$$

ES5: elencare i cognomi e i nomi dei pazienti non medici.

È sufficiente anteporre il NOT alla quantificazione esistenziale dell'esercizio precedente.

$$\begin{aligned}
 \mathbf{T} &: p.Nome, p.Cognome \\
 \mathbf{L} &: p(PAZIENTI) \\
 \mathbf{F} &: \neg \exists m(MEDICI)(m.Nome = p.Nome \wedge m.Cognome = p.Cognome)
 \end{aligned} \tag{4.12}$$

ES6: elencare i pazienti con almeno un ricovero in ogni reparto.

$$\begin{aligned}
 \mathbf{T} &: p.Nome, p.Cognome \\
 \mathbf{L} &: p(PAZIENTI) \\
 \mathbf{F} &: \forall r(REPARTI) \exists r'(RICOVERI)(p.COD = r'.PAZ \wedge r.COD = r'.Reparto)
 \end{aligned} \tag{4.13}$$

ES7: elencare i pazienti ricoverati due o più volte.

$$\begin{aligned}
 \mathbf{T} &: p.Nome, p.Cognome \\
 \mathbf{L} &: p(PAZIENTI) \\
 \mathbf{F} &: \exists r'(RICOVERI)(\exists r''(RICOVERI)(p.COD = r'.PAZ \wedge p.COD = r''.PAZ \\
 &\wedge r'.Inizio \neq r''.Inizio))
 \end{aligned} \tag{4.14}$$

4.1.3 Negazione essenziale

Lo schema generale di soluzione di interrogazioni contenenti una negazione essenziale è

$$R = U - P \tag{4.15}$$

In calcolo relazionale, esprimiamo questo schema con il seguente pattern di soluzione:

$$\mathbf{F} = formulaU \wedge \neg formulaP \tag{4.16}$$

ES1: elencare i pazienti ricoverati una sola volta.

$$\begin{aligned}
 \mathbf{T} &: p.Nome, p.Cognome \\
 \mathbf{L} &: p(PAZIENTI) \\
 \mathbf{F} &: \exists r(RICOVERI)(p.COD = r.PAZ) \wedge \\
 &\quad \neg \exists r'(RICOVERI)(\exists r''(RICOVERI)(p.COD = r'.PAZ \wedge p.COD = r''.PAZ))
 \end{aligned}
 \tag{4.17}$$

ES2: elencare i medici non primari.

$$\begin{aligned}
 \mathbf{T} &: m.Nome, m.Cognome \\
 \mathbf{L} &: m(MEDICI) \\
 \mathbf{F} &: \neg \exists r(REPARTI)(r.Primario = m.MATR)
 \end{aligned}
 \tag{4.18}$$

In questo caso apparentemente manca l'universo del discorso, ma è implicitamente definito nella variabile libera m .

4.1.4 Prodotto cartesiano

Il prodotto cartesiano viene reso utilizzando la seguente sintassi:

$$\begin{aligned}
 \mathbf{T} &: x.*, y* \\
 \mathbf{L} &: x(R), y(S)
 \end{aligned}
 \tag{4.19}$$

4.1.5 Unione

L'unione insiemistica dell'algebra relazionale non è esprimibile con il calcolo relazionale con dichiarazione di range.

4.1.6 Limiti del calcolo relazionale

Il problema dell'inesprimibilità dell'unione insiemistica non riguarda tutti i linguaggi dichiarativi, ma soltanto il calcolo relazionale su tuple con dichiarazione di range.

Per superare questa limitazione, viene introdotto il **calcolo relazionale sui domini**.

Esempio

Consideriamo la seguente base dati:

$$\begin{aligned}
 S(\text{Studenti}) &= (\underline{MATR}, Nome, Indirizzo) \\
 E(\text{Esami}) &= (\underline{MATR}, Corso, Indirizzo) \\
 O(\text{Offerta Formativa}) &= (\underline{Corso}, \underline{Indirizzo})
 \end{aligned}
 \tag{4.20}$$

Elencare gli studenti che hanno superato tutti gli esami del loro indirizzo.

$$\begin{aligned}
 & \mathbf{T} : s.* \\
 & \mathbf{L} : s(S) \\
 & \mathbf{F} : \forall o(O)(o.Indirizzo = s.Indirizzo \\
 & \rightarrow \exists e(E)(e.MATR = s.MATR \wedge e.Corso = o.Corso \wedge e.Indirizzo = o.Indirizzo)) \\
 & \quad (4.21)
 \end{aligned}$$

- Con la quantificazione universale scorro tutti i corsi dell'offerta formativa.
- Con l'antecedente mi soffermo sui corsi dell'indirizzo dello studente.
- Con il conseguente verifico che lo studente abbia superato l'esame
- Se scorro corsi che non sono dell'indirizzo dello studente, l'antecedente è falso e l'implicazione rimane vera.

Esempio

Consideriamo la seguente base dati:

$$\begin{aligned}
 A(\text{Autori}) & : (\underline{COD}, \text{Cognome}, \text{Nome}) \\
 P(\text{Pubblicazioni}) & : (\underline{ID}, \text{Titolo}, \text{Anno}) \\
 F(\text{Firma}) & : (\underline{A}, \underline{P})
 \end{aligned} \quad (4.22)$$

Trovare gli autori che hanno pubblicato solo in collaborazione (autori che non hanno mai pubblicato una pubblicazione di cui sono gli unici firmatari)

$$\begin{aligned}
 & \mathbf{T} : a.* \\
 & \mathbf{L} : a(A) \\
 & \mathbf{F} : \forall f(F)(f.A = a.COD \rightarrow \exists f'(F)(f'.P = f.P \wedge f'.A \neq a.COD)) \\
 & \quad (4.23)
 \end{aligned}$$

Capitolo 5

Normalizzazione

5.1 Ridondanze e anomalie

Per introdurre i primi concetti, utilizziamo un esempio.

Consideriamo una relazione (*Impiegato, Stipendio, Progetto, Bilancio, Funzione*), avente le seguente proprietà:

- Lo stipendio di ciascun impiegato è unico ed è funzione del solo impiegato, indipendentemente dai progetti a cui partecipa.
- Il bilancio di ciascun progetto è unico e dipende dal solo progetto, indipendentemente dagli impiegati che vi partecipano.

Questa relazione presenta alcune problematiche:

- Il valore dello stipendio di ciascun impiegato è ripetuto in tutte le tuple relative ad esso: si ha quindi una **ridondanza**.
- Se lo stipendio di un impiegato varia, è necessario andarne a modificare il valore in tutte le tuple corrispondenti, affinché la dipendenza continui a valere. Questo inconveniente, che comporta la necessità di effettuare più modifiche contemporaneamente, va sotto il nome di **anomalia di aggiornamento**.
- Se un impiegato interrompe la partecipazione a tutti i progetti senza lasciare l'azienda, e quindi tutte le corrispondenti tuple vengono eliminate, non è possibile conservare traccia del suo nome e del suo stipendio (a meno di ammettere valori nulli sull'unica chiave, il che è inammissibile), che potrebbero rimanere di interesse. Questo problema viene indicato come **anomalia di cancellazione**.
- Analogamente, se si hanno informazioni su un nuovo impiegato, non è possibile inserirle finché questi non viene assegnato a un progetto; in questo caso parliamo di **anomalia di inserimento**.

Una motivazione intuitiva della presenza di questi inconvenienti può essere la seguente: abbiamo usato un'unica relazione per rappresentare informazioni eterogenee.

Generalizzando, possiamo arrivare alle seguenti conclusioni, che evidenziano i difetti presentati da relazioni che riuniscono concetti fra loro disomogenei:

- È possibile che alcuni dati debbano essere ripetuti in diverse tuple, senza aggiungere in tal modo informazioni significative.
- Se alcune informazioni sono ripetute in modo ridondante, il relativo aggiornamento (concettualmente atomico) deve essere ripetuto per ciascuna occorrenza dei relativi dati.
Il fatto che i linguaggi di manipolazione, per esempio SQL, permettano aggiornamenti multipli risolve il problema solo dal punto di vista del programmatore ma non da quello del sistema, perché comunque le tuple della base di dati vanno aggiornate tutte e quindi si deve fisicamente accedere a ciascuna di esse.
- La cancellazione di una tupla, motivata dal fatto che non è più valido l'intero insieme di concetti da essa espressi, per esempio perché uno di essi non sussiste più, può comportare l'eliminazione di tutti i concetti in questione, cioè anche di quelli che conservano la loro validità.
- L'inserimento di informazioni relative a uno solo dei concetti di pertinenza per una relazione non è possibile se non esiste un intero insieme di concetti in grado di costruire una tupla completa (o almeno la sua chiave primaria).

5.2 Dipendenze funzionali

La **dipendenza funzionale** è un particolare vincolo di integrità per il modello relazionale, che descrive legami di tipo funzionale tra gli attributi di una relazione.

Data una relazione r su uno schema $R(X)$ e due sottoinsiemi di attributi non vuoti Y e Z di X , diremo che esiste su r una **dipendenza funzionale fra Y e Z** se, per ogni coppia di tuple t_1, t_2 di r aventi gli stessi valori sugli attributi Y , risulta che t_1 e t_2 hanno gli stessi valori anche sugli attributi Z .

Una dipendenza funzionale tra gli attributi Y e Z viene generalmente indicata con la notazione $Y \rightarrow Z$ e, come gli altri vincoli di integrità, viene associata a uno schema: una relazione su quello schema verrà considerata corretta se soddisfa tale dipendenza funzionale.

Ci sono alcune osservazioni da fare:

1. Se l'insieme Z è composto dagli attributi A_1, \dots, A_k , allora una relazione soddisfa $Y \rightarrow Z$ se e solo se esso soddisfa tutte le k dipendenze $Y \rightarrow A_1, \dots, Y \rightarrow A_k$.
Quindi, possiamo assumere che le dipendenze abbiano la forma $Y \rightarrow A$, in cui A è un singolo attributo.

2. Una dipendenza funzionale $Y \rightarrow A$ è **non banale** se A non compare tra gli attributi di Y .
3. Se prendiamo una chiave K di una relazione r , si può facilmente verificare che esiste una dipendenza funzionale tra K e ogni altro attributo dello schema di r .
Questo perché, per definizione stessa di vincolo di chiave, non possono esistere due tuple con gli stessi valori su K e quindi una dipendenza funzionale che ha K al primo membro sarà sempre soddisfatta.
4. Il vincolo di dipendenza funzionale **generalizza** il vincolo di chiave. Più precisamente, possiamo dire che una dipendenza funzionale $Y \rightarrow Z$ su uno schema $R(X)$ degenera nel vincolo di chiave se l'unione di Y e Z è pari X .

5.3 Teoria di Armstrong

La teoria di Armstrong ci permette di verificare le equivalenze senza ricorrere alla definizione di vincolo di dipendenza funzionale.

5.3.1 Assiomi della teoria di Armstrong

Riflessività

$$\text{Se } Y \subseteq X \text{ allora } X \rightarrow Y \quad (5.1)$$

Unione

$$\begin{aligned} \text{Se } X \rightarrow Y \text{ e } X \rightarrow Z \text{ allora } X \rightarrow YZ \\ \text{Dove } YZ = Y \cup Z \end{aligned} \quad (5.2)$$

Transitività

$$\text{Se } X \rightarrow Y \text{ e } Y \rightarrow Z \text{ allora } X \rightarrow Z \quad (5.3)$$

5.3.2 Vincolo di dipendenza funzionale

Il vincolo della dipendenza funzionale è un modello (in senso matematico) della teoria di Armstrong.

Un modello calza su una teoria quando gli assiomi della teoria sono rispettati dal modello.

La teoria di Armstrong, dal punto di vista del modello dei vincoli delle dipendenze funzionali è una teoria corretta: posso quindi applicare i teoremi della teoria di Armstrong al modello dei vincoli di dipendenza funzionale.

Teorema dell'espansione

Data una dipendenza funzionale $X \rightarrow Y$ e un insieme di attributi W , allora:

$$WX \rightarrow WY$$

Teorema di decomposizione

Data una dipendenza funzionale $X \rightarrow YZ$, allora

$$X \rightarrow Y \wedge X \rightarrow Z$$

Teorema di pseudo-transitività

Date $X \rightarrow Y$, $WY \rightarrow Z$, allora

$$WX \rightarrow Z$$

Teorema del prodotto

Date le dipendenze funzionali $X \rightarrow Y$ e $W \rightarrow Z$, allora

$$WX \rightarrow YZ$$

5.3.3 Attributi estranei

Abbiamo le seguenti dipendenze funzionali F :

$$\begin{aligned} ABCD &\rightarrow E \\ B &\rightarrow C \end{aligned} \tag{5.4}$$

Allora, C è un attributo estraneo e si può cancellare dalla prima dipendenza, che diventa $ABD \rightarrow E$.

5.4 Forma normale di Boyce e Codd**5.4.1 Definizione di forma normale di Boyce e Codd**

Possiamo introdurre delle proprietà, dette **forme normali**, definite con riferimento alle dipendenze funzionali, che sono soddisfatte *quando non ci sono anomalie*.

Le ridondanze e le anomalie sono causate dalle dipendenze funzionali $X \rightarrow A$ che permettono la presenza di più tuple tra loro uguali sugli attributi in X , cioè dalle dipendenze funzionali $X \rightarrow A$ tali che X non contiene una chiave.

Precisiamo questa considerazione per mezzo della più importante delle forme normali, detta di Boyce e Codd: una relazione r è **in forma normale di Boyce e Codd** se, per ogni dipendenza funzionale (non banale) $X \rightarrow A$ definita su di essa, X contiene una chiave K di r , cioè X è superchiave per r .

Anomalie e ridondanze non si presentano per relazioni in forma normale di Boyce e Codd, perché i concetti indipendenti sono separati, uno per relazione.

5.4.2 Decomposizione in forma normale di Boyce e Codd

Data una relazione che non soddisfa la forma normale di Boyce e Codd è possibile, in molti casi, sostituirla con due o più relazioni normalizzate attraverso un processo detto di **normalizzazione**.

Questo processo si fonda su un semplice criterio: se una relazione rappresenta più concetti indipendenti, allora va decomposta in relazioni più piccole, una per ogni concetto.

In molti casi pratici, la decomposizione può essere effettuata producendo tante relazioni quante sono le dipendenze funzionali definite (o meglio, le dipendenze funzionali con diverso primo membro).

5.5 Proprietà delle decomposizioni

5.5.1 Decomposizione senza perdita

Data una relazione r su un insieme di attributi X , se X_1 e X_2 sono due sottoinsiemi di X la cui unione sia pari a X stesso, allora il join delle due relazioni ottenute per proiezione da r su X_1 e X_2 , rispettivamente, è una relazione che contiene tutte le tuple di r , più eventualmente altre, che possiamo chiamare *spurie*.

Diciamo che r si **decompone senza perdita** su X_1 e X_2 se il join delle due proiezioni è uguale a r stessa (cioè non contiene tuple spurie).

È chiaramente desiderabile, anzi, è un requisito irrinunciabile, che una decomposizione effettuata a fini di normalizzazione sia senza perdita.

È possibile individuare una condizione che garantisce la decomposizione senza perdita di una relazione.

Sia r una relazione su X e siano X_1 e X_2 sottoinsiemi di X tali che $X_1 \cup X_2 = X$, e sia $X_0 = X_1 \cap X_2$. Allora, r si decompone senza perdita su X_1 e X_2 se soddisfa la dipendenza funzionale $X_0 \rightarrow X_1$ oppure la dipendenza funzionale $X_0 \rightarrow X_2$.

In altre parole, r si decompone senza perdita su due relazioni se l'insieme degli attributi comuni alle due relazioni è chiave per almeno una delle relazioni decomposte.

Questa condizione è sufficiente ma non necessaria per la decomposizione senza perdita: esistono, infatti, istanze di relazione che non soddisfano nessuna delle due dipendenze, ma al tempo stesso si decompongono senza perdita.

Al tempo stesso, la condizione in questione che tutte le istanze di relazione che soddisfano un dato insieme di dipendenze si decompongano senza perdita, e questo è un risultato utilizzabile in pratica: quando decomponiamo una relazione in due parti, se l'insieme degli attributi comuni è chiave per una delle due relazioni, allora possiamo essere certi che tutte le istanze della relazione si decompongono senza perdita.

5.5.2 Conservazione delle dipendenze

In ogni decomposizione, ciascuna delle dipendenze funzionali dello schema originario dovrebbe coinvolgere attributi che compaiono tutti insieme in uno degli schemi decomposti. In questo modo è possibile garantire, sullo schema decomposto, il soddisfacimento degli stessi vincoli il cui soddisfacimento degli stessi vincoli il cui soddisfacimento è garantito dallo schema originario.

Diremo che una decomposizione che soddisfa tale proprietà **conserva le dipendenze** dello schema originario.

5.5.3 Qualità delle decomposizioni

Possiamo affermare che le decomposizioni dovrebbero sempre soddisfare le proprietà di **decomposizione senza perdita** e **conservazione delle dipendenze**:

- La **decomposizione senza perdita** garantisce che le informazioni nella relazione originaria siano ricostruibili con precisione (cioè senza informazioni spurie) a partire da quelle rappresentate nelle relazioni decomposte.
In tal caso, interrogando le relazioni decomposte, otteniamo gli stessi risultati che otterremmo interrogando la relazione originaria.
- La **conservazione delle dipendenze** garantisce che le relazioni decomposte abbiano la stessa capacità della relazione originaria di rappresentare i vincoli di integrità (e cioè le proprietà della realtà di interesse) e quindi di rilevare aggiornamenti illeciti: a ogni aggiornamento lecito (rispettivamente illecito) sulla relazione originaria corrisponde un aggiornamento lecito (rispettivamente illecito) sulle relazioni decomposte.
Ovviamente sono possibili sulle relazioni decomposte ulteriori aggiornamenti, legati ai singoli concetti rappresentati in ciascuna di esse, che non hanno un corrispettivo sulla relazione originaria, senza però corrispondere a violazioni dei vincoli: si tratta degli aggiornamenti impossibili sulle relazioni non normalizzate a causa delle anomalie.

Di conseguenza, consideriamo accettabili, cioè di qualità sufficiente, solo le decomposizioni che soddisfano queste due proprietà.

Dato uno schema che violi una forma normale, l'attività di normalizzazione è quindi volta a ottenere una decomposizione che sia senza perdita, che conservi le dipendenze e contenga relazioni in forma normale.

5.6 Terza forma normale

5.6.1 Limitazioni della forma normale di Boyce e Codd

In alcuni casi, la forma normale di Boyce e Codd non è raggiungibile: esistono schemi che violano la forma normale di Boyce e Codd per i quali non esiste alcuna decomposizione

che conservi le dipendenze.

5.6.2 Definizione di terza forma normale

Per trattare i casi in cui la forma normale di Boyce e Codd non è raggiungibile, si ricorre a una forma normale meno restrittiva.

Diciamo che una relazione r è in **terza forma normale** se, per ogni dipendenza funzionale non banale $X \rightarrow A$ definita su di essa, almeno una delle seguenti condizioni è verificata:

- X contiene una chiave K di r
- A appartiene ad almeno una chiave di r

In sostanza, la terza forma normale è meno forte della forma normale di Boyce e Codd e quindi non offre le medesime garanzie di qualità per una relazione; ha però, rispetto a essa il vantaggio di essere sempre ottenibile.

È possibile dimostrare che una qualunque relazione che non soddisfa la terza forma normale è certamente decomponibile senza perdita e con conservazione delle dipendenze in relazioni in terza forma normale.

5.6.3 Decomposizione in terza forma normale

Per decomporre una relazione in terza forma normale, si può procedere come suggerito nel caso della forma normale di Boyce e Codd: una relazione che non soddisfa la terza forma normale si decompone in relazioni ottenute per proiezione sugli attributi corrispondenti alle dipendenze funzionali, con l'unica accortezza di mantenere sempre una relazione che contiene una chiave della relazione originaria.

Una decomposizione tesa a ottenere la terza forma normale produce nella maggior parte dei casi schemi in forma normale di Boyce e Codd. In particolare, si può dimostrare che, se una relazione ha solo una chiave, allora le due forme normali coincidono, cioè una relazione in terza forma normale è anche in forma normale di Boyce e Codd.

5.7 Teoria delle dipendenze e normalizzazione

Questa sezione mostra come i più importanti concetti appena discussi possano essere formalizzati, arrivando a un processo di normalizzazione realizzabile in modo algoritmico.

Data una relazione e un insieme di dipendenze funzionali definite su di essa, ci interessa generare una decomposizione della relazione che contenga solo relazioni in forma normale e soddisfi le qualità di decomposizione senza perdita e conservazione delle dipendenze.

Poiché, come abbiamo visto, questo obiettivo non è sempre raggiungibile per la forma normale di Boyce e Codd, lo perseguiremo per la terza forma normale.

In linea di massima, il procedimento è quello già illustrato informalmente: definire una

relazione per ciascun gruppo di dipendenze funzionali fra loro strettamente correlate. Il procedimento va formalizzato per definire bene l'insieme di dipendenze di interesse e completato con una verifica finale, che può portare a un passo aggiuntivo.

5.7.1 Implicazione di dipendenze funzionali

Diciamo che un insieme di dipendenze funzionali F **implica** un'altra dipendenza f se ogni relazione che soddisfa tutte le dipendenze in F soddisfa anche f .

Chiusura di un insieme

Dati F e f , come verifichiamo se F implica f ?

Per procedere definiamo un concetto molto utile. Siano dati uno schema di relazione $R(U)$ e un insieme di dipendenze funzionali F definite sugli attributi in U . Sia X un insieme di attributi contenuti in U (cioè $X \subseteq U$).

La **chiusura** di X rispetto ad F , indicata con X_F^+ , è l'insieme degli attributi che dipendono funzionalmente da X (esplicitamente o implicitamente):

$$X_F^+ = \{A : A \in U \wedge F \text{ implica } X \rightarrow A\} \quad (5.5)$$

Questo insieme può essere molto utile: se vogliamo vedere se $X \rightarrow A$ è implicata da F , basta vedere se A appartiene a X_F^+ , a patto di saper calcolare quest'ultimo.

In effetti, questa è una strada valida, in quanto esiste un algoritmo che ci permette di calcolare X_F^+ .

Input: un insieme X di attributi e un insieme F di dipendenze

Output: un insieme X_P di attributi

1. Inizializziamo X_P con l'insieme di input X
2. Esaminiamo le dipendenze in F : se esiste una dipendenza $Y \rightarrow A$ con $Y \subseteq X_P$ e $A \notin X_P$ allora aggiungiamo A a X_P
3. Ripetiamo il passo 2 fino al momento in cui non vi sono ulteriori attributi che possono essere aggiunti X_P

Il concetto di chiusura è utile anche per formalizzare il concetto di dipendenza funzionale e quello di chiave: un insieme di attributi K è chiave per uno schema di relazione $R(U)$ su cui è definito un insieme di dipendenze funzionali F se F implica $K \rightarrow U$. Di conseguenza, l'algoritmo appena mostrato può essere utilizzato per verificare se un insieme è chiave.

Esempio

Consideriamo la seguente relazione:

Impiegato	Stipendio	Progetto	Bilancio	Funzione
Rossi	20.000	Marte	2.000	Tecnico
Verdi	35.000	Giove	15.000	Progettista
Verdi	35.000	Venere	15.000	Progettista
Neri	55.000	Venere	15.000	Direttore
Neri	55.000	Giove	15.000	Consulente
Neri	55.000	Marte	2.000	Consulente
Mori	48.000	Marte	2.000	Direttore
Mori	48.000	Venere	15.000	Progettista
Bianchi	48.000	Venere	15.000	Progettista
Bianchi	48.000	Giove	15.000	Direttore

Indichiamo gli attributi di questo schema con le rispettive iniziali.

Possiamo verificare che gli attributi IP formano una chiave, in quanto:

$$\begin{aligned}
 IP^+ &= ISPBF \\
 I^+ &= IS \\
 P^+ &= PB
 \end{aligned}
 \tag{5.6}$$

La prima uguaglianza dimostra che IP è superchiave, le ultime due che nessun sottoinsieme proprio di IP è superchiave.

Il calcolo di IP^+ si esegue inizializzando l'insieme di lavoro a IP e aggiungendo (in qualunque ordine):

- S utilizzando $I \rightarrow S$
- B utilizzando $P \rightarrow B$
- F utilizzando $IP \rightarrow F$

Invece, I^+ si calcola partendo da I e aggiungendo solo S (utilizzando $I \rightarrow S$); le altre due dipendenze non sono utilizzabili, perché P non appartiene all'insieme di lavoro.

5.7.2 Coperture di insiemi di dipendenze funzionali

Può essere utile sostituire a un insieme di dipendenze funzionali un altro che specifichi, nella sostanza, le stesse proprietà, ma che sia più semplice da gestire.

Due insiemi di dipendenze funzionali F_1 e F_2 sono **equivalenti** se F_1 implica ciascuna dipendenza in F_2 e viceversa.

Se due insiemi sono equivalenti, diciamo anche che ognuno è **copertura** dell'altro.

Si può dimostrare che, dati due insiemi F_1 e F_2 equivalenti, una relazione soddisfa F_1 se e solo se essa soddisfa F_2 . Questa proprietà giustifica, quindi l'uso del termine **equivalenza** e la possibilità di utilizzare, dato un insieme di dipendenze, un altro, a esso equivalente, ma più semplice, per esempio con meno dipendenze o meno attributi.

Diciamo che un insieme F è:

- **Non ridondante:** se non esiste dipendenza $f \in F$ tale che $F - \{f\}$ implica f .
- **Ridotto:** se non è ridondante e non esiste un insieme F' equivalente a F ottenuto eliminando attributi dai primi membri di una o più dipendenze di F .

Avendo già visto come si verifica l'implicazione, possiamo dire che il calcolo di una copertura non ridondante e di una ridotta, dato un insieme di dipendenze è abbastanza semplice, in quanto entrambe le definizioni si basano sul concetto di implicazione.

Per **trovare una copertura non ridondante** è sufficiente esaminare ripetutamente le dipendenze dell'insieme dato, eliminando quelle implicate da altre, fermandosi quando non ve ne sono più; l'insieme rimasto è una copertura non ridondante di quello iniziale.

Per **trovare una copertura ridotta**, per un qualunque insieme di dipendenze funzionali, possiamo procedere in tre passi:

- Sostituiamo l'insieme dato con quello equivalente che ha tutti i secondi membri costituiti da singoli attributi
- Eliminiamo le dipendenze ridondanti
- Per ogni dipendenza verifichiamo se esistono attributi eliminabili dal primo membro

Esempio

Consideriamo alcuni insiemi di dipendenze:

$$\begin{aligned} F_1 &= \{A \rightarrow B, AB \rightarrow C, A \rightarrow C\} \\ F_2 &= \{A \rightarrow B, AB \rightarrow C\} \\ F_3 &= \{A \rightarrow B, A \rightarrow C\} \end{aligned} \tag{5.7}$$

Notiamo che:

- F_1 è ridondante, perché $\{A \rightarrow B, AB \rightarrow C\}$ implica $A \rightarrow C$: F_1 è equivalente a F_2
- F_2 è non ridondante ma non è ridotto, perché B può essere eliminato dal primo membro della seconda dipendenza: F_2 è equivalente a F_3
- F_3 è ridotto

5.7.3 Sintesi di schemi in terza forma normale

Dopo aver esaminato il concetto di copertura, possiamo mostrare come ottenere, in modo algoritmico una decomposizione in terza forma normale (che soddisfi le proprietà di conservazione delle dipendenze e decomposizione senza perdita) per un qualunque schema.

Ricordiamo la definizione di **terza forma normale**: uno schema di relazione $R(U)$ con il suo insieme di dipendenze funzionali F è in terza forma normale se, per ogni dipendenza funzionale non banale $X \rightarrow A \in F$, almeno una delle seguenti condizioni è verificata:

- X contiene una chiave K di r : cioè $X_F^+ = U$
- A è contenuto in almeno una chiave di r : esiste un insieme di attributi $K \subseteq U$ tale che $K_F^+ = U$ e $(K - A)_F^+ \subset U$

L'algoritmo per la decomposizione procede come segue:

1. Viene calcolata una copertura ridotta G di F
2. G viene partizionato in sottoinsiemi G_1, \dots, G_k tali che a ogni insieme appartengano dipendenze che hanno primi membri con la stessa chiusura.
Cioè: $X \rightarrow A$ e $Y \rightarrow B$ appartengono alla stessa partizione se e solo se $X_G^+ = Y_G^+$
3. Viene costruito un insieme \mathcal{U} di sottoinsiemi di U , uno per ciascuna partizione di dipendenze, con tutti gli attributi coinvolti nella partizione.
4. Se un elemento di \mathcal{U} è propriamente contenuto in un altro, allora esso viene eliminato da \mathcal{U} .
5. Viene costruito uno schema di basi dati con uno schema di relazione $R_i(U_i)$ per ciascun elemento:
 $U_i \in \mathcal{U}$ con associate le dipendenze in G i cui attributi sono tutti contenuti in U_i .
6. Se nessuno degli U_i costituisce una chiave per la relazione originaria $R(U)$, allora viene calcolata una chiave K di $R(U)$ e viene aggiunto allo schema generato al passo precedente uno schema di relazione sugli attributi K , senza dipendenze.

Questo algoritmo va spesso sotto il nome di **algoritmo di sintesi di schemi in terza forma normale**, perché costruisce lo schema finale a partire dalle dipendenze.

Capitolo 6

Architettura dei DBMS

Una transazione è un'**unità di programma**:

- Una transazione ha un **begin transaction** che comunica al DBMS la richiesta di interazione con esso da parte dell'applicazione.
- Il DBMS identifica l'inizio della transazione T_i e la abbina in modo univoco con l'utente/applicazione che ne ha fatto richiesta.
- Il DBMS, nell'ambito di T_i , riceve dei comandi DML in sequenza e le abbina alla transazione.

I DBMS gestiscono le transazioni garantendo, per ogni transazione T_i , il soddisfacimento delle proprietà *ACID*:

- Atomicità
- Consistenza
- Isolamento
- Durabilità

La seguente figura rappresenta un modello di architettura del DBMS:

Il **gestore delle transazioni** riceve le transazioni e ne gestisce il ciclo di vita inviando i comandi DML alle componenti sottostanti che devono eseguire l'azione richiesta.

Il **serializzatore** è responsabile della proprietà di *isolamento* (quando il programmatore progetta una transazione, lo fa nell'ipotesi che il DBMS la esegua in modo isolato rispetto alle altre transazioni).

Il **gestore del ripristino**, oltre a gestire il ripristino in seguito a guasti, è responsabile della proprietà di *atomicità* (la transazione dev'essere eseguita interamente o per nulla). Il gestore del ripristino garantisce anche l'*integrità* dei dati (se i dati vengono corrotti, il gestore del ripristino cerca di recuperare il dato corretto oppure forza un abort della

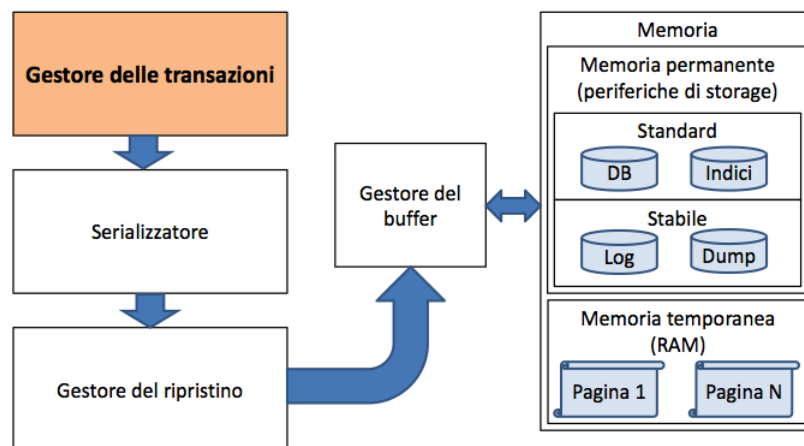


Figura 6.1: Architettura dei DBMS

transazione).

Il **gestore del buffer** è responsabile della **durabilità** (persistenza dei dati del DBMS).

La **consistenza** è di competenza sia del **gestore delle transazioni** (verifica di tutti i vincoli della base di dati) che del serializzatore.

6.1 Memoria centrale

Consideriamo la nostra area di lavoro, cioè la memoria centrale.

Nell'area di lavoro abbiamo la transizione T_i che richiede la lettura/modifica di un record. Quando una transizione richiede l'attività su un record, invia delle richieste al gestore del buffer.

La transazione T_i invia una richiesta chiamata **fix pid**, che significa: la transazione ha bisogno di avere a disposizione una pagina dati di indirizzo **pid**.

6.1.1 Le pagine

Tutti i dati del database (tabelle, indici, log e dump) sono organizzati in pagine.

Le pagine hanno dimensioni che dipendono dal sistema (anche variabili), e contengono i record.

Se la transazione ha bisogno di lavorare su un ben determinato record (tupla), bisogna cercare una pagina con un certo indirizzo **pid** che si presume contenga il record desiderato.

Caricamento delle pagine

La transazione per lavorare su un dato ha sempre bisogno di chiedere al buffer la disponibilità di una precisa pagina:

1. Il **gestore del buffer** caricherà la pagina nel buffer stesso, prendendola dalla periferica di storage se non già presente nel buffer.
2. Il gestore del buffer prende i dati memorizzati nel disco (settori di tracce sul disco) e li mette nella cache presente nella periferica stessa.
3. Dalla cache vengono poste nel buffer della memoria centrale.
4. Il gestore del buffer risponde alla transazione inviandole l'indirizzo della memoria centrale in cui trovare la pagina.
5. La transazione acquisisce, tramite il comando **fix pid**, il puntatore alla zona di memoria (buffer) che contiene la pagina.
6. In caso di lettura, si accontenta di leggere i dati.
7. In caso di scrittura o modifica, avviene il processo inverso, ovvero, la transazione modifica i dati *nel buffer*.
8. È il gestore del buffer che si occupa di copiare la pagina con le modifiche nella periferica.

Semplificazione

La pagina è composta da un certo numero di blocchi fisici, che a loro volta sono composti da un certo numero di settori.

In questo corso, però, confonderemo il blocco fisico con la pagina (dimensione blocco = dimensione pagina).

Le dimensioni caratteristiche dei blocchi fisici vanno dai 512 Byte ai 4KB, mentre le dimensioni delle pagine vanno dai 4KB agli 8KB.

Il dimensionamento della pagina è importante perché influisce sulle prestazioni dell'intero DBMS.

Tempi di trasferimento

Ricordiamo che i tempi di trasferimento di una pagina tra periferiche di storage e RAM (e viceversa) sono dell'ordine dei millisecondi ($10^{-3}s$).

I tempi di trasferimento di pagine in RAM sono dell'ordine dei nanosecondi ($10^{-9}s$).

Il fattore di differenza è 10^6 : dunque, il costo temporale è tutto sulla movimentazione delle pagine dalla periferica alla memoria centrale (e viceversa).

Nonostante i progressi tecnologici, il divario ha ancora lo stesso ordine di grandezza.

Struttura di una pagina

La pagina contiene:

- **Array degli offset**
- **Contatore CP**: conta i programmi che hanno avuto l'OK di accesso alla pagina
- **Set Dirty Bit**: vale 1 se la pagina è stata modificata.
- **Stack dei records**: contenuto della pagina.
- **Bit di parità**

Per l'identificazione di un record è necessario fornire due valori:

$$RID[PID, i]$$

Dove PID è l'identificatore della pagina, mentre i è un offset.

L'architettura di RID , con indice e offset, consente al gestore delle pagine (il filesystem) di modificare la posizione dei records a piacimento all'interno dell'area dati senza influenzare gli indirizzi esterni.

Parte II

SQL

Capitolo 7

Introduzione

Il linguaggio SQL è il linguaggio di riferimento per tutti i DBMS relazionali, e implementa sia il DDL (Data Definition Language) che il DML (Data Manipulation Language).

SQL è un linguaggio che ci permette di interrogare e gestire le basi di dati. Ciascuna interrogazione è implementata per mezzo di costrutti di programmazione detti **query**.

Per il linguaggio SQL sono stati realizzati diversi standard e diverse implementazioni.

7.1 PostgreSQL

PostgreSQL è un DBMS opensource, conforme agli standard SQL, che aggiunge ai database relazionali caratteristiche che lo rendono classificabile come object-relational. Aggiunge inoltre funzionalità quali vincoli, trigger e rules.

PostgreSQL è basato su un'architettura Client/Server, ovvero la connessione è frutto di due processi:

- Il processo **server**, chiamato postgres o postmaster, che accetta le connessioni dai client e ha il compito di interagire con i database su delega del client che ha fatto richiesta. Spesso gira su un altro computer, più potente, e accetta richieste contemporaneamente da più client.
- Un **client** che richiede di effettuare determinate operazioni su uno o più database. Un client può essere: un'applicazione grafica, una pagina web, un programma scritto da un utente. pgAdmin, l'utility che permette di impartire comandi SQL interattivamente, è un esempio di client.

7.2 Domini dei dati

7.2.1 Caratteri e stringhe

Le **stringhe a lunghezza fissa** vengono dichiarate con la seguente sintassi:

```
1      char ( length )
```

Stringhe

Le **stringhe a lunghezza variabile** vengono dichiarate con la seguente sintassi:

```
4      varchar ( length )
```

Stringhe

Quando la lunghezza di una stringa non viene specificata, si assume che sia 1 (**singolo carattere**).

Esempi:

- *char*(100): stringa di esattamente 100 caratteri
- *varchar*(100): stringa di lunghezza variabile fino a un massimo di 100 caratteri
- *char*: un singolo carattere

7.2.2 Tipi numerici esatti

I numeri decimali vengono rappresentati in maniera esatta con una delle seguenti sintassi:

```
1      decimal ( precisione , scala )  
3      numeric ( precisione , scala )
```

Numeri decimali

- *precisione*: numero totale di cifre decimali (opzionale). Per *decimal* la precisione specificata è quella minima garantita, per *numeric* è quella esatta.
- *scala*: numero di cifre decimali dopo la virgola (opzionale).

I numeri interi vengono rappresentati con una delle seguenti sintassi:

```
2      integer  
      smallint
```

Numeri interi

La rappresentazione dei numeri interi è dipendente dalla specifica implementazione.

Esempi:

- *decimal*(6,4): decimali da -99.9999 a +99.999
- *numeric*(4): decimali da -9999 a +9999
- *integer*: interi a 32 bit (solitamente)

7.2.3 Tipi numerici approssimati

I numeri in virgola mobile con mantissa ed esponente vengono rappresentati con la seguente sintassi:

	<code>float (precisione)</code>
2	<code>real</code>
4	<code>double precision</code>

Numeri approssimati

- *precisione*: numero di cifre binarie per la mantissa

7.2.4 Date

Possiamo usare la seguente sintassi per rappresentare istanti temporali:

- *date*: per le date. Comprende i sottocampi *year*, *month*, *day*.
- *time*(*precisione*): per gli orari. Comprende i sottocampi *hour*, *minute*, *second*.
- *timestamp*(*precisione*): per specificare sia date che orari. Comprende i sottocampi *year*, *month*, *day*, *hour*, *minute*, *second*.
- *precisione*: numero di cifre per le frazioni di secondo (opzionale)

Esempi:

- *timestamp*(2): per memorizzare eventi di log con precisione al centesimo di secondo.

7.2.5 Intervalli

Per rappresentare gli intervalli temporali, usiamo la sintassi:

```
1 interval unita1 [to unita2]
```

Intervalli Temporali

- *unita1*: unità di tempo più grossolana
- *unita2* (opzionale): unità di tempo più fine

Esempi:

- *interval year*: esprime intervalli in anni.
- *interval year to month*: esprime intervalli misurati in anni e mesi (e.g. un anno e due mesi).
- *interval day to second(2)*: esprime intervalli misurati in giorni, ore, minuti, secondi e centesimi di secondo.
- *interval year to minute*: non è permesso (non si può passare in modo preciso da *month* a *day* perché i mesi hanno durata diversa).

7.2.6 Altri domini

- *boolean*: valori booleani.
- *bigint*: interi "grossi"
- *blob*: Binary Large Object (immagini, video, file di vario tipo) (in PostgreSQL *bytea* oppure *lo*).
- *clob*: Character Large Object (lunghi file di testo) (in PostgreSQL *text*).

7.2.7 Domini personalizzati

È possibile creare domini personalizzati (le parti tra [] sono opzionali:

```
1 create domain NomeDominio as TipoDato
2 [default ValoreDefault] [Vincoli]
```

Domini personalizzati

- *NomeDominio*: nome del nuovo dominio

- *TipoDato*: nome del tipo di base
- *default ValoreDefault*: valore assegnato in automatico, se non specificato esplicitamente (opzionale)
- *Vincoli*: insieme di vincoli sui valori assunti dal dominio personalizzato (opzionale)

I domini personalizzati sono comunque semplici (no array, no struct, no record).

Esempio 1: voto di un esame

```

1      create domain Voto
2      as smallint default NULL
3      check (value >= 18) AND (value <= 30);
4

```

Domini personalizzati –Esempio 1

Esempio 2: temperatura corporea (gradi Celsius)

```

1      create domain Temperatura
2      as decimal(3, 1) default NULL
3      check (value >= 35) AND (value <= 42);
4

```

Domini personalizzati –Esempio 2

L'utilità dei nuovi domini è legata alla possibilità di astrazione.

Esempio 3: per passare dai gradi Celsius ai gradi Fahrenheit è sufficiente cambiare una volta per tutte il dominio di Temperatura:

```

1      create domain Temperatura
2      as decimal(4, 1) default NULL
3      check (value >= 95) AND (value <= 107.6);
4

```

Domini personalizzati –Esempio 3

7.3 Definizione di tabelle

Per creare e definire una nuova tabella utilizziamo la seguente sintassi:

```

1      create table NomeTabella(
2          NomeAttributo1 Dominio1 [ValoreDefault1][Vincoli1
3          ]
4          ...
5      )

```



```

1      Nome AttributoN DominioN [ValoreDefaultN][
2      VincoliN]
3      );

```

Creare nuova tabella

Esempio: *Dipartimento*(Nome, Indirizzo, Città)

```

1      create table Dipartimento(
2          Nome varchar(20) primary key,
3          Indirizzo varchar(50),
4          Città varchar(20) not null
5      );

```

Esempio Dipartimento

7.3.1 Valori di default

Per ogni attributo può essere specificato un valore predefinito che verrà usato se, nell'inserimento di una riga, non viene specificato un valore per quell'attributo.

Esempio:

```

1      NumeroFigli smallint default 0,
2      Email varchar(64) default 'unito.it',
3      StatoCivile varchar(20) default 'libero'

```

Esempio Valori Default

Se non si specifica esplicitamente un valore di default, viene usato *null*.

7.4 Definizione dei vincoli

I vincoli servono a definire proprietà che devono essere verificate da ogni istanza della base di dati per garantirne l'integrità (vincoli di integrità).

Si differenziano in:

- **Vincoli intrarelazionali:** relativi a una sola tabella
- **Vincoli interrelazionali:** relativi a più tabelle

Si possono specificare:

- Contestualmente alla definizione degli attributi

- Alla fine della definizione della tabella

Si possono usare vincoli predefiniti oppure si può specificare l'espressione logica che il vincolo deve verificare.

In alcune occasioni può essere utile assegnare un nome a un vincolo. Per farlo si utilizza la seguente sintassi dopo tutte le definizioni di attributi:

```
1      constraint NomeVincolo DefinizioneVincolo
```

7.4.1 Vincoli intrarelazionali predefiniti

Vincolo *not null*:

- È un vincolo di tupla che indica che il valore nullo non è ammesso come valore per un determinato attributo, quindi rende l'attributo obbligatorio
- Se l'attributo non viene specificato in fase di inserimento e non si è specificato un valore di default, si viola il vincolo d'integrità e l'operazione è annullata
- Sintassi

```
1      NomeAttributo Dominio { } not null
```

- Se per l'attributo viene specificato un valore di default è possibile effettuare l'inserimento anche senza specificarne il valore.

Esempio

```
1      Citta varchar(20) not null
```

Vincolo *unique*:

- È un vincolo di tabella che indica che il valore di un attributo o le combinazioni di valori su un insieme di attributi sono una superchiave:

$$\text{righe diverse} \leftrightarrow \text{valori diversi}$$

- Fa eccezione il valore nullo (che può comparire in più righe senza violare il vincolo).

- Sintassi (vincolo su un solo attributo):

```
1 NomeAttributo Dominio unique
```

Esempio

```
1 Matricola varchar(6) unique
```

Vincolo *unique* su insiemi di attributi:

- Va specificato dopo la definizione degli attributi della tabella.
- Sintassi:

```
1 NomeAttributo1 Dominio1
  NomeAttributo2 Dominio2
3 ...
  unique (NomeAttributo1 , NomeAttributo2)
5
```

Esempio

```
2 Nome varchar(255) ,
  Cognome varchar(255) ,
4 unique(Nome, Cognome)
```

Vincolo *primary key* di chiave primaria:

- È un vincolo di tabella che indica che un attributo o un insieme di attributi sono la chiave primaria.
- Gli attributi così definiti non possono assumere valore nullo.
- Può esserci un solo vincolo primary key per ogni tabella.
- Vincolo *primary key* su un solo attributo: va specificato nella definizione dell'attributo mediante la seguente sintassi:

```

1      NomeAttributo Dominio primary key

```

- Vincolo *primary key* su un insieme di attributi: va specificato dopo la definizione degli attributi mediante la seguente sintassi:

```

1      NomeAttributo1 Dominio1 ,
      NomeAttributo2 Dominio2 ,
3      ...
      primary key ( NomeAttributo1 ,
5      NomeAttributo2 , ... )

```

Vincoli interrelazionali predefiniti

Un **vincolo di integrità referenziale** (di **chiave esterna**):

- Crea un vincolo tra i valori uno o più attributi della tabella (interna) su cui è definito e uno o più attributi di un'altra tabella (esterna).
- Per ogni riga della tabella interna, il valore dell'attributo specificato nel vincolo, se diverso da *null*, dev'essere presente tra i valori del corrispondente attributo della tabella esterna.
- Sintassi con un solo attributo:

```

      NomeAttributo references
      NomeTabellaEsterna( NomeAttributoEsterno )
2

```

- Sintassi con più attributo:

```

1      NomeAttributo1 Dominio1 ,
      NomeAttributo2 Dominio2 ,
3      ...
      foreign key ( NomeAttributo1 ,
      NomeAttributo2 , ... )
5      references NomeTabellaEsterna (
      NomeAttributoEsterno1 , Nome AttributoEsterno2 , ... )

```

L'ordine è importante! Ciascun attributo viene mappato al corrispondente secondo l'ordine con cui lo specifichiamo.

Violazione del vincolo di integrità referenziale

In generale, quando un vincolo viene violato, il DBMS rifiuta l'operazione che causerebbe la violazione e segnala un errore.

Con i vincoli di integrità referenziale si possono specificare altre reazioni da adottare in caso di violazione.

Casi in cui può avvenire una violazione di un vincolo di integrità referenziale:

1. Inserimento di una nuova riga nella tabella interna.
2. Modifica, nella tabella interna, di un valore dell'attributo referente.
3. Modifica, nella tabella esterna, di un valore dell'attributo riferito.
4. Cancellazione di una riga nella tabella esterna.

I primi due casi riguardano un cambiamento della **tabella interna**, gli ultimi due un cambiamento della **tabella esterna**.

Quando cambia la tabella interna (casi 1 e 2), l'operazione viene semplicemente **rifiutata**. Quando cambia la tabella esterna (casi 3 e 4), si può specificare quali variazioni apportare alla tabella interna.

Questa asimmetria nel comportamento deriva dal fatto che dal punto di vista applicativo la tabella esterna ricopre il ruolo di tabella principale (**master**) e quella interna di secondaria (**slave**) che deve adeguarsi alle variazioni.

In seguito a una **modifica** (caso 3), le reazioni possono essere le seguenti:

1. **cascade**: il nuovo valore dell'attributo della tabella esterna viene riportato in tutte le corrispondenti righe della tabella interna.
2. **set null**: all'attributo referente viene assegnato il valore nullo al posto del valore modificato nella tabella esterna.
3. **set default**: all'attributo referente viene assegnato il valore di default al posto del valore modificato nella tabella esterna.
4. **no action**: nessuna reazione (e quindi la modifica non viene consentita).

In seguito a una **cancellazione** (caso 4), le reazioni possono essere le seguenti:

1. **cascade**: tutte le righe della tabella interna corrispondenti alla riga cancellata nella tabella esterna vengono cancellate.

2. **set null**: all'attributo referente viene assegnato il valore nullo al posto del valore cancellato nella tabella esterna.
3. **set default**: all'attributo referente viene assegnato il valore di default al posto del valore cancellato nella tabella esterna.
4. **no action**: nessuna reazione (e quindi la cancellazione non viene consentita).

La sintassi per specificare il comportamento in caso di violazione di un vincolo interrelazionale è la seguente:

```
1      //Per la modifica , subito dopo il vincolo
      on update Reazione
3      //Per la cancellazione , subito dopo il vincolo
      on delete Reazione
5
```

Esempio

```
2      create table Ricoveri (
      PAZ varchar(4) ,
4      Inizio date ,
      Fine date ,
      Reparto char ,
6      primary key (PAZ, Inizio) ,
      foreign key (PAZ) references Pazienti(COD)
8          on update cascade
          on delete no action ,
10     foreign key (Reparto) references Reparti (COD)
          on update cascade
12     on delete set null
14 );
```

Creazione tabella Ricoveri

7.5 Modifiche alle definizioni

SQL permette di modificare la definizione di tabelle, domini, vincoli precedentemente introdotti:

```
1      alter (modifica),  
      drop (cancellazione),  
3      add (aggiunta)
```

Modifica definizioni

Capitolo 8

Query di Base

Le operazioni di interrogazione in SQL vengono specificate per mezzo dell'istruzione `select`.

Vediamone la struttura essenziale:

```
1      select Attributo1 [[as] AliasAttributo1]...  
      from Tabella1 [[as] AliasTabella1], ...  
3      [where Condizione]
```

Le tre parti di cui si compone un'istruzione `select` vengono spesso chiamate:

- Clausola `select`
- Clausola `from`
- Clausola `where`

L'interrogazione SQL seleziona, tra le righe che appartengono al prodotto cartesiano delle tabelle elencate nella clausola `from`, quelle che soddisfano le condizioni espresse nell'argomento della clausola `where`.

Il risultato dell'esecuzione di un'interrogazione SQL è quindi una tabella con una riga per ogni riga prodotta dalla clausola `from` e filtrata dalla clausola `where`, le cui colonne si ottengono dalla valutazione delle espressioni che appaiono nella clausola `select`.

Ogni colonna viene eventualmente ridenominata con l'Alias specificato.

Esempio 1

Consideriamo la seguente base dati:

IMPIEGATO(Nome, Cognome, *Dipartimento*, *Ufficio*, *Stipendio*, *Citta*)
DIPARTIMENTO(Nome, *Indirizzo*, *Citta*) (8.1)

Estrarre lo stipendio degli impiegati di cognome "Rossi":


```
1      select
2          Stipendio as Salario
3      from
4          Impiegato
5      where
6          Cognome = 'Rossi'
```

Esempio 1

Se non vi sono impiegati di cognome "Rossi", l'interrogazione restituirà un insieme vuoto, altrimenti, un insieme con tante righe quanti sono tali impiegati.

8.1 Clausola select

La clausola `select` specifica gli elementi dello schema della tabella risultato. Come argomento della `select` può anche comparire il carattere speciale `*`, che rappresenta la selezione di tutti gli attributi delle tabella indicate nelle `from`.

Inoltre, nella `select` possono comparire generiche espressioni sul valore degli attributi di ciascuna riga selezionata.

Esempio 2

Consideriamo la seguente base dati:

IMPIEGATO(Nome, *Cognome*, *Dipartimento*, *Ufficio*, *Stipendio*, *Citta*)
DIPARTIMENTO(Nome, *Indirizzo*, *Citta*) (8.2)

Estrarre tutte le informazioni relative agli impiegati di cognome "Rossi"

```
1      select *
2      from
3          Impiegato
4      where
5          Cognome = 'Rossi'
```

Esempio 2

Esempio 3

Consideriamo la seguente base dati:

IMPIEGATO(Nome, *Cognome*, *Dipartimento*, *Ufficio*, *Stipendio*, *Citta*)
DIPARTIMENTO(Nome, *Indirizzo*, *Citta*) (8.3)

Estrarre lo stipendio mensile dell'impiegato che ha cognome "Bianchi"

```
1      select
2          Stipendio/12 as StipendioMensile
3      from
4          Impiegato
5      where
6          Cognome = 'Bianchi'
```

Esempio 3

8.2 Clausola from

Quando si desidera formulare un'interrogazione che coinvolge righe appartenenti a più di una tabella, si pone come argomento della *from* l'insieme di tabelle alle quali si vuole accedere. Sul prodotto cartesiano delle tabelle elencate verranno poi applicate le condizioni della *where*.

Quindi, un join può essere specificato indicando in modo esplicito le condizioni che esprimono il legame tra le diverse tabelle.

Quando abbiamo a che fare con più tabelle, può accadere che compaiano attributi omonimi: in questo caso è possibile utilizzare gli alias oppure utilizzare la notazione "." specificando prima del punto la tabella da cui prendiamo l'attributo (Tabella.Attributo).

Esempio 4

Consideriamo la seguente base dati:

IMPIEGATO(Nome, *Cognome*, *Dipartimento*, *Ufficio*, *Stipendio*, *Città*)
DIPARTIMENTO(Nome, *Indirizzo*, *Città*) (8.4)

Estrarre i nomi degli impiegati e le città in cui lavorano

```
2      select
3          Impiegato.Nome,
4          Impiegato.Cognome,
5          Dipartimento.Città
6      from
7          Impiegato, Dipartimento
8      where
9          Impiegato.Dipartimento = Dipartimento.Nome
```

Esempio 4

Esempio 5

Consideriamo la seguente base dati:

IMPIEGATO(Nome, Cognome, *Dipartimento*, *Ufficio*, *Stipendio*, *Citta*)
DIPARTIMENTO(Nome, *Indirizzo*, *Citta*) (8.5)

Estrarre i nomi degli impiegati e le città in cui lavorano utilizzando degli alias per le tabelle

```

select
    I.Nome,
    I.Cognome,
    D.Citta
from
    Impiegato as I,
    Dipartimento as D
where
    I.Dipartimento = D.Nome

```

Esempio 5

8.3 Clausola where

La clausola *where* ammette come argomento un'espressione booleana costituita combinando predicati semplici con gli operatori *AND*, *OR* e *NOT*.

Ciascun predicato utilizza gli operatori di confronto (*=*, *<>*, *<*, *<=*, *>*, *>=*) per confrontare da un lato un'espressione costruita a partire dai valori degli attributi per la riga, e dall'altro lato un valore costante o un'altra espressione.

La sintassi assegna la precedenza all'operatore *NOT*, ma non definisce una relazione di precedenza tra l'*AND* e l'*OR*. Se è necessario esprimere un'interrogazione che richieda sia l'uso dell'*AND* che del *NOT*, conviene esplicitare l'ordine di valutazione mediante le parentesi.

Oltre ai normali predicati di confronto, SQL mette a disposizione un operatore *like* per il confronto di stringhe, che permette di effettuare confronti con stringhe in cui compaiono caratteri speciali (*_*) oppure *%*.

Il carattere *_* rappresenta nel confronto un carattere arbitrario, mentre il carattere *%* rappresenta una stringa di un numero arbitrario (eventualmente nullo) di caratteri arbitrari. Un confronto del tipo *like ab%ba_* sarà soddisfatto da una qualunque stringa di caratteri che inizi con *ab* e la stringa *ba* prima dell'ultima posizione.

Esempio 6

Consideriamo la seguente base dati:

IMPIEGATO(Nome, Cognome, *Dipartimento*, *Ufficio*, *Stipendio*, *Citta*)
DIPARTIMENTO(Nome, *Indirizzo*, *Citta*) (8.6)

Estrarre il nome e il cognome degli impiegati che lavorano nell'ufficio 20 del dipartimento *Amministrazione*

```
1      select
2          I . Nome ,
3          I . Cognome
4      from
5          Impiegato as I
6      where
7          I . Ufficio = 20
8          and I . Dipartimento = 'Amministrazione'
```

Esempio 6

Esempio 7

Consideriamo la seguente base dati:

IMPIEGATO(Nome, Cognome, *Dipartimento*, *Ufficio*, *Stipendio*, *Citta*)
DIPARTIMENTO(Nome, *Indirizzo*, *Citta*) (8.7)

Estrarre i nomi e i cognomi degli impiegati che lavorano al dipartimento *Amministrazione* o al dipartimento *Produzione*.

```
2      select
3          I . Nome ,
4          I . Cognome
5      from
6          Impiegato as I
7      where
8          I . Dipartimento = 'Amministrazione'
9          or I . Dipartimento = 'Produzione'
```

Esempio 7

Esempio 8

Consideriamo la seguente base dati:

IMPIEGATO(Nome, Cognome, *Dipartimento*, *Ufficio*, *Stipendio*, *Citta*)
DIPARTIMENTO(Nome, *Indirizzo*, *Citta*) (8.8)

Estrarre i nomi propri degli impiegati di cognome "Rossi" che lavorano al dipartimento *Produzione* o *Amministrazione*.

```
select
    I.Nome
from
    Impiegato as I
where
    (I.Dipartimento = 'Produzione'
    or I.Dipartimento = 'Amministrazione')
    and I.Cognome = 'Rossi'
```

Esempio 8

Esempio 9

Consideriamo la seguente base dati:

IMPIEGATO(Nome, Cognome, *Dipartimento*, *Ufficio*, *Stipendio*, *Citta*)
DIPARTIMENTO(Nome, *Indirizzo*, *Citta*) (8.9)

Estrarre gli impiegati che hanno un cognome che ha una *o* in seconda posizione e finisce per *i*.

```
select
    I.Nome, I.Cognome
from
    Impiegato as I
where
    I.Cognome like '_o%i'
```

Esempio 9

8.4 Gestione dei valori nulli

Un valore nullo in un attributo può significare che tale attributo non è applicabile, o che il valore è applicabile ma non è conosciuto o anche che non si sa in quale delle due situazioni

ci troviamo.

Per selezionare i termini con valori nulli, SQL fornisce il predicato `is null`, la cui sintassi è semplicemente

$$\text{Attributo is [not] null}$$

Il predicato risulta vero solo se l'attributo ha valore nullo, mentre il predicato `is not null` è la sua negazione.

I valori nulli hanno un particolare impatto sulla valutazione dei normali predicati.

Consideriamo un semplice predicato di confronto fra un attributo della tabella T e un valore costante:

$$A > c$$

Questo predicato sarà vero per le righe in cui l'attributo A è superiore a c .

Osserviamo che ci sono due soluzioni per gestire il caso in cui A sia nullo: la prima soluzione, più immediata, usa la **logica a due valori** e prevede semplicemente di considerare falso il predicato.

La seconda soluzione, quella adottata in SQL a partire da SQL-2, fa uso di una **logica a tre valori**, in cui un predicato semplice restituisce il valore *unknown* quando uno qualsiasi dei suoi termini ha valore nullo. Fa eccezione, ovviamente, il predicato `is null`, che restituisce sempre o vero o falso.

La differenza tra le soluzioni basate sulle logiche a due e tre valori emerge solo quando si valutano espressioni complicate. In alcuni casi il comportamento del sistema in presenza di valori nulli può diventare molto poco intuitivo, richiedendo una maggiore attenzione.

8.5 Duplicati

Una significativa differenza tra SQL e algebra relazionale è data dalla gestione dei duplicati. Mentre in algebra una tabella viene vista come una relazione dal punto di vista matematico, e quindi come un insieme di elementi (tuple) diversi tra loro, in SQL si possono avere in una tabella più righe uguali (dette duplicati), ovvero righe con gli stessi valori su tutti gli attributi.

Per emulare il comportamento dell'algebra relazionale, sarebbe necessario effettuare l'eliminazione dei duplicati tutte le volte che si eseguono operazioni di proiezione. L'operazione di eliminazione di duplicati, però, è molto costosa, e spesso non è necessaria, in quanto in molti casi il risultato non contiene duplicati.

Per esempio, quando il risultato include una chiave per ogni tabella che compare nella `from`, la tabella risultato non può contenere più esemplari della stessa riga. Per questo in SQL si è stabilito di permettere la presenza di duplicati all'interno delle tabelle, lasciando al programmatore il compito di specificare esplicitamente quando l'operazione di rimozione di duplicati è necessaria.

L'eliminazione dei duplicati è specificata con la parola chiave `distinct`, da porre immediatamente dopo la parola chiave `select`.

La sintassi prevede che si possa anche specificare la parola chiave `all` al posto di `distinct`, indicando che si intendono mantenere tutti i duplicati. L'indicazione della parola `all` è opzionale, in quanto, come già detto, il mantenimento dei duplicati costituisce l'opzione di default.

Esempio 10

Consideriamo la seguente base dati:

IMPIEGATO(*Nome*, *Cognome*, *Dipartimento*, *Ufficio*, *Stipendio*, *Città*)
DIPARTIMENTO(*Nome*, *Indirizzo*, *Città*) (8.10)

Estrarre gli impiegati di cognome "Rossi" che abitano nella stessa città, presentando eventualmente più volte lo stesso valore di *Città*.

```
select  
    I.Città  
from  
    Impiegato as I  
where  
    I.Cognome = 'Rossi'
```

Esempio 10

Esempio 11

Consideriamo la seguente base dati:

IMPIEGATO(*Nome*, *Cognome*, *Dipartimento*, *Ufficio*, *Stipendio*, *Città*)
DIPARTIMENTO(*Nome*, *Indirizzo*, *Città*) (8.11)

Estrarre gli impiegati di cognome "Rossi" che abitano nella stessa città, facendo comparire al più una volta ogni città.

```
select distinct  
    I.Città  
from  
    Impiegato as I  
where  
    I.Cognome = 'Rossi'
```

Esempio 11

8.6 Uso di variabili

Abbiamo già visto come nelle interrogazioni SQL sia possibile associare un nome alternativo, detto *alias*, alle tabelle che compaiono come argomento della clausola `from`.

Il nome viene usato per far riferimento alla tabella nel contesto dell'interrogazione. Questa funzionalità può essere sfruttata per far riferimento a una tabella in modo compatto, ricorrendo a brevi alias ed evitando così di scrivere per esteso il nome della tabella tutte le volte che ne viene richiesto l'uso. Vi sono, però, altre ragioni per utilizzare gli alias.

Per prima cosa, utilizzando gli alias è possibile fare accesso più volte alla stessa tabella, come avviene nel calcolo relazionale quando si usano più variabili associate alla stessa tabella e in modo simile all'uso dell'operatore di ridenominazione ρ dell'algebra relazionale. Tutte le volte che si introduce un alias per una tabella si dichiara in effetti una variabile che rappresenta le righe della tabella di cui è alias. Quando una tabella compare una sola volta in un'interrogazione, non c'è differenza tra l'interpretare l'alias come uno pseudonimo o come una nuova variabile. Quando una tabella compare invece più volte, è necessario considerare l'alias come una nuova variabile.

Esempio 12

Consideriamo la seguente base dati:

$IMPIEGATO(\underline{Nome}, \underline{Cognome}, Dipartimento, Ufficio, Stipendio, Città)$
 $DIPARTIMENTO(\underline{Nome}, Indirizzo, Città)$
(8.12)

Estrarre tutti gli impiegati che hanno lo stesso cognome (ma diverso nome) di impiegati dipartimento *Produzione*.

```

select distinct
  I1.Nome,
  I1.Cognome
from
  Impiegato as I1
  Impiegato as I2
where
  I1.Cognome = I2.Cognome
and I1.Nome <> I2.Nome
and I2.Dipartimento = 'Produzione'
```

Esempio 12

Questa interrogazione confronta ciascuna riga di *IMPIEGATO* con tutte le righe di *IMPIEGATO* associate al dipartimento *Produzione*.

Si osservi che in questa interrogazione ogni riga con *Produzione* come valore dell'attributo *Dipartimento* viene confrontata anche con se stessa, ma il confronto della riga con se

stessa non sarà mai soddisfatto, in quanto il predicato di disuguaglianza sull'attributo *Nome* non potrà mai essere vero.

Per illustrare l'esecuzione di quest'interrogazione si può immaginare che al momento della definizione degli alias, vengano create due diverse tabelle associate alle variabili *I1* e *I2*, ciascuna con tutte le righe di *IMPIEGATO*; ciascuna variabile assumerà quindi ciascun valore di tupla in modo indipendente dell'altra variabile.

8.7 Ordinamento

Mentre una relazione è costituita da un insieme non ordinato di tuple, nell'uso reale delle basi di dati sorge spesso il bisogno di costruire un ordine sulle righe delle tabelle.

SQL permette di specificare un ordinamento delle righe del risultato di un'interrogazione tramite la clausola `order by`, con la quale si chiude l'interrogazione. La clausola rispetta la seguente sintassi:

```
order by AttributoDiOrdinamento1 [asc|desc] , AttributoDiOrdinamento2 [asc|desc]
```

In questo modo si specificano gli attributi che devono essere utilizzati per l'ordinamento: per prima cosa le righe vengono ordinate in base al primo attributo nell'elenco; per righe che hanno lo stesso valore sul primo attributo, si considerano i valori degli attributi successivi, in sequenza.

L'ordine su ciascun attributo può essere ascendente o discendente, a seconda che si sia usato il quantificatore `asc` o `desc`. Se il quantificatore è omissso, so assume un ordinamento ascendente.

Esempio 13

Consideriamo la seguente base dati:

$$\begin{array}{l} \text{IMPIEGATO}(\underline{\text{Nome}}, \underline{\text{Cognome}}, \text{Dipartimento}, \text{Ufficio}, \text{Stipendio}, \text{Citta}) \\ \text{DIPARTIMENTO}(\underline{\text{Nome}}, \text{Indirizzo}, \text{Citta}) \end{array} \quad (8.13)$$

Estrarre i nomi propri di tutti gli impiegati del dipartimento *Produzione* ordinati in ordine decrescente.

```

2      select distinct
3          I.Nome
4      from
5          Impiegato as I
6      where
7          I.Dipartimento = 'Produzione'
8      orderby I.Nome desc

```

Esempio 13

Capitolo 9

Join

Abbiamo visto come specificare il join attraverso una clausola `where` sul prodotto cartesiano.

SQL implementa anche una sintassi più immediata, attraverso l'utilizzo della parola chiave `join`.

```
1      select
2      from
3          Tabella1 [as Alias1]
4          [TipoJoin] join
5          Tabella2 [as Alias2]
6              on CondizioneJoin
7      [where AltraCondizione]
```

Sintassi del join

Mediante questa sintassi la condizione di join non compare come argomento della clausola `where`, ma viene invece spostata nell'ambito della clausola `from`, associata alle tabelle che vengono coinvolte nel join.

Il parametro `TipoJoin` specifica qual è il tipo di join da usare, e ad esso si possono sostituire i termini `inner`, `right outer`, `left outer`, `full outer` (il qualificatore `outer` è opzionale).

Con il join interno le righe che vengono coinvolte nel join sono in generale un sottoinsieme delle righe di ciascuna tabella. Può, infatti, capitare che alcune righe non vengano considerate in quanto non esiste una corrispondente riga nell'altra tabella per cui la condizione sia soddisfatta. Questo comportamento spesso non rispetta le esigenze delle applicazioni, le quali, all'eliminazione delle righe operata dal join, possono preferire di mantenere le righe, introducendo dei valori nulli per rappresentare l'assenza di informazioni provenienti dall'altra tabella.

Il join esterno, invece, esegue un join mantenendo tutte le righe che fanno parte di una o entrambe le tabelle coinvolte.

Esistono tre varianti dei join esterni: `left`, `right` e `full`. Il `left join` fornisce come risultato il join esterno esteso con le righe della tabella che compare a sinistra per le quali non esiste una corrispondente riga nella tabella di destra.

Il `right join` si comporta in modo simmetrico (conserva le righe escluse della tabella di destra).

Infine, il `full join` restituisce il join interno esteso con le righe escluse di entrambe le tabelle.

Possiamo, inoltre, realizzare il `natural join` utilizzando nel join di due tabelle una condizione implicita di uguaglianza su tutti gli attributi caratterizzati dallo stesso nome. Nonostante il vantaggio di una scrittura più compatta, il join naturale non è normalmente consigliabile (e spesso non è offerto dai sistemi commerciali). Un motivo è che un'interrogazione che usa il join naturale può introdurre dei rischi nelle applicazioni, in quanto il suo comportamento può mutare profondamente al variare dello schema delle tabelle.

Esempio 1

Consideriamo la seguente base dati:

$$\begin{aligned} &IMPIEGATO(\underline{Nome}, \underline{Cognome}, Dipartimento, Ufficio, Stipendio, Città) \\ &DIPARTIMENTO(\underline{Nome}, Indirizzo, Città) \end{aligned} \quad (9.1)$$

Estrarre i nomi degli impiegati e le città in cui lavorano utilizzando degli alias per le tabelle

```

8      select
9          I.Nome,
10         I.Cognome,
11         D.Città
12      from
13         Impiegato as I
14         join
15         Dipartimento as D
16         on (I.Dipartimento = D.Nome)

```

Esempio 1

Esempio 2

Consideriamo la seguente base dati:

$$\begin{aligned} &GUIDATORE(\underline{NroPatente}, Nome, Cognome) \\ &AUTOMOBILE(\underline{Targa}, Modello, NroPatente) \end{aligned} \quad (9.2)$$

Estrarre i guidatori con le automobili loro associate, mantenendo nel risultato anche i guidatori senza automobile

```
8      select
9          G.Nome,
10         G.Cognome
11      from
12         Guidatore as G
13         left join
14         Automobile as A
15             on (G.NroPatente = A.NroPatente)
```

Esempio 2

Esempio 3

Consideriamo la seguente base dati:

GUIDATORE(NroPatente, Nome, Cognome)
AUTOMOBILE(Targa, Modello, NroPatente) (9.3)

Estrarre tutti i guidatori e tutte le auto, mostrando tutte le relazioni esistenti tra di essi.

```
8      select
9          G.Nome,
10         G.Cognome,
11         G.NroPatente,
12         A.Targa,
13         A.Modello
14      from
15         Guidatore as G
16         full join
17         Automobile as A
18             on (G.NroPatente = A.NroPatente)
```

Esempio 3

Esempio 4

Consideriamo la seguente base dati:

GUIDATORE(NroPatente, Nome, Cognome)
AUTOMOBILE(Targa, Modello, NroPatente) (9.4)

Estrarre tutti i guidatori e tutte le auto, mostrando tutte le relazioni esistenti tra di essi.

```
10      select
11          G.Nome,
12          G.Cognome,
13          G.NroPatente,
14          A.Targa,
15          A.Modello
16      from
17          Guidatore as G
18          natural join
19          Automobile as A
```

Esempio 4

Capitolo 10

Aggregati

10.1 Operatori aggregati

Gli operatori aggregati costituiscono una delle più importanti estensioni di SQL rispetto all'algebra relazionale. Infatti, in algebra relazionale tutte le condizioni vengono valutate su una tupla alla volta, la condizione è sempre un predicato che viene valutato su ciascuna tupla indipendentemente da tutte le altre.

Spesso, però, nei contesti reali è richiesto di valutare delle proprietà che dipendono da insiemi di tuple.

Gli operatori aggregati vengono gestiti come un'estensione delle normali interrogazioni. Prima viene normalmente eseguita l'interrogazione, considerando solo le parti `from` e `where`.

L'operatore aggregato viene poi applicato alla tabella contenente il risultato dell'interrogazione.

10.1.1 Operatore `count`

L'operatore `count` calcola il numero di tuple che compongono la tabella risultante dall'interrogazione, usando la seguente sintassi:

```
count (<* | [distinct|all] listaAttributi>)
```

La prima opzione `*` restituisce il numero di righe.

L'opzione `distinct` restituisce il numero di diversi valori degli attributi in `ListaAttributi`.

L'opzione `all` restituisce il numero di righe che possiedono valori diversi dal valore nullo per gli attributi in `ListaAttributi`.

Se si specifica un attributo e si omette `distinct` o `all`, si assume `all` come default.

Esempio 1

Consideriamo la seguente base dati:

IMPIEGATO(Nome, *Cognome*, *Dipartimento*, *Ufficio*, *Stipendio*, *Citta*)
DIPARTIMENTO(Nome, *Indirizzo*, *Citta*) (10.1)

Estrarre il numero di impiegati del dipartimento *Produzione*

```
18      select
19          count(*)
20      from
21          Impiegato as I
22      where
23          I.Dipartimento = 'Produzione'
```

Esempio 1

Esempio 2

Consideriamo la seguente base dati:

IMPIEGATO(Nome, *Cognome*, *Dipartimento*, *Ufficio*, *Stipendio*, *Citta*)
DIPARTIMENTO(Nome, *Indirizzo*, *Citta*) (10.2)

Estrarre il numero di diversi valori dell'attributo *Stipendio* fra tutte le righe di *IMPIEGATO*

```
16      select
17          count (distinct I.Stipendio)
18      from
19          Impiegato as I
```

Esempio 2

Esempio 3

Consideriamo la seguente base dati:

IMPIEGATO(Nome, *Cognome*, *Dipartimento*, *Ufficio*, *Stipendio*, *Citta*)
DIPARTIMENTO(Nome, *Indirizzo*, *Citta*) (10.3)

Estrarre il numero di righe che possiedono un valore non nullo per l'attributo *Nome*

```

20      select
21          count ( all I.Nome)
22      from
23          Impiegato as I
24

```

Esempio 3

10.1.2 Altri operatori aggregati

Gli altri operatori aggregati (`sum`, `max`, `min`, `avg`), invece, ammettono come argomento un attributo o un'espressione, eventualmente preceduta dalle parole chiave `distinct` o `all`.

Le funzioni aggregate `sum` e `avg` ammettono come argomento solo espressioni che rappresentano valori numerici o intervalli di tempo.

Le funzioni `min` e `max` richiedono solamente che sull'espressione sia definito un ordinamento, per cui si possono applicare anche su stringhe di caratteri o su istanti di tempo

`<sum | max | min | avg> ([distinct | all] AttEsp)`

Gli operatori si applicano sulle righe che soddisfano la condizione presente nella clausola `where` e hanno il seguente significato:

- `sum`: restituisce la somma dei valori posseduti dall'espressione
- `max` e `min`: restituiscono rispettivamente il valore massimo e minimo
- `avg`: restituisce la media dei valori (ossia il risultato della divisione di `sum` per `count`)

Le parole chiave `distinct` e `all` hanno il significato che abbiamo già visto: `distinct` elimina i duplicati, mentre `all` trascura solo i valori nulli; l'uso di `distinct` o `all` con gli operatori `max` e `min` non ha effetto sul risultato.

Esempio 4

Consideriamo la seguente base dati:

$$\begin{array}{l}
 IMPIEGATO(\underline{Nome}, \underline{Cognome}, Dipartimento, Ufficio, Stipendio, Città) \\
 DIPARTIMENTO(\underline{Nome}, Indirizzo, Città)
 \end{array}
 \quad (10.4)$$

Estrarre la somma degli stipendi del dipartimento 'Amministrazione'

```

22      select
23          sum(I.Stipendio)
24      from
25          Impiegato as I
26

```



```

26      where
        l.Dipartimento = 'Amministrazione'

```

Esempio 4

10.2 Interrogazioni con raggruppamento

Abbiamo caratterizzato gli operatori aggregati come gli operatori che vengono applicati a un insieme di righe. Molto spesso, però, sorge l'esigenza di applicare l'operatore aggregato separatamente a sottoinsiemi di righe.

Per poter utilizzare in questo modo l'operatore aggregato, SQL mette a disposizione la clausola `group by`, che permette di specificare come dividere le tabelle in sottoinsiemi. La clausola ammette come argomento un insieme di attributi e l'interrogazione raggrupperà le righe che possiedono gli stessi valori per questo insieme di attributi.

Inizialmente, l'interrogazione viene eseguita come se la clausola non esistesse, selezionando gli attributi che compaiono come argomenti della clausola `group by` o che compaiono all'interno dell'espressione argomento dell'operatore aggregato.

La tabella ottenuta viene poi analizzata, dividendo le righe in insiemi caratterizzati dallo stesso valore degli attributi che compaiono come argomento della clausola `group by`.

Dopo che le righe sono state raggruppate in sottoinsiemi, l'operatore aggregato viene applicato separatamente su ogni sottoinsieme. Il risultato dell'interrogazione è costituito da una tabella con righe che contengono l'esito della valutazione dell'operatore aggregato affiancato al valore dell'attributo che è stato usato per l'aggregazione.

La sintassi impone che, in un'interrogazione che fa uso della clausola `group by`, possa comparire come argomento della `select` solamente un sottoinsieme degli attributi usati nella clausola `group by`. Per questi attributi, infatti, ciascuna tupla del sottoinsieme sarà caratterizzata dallo stesso valore.

Esempio 5

Consideriamo la seguente base dati:

$$\begin{array}{l}
 \text{IMPIEGATO}(\underline{\text{Nome}}, \underline{\text{Cognome}}, \text{Dipartimento}, \text{Ufficio}, \text{Stipendio}, \text{Citta}) \\
 \text{DIPARTIMENTO}(\underline{\text{Nome}}, \text{Indirizzo}, \text{Citta})
 \end{array} \quad (10.5)$$

Estrarre la somma degli stipendi di tutti gli impiegati dello stesso dipartimento.

```

12      select
        l.Dipartimento , sum(l.Stipendio)
14      from
        Impiegato as l

```

```
16      group by  
        I.Dipartimento
```

Esempio 5

Esempio 6

Consideriamo la seguente base dati:

IMPIEGATO(Nome, *Cognome*, *Dipartimento*, *Ufficio*, *Stipendio*, *Citta*)
DIPARTIMENTO(Nome, *Indirizzo*, *Citta*) (10.6)

Estrarre i dipartimenti, il numero di impiegati di ciascun dipartimento, e la città in cui il dipartimento ha sede.

```
10      select  
11          I.Dipartimento  
12          count(*)  
13          D.Citta  
14      from  
15          Impiegato as I  
16      join Dipartimento D  
17          on (I.Dipartimento = D.Nome)  
18      group by  
19          I.Dipartimento , D.Citta  
20
```

Esempio 6

10.2.1 Predicati sui gruppi

Abbiamo visto come, tramite la clausola `group by`, le righe possano venire raggruppate in sottoinsiemi. Un'applicazione può aver bisogno di considerare solo i sottoinsiemi che soddisfano certe condizioni.

Se le condizioni che i sottoinsiemi devono soddisfare sono verificabili al livello delle singole righe, allora basta porre gli opportuni predicati come argomento della clausola `where`.

Se invece le condizioni sono delle condizioni di tipo aggregato, sarà necessario utilizzare un nuovo costrutto, la clausola `having`.

La clausola `having` descrive le condizioni che si devono applicare al termine dell'esecuzione di un'interrogazione che fa uso della clausola `group by`. Ogni sottoinsieme di righe costruito dalla `group by` fa parte del risultato dell'interrogazione solo se il predicato argomento della `having` risulta soddisfatto.

Il funzionamento dell'interrogazione è piuttosto semplice: si procede seguendo gli stessi passi descritti per le interrogazioni con `group by`. Dopo aver raggruppato le righe in base agli attributi specificati, viene valutato il predicato argomento della clausola `having`, selezionando le righe che lo soddisfano.

La sintassi permette anche la definizione di interrogazioni che presentano la clausola `having` senza una corrispondente clausola `group by`.

In questo caso l'intero insieme di righe è trattato come un unico raggruppamento, ma questo ha in generale un limitato campo di applicabilità, perché, se la condizione non è soddisfatta il risultato sarà vuoto.

Come la clausola `where`, anche la clausola `having` ammette come argomento un'espressione booleana su predicati semplici. I predicati semplici sono normalmente confronti tra il risultato della valutazione di un operatore aggregato e una generica espressione; sintatticamente è ammessa anche la presenza diretta degli attributi argomento della `group by`, ma è preferibile raccogliere tutte le condizioni su questi attributi nell'ambito della clausola `where`.

Per sapere quali predicati di un'interrogazione che fa uso del raggruppamento vanno dati come argomento della clausola `where` e quali come argomento della clausola `having`, basta rispettare il seguente criterio: solo i predicati in cui compaiono operatori aggregati devono essere argomento della clausola `having`.

Esempio 7

Consideriamo la seguente base dati:

IMPIEGATO(Nome, Cognome, *Dipartimento*, *Ufficio*, *Stipendio*, *Citta*)
DIPARTIMENTO(Nome, *Indirizzo*, *Citta*) (10.7)

Estrarre i dipartimenti che spendono più di 100 in stipendi

```
10  select
11      l.Dipartimento ,
12      sum(l.Stipendio) as SommaStipendi
13  from
14      Impiegato as l
15  group by
16      l.Dipartimento
17  having
18      sum(Stipendio) > 100
```

Esempio 7

Esempio 8

Consideriamo la seguente base dati:

IMPIEGATO(Nome, Cognome, *Dipartimento*, *Ufficio*, *Stipendio*, *Citta*)
DIPARTIMENTO(Nome, *Indirizzo*, *Citta*) (10.8)

Estrarre i dipartimenti per cui la media degli stipendi degli impiegati che lavorano nell'ufficio 20 è superiore a 25.

```
10  select
    1. Dipartimento
12  from
    1. Impiegato as I
14  where
    1. Ufficio = 20
16  group by
    1. Dipartimento
18  having
    avg(I. Stipendio) > 25
20
```

Esempio 8

Capitolo 11

Operatori Insiemistici

SQL mette a disposizione anche degli operatori insiemistici, simili a quelli disponibili nell'algebra relazionale.

Gli operatori disponibili sono quelli di `union`, `intersect`, `except`, aventi significato analogo ai corrispettivi in algebra relazionale.

Notiamo che ogni interrogazione che faccia uso di `intersect` ed `except` può essere espressa utilizzando altri costrutti del linguaggio (tipicamente usando interrogazioni nidificate, esposte nel capitolo 12). Al contrario, il comando `union` arricchisce il potere espressivo di SQL e permette di scrivere interrogazioni altrimenti non formulabili.

La sintassi per l'uso degli operatori insiemistici è la seguente:

```
2  select ... SQL ...
   <union | intersect | except> [ all ]
4  select ... SQL ...
```

Gli operatori insiemistici, al contrario degli altri costrutti, assumono come default di eseguire un'eliminazione dei duplicati, interpretando meglio il tipico significato matematico di questi operatori. Qualora nell'interrogazione si voglia adottare una diversa interpretazione e si vogliano preservare i duplicati, è sufficiente specificare il parametro `all`.

SQL, a differenza dell'algebra relazionale, non richiede che gli schemi su cui vengono effettuate le operazioni insiemistiche siano identici, ma solo che gli attributi siano in pari numero e abbiano domini compatibili. La corrispondenza tra gli attributi non si basa sul nome ma sulla loro posizione: se gli attributi hanno nome diverso, il risultato normalmente usa i nomi del primo operando.

Esempio 1

Consideriamo la seguente base dati:

IMPIEGATO(Nome, Cognome, *Dipartimento*, *Ufficio*, *Stipendio*, *Citta*)
DIPARTIMENTO(Nome, *Indirizzo*, *Citta*) (11.1)

Estrarre nomi e cognomi degli impiegati.

```
select
  I.Nome
from
  Impiegato as I

union

select
  I.Cognome
from
  Impiegato as I
```

Esempio 1

Esempio 2

Consideriamo la seguente base dati:

IMPIEGATO(Nome, Cognome, *Dipartimento*, *Ufficio*, *Stipendio*, *Citta*)
DIPARTIMENTO(Nome, *Indirizzo*, *Citta*) (11.2)

Estrarre i nomi e i cognomi di tutti gli impiegati, eccetto quelli appartenenti al dipartimento *Amministrazione*, mantenendo i duplicati.

```
select
  I.Nome
from
  Impiegato as I
where
  I.Dipartimento <> 'Amministrazione'

union all

select
  I.Cognome
from
```

```

    Impiegato as I
where
    I.Dipartimento <> 'Amministrazione'

```

Esempio 2

Esempio 3

Consideriamo la seguente base dati:

$IMPIEGATO(\underline{Nome}, \underline{Cognome}, Dipartimento, Ufficio, Stipendio, Città)$
 $DIPARTIMENTO(\underline{Nome}, Indirizzo, Città)$
(11.3)

Estrarre i cognomi di impiegati che sono anche nomi

```

select
I.Nome
from
Impiegato as I

intersect

select
I.Cognome
from
Impiegato as I

```

Esempio 3

Esempio 3

Consideriamo la seguente base dati:

$IMPIEGATO(\underline{Nome}, \underline{Cognome}, Dipartimento, Ufficio, Stipendio, Città)$
 $DIPARTIMENTO(\underline{Nome}, Indirizzo, Città)$
(11.4)

Estrarre i nomi degli impiegati che non sono cognomi di qualche impiegato

```

select
I.Nome
from
Impiegato as I

except

```

```
44      select  
      I.Cognome  
46      from  
      Impiegato as I
```

Esempio 3

Capitolo 12

Query Nidificate

Finora abbiamo analizzato interrogazioni in cui l'argomento della clausola `where` si basa su condizioni composte da predicati semplici mediante gli operatori logici, in cui ciascun predicato rappresenta un semplice confronto tra due valori.

SQL ammette anche l'uso di predicati con una struttura più complessa, in cui si confronta un valore (ottenuto come risultato di un'espressione valutata sulla singola riga) con il risultato dell'esecuzione di un'interrogazione SQL. L'interrogazione che viene usata per il confronto viene definita direttamente nella clausola `where`.

Si parla in questo caso di **interrogazioni nidificate**.

Nel caso più tipico, l'espressione che compare come primo membro del confronto è il semplice nome di un attributo. Se in un predicato si confronta un attributo con il risultato di un'interrogazione, sorge il problema di **disomogeneità dei termini del confronto**. Infatti, da una parte abbiamo il risultato dell'esecuzione di un'interrogazione SQL (in generale un insieme di valori), mentre dall'altra abbiamo il valore dell'attributo per la particolare riga.

La soluzione offerta da SQL consiste nell'estendere, con le parole chiave `all` o `any`, i normali operatori di confronto.

La parola chiave `any` specifica che la riga soddisfa la condizione se risulta vero il confronto (con l'operatore specificato) tra il valore dell'attributo per la riga e almeno uno degli elementi restituiti dall'interrogazione.

La parola chiave `all` specifica che la riga soddisfa la condizione solo se tutti gli elementi restituiti dall'interrogazione nidificata rendono vero il confronto.

La sintassi richiede la compatibilità di dominio tra l'attributo restituito dall'interrogazione nidificata e l'attributo con cui avviene il confronto.

Esempio 1

Consideriamo la seguente base dati:

IMPIEGATO(Nome, Cognome, *Dipartimento*, *U f ficio*, *Stipendio*, *Citta*)
DIPARTIMENTO(Nome, *Indirizzo*, *Citta*) (12.1)

Estrarre gli impiegati che lavorano in dipartimenti situati a Firenze.

```

select
  *
from
  Impiegato
where
  Dipartimento = any (
    select Nome
    from Dipartimento
    where Citta = 'Firenze')

```

Esempio 1

Esempio 2

Consideriamo la seguente base dati:

IMPIEGATO(Nome, Cognome, *Dipartimento*, *U f ficio*, *Stipendio*, *Citta*)
DIPARTIMENTO(Nome, *Indirizzo*, *Citta*) (12.2)

Estrarre gli impiegati che hanno lo stesso nome di un impiegato del dipartimento *Produzione*.

```

select
  I1 . Nome
from
  Impiegato I1 ,
  Impiegato I2
where
  I1 . Nome = I2 . Nome
and I2 . Dipartimento = 'Produzione'

```

Esempio 2 –Formulazione a

```

select
  Nome
from
  Impiegato

```

```

6      where
      Nome = any (
8          select Nome
          from Impiegato
          where Dipartimento = 'Produzione')

```

Esempio 2 –Formulazione b

Esempio 3

Consideriamo la seguente base dati:

IMPIEGATO(Nome, *Cognome*, *Dipartimento*, *Ufficio*, *Stipendio*, *Citta*)
DIPARTIMENTO(Nome, *Indirizzo*, *Citta*) (12.3)

Estrarre i dipartimenti in cui non lavorano persone di cognome 'Rossi'

```

50      select
      Nome
52      from
      Dipartimento
54      where
      Nome <> all(
          select Dipartimento
56          from Impiegato
          where Cognome = 'Rossi')
58

```

Esempio 3

Esempio 4

Consideriamo la seguente base dati:

IMPIEGATO(Nome, *Cognome*, *Dipartimento*, *Ufficio*, *Stipendio*, *Citta*)
DIPARTIMENTO(Nome, *Indirizzo*, *Citta*) (12.4)

Estrarre il dipartimento dell'impiegato che guadagna lo stipendio massimo (usando l'operatore aggregato max)

```

1      select
      Dipartimento
3      from
      Impiegato
5      where
      Stipendio = any(

```

```
7      select max(Stipendio)
9      from Impiegato)
```

Esempio 4 –Formulazione a

Estrarre il dipartimento dell'impiegato che guadagna lo stipendio massimo (usando solo un'interrogazione nidificata).

```
2      select
3      Dipartimento
4      from
5      Impiegato
6      where
7      Stipendio >= all(
8      select Stipendio
9      from Impiegato)
```

Esempio 4 –Formulazione b

12.1 Interrogazioni nidificate complesse

Un'interpretazione molto semplice e intuitiva delle interrogazioni nidificate consiste nell'assumere che l'interrogazione nidificata venga eseguita prima di analizzare le righe dell'interrogazione esterna.

Il risultato dell'interrogazione può essere salvato in una tabella temporanea e il controllo sulle righe dell'interrogazione esterna può essere fatto accedendo direttamente al risultato temporaneo. Questa interrogazione corrisponde, tra l'altro, a un meccanismo di esecuzione efficiente, in cui l'interrogazione nidificata viene eseguita una sola volta.

Talvolta l'interrogazione nidificata fa riferimento al contesto dell'interrogazione che la racchiude; tipicamente ciò accade tramite una variabile definita nell'ambito della query più esterna e usata nell'ambito della query più interna (si parla di un **passaggio di binding** da un contesto all'altro).

La presenza del meccanismo di passaggio di binding arricchisce il potere espressivo di SQL. In questo caso l'interpretazione semplice data precedentemente alle query nidificate non vale più; bisogna, a questo punto, ricostruire l'interpretazione standard delle interrogazioni SQL, per cui prima si costruisce il prodotto cartesiano delle tabelle e successivamente si applicano a ciascuna riga del prodotto le condizioni che compaiono nella clausola *where*. L'interrogazione nidificata è un componente della clausola *where* e dovrà anch'essa essere valutata separatamente per ogni riga prodotta nella valutazione della query esterna.

Così, la nuova interpretazione è la seguente: per ogni riga della query esterna, valutiamo per prima cosa la query nidificata, quindi calcoliamo il predicato a livello di riga

sulla query esterna. Tale processo può essere ripetuto un numero arbitrario di volte, pari al numero di nidificazioni che possono essere utilizzate nella query; con query così complicate si perdono però le caratteristiche di leggibilità di SQL.

Per quanto riguarda la visibilità (o **scope**) delle variabili SQL, vale la restrizione che una variabile è utilizzabile solo all'interno della query in cui è definita o nell'ambito di una query nidificata (a qualsiasi livello) all'interno di essa.

Se un'interrogazione possiede interrogazioni nidificate allo stesso livello (su predicati distinti), le variabili introdotte nella clausola *from* di una query non potranno essere usate nell'ambito di un'altra query.

Introduciamo ora l'operatore logico *exists*, che ammette come parametro un'interrogazione nidificata e restituisce il valore vero solo se l'interrogazione fornisce un risultato non vuoto.

Questo operatore può essere utilizzato in modo significativo solo quando si ha un passaggio di binding tra l'interrogazione esterna e quella nidificata.

Esempio 5

Consideriamo la seguente base dati:

PERSONA(CodFiscale, Nome, Cognome, Città) (12.5)

Estrarre le persone che hanno degli omonimi (ovvero persone con lo stesso nome e cognome, ma diverso codice fiscale) usando le query nidificate.

```

1  select *
2  from Persona P\
3  where exists (
4      select *
5      from Persona P2
6      where
7          P1.Nome = P2.Nome
8          and P1.Cognome = P2.Cognome
9          and P1.CodFiscale <> P2.CodFiscale)
10

```

Esempio 5 –Formulazione a

Estrarre le persone che hanno degli omonimi (ovvero persone con lo stesso nome e cognome, ma diverso codice fiscale) senza usare le query nidificate.

```

1  select P.*
2  from
3      Persona P1,
4      Persona P2
5

```

```
5      where
      P1.Nome = P2.Nome
7      and P1.Cognome = P2.Cognome
      and P1.CodFiscale <> P2.CodFiscale
9
```

Esempio 5 –Formulazione b

Esempio 6

Consideriamo la seguente base dati:

$$\begin{array}{l} CANTANTE(\underline{Nome}, \underline{Canzone}) \\ AUTORE(\underline{Nome}, \underline{Canzone}) \end{array} \quad (12.6)$$

Estrarre i cantautori puri, ovvero i cantanti che hanno eseguito solo canzoni di cui erano anche autori.

```
2      select Nome
      from Cantante
      where Nome not in (
4          select Nome
          from Cantante C
          where Nome not in (
6              select nome
              from Autore
              where Autore.Canzone = C.Canzone))
8
10
```

Esempio 6 –Formulazione a

```
1      select Nome
      from Cantante
3
      except
5
      select Nome
      from Cantante C
7      where Nome not in (
9          select Nome
          from Autore
          where Autore.Canzone = C.Canzone)
11
```

Esempio 6 –Formulazione b

Esempio 5

Consideriamo la seguente base dati:

PERSONA(CodFiscale, Nome, Cognome, Città) (12.7)

Estrarre le persone che hanno degli omonimi (ovvero persone con lo stesso nome e cognome, ma diverso codice fiscale) usando le query nidificate.

```
12 select *  
13 from Persona P\  
14 where exists (  
15   select *  
16   from Persona P2  
17   where  
18     P1.Nome = P2.Nome  
19   and P1.Cognome = P2.Cognome  
20   and P1.CodFiscale <> P2.CodFiscale)
```

Esempio 5 –Formulazione a

Estrarre le persone che hanno degli omonimi (ovvero persone con lo stesso nome e cognome, ma diverso codice fiscale) senza usare le query nidificate.

```
10 select P.*  
11 from  
12   Persona P1,  
13   Persona P2  
14 where  
15   P1.Nome = P2.Nome  
16 and P1.Cognome = P2.Cognome  
17 and P1.CodFiscale <> P2.CodFiscale
```

Esempio 5 –Formulazione b

Capitolo 13

Modifica dei dati

La parte di DML (Data Manipulation Language) comprende i comandi per interrogare e modificare il contenuto della base di dati.

13.1 Inserimento

Il comando di inserimento di righe nella base di dati presenta due sintassi alternative: la prima permette di inserire **singole righe** all'interno delle tabelle. L'argomento della clausola `values` rappresenta esplicitamente i valori degli attributi della singola riga.

```
2      insert into NomeTabella [ ListaAttributi ]  
      values ( ListaValori )
```

La seconda forma, invece, permette di aggiungere **insiemi di righe**, estratti dal contenuto della base di dati.

```
2      insert into NomeTabella [ ListaAttributi ]  
      select ...
```

Ciascuna forma del comando possiede uno specifico campo di applicazione: la prima forma è quella tipicamente usata all'interno dei programmi per riempire una tabella con i dati forniti direttamente dagli utenti.

Ogni uso del comando di `insert` è generalmente associato al riempimento di una **maschera** (o **form**), ovvero un'interfaccia di facile uso in cui all'utente vengono presentati sul video il nome dei vari attributi e appositi spazi in cui immettere i relativi valori.

La seconda forma permette invece di inserire dati in una tabella a partire da altre informazioni presenti nella base di dati.

Se in un inserimento non vengono specificati i valori di tutti gli attributi della tabella, agli attributi mancanti viene assegnato il valore di default, o in assenza di questo, il valore nullo; se l'inserimento viola un vincolo `not null` definito sull'attributo, l'inserimento viene rifiutato.

Si noti, infine, che la corrispondenza tra gli attributi della tabella e i valori da inserire è data dall'ordine in cui compaiono i termini nella definizione della tabella. Perciò, al primo attributo che compare in `ListaValori` (per la prima forma del comando) o al primo elemento della clausola `select` (per la seconda forma) deve corrispondere il primo attributo che compare in `ListaAttributi` (o nella definizione della tabella se `ListaAttributi` è omesso), e così via per gli altri attributi.

13.2 Cancellazione

Il comando `delete` elimina righe dalle tabelle della base di dati, seguendo la semplice sintassi:

```
delete from NomeTabella where [Condizione]
```

Quando la condizione argomento della clausola `where` non viene specificata, il comando cancella tutte le righe della tabella, altrimenti vengono rimosse solo le righe che soddisfano la condizione.

Qualora esista un vincolo di integrità referenziale con politica di cascade in cui la tabella viene referenziata, allora la cancellazione di righe dalla tabella può comportare la cancellazione di righe appartenenti ad altre tabelle (e si può generare una reazione a catena se queste cancellazioni a loro volta causano la cancellazione di righe di altre tabelle).

13.3 Modifica

Il comando `update` presenta una sintassi più complicata rispetto agli altri due:

```
2      update NomeTabella
      set Attributo = <Espressione | select ...SQL... | null
      | default>
      {, Attributo = <Espressione | select ...SQL... | null |
4      default> }
      [where Condizione]
```

Il comando `update` permette di aggiornare uno o più attributi delle righe di `NomeTabella` che soddisfano l'eventuale `Condizione`.

Se il comando non presenta la clausola `where`, si suppone che la condizione sia soddisfatta e si esegue la modifica su tutte le righe. Il nuovo valore cui viene posto l'attributo può essere:

1. Il risultato della valutazione di un'espressione sugli attributi della tabella, che può anche far riferimento al valore corrente dell'attributo che verrà modificato dal comando.
2. Il risultato di una generica interrogazione SQL
3. Il valore nullo
4. Il valore di default per il dominio

La natura **set-oriented** di SQL presenta alcune particolarità di cui è necessario tenere conto.

Consideriamo lo schema

IMPIEGATO(Nome, *Cognome*, *Dipartimento*, *Ufficio*, *Stipendio*, *Citta*)
DIPARTIMENTO(Nome, *Indirizzo*, *Citta*) (13.1)

Vogliamo modificare gli stipendi dei dipendenti, aumentando del 10% gli stipendi inferiori a 30, e del 15% agli stipendi superiori.

Un modo per aggiornare in questo modo la base dati consiste nell'eseguire questo comando:

```
2      update Impiegato
3      set I.Stipendio = I.Stipendio * 1.1
4      where I.Stipendio <= 30
5
6      update Impiegato
7      set I.Stipendio = I.Stipendio * 1.15
8      where I.Stipendio > 30
```

Il problema di questa soluzione è che se consideriamo un dipendente con uno stipendio iniziale di 30, questo soddisferà la condizione del primo comando di aggiornamento, per cui l'attributo *Stipendio* verrà posto pari a 33. Ma, a questo punto, la riga soddisferà anche le condizioni del secondo comando di aggiornamento, per cui lo stipendio sarà nuovamente modificato.

Il risultato finale è che per questa riga l'aumento complessivo sarà del 26.5%, violando quindi i requisiti di partenza.

Il problema ha origine nel carattere *set-oriented* di SQL. Con un linguaggio *tuple-oriented* sarebbe possibile analizzare le righe una a una e applicare o l'una o l'altra delle modifiche a seconda del valore dello stipendio.

In questo caso una semplice soluzione consiste nell'invertire l'ordine di esecuzione dei due comandi, aumentando prima gli stipendi superiori e poi i rimanenti.

In casi più complicati risulta necessario introdurre degli aggiornamenti intermedi, fare uso

di costrutti avanzati, oppure cambiare completamente approccio e scrivere un programma in un tradizionale linguaggio di programmazione.

Parte III

Progettazione

Capitolo 14

Metodologie e Modelli per il progetto

14.1 Introduzione alla progettazione

14.1.1 Il ciclo di vita dei sistemi informativi

La progettazione di una base di dati costituisce solo una delle componenti del processo di sviluppo di un sistema informativo, e va quindi inserita in un contesto più ampio, quello del **ciclo di vita** di un sistema informatico.

Il ciclo di vita di un sistema informativo comprende, generalmente, le seguenti attività:

- **Studio di fattibilità:** serve a definire, in maniera per quanto possibile precisa, i costi delle varie alternative possibili e a stabilire le priorità di realizzazione delle varie componenti del sistema.
- **Raccolta e analisi dei requisiti:** consiste nell'individuazione e nello studio delle proprietà e delle funzionalità che il sistema informativo dovrà avere.
Questa fase richiede un'interazione con gli utenti del sistema e produce una descrizione completa, ma generalmente informale, dei dati coinvolti (anche in termini di previsione sul carico applicativo) e delle operazioni su di essi (anche in termini di previsione della loro frequenza).
Vengono inoltre stabiliti i requisiti software e hardware del sistema informativo.
- **Progettazione:** si divide generalmente in **progettazione dei dati** e **progettazione delle applicazioni**.
Nella prima si individua la struttura e l'organizzazione che i dati dovranno avere, nell'altra si definiscono le caratteristiche dei programmi applicativi.
Le due attività sono complementari e possono procedere in parallelo o in cascata.
Le descrizioni dei dati e delle applicazioni prodotte in questa fase sono formali e fanno riferimento a specifici modelli.

- **Implementazione:** consiste nella realizzazione del sistema informativo secondo la struttura e le caratteristiche definite nella fase di progettazione. Viene costruita e popolata la base di dati e viene prodotto il codice dei programmi.
- **Validazione e collaudo:** serve a verificare il corretto funzionamento e la qualità del sistema informativo. La sperimentazione deve prevedere, per quanto possibile, tutte le condizioni operative.
- **Funzionamento:** in questa fase il sistema informativo diventa operativo ed esegue i compiti per i quali era stato originariamente progettato. Se non si verificano malfunzionamenti o revisioni delle funzionalità del sistema, questa attività richiede solo operazioni di gestione e manutenzione.

Il processo non è quasi mai strettamente sequenziale in quanto, spesso, durante l'esecuzione di una delle attività citate, bisogna rivedere decisioni prese nell'attività precedente. Si ottiene proprio un **ciclo** di operazioni.

Inoltre, si aggiunge talvolta alle attività citate quella di **prototipazione**, che consiste nell'uso di specifici strumenti software per la realizzazione rapida di una versione semplificata del sistema informativo.

Le basi di dati costituiscono in effetti solo una delle componenti del sistema, ma il loro ruolo centrale i dati hanno in un sistema informativo giustifica uno studio autonomo relativo alla **progettazione delle basi di dati**.

Ci interesseremo solo agli aspetti dello sviluppo dei sistemi informativi che riguardano da vicino il progetto delle basi di dati, focalizzando l'attenzione in particolare sulla terza fase del ciclo, discutendo anche alcuni aspetti riguardanti l'attività di raccolta e analisi dei dati.

Questa maniera di procedere è coerente con l'approccio allo sviluppo dei sistemi informativi basato sui dati, in cui l'attenzione è centrata sui dati e sulle loro proprietà.

Questo approccio prevede prima la progettazione della base di dati e, successivamente, la realizzazione delle applicazioni che la utilizzano.

14.1.2 Metodologie di progettazione e basi di dati

Un aspetto che vale la pena di precisare è che cosa si intende per **metodologia di progettazione** e quali sono le priorità che una metodologia deve garantire.

In buona sostanza, una metodologia di progettazione consiste in:

- Una **decomposizione** dell'intera attività di progetto in passi successivi indipendenti tra loro.
- Una serie di **strategie** da seguire nei vari passi e alcuni **criteri** per la scelta in caso di alternative.

- Alcuni **modelli di riferimento** per descrivere i dati di ingresso e uscita delle varie fasi.

Le proprietà che una metodologia deve garantire sono principalmente:

- La **generalità** rispetto alle applicazioni e ai sistemi in gioco (e quindi la possibilità di utilizzo indipendentemente dal problema allo studio e dagli strumenti a disposizione).
- La **qualità del prodotto** in termini di correttezza, competenza ed efficienza rispetto alle risorse impiegate.
- La **facilità d'uso** delle strategie e dei modelli di riferimento.

Nell'ambito delle basi di dati si è consolidata negli anni una metodologia di progetto che ha dato prova di soddisfare pienamente queste tre proprietà.

Tale metodologia è articolata in tre fasi principali da effettuare in cascata e si fonda sul seguente principio: separare in maniera netta le decisioni relative a *cosa* rappresentare in una base di dati (prima fase), da quelle relative a *come* farlo (seconda e terza fase).

- **Progettazione concettuale:** in questa fase rappresentiamo le specifiche informali della realtà di interesse in termini di una descrizione formale e completa, ma indipendente dai criteri di rappresentazione utilizzati nei sistemi di gestione di basi di dati.

Il prodotto di questa fase è detto **schema concettuale** e fa riferimento a un **modello concettuale** dei dati.

I modelli concettuali ci permettono di descrivere l'organizzazione dei dati a un alto livello di astrazione, senza tenere conto degli aspetti implementativi.

In questa fase, infatti, il progettista deve cercare di rappresentare il **contenuto informativo** della base di dati, senza occuparsi né delle modalità con le quali queste informazioni verranno codificare in un sistema reale, né dell'efficienza dei programmi che faranno uso di queste informazioni.

- **Progettazione logica:** in questa fase traduciamo lo schema concettuale definito nella fase precedente, in termini del modello di rappresentazione dei dati adottato dal sistema di gestione di base di dati a disposizione.

Il prodotto di questa fase viene denominato **schema logico** della base di dati e fa riferimento a un **modello logico** dei dati.

Un modello logico ci permette di descrivere i dati secondo una rappresentazione ancora indipendente dai dettagli fisici, ma concreta perché disponibile nei sistemi di gestione di base di dati.

In questa fase, le scelte progettuali si basano, tra l'altro, su criteri di ottimizzazione delle operazioni da effettuare sui dati.

Si fa comunemente uso anche di tecniche formali di verifica della qualità dello schema logico ottenuto. Nel caso del modello relazionale dei dati, la tecnica comunemente utilizzata è quella della **normalizzazione**.

- **Progettazione fisica:** in questa fase lo schema logico viene completato con la specifica dei parametri fisici di memorizzazione dei dati (organizzazione dei file e degli indici).
Il prodotto di questa fase viene denominato **schema fisico** e fa riferimento a un **modello fisico** dei dati.

È necessario fare una distinzione tra **specifiche sui dati**, che riguardano il contenuto della base di dati, e **specifiche sulle operazioni**, che riguardano l'uso che utenti e applicazioni fanno della base di dati.

Nella progettazione concettuale si fa uso soprattutto delle specifiche sui dati, mentre le specifiche sulle operazioni servono solo a verificare che lo schema concettuale sia completo, contenga cioè le informazioni necessarie per eseguire tutte le operazioni previste.

Nella progettazione logica, invece, lo schema concettuale in ingresso riassume le specifiche sui dati, mentre le specifiche sulle operazioni si utilizzano, insieme alle previsioni sul carico applicativo, per ottenere uno schema logico che renda tali operazioni eseguibili in maniera efficiente.

In questa fase bisogna anche conoscere il modello logico adottato, ma non è ancora necessario conoscere il particolare DBMS scelto (solo la categoria cui appartiene).

Infine, nella progettazione fisica si fa uso dello schema logico e delle specifiche sulle operazioni per ottimizzare le prestazioni del sistema.

In questa fase è necessario anche tenere conto delle caratteristiche del particolare sistema di gestione di basi di dati utilizzato.

Il risultato della progettazione di una base di dati non è solo lo schema fisico, ma è costituito anche dallo schema concettuale e dallo schema logico.

Lo schema concettuale fornisce infatti una rappresentazione della base di dati di alto livello, che può essere molto utile a scopo documentativo, mentre lo schema logico fornisce una descrizione concreta del contenuto della base di dati che, prescindendo dagli aspetti implementativi, è il riferimento per le operazioni di interrogazione e aggiornamento.

Nel caso della progettazione di una base di dati relazionale basata sull'uso del più diffuso modello concettuale dei dati, il modello Entità-Relazione.

A partire da requisiti rappresentati da documenti e moduli di vario genere, acquisiti anche attraverso l'interazione con gli utenti, viene costruito uno schema E-R che descrive a livello concettuale la base di dati.

Questa rappresentazione viene poi tradotta in uno schema relazionale, costituito da una collezione di tabelle.

Infine, i dati vengono descritti da un punto di vista fisico (tipo e dimensione dei vari campi) e vengono specificate strutture ausiliarie, come gli indici, per l'accesso efficiente ai dati.

14.2 Il modello E-R

Il modello Entità-Relazione è un **modello concettuale** di dati, e, come tale, fornisce una serie di strutture, dette **costrutti**, atte a descrivere la realtà d'interesse in una maniera facile da comprendere e che prescinde dai criteri di organizzazione dei dati nei calcolatori. Questi costrutti vengono utilizzati per definire **schemi** che descrivono l'organizzazione e la struttura delle **occorrenze** dei dati, ovvero, dei valori assunti al variare del tempo.

I costrutti del modello E-R sono presentati nella tabella 14.1.

14.2.1 I costrutti principali del modello

Entità. Rappresentano classi di oggetti (es. fatti, cose, persone) che hanno proprietà comuni ed esistenza "autonoma" ai fini dell'applicazione d'interesse.

Si osservi che un'occorrenza di entità non è un valore che identifica un oggetto, ma è l'oggetto stesso.

Un'interessante conseguenza di questo fatto è che un'occorrenza di entità ha un'esistenza (e un'identità) indipendente dalle proprietà ad esso associate.

In questo il modello E-R presenta una marcata differenza rispetto al modello relazionale nel quale non possiamo rappresentare un oggetto senza conoscere alcune sue proprietà.

In uno schema, ogni entità ha un nome che la identifica univocamente e viene rappresentata graficamente mediante un rettangolo con il nome dell'entità all'interno.

Relazioni. Rappresentano legami logici, significativi per l'applicazione d'interesse tra due o più entità.

Un'occorrenza di relazione è un'ennupla (coppia nel caso di relazione binaria) costituita da occorrenze di entità, una per ciascuna delle entità coinvolte.

In uno schema E-R, ogni relazione ha un nome che la identifica univocamente e viene rappresentata graficamente mediante un rombo, con il nome della relazione all'interno, e da linee che connettono la relazione con ciascuna delle sue componenti.

Possono esistere diverse che coinvolgono le stesse entità (E.G. IMPIEGATO e CITTA' possono essere in relazione SEDE oppure RESIDENZA).

Nella scelta dei nomi di relazione è preferibile utilizzare sostantivi invece che verbi, in maniera da non indurre ad assegnare un "verso" alla relazione. (E.g. SEDE DI LAVORO è preferibile a LAVORA IN).

L'insieme delle occorrenze di una relazione del modello E-R è, a tutti gli effetti, una relazione matematica tra le occorrenze delle entità coinvolte, ossia, è un sottoinsieme del loro prodotto cartesiano.



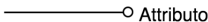
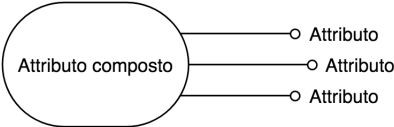


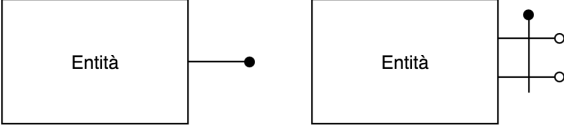
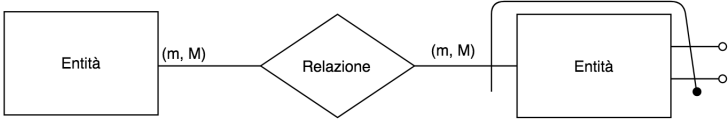

Questo significa che tra le occorrenze di una relazione del modello E-R non ci possono essere ennuple ripetute.

È anche possibile avere relazioni **ricorsive**, ovvero relazioni tra un'entità e se stessa.

È infine possibile avere relazioni *n*-arie, relazioni, cioè, che coinvolgono più di due entità.

Attributi. Descrivono le proprietà elementari di entità o relazioni che sono di interes-

Tabella 14.1: Costrutti del modello ER

Costrutti	Rappresentazione grafica
Entità	
Relazione	
Attributo semplice	
Attributo composto	
Cardinalità di relazione	
Cardinalità di attributo	
Identificatore interno	
Identificatore esterno	
Generalizzazione	

se ai fini dell'applicazione.

Un attributo associa a ciascuna occorrenza di entità (o di relazione) un valore appartenente a un insieme, detto **dominio**, che contiene i valori ammissibili per l'attributo. I domini non vengono riportati nello schema, ma sono generalmente descritti nella documentazione associata.

Può risultare comodo, qualche volta, raggruppare attributi di una medesima entità o relazione che presentano affinità nel loro significato o uso: l'insieme di attributi che si ottiene in questa maniera viene detto **attributo composto**. Per ridurre la complessità degli schemi, gli attributi composti verranno usati raramente nel seguito preferendo usare, per quanto possibile, attributi atomici.

Cardinalità delle relazioni. Vengono specificate per ciascuna partecipazione di entità a una relazione e descrivono il numero minimo e massimo di occorrenze di relazione a cui una occorrenza dell'entità può partecipare. Dicono quindi quante volte, in una relazione tra entità, un'occorrenza di una di queste entità può essere legata a occorrenze delle altre entità coinvolte.

E.G.: se in una relazione ASSEGNAIMENTO tra due entità IMPIEGATO e INCARICO specifichiamo per la prima entità una cardinalità minima 1 a 1 e una cardinalità massima pari a 5, vogliamo indicare che un impiegato può partecipare a un minimo di 1 occorrenza e a un massimo di 5 occorrenze della relazione ASSEGNAIMENTO.

In linea di principio è possibile assegnare un qualunque intero non negativo a una cardinalità di una relazione con l'unico vincolo che la cardinalità minima deve essere minore o uguale della cardinalità massima. In realtà, nella maggior parte dei casi, è sufficiente utilizzare solo tre valori: zero, uno e il simbolo N (che indica genericamente un intero maggiore di uno). In particolare:

- Per la cardinalità minima, zero o uno: nel primo caso si dice che la partecipazione dell'entità relativa è **opzionale**, nel secondo si dice che la partecipazione è **obbligatoria**.
- Per la cardinalità massima, uno o molti (N): nel primo caso la partecipazione dell'entità relativa può essere vista come una funzione (parziale se la cardinalità minima vale zero) che associa a un'occorrenza dell'entità una sola occorrenza (o nessuna) dell'altra entità che partecipa alla relazione; nel secondo, invece, c'è un'associazione con un numero arbitrario di occorrenze dell'altra entità.

Osservando le cardinalità massime, è possibile classificare le relazioni binarie in base al tipo di corrispondenza che viene stabilita tra le occorrenze delle entità coinvolte:

- Le relazioni aventi cardinalità massima pari a uno per entrambe le entità coinvolte definiscono una corrispondenza uno a uno tra le occorrenze di tali entità e vengono quindi denominate **relazioni uno a uno**.
- Le relazioni aventi un'entità cardinalità massima pari a uno e l'altra con cardinalità massima pari a N sono denominate **relazioni uno a molti**.

- Le relazioni aventi cardinalità massima pari a N per entrambe le entità coinvolte, vengono denominate **relazioni molti a molti**.

Per le cardinalità minime, invece, è importante notare che il caso di partecipazione obbligatoria per tutte le entità coinvolte è piuttosto raro, perché quando si aggiunge una nuova occorrenza di entità molto spesso non sono note (o addirittura non esistono) le corrispondenti occorrenze delle entità a esse collegate.

Le relazioni non sono necessariamente binarie, possono coinvolgere n attributi, e in tal caso sono dette n -arie. Nelle relazioni n -arie le entità coinvolte partecipano quasi sempre con cardinalità massima pari a N .

Nel caso in cui un'entità partecipa a una relazione n -aria con cardinalità massima pari a uno, significa che ogni sua occorrenza può essere legata a una sola occorrenza della relazione e quindi a un'unica ennupla di occorrenze delle altre entità coinvolte nella relazione. Questo significa che è possibile (e risulta a volte più naturale) eliminare la relazione n -aria e legare direttamente tale entità con altre entità, mediante delle relazioni binarie di tipo uno a molti.

Cardinalità degli attributi. Possono essere specificate per gli attributi di entità o relazioni e descrivono il numero minimo e massimo di valori dell'attributo associati a ogni occorrenza di entità o relazione. Nella maggior parte dei casi, la cardinalità di un attributo è pari a (1, 1) e viene omessa. In questi casi l'attributo rappresenta sostanzialmente una funzione che associa a ogni occorrenza di entità un solo valore dell'attributo.

Il valore per un certo attributo può essere però nullo, oppure possono esistere diversi valori di un certo attributo per un'occorrenza di entità. Queste situazioni si possono rappresentare associando all'attributo in questione una cardinalità minima pari a zero nel primo caso e una cardinalità massima pari a N , nel secondo.

In maniera simile alle partecipazioni delle occorrenze di entità alle relazioni, diremo che un attributo con cardinalità minima pari a zero è **opzionale** per la relativa entità o relazione, mentre è **obbligatorio** se la cardinalità minima è pari a uno. Diremo che un attributo è **multivalore** se la sua cardinalità massima è pari a N .

In molte situazioni reali accade che certe informazioni non sono disponibili, ed è quindi utile avere la possibilità di specificare attributi opzionali. Gli attributi multivalore vanno invece utilizzati con maggiore cautela, perché essi rappresentano situazioni che possono essere modellate, in alcune occasioni con entità a sé, legate da relazioni uno a molti (o molti a molti) con l'entità cui si riferiscono.

Identificatori delle entità. Vengono specificati per ciascuna entità di uno schema e descrivono i concetti (attributi e/o entità) dello schema che permettono di identificare in maniera univoca le occorrenze delle entità.

- In molti casi uno o più attributi di un'entità sono sufficienti a individuare un identificatore: si parla in questo caso di **identificatore interno** (detto anche **chiave**).
E.G.: un identificatore interno per l'entità AUTOMOBILE con attributi Targa, Colore, Modello è l'attributo Targa, in quanto non possono esistere due automobili con la stessa Targa.

- Alcune volte, però, gli attributi di un'entità non sono sufficienti a identificare univocamente le sue occorrenze.

E.G.: Consideriamo l'entità *STUDENTE*: potrebbe sembrare, a prima vista, che l'attributo *Matricola* possa essere un identificatore per tale entità, ma ciò non è vero: studenti iscritti a università diverse possono avere lo stesso numero di matricola. Per identificare univocamente uno studente serve, oltre al numero di matricola, anche la relativa università. Quindi, un identificatore corretto per l'entità *STUDENTE* è costituito dall'attributo *Matricola* e dell'entità *UNIVERSITA'*.

Osserviamo che un'entità E può essere identificata da altre entità solo se tali entità sono coinvolte in una relazione a cui E partecipa con cardinalità $(1, 1)$. Nei casi in cui l'identificazione di un'entità è ottenuta utilizzando altre entità si parla di **identificatore esterno**.

E.G.: Nell'esempio precedente, l'identificazione è resa possibile dalla relazione uno a molti tra le entità *UNIVERSITA'* e *STUDENTE*, che associa a ogni studente una e una sola università. Se questa relazione non esistesse, l'identificazione univoca attraverso un'altra entità non sarebbe possibile.

Sulla base di quanto detto sugli identificatori, è possibile fare alcune considerazioni generali:

- Un identificatore può coinvolgere uno o più attributi, ognuno dei quali deve avere cardinalità $(1, 1)$
- Un'identificazione esterna può coinvolgere una o più entità, ognuna delle quali deve essere membro di una relazione alla quale l'entità da identificare partecipa con cardinalità $(1, 1)$
- Un'identificazione esterna può coinvolgere un'entità che è a sua volta identificata esternamente, purché non vengano generati, in questa maniera, cicli di identificazioni esterne.
- Ogni entità deve avere almeno un identificatore (interno o esterno), ma ne può avere in generale più d'uno; nel caso di più identificatori, gli attributi e le entità coinvolte in alcune identificazioni (tranne una) possono essere opzionali (cardinalità minima uguale a zero).

Generalizzazioni. Rappresentano legami logici tra un'entità E , detta entità **genitore** e una o più entità E_1, \dots, E_n dette entità **figlie**, di cui E è più generale, nel senso che le comprende come caso particolare. Si dice in questo caso che E è **generalizzazione** di E_1, \dots, E_n e che le entità E_1, \dots, E_n sono **specializzazioni** dell'entità E .

Tra le entità coinvolte in una generalizzazione valgono le seguenti proprietà generali:

- Ogni occorrenza di un'entità figlia è anche un'occorrenza dell'entità genitore.
- Ogni proprietà dell'entità genitore (attributi, identificatori, relazioni e altre generalizzazioni) è anche una proprietà delle entità figlie. Questa proprietà delle generalizzazioni è nota sotto il nome di **ereditarietà**.

Le generalizzazioni possono essere classificate sulla base di due proprietà tra loro ortogonali:

- Una generalizzazione è **totale** se ogni occorrenza dell'entità genitore è un'occorrenza di almeno una delle entità figlie, altrimenti è **parziale**.
- Una generalizzazione è **esclusiva** se ogni occorrenza dell'entità genitore è al più un'occorrenza di una delle entità figlie, altrimenti è **sovrapposta**.

Esempi:

- Una generalizzazione tra PERSONA, UOMO e DONNA è totale (gli uomini e le donne costituiscono tutte le persone) ed esclusiva (una persona o è uomo o è donna).
- Una generalizzazione tra l'entità PROFESSIONISTA, INGEGNERE e DOTTORE è invece parziale (esistono altre professioni oltre a queste due) ed esclusiva (assumiamo che ogni professionista abbia una sola professione principale).
- **E.G. 3:** Una generalizzazione tra l'entità PERSONA, STUDENTE e LAVORATORE è una generalizzazione parziale (esistono persone che non sono né studenti né lavoratori) e sovrapposta (esistono studenti che sono anche lavoratori).

In realtà le generalizzazioni sovrapposte possono essere facilmente trasformate in generalizzazioni esclusive aggiungendo una o più entità figlie, per rappresentare i concetti che costituiscono le intersezioni delle entità che si sovrappongono.

E.G.: nel caso degli studenti e dei lavoratori è sufficiente aggiungere l'entità STUDENTELAVORATORE per ottenere una generalizzazione esclusiva.

Quindi, nel seguito assumeremo che le generalizzazioni siano sempre esclusive, in quanto questa scelta rende più semplice la traduzione verso il modello relazionale.

14.3 Panoramica finale sul modello E-R

Abbiamo detto più volte che gli schemi E-R costituiscono utili strumenti nell'attività di progettazione di basi di dati. In realtà, tali schemi fornendo rappresentazioni astratte dei dati di un'applicazione, possono essere utilizzati con profitto anche per attività non strettamente legate alla progettazione.

Si possono fare i seguenti esempi:

- Gli schemi E-R possono essere utilizzati a scopo documentativo, poiché sono facilmente comprensibili anche da non specialisti di basi di dati.
- Gli schemi E-R possono essere utilizzati per descrivere i dati di un sistema informativo già esistente (per esempio, per integrarlo con altri) e, nel caso di sistema costituito da diversi sottosistemi, c'è il vantaggio di poter rappresentare le varie componenti con un linguaggio astratto e quindi unificante.

- Gli schemi E-R possono essere utilizzati per comprendere, in caso di modifica dei requisiti di un'applicazione, su quali porzioni del sistema si deve operare e in cosa consistono le modifiche da effettuare.

14.4 Documentazione di schemi E-R

Uno schema E-R non è quasi mai sufficiente, da solo, a rappresentare nel dettaglio tutti gli aspetti di un'applicazione, per varie ragioni.

- Innanzitutto in uno schema E-R compaiono solo i nomi dei vari concetti in esso presenti ma questo può essere insufficiente per comprenderne il significato. Nel caso di schemi particolarmente complessi, inoltre, può accadere di non riuscire a rappresentare in maniera comprensibile ed esaustiva i vari concetti.
- Infine, in certi casi addirittura impossibile rappresentare alcune proprietà dei dati attraverso i costrutti che il modello E-R mette a disposizione. Per esempio, è impossibile rappresentare proprietà che fanno riferimento a due concetti indipendenti (direzione e afferenza) descritti da due relazioni e non esistono costrutti del modello che ci permettono di correlare due relazioni. Inoltre, non è possibile esprimere proprietà che corrispondono a vincoli di integrità sui dati. Infatti, mentre il modello E-R è sufficientemente espressivo per rappresentare dati, risulta meno adatto a rappresentare vincoli complessi su di essi.

In conclusione, risulta indispensabile corredare ogni schema E-R con una documentazione di supporto, che possa servire a facilitare l'interpretazione dello schema stesso e a descrivere proprietà dei dati rappresentati che non possono essere espresse direttamente dai costrutti del modello.

Le strutture utilizzate per documentare uno schema E-R non vanno intese come nuovi costrutti di rappresentazione, ma come strumenti, non formali, atti a completare e arricchire la descrizione dei dati di un'applicazione fatta con un modello concettuale. Vanno quindi considerate come strumenti di supporto all'analisi concettuale ma non possono certamente sostituirsi a essa.

14.4.1 Regole aziendali

Uno degli strumenti più utilizzati per la descrizione di proprietà di un'applicazione che non sono rappresentabili direttamente con i modelli concettuali è quello delle **regole aziendali (business rules)**.

Questa terminologia deriva dal fatto che, nella maggior parte dei casi, si vuole esprimere proprio una "regola" del particolare dominio applicativo che stiamo esaminando.

Possiamo classificare le regole aziendali a seconda della loro natura:

- *Descrizione di un concetto* rilevante per l'applicazione, ovvero la descrizione precisa di un'entità, di un'attributo o di una relazione del modello ER.
- *Vincolo di integrità* sui dati dell'applicazione, sia esso la documentazione di un vincolo espresso con qualche costrutto del modello ER, o la descrizione di un vincolo non esprimibile mediante i costrutti del modello.
- *Derivazione*: un concetto che può essere ottenuto, attraverso un'inferenza o un calcolo aritmetico, da altri concetti dello schema.

Per le regole del primo tipo si fa in genere ricorso a frasi in linguaggio naturale.

Le regole che descrivono vincoli di integrità e derivazioni sono, invece, più adatte a definizioni formali.

Le regole che descrivono vincoli di integrità possono essere espresse sotto forma di **asserzioni**, ovvero affermazioni che devono essere sempre verificate nella nostra base dati. Tali affermazioni, per motivi di chiarezza, devono essere atomiche (non possono essere decomposte in frasi che costituiscono esse stesse delle asserzioni).

Una struttura predefinita per enunciare regole aziendali sotto forma di asserzioni potrebbe essere la seguente:

$$< \text{concetto} > \quad \text{deve/non deve} \quad < \text{espressione su concetti} > \quad (14.1)$$

dove i concetti citati possono corrispondere o a concetti che compaiono nello schema ER a cui si fa riferimento, oppure a concetti derivabili da essi.

Le regole aziendali che esprimono derivazioni possono essere espresse specificando le operazioni (aritmetiche o di altro genere) che permettono di ottenere il concetto derivato.

Una possibile sintassi è la seguente:

$$< \text{concetto} > \quad \text{si ottiene} \quad < \text{operazione su concetti} > \quad (14.2)$$

Ovviamente, quando lo schema concettuale viene tradotto in una base di dati (fase di progettazione logica o fisica), le regole aziendali non descrittive vanno codificate per garantire la consistenza dei dati rispetto alle proprietà che esse rappresentano.

14.4.2 Tecniche di documentazione

Uno schema ER va corredato con una documentazione di supporto, per facilitare l'interpretazione dello schema stesso e per descrivere proprietà dei dati che non possono essere espresse direttamente dai costrutti del modello.

La documentazione dei vari concetti rappresentati in uno schema, ovvero le regole aziendali di tipo descrittivo, può essere prodotta facendo uso di un **dizionario dei dati**. Esso è composto da due tabelle: la prima descrive le *entità* dello schema con il nome, una

definizione informale in linguaggio naturale, l'elenco di tutti gli attributi (con eventuali descrizioni associate) e i possibili identificatori.

L'altra tabella descrive le *relazioni* con il nome, una loro descrizione informale, l'elenco degli attributi (con eventuali descrizioni) e l'elenco delle entità coinvolte insieme alla loro cardinalità di partecipazione.

L'uso del dizionario è particolarmente importante nei casi in cui lo schema è complesso, (molti concetti collegati in maniera articolata) e risulta pesante specificare direttamente sullo schema tutti gli attributi di entità e relazioni.

Per quel che riguarda le altre regole aziendali, si può far ricorso ancora a una tabella, nella quale vengono elencate le varie regole, specificando di volta in volta la loro tipologia.

Capitolo 15

Progettazione Concettuale

15.1 Raccolta e analisi dei requisiti

Per **raccolta dei requisiti** si intende la completa individuazione dei problemi che l'applicazione da realizzare deve risolvere e le caratteristiche che tale applicazione dovrà avere. Per caratteristiche del sistema si intendono sia gli aspetti statici (i dati) che gli aspetti dinamici (le operazioni sui dati).

Dato che i requisiti vengono inizialmente raccolti in specifiche espresse generalmente in linguaggio naturale, risultano spesso ambigui e disorganizzati.

L'**analisi dei requisiti** consiste nel chiarimento e nell'organizzazione delle specifiche dei requisiti.

15.1.1 Fonti dei requisiti

Per la raccolta dei requisiti possiamo rivolgerci a diverse fonti, che rientrano solitamente in una delle seguenti categorie:

- **Utenti dell'applicazione:** le operazioni si acquisiscono mediante opportune interviste, oppure attraverso una documentazione scritta che gli utenti possono avere predisposto appositamente.
- **Documentazione esistente:** documentazione che ha qualche attinenza con il problema che stiamo studiando (moduli, regolamenti interni, procedure aziendali, normative).
È richiesta, in questo caso, un'attività di raccolta e selezione che viene assistita dagli utenti, ma è sempre a carico del progettista.
- **Realizzazioni pre-esistenti:** applicazioni che devono essere rimpiazzate oppure che devono interagire in qualche maniera con il sistema da realizzare.
La conoscenza delle caratteristiche di queste applicazioni potrebbe fornirci importanti informazioni anche in relazione ai problemi esistenti che è necessario risolvere.

15.1.2 Specifica dei requisiti

La specifica dei requisiti raccolti avviene spesso facendo uso di descrizioni in linguaggio naturale. Sappiamo, però, che il linguaggio naturale è fonte di ambiguità e fraintendimenti.

È molto importante, quindi, effettuare una profonda analisi del testo che descrive le specifiche per filtrare le eventuali inesattezze e i termini ambigui presenti.

Ecco alcune regole generali per ottenere una specifica dei requisiti più precisa e senza ambiguità:

1. **Scegliere il corretto livello di astrazione:** è bene evitare di utilizzare termini *troppo generici* o *troppo specifici*, che rendono poco chiaro un concetto.
2. **Standardizzare la struttura delle frasi:** nella specifica dei requisiti preferibile utilizzare sempre lo stesso stile sintattico.
Per esempio:

Per *< dato >* rappresentiamo *< insieme di proprieta >* (15.1)

3. **Evitare frasi contorte:** le definizioni devono essere semplici e chiare.
4. **Individuare sinonimi/omonimi e unificare i termini:** i sinonimi indicano termini diversi con lo stesso significato; gli omonimi indicano termini uguali con diversi significati.
Queste situazioni possono generare ambiguità e vanno chiarite: nel caso di sinonimi unificando i termini, nel caso di omonimi utilizzando termini diversi o specificandoli meglio.
5. **Rendere esplicito il riferimento tra termini:** può succedere che l'assenza di di un contesto di riferimento renda alcuni concetti ambigui: in questi casi bisogna esplicitare il riferimento tra termini.
6. **Costruire un glossario dei termini:** è molto utile per la comprensione e la precisazione dei termini usati, definire un glossario che, per ogni termine, contenga: una breve descrizione, possibili sinonimi e altri termini contenuti nel glossario con i quali esiste un legame logico.

Termine	Descrizione	Sinonimi	Collegamenti
...

Tabella 15.1: Glossario dei termini

15.1.3 Specifiche sulle operazioni

Accanto alle specifiche sui dati, vanno raccolte le specifiche sulle operazioni da effettuare su tali dati. È opportuno cercare di impiegare la stessa terminologia usata per i dati (possiamo fare riferimento al glossario dei termini) e informarci anche sulla frequenza con la quale le varie operazioni vengono eseguite.

15.2 Criteri per la rappresentazione dei dati

15.2.1 Criteri generali di rappresentazione

Pur specificando che non esiste una rappresentazione univoca di un insieme di specifiche, è utile avere delle indicazioni sulle scelte più opportune.

1. Se un concetto ha proprietà significative e/o descrive classi di oggetti con esistenza autonoma, è opportuno rappresentarlo con una **entità**.
2. Se un concetto ha una struttura semplice e non possiede proprietà rilevanti associate, è opportuno rappresentarlo con un attributo di un altro concetto a cui si riferisce.
3. Se sono state individuate due (o più) entità e nei requisiti compare un concetto che le associa, questo concetto può essere rappresentato da una relazione.
4. Se uno o più concetti risultano essere casi particolari di un altro, è opportuno rappresentarli facendo uso di una generalizzazione.

15.2.2 Qualità di uno schema concettuale

Nella costruzione di uno schema concettuale vanno garantite alcune proprietà generali che uno schema concettuale di buona qualità deve possedere.

Correttezza

Uno schema concettuale è **corretto** quando utilizza propriamente i costrutti messi a disposizione dal modello concettuale di riferimento.

Come avviene nei linguaggi di programmazione, gli errori possono essere di due tipi:

- **Errori sintattici:** riguardano un uso non ammesso di costrutti.
E.g. una generalizzazione tra relazioni invece che tra entità.
- **Errori semantici:** riguardano un uso di costrutti che non rispetta la loro definizione.
E.g. l'uso di una relazione per descrivere il fatto che un'entità è una specializzazione di un'altra

Completezza

Uno schema concettuale è **completo** quando rappresenta tutti i dati di interesse e quando tutte le operazioni possono essere eseguite a partire dai concetti descritti nello schema.

Leggibilità

Uno schema concettuale è **leggibile** quando rappresenta i requisiti in maniera naturale e facilmente comprensibile.

Per garantire questa proprietà è necessario rendere lo schema autoesplicativo, per esempio, mediante una scelta opportuna dei nomi da dare ai concetti.

Inoltre, la leggibilità dipende anche da fattori puramente estetici. Ecco alcuni suggerimenti per rendere lo schema più leggibile:

- Disporre i costrutti su una griglia scegliendo come elementi centrali quelli con più legami (relazioni) con altri.
- Tracciare solo linee perpendicolari e cercare di minimizzare le intersezioni.
- Disporre le entità che sono generalizzazioni sopra le relative entità figlie.

Minimalità

Uno schema è **minimale** quando tutte le specifiche sui dati sono rappresentate una sola volta nello schema. Uno schema, quindi, non è minimale quando esistono delle **ridondanze**, ovvero concetti che possono essere derivati da altri.

A differenza delle altre proprietà, comunque, non sempre una ridondanza è indesiderata, ma può nascere da precise scelte di progettazione.

Capitolo 16

Progettazione Logica

L'obiettivo della progettazione logica è quello di costruire uno schema logico in grado di descrivere in maniera corretta ed efficiente tutte le informazioni contenute nello schema E-R prodotto nella fase di progettazione concettuale.

Non si tratta di una semplice traduzione da un modello a un altro, in quanto, prima di passare allo schema logico, lo schema E-R va ristrutturato per soddisfare due esigenze: quella di semplificare la traduzione e quella di ottimizzare il progetto.

La semplificazione dello schema si rende necessaria perché non tutti i costrutti del modello E-R hanno una traduzione naturale nei modelli logici. Per esempio, mentre un'entità può essere facilmente rappresentata da una relazione del modello relazionale (avente gli stessi attributi dell'entità), per le generalizzazioni esistono varie alternative.

Inoltre, mentre la progettazione concettuale ha come obiettivo la rappresentazione accurata e naturale dei dati d'interesse dal punto di vista del significato che hanno nell'applicazione, la progettazione logica costituisce la base per l'effettiva realizzazione dell'applicazione e deve tenere conto, per quanto possibile, delle sue prestazioni: questa necessità può portare a una ristrutturazione dello schema concettuale.

16.1 Fasi della progettazione logica

Le principali fasi della progettazione logica sono le seguenti:

- **Ristrutturazione dello schema E-R:** è una fase indipendente dal modello logico scelto e si basa su criteri di ottimizzazione dello schema e di semplificazione della fase successiva.
- **Traduzione verso il modello logico:** fa riferimento a uno specifico modello logico (nel nostro caso il modello relazionale) e può includere un'ulteriore ottimizzazione che si basa sulle caratteristiche del modello logico stesso.

I dati in ingresso della prima fase sono lo schema concettuale prodotto nella fase precedente e il carico applicativo previsto, in termini di dimensione dei dati e caratteristiche delle operazioni.

Il risultato che si ottiene è uno schema E-R ristrutturato, che non è più uno schema concettuale nel senso stretto del termine, in quanto costituisce una rappresentazione dei dati che tiene conto degli aspetti realizzativi.

Questo schema e il modello logico scelto costituiscono i dati in ingresso della seconda fase, che produce lo schema logico della nostra base di dati.

In questa seconda fase è possibile effettuare verifiche della qualità dello schema ed eventuali ulteriori ottimizzazioni mediante tecniche basate sulle caratteristiche del modello logico.

La tecnica usata nell'ambito del modello relazionale è la **normalizzazione**.

Lo schema logico finale, i vincoli d'integrità definiti su di esso e la relativa documentazione costituiscono i prodotti finali della progettazione logica.

16.2 Analisi delle prestazioni su schemi E-R

Uno schema E-R può essere modificato per ottimizzare alcuni **indici di prestazione** del progetto. Facciamo riferimento ad indici di prestazione e non di prestazioni perché, in realtà, le prestazioni di una base dati non sono valutabili in maniera precisa in sede di progettazione logica, in quanto dipendenti anche da parametri fisici, dal sistema di gestione di basi di dati che verrà utilizzato e da altri fattori difficilmente prevedibili in questa fase. È comunque possibile, facendo uso di alcune schematizzazioni, effettuare studi di massima dei due parametri che generalmente regolano le prestazioni dei sistemi software:

- **Costo di un'operazione:** viene valutato in termini di numero di occorrenze di entità e associazioni che mediamente vanno visitate per rispondere a un'operazione sulla base di dati.
- **Occupazione di memoria:** viene valutato in termini dello spazio di memoria (misurato, per esempio, in numero di byte) necessario per memorizzare i dati descritti dallo schema.

Per studiare questi parametri, abbiamo bisogno di conoscere (oltre allo schema) le seguenti informazioni:

- **Volume dei dati:**
 - Numero di occorrenze di ogni entità e associazione dello schema
 - Dimensioni di ciascun attributo
- **Caratteristiche delle operazioni:**
 - Tipo dell'operazione (interattiva o batch)
 - Frequenza (numero medio di esecuzioni in un certo intervallo di tempo)
 - Dati coinvolti (entità e/o associazioni)

Tabella 16.1: Tavola dei volumi

Concetto	Tipo	Volume
Impiegato	E	2000
Afferenza	R	1900
Dipartimento	E	80
Sede	E	10

Tabella 16.2: Tavola delle operazioni

Operazioni	Tipo	Frequenza
Op.1	I	50 al giorno
Op.2	I	100 al giorno
Op.3	I	10 al giorno
Op.4	B	2 a settimana

Sebbene un'analisi delle prestazioni che fa riferimento a un numero ristretto di operazioni può sembrare riduttiva rispetto al reale carico della base di dati va notato che le operazioni sulle basi di dati seguono la cosiddetta regola **ottanta-venti**.

In base a questa regola, l'ottanta per cento del carico è generato dal venti per cento delle operazioni. Questo fatto ci consente di valutare adeguatamente il carico concentrandoci solo sulle operazioni principali previste.

Nella **tavola dei volumi** vengono riportati tutti i concetti dello schema (entità e associazioni) con il volume previsto a regime: Il volume dei dati e le caratteristiche generali delle operazioni possono essere descritti facendo uso di tabelle, dette **tavole delle operazioni**, in cui vengono riportati, per ogni operazione, la frequenza prevista e un simbolo che indica se l'operazione è interattiva (*I*) oppure batch (*B*). Nella tavola dei volumi, il numero delle occorrenze delle associazioni dipende da due parametri: il numero di occorrenze delle entità coinvolte nelle associazioni e il numero (medio) di partecipazioni di un'occorrenza di entità alle occorrenze di associazioni.

A sua volta, il secondo parametro dipende dalle cardinalità delle associazioni.

Per ogni operazione, possiamo inoltre descrivere graficamente i dati coinvolti con uno **schema di operazione** che consiste nel frammento dello schema E-R interessato dall'operazione, sul quale viene disegnato il *cammino logico* da percorrere per accedere alle informazioni di interesse.

Una volta ottenute queste informazioni, possiamo fare una stima del costo di un'operazione sulla base di dati contando il numero di accessi alle occorrenze di entità e associazioni necessario per eseguire l'operazione.

Tabella 16.3: Tavola degli accessi

Concetto	Costrutto	Accessi	Tipo
Impiegato	Entità	1	L
Afferenza	Relazione	1	L
Dipartimento	Entità	1	L
Impiegato	Entità	1	L

Per esprimere le modalità di accesso alle informazioni utilizziamo la **tavola degli accessi**, che è nella forma: Nell'ultima colonna di questa tabella viene riportato il tipo di accesso: *L* per accesso in lettura e *S* per accesso in scrittura. Questa distinzione va fatta perché, generalmente, le operazioni di scrittura sono più onerose di quelle in lettura (in quanto devono essere eseguite in modo esclusivo e possono richiedere l'aggiornamento di **indici**, che sono strutture ausiliarie per l'accesso efficiente ai dati).

16.3 Ristrutturazione di schemi E-R

La fase di ristrutturazione di uno schema E-R si può suddividere in una serie di passi da effettuare in sequenza:

- **Analisi delle ridondanze:** si decide se eliminare o mantenere eventuali ridondanze presenti nello schema.
- **Eliminazione delle generalizzazioni:** tutte le generalizzazioni presenti nello schema vengono analizzate e sostituite da altri costrutti.
- **Partizionamento/accorpamento di entità e associazioni:** si decide se è opportuno partizionare concetti dello schema (entità e/o associazioni) in più concetti o, viceversa, accorpare concetti separati da un unico concetto.
- **Scelta degli identificatori principali:** si seleziona un identificatore per quelle entità che ne hanno più di uno.

16.3.1 Analisi delle ridondanze

Ricordiamo che una **ridondanza** in uno schema concettuale corrisponde alla presenza di un dato che può essere derivato (cioè ottenuto attraverso una serie di operazioni) da altri dati. In particolare, in uno schema E-R si possono presentare diversi tipi di ridondanza:

- **Attributi derivabili, occorrenza per occorrenza,** da altri attributi della stessa entità (o associazione).

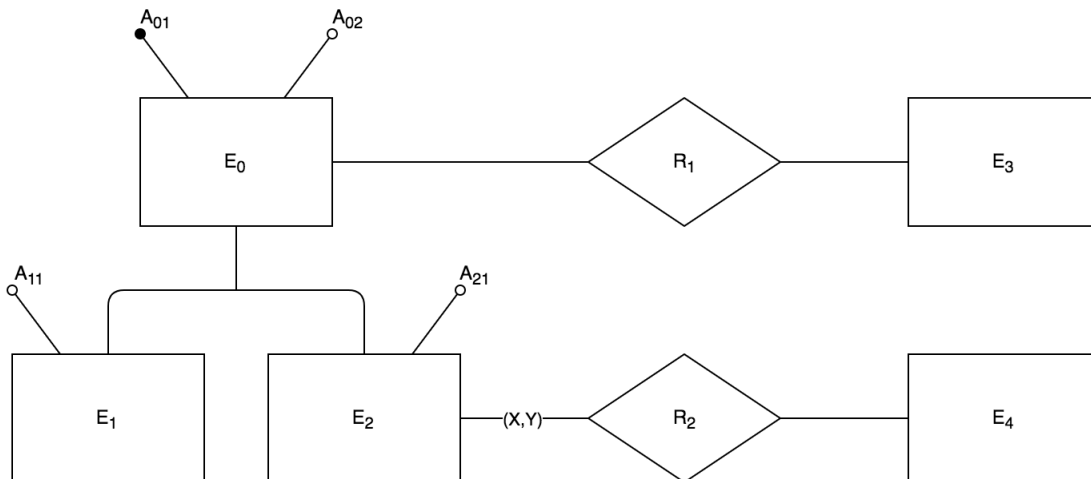


Figura 16.1: Generalizzazione: esempio

- Attributi derivabili da attributi di altre entità (o associazioni) di solito attraverso funzioni aggregative.
- Attributi derivabili da operazioni di conteggio di occorrenze. Si tratta, in effetti, di una variante del caso precedente, che viene però discusso separatamente perché molto frequente in pratica.
- Associazioni derivabili dalla composizione di altre associazioni in presenza di cicli. Notiamo che non sempre la presenza di un ciclo genera ridondanze!

La presenza di un dato derivato presenta un vantaggio e alcuni svantaggi. Il vantaggio è una riduzione degli accessi necessari per calcolare il dato derivato, gli svantaggi sono una maggiore occupazione di memoria (che è, comunque, spesso un costo trascurabile) e la necessità di effettuare operazioni aggiuntive per mantenere il dato derivato aggiornato. La decisione di mantenere o eliminare una ridondanza va quindi presa confrontando costo di esecuzione delle operazioni che coinvolgono il dato ridondante e relativa occupazione di memoria, nei casi di presenza e assenza della ridondanza.

16.3.2 Eliminazione delle generalizzazioni

Dato che i sistemi tradizionali per la gestione delle basi di dati non consentono di rappresentare direttamente una generalizzazione, risulta spesso necessario trasformare questo costrutto in altri costrutti del modello E-R per i quali esiste invece una implementazione naturale: le entità e le associazioni.

Per rappresentare una generalizzazione mediante entità e associazioni abbiamo essenzialmente tre alternative possibili, che si eseguono attraverso le seguenti ristrutturazioni:

1. Accorpamento delle figlie della generalizzazione nel genitore.

Le entità E_1 ed E_2 vengono eliminate e le loro proprietà (attributi e partecipazioni ad associazioni e generalizzazioni) vengono aggiunte all'entità genitore E_0 .

A tale entità viene aggiunto un ulteriore attributo che serve a descrivere il "tipo" di una occorrenza di E_0 , cioè se tale occorrenza apparteneva a E_1 oppure a E_2 o, nel caso di generalizzazione non totale, a nessuna di esse.

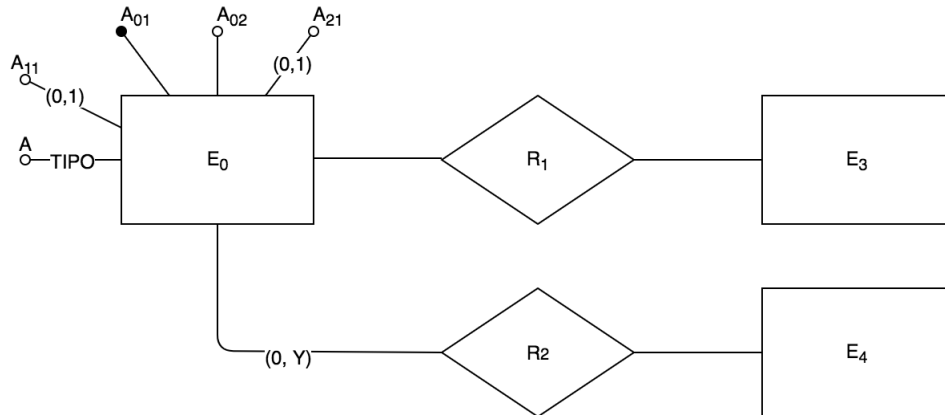


Figura 16.2: Ristrutturazione n.1

2. Accorpamento del genitore della generalizzazione nelle figlie.

L'entità genitore E_0 viene eliminata e, per la proprietà dell'ereditarietà i suoi attributi, il suo identificatore e le relazioni a cui tale entità partecipava, vengono aggiunti alle entità figlie E_1 ed E_2 .

Le relazioni R_{11} e R_{12} rappresentano rispettivamente la restrizione della relazione R_1 sulle occorrenze delle entità E_1 ed E_2 .

3. Sostituzione della generalizzazione con associazioni.

La generalizzazione si trasforma in due associazioni uno a uno, che legano rispettivamente l'entità genitore con le entità figlie E_1 ed E_2 . Non ci sono trasferimenti di attributi o associazioni e le entità E_1 ed E_2 sono identificate esternamente dall'entità E_0 .

Nello schema ottenuto vanno aggiunti, però, dei vincoli: ogni occorrenza di E_0 non può partecipare contemporaneamente a R_{G1} e R_{G2} ; inoltre, se la generalizzazione è totale, ogni occorrenza di E_0 deve partecipare o a un'occorrenza di R_{G1} oppure a un'occorrenza di R_{G2} .

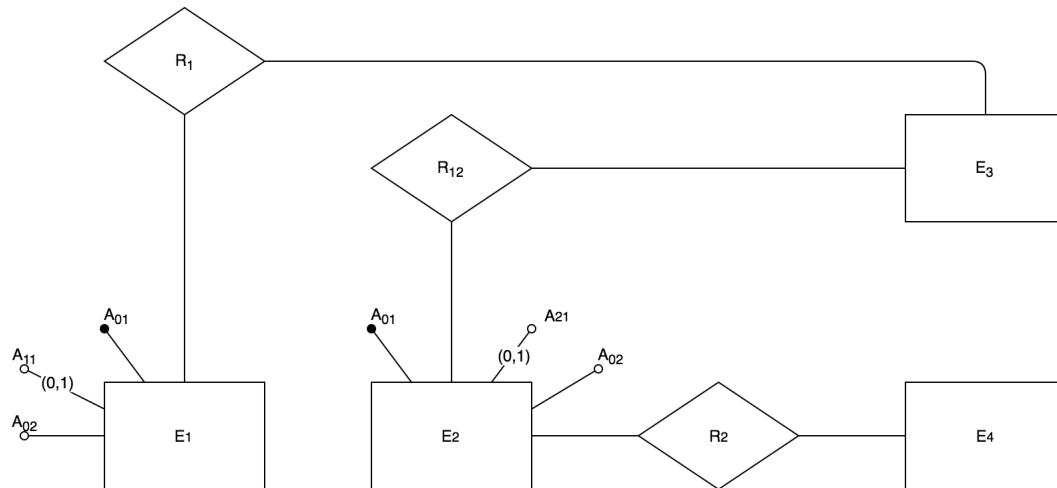


Figura 16.3: Ristrutturazione n.2

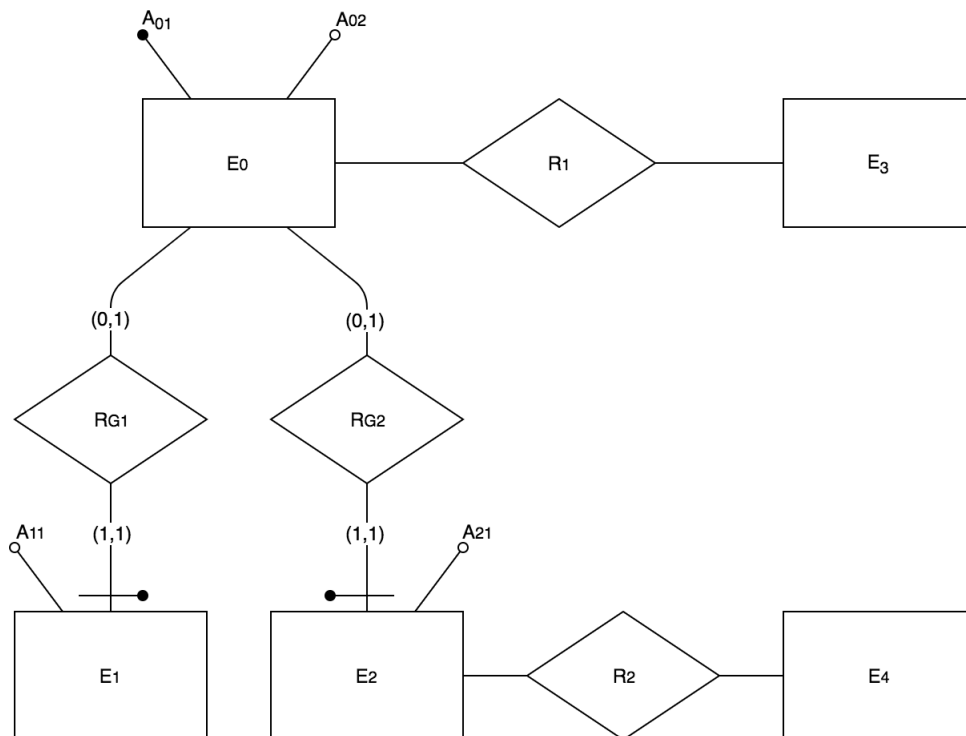


Figura 16.4: Ristrutturazione n.3

La scelta fra le varie alternative può essere fatta in maniera analoga a quanto fatto per i dati derivati, considerando vantaggi e svantaggi di ognuna delle scelte possibili relativamente all'occupazione di memoria e al costo delle operazioni coinvolte. È comunque possibile stabilire alcune regole di carattere generale:

1. **Alternativa n. 1:** conveniente quando le operazioni non fanno molta distinzione tra le occorrenze e gli attributi di E_0 , E_1 ed E_2 .
In questo caso, infatti, anche se abbiamo uno spreco di memoria per la presenza di valori nulli, la scelta ci assicura un numero minore di accessi rispetto alle altre, nelle quali le occorrenze e gli attributi sono distribuiti tra le varie entità.
2. **Alternativa n. 2:** possibile solo se la generalizzazione è totale, altrimenti le occorrenze di E_0 che non sono occorrenze né di E_1 né di E_2 non sarebbero rappresentate. È conveniente quando non ci sono operazioni che si riferiscono solo ad occorrenze di E_1 oppure di E_2 , e dunque fanno delle distinzioni tra tali entità.
In questo caso abbiamo un risparmio di memoria rispetto all'alternativa n.1 perché, in linea di principio, gli attributi non assumono mai valori nulli. Inoltre, c'è una riduzione degli accessi rispetto all'alternativa n.3 perché non si deve visitare E_0 per accedere ad alcuni attributi di E_1 ed E_2 .
3. **Alternativa n. 3:** conveniente quando la generalizzazione non è totale (sebbene ciò non sia necessario) e ci sono operazioni che si riferiscono solo a occorrenze di E_1 (E_2) oppure di E_0 , e dunque fanno delle distinzioni tra entità figlia ed entità genitore.
In questo caso abbiamo un risparmio di memoria rispetto all'alternativa n.1, per l'assenza di valori nulli, ma c'è un incremento degli accessi per mantenere la consistenza delle occorrenze rispetto ai vincoli introdotti.

La ristrutturazione delle generalizzazioni è un tipico caso per il quale il semplice conteggio delle istanze e degli accessi non è sempre sufficiente per scegliere la migliore alternativa possibile.

Infatti, in base a quanto abbiamo detto, sembra che l'alternativa n.3 non convenga quasi mai perché richiede molti più accessi a occorrenze delle altre per eseguire le operazioni sui dati. Questa ristrutturazione, però, ha il grosso vantaggio di generare entità con pochi attributi. Come vedremo in seguito, questo si traduce a livello pratico, in strutture logiche di piccole dimensioni per le quali un accesso fisico permette di recuperare molti dati (tuple) in una sola volta.

In alcuni casi, quindi, va effettuata un'analisi più fine, che tiene conto di altri fattori quali le dimensioni dei domini degli attributi e la quantità di dati che è possibile recuperare con una sola operazione di accesso a memoria secondaria.

16.3.3 Partizionamento/accorpamento di concetti

Entità e associazioni in uno schema E-R possono essere partizionati o accorpati per garantire una maggiore efficienza delle operazioni in base al seguente principio: gli accessi si riducono separando attributi di uno stesso concetto a cui accediamo attraverso operazioni diverse e raggruppando attributi di concetti diversi a cui accediamo attraverso medesime operazioni. Le stesse tecniche discusse per l'analisi delle generalizzazioni possono essere usate per prendere decisioni di questo tipo.

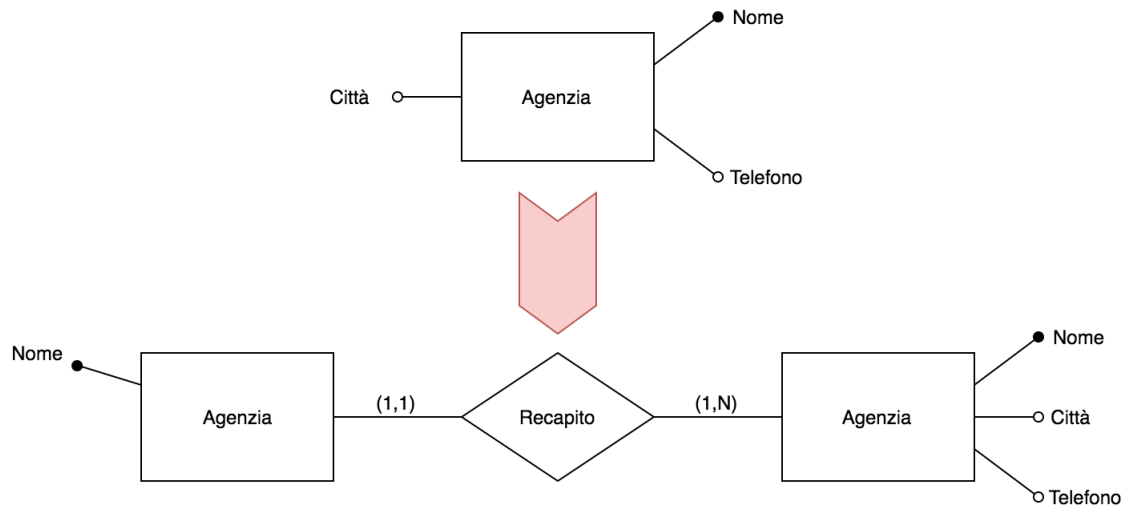


Figura 16.5: Eliminazione di attributi multivalore

È importante notare che, in molti casi, i problemi di partizionamento/accorpamento possono essere rinviati alla fase di progettazione fisica.

Partizionamenti di entità

Esistono sostanzialmente due tipi di partizionamenti di entità:

- **Partizionamenti verticali:** il concetto viene suddiviso operando sui suoi attributi. I partizionamenti verticali generano entità con pochi attributi che possono essere tradotte in strutture logiche sulle quali, con un solo accesso, è possibile recuperare molti dati.
- **Partizionamenti orizzontali:** la suddivisione avviene sulle occorrenze dell'entità. Questo tipo di partizionamenti hanno un effetto collaterale: quello di dover duplicare tutte le associazioni a cui l'entità originaria partecipa. Questo fenomeno può avere delle ripercussioni negative sulle prestazioni del sistema.

Come per le generalizzazioni, anche in questo caso il semplice conteggio delle occorrenze e degli accessi, non è sempre sufficiente per scegliere la migliore alternativa possibile.

Eliminazione di attributi multivalore

Questa ristrutturazione si rende necessaria perché, come per le generalizzazioni, il modello relazionale non permette di rappresentare in maniera diretta questo tipo di attributo. Per eliminare un attributo che ha cardinalità $(1, N)$, partizioniamo l'entità che lo contiene in due entità, tra loro legate da una nuova relazione.

Accorpamento di entità

L'accorpamento è l'operazione inversa del partizionamento.

Immaginiamo di avere una relazione R tra due entità E_1 ed E_2 . Nel caso in cui le operazioni sull'entità E_1 richiedano sempre i dati relativi ad E_2 , vogliamo risparmiare gli accessi necessari per risalire a questi dati attraverso R . Per farlo, aggiungiamo gli attributi di E_2 ad E_1 , ed eliminiamo E_2 .

Un effetto collaterale di questa ristrutturazione è la possibile presenza di valori nulli causate dalla partecipazione opzionale dell'entità E_1 alla relazione R .

In genere, gli accorpamenti vengono effettuati su associazioni di tipo uno a uno, raramente su relazioni uno a molti e quasi mai su relazioni molti a molti. Questo perché gli accorpamenti di entità legate da un'associazione uno a molti o molti a molti generano ridondanze. In particolare, si possono presentare ridondanze su attributi non chiave dell'entità che partecipava all'associazione originaria con cardinalità massima pari a N .

La presenza di ridondanze può comunque essere discussa efficacemente attraverso la tecnica della **normalizzazione**.

Altri tipi di partizionamento/accorpamento

I discorsi fatti finora sul partizionamento/accorpamento possono essere estesi alle associazioni. In alcuni casi, infatti, può essere conveniente decomporre un'associazione tra due entità in due (o più) associazioni tra le medesime entità, per separare occorrenze dell'associazione originarie a cui effettuiamo l'accesso sempre separatamente o, viceversa, accorpare due (o più) associazioni tra le medesime entità (che si riferiscono a due aspetti dello stesso concetto) in un'unica associazione, quando accediamo alle relative occorrenze vengono sempre contemporaneamente.

16.3.4 Scelta degli identificatori principali

La scelta degli identificatori principali è essenziale nelle traduzioni verso il modello relazionale perché in questo modello le chiavi vengono utilizzate per stabilire legami tra dati in relazioni diverse.

Inoltre, i sistemi di gestione di basi di dati richiedono generalmente di specificare una **chiave primaria** sulla quale vengono costruite automaticamente delle strutture ausiliarie, dette **indici**, per il reperimento efficiente di dati.

Quindi, nei casi in cui esistono entità per le quali sono stati specificati più identificatori, bisogna decidere quale di questi identificatori verrà utilizzato come chiave primaria.

I criteri di decisione per effettuare questa scelta sono i seguenti:

- Gli attributi con valori nulli non possono costituire identificatori principali. Tali attributi, infatti, non garantiscono l'accesso a tutte le occorrenze dell'entità corrispondente.

- Un identificatore composto da uno o da pochi attributi è da preferire a identificatori costituiti da molti attributi. Questo garantisce che le strutture ausiliarie create per accedere ai dati (gli indici) siano di dimensioni ridotte, permette un risparmio di memoria nella realizzazione dei legami logici tra le varie relazioni e facilita le operazioni di join.
- Per le stesse motivazioni esposte al punto precedente, un identificatore interno è da preferire a un identificatore esterno, che magari coinvolge diverse entità. Infatti, gli identificatori esterni vengono tradotti in chiavi che includono gli identificatori delle entità coinvolte nell'identificazione esterna: chiaramente in questa maniera si possono generare chiavi con molti attributi.
- Un identificatore che viene utilizzato da molte operazioni per accedere alle occorrenze di un'entità è da preferire rispetto agli altri. In questa maniera tali operazioni possono essere eseguite efficientemente perché possono trarre vantaggio dagli indici creati automaticamente dal DBMS.

A questo punto, se nessuno degli identificatori candidati soddisfa tali requisiti, è possibile pensare di introdurre un ulteriore attributo all'entità: questo attributo conterrà valori speciali (detti **codici**) generati appositamente per identificare le occorrenze delle entità. È comunque consigliabile tenere traccia, in questa fase, anche degli identificatori non selezionati come principali ma che vengono utilizzati da qualche operazione per accedere ai dati. Infatti, per questi identificatori è possibile definire, in sede di progettazione fisica, degli **indici secondari**. Gli indici secondari consentono l'accesso efficiente ai dati e possono essere usati in alternativa agli indici definiti automaticamente sugli identificatori principali.

16.4 Traduzione verso il modello relazionale

La seconda fase della progettazione logica corrisponde a una traduzione tra modelli di dati diversi: a partire da uno schema E-R ristrutturato si costruisce uno **schema logico equivalente**, in grado cioè di rappresentare le medesime informazioni.

Coerentemente con quanto già detto, facciamo riferimento a una versione semplificata del modello E-R, che non contiene generalizzazioni e attributi multivalore, e nella quale ogni entità ha un solo identificatore.

16.4.1 Entità e associazioni molti a molti

Consideriamo il seguente schema:

La sua traduzione naturale nel modello relazionale prevede:

- Per ogni entità, una relazione con lo stesso nome avente per attributi i medesimi attributi dell'entità e per chiave il suo identificatore.

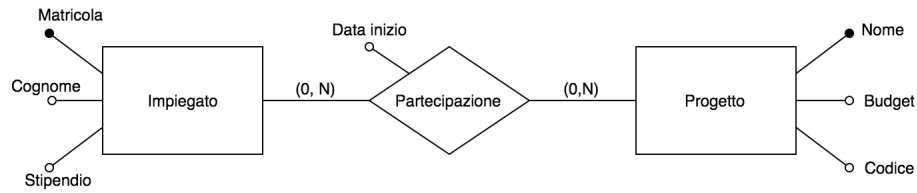


Figura 16.6: Relazione molti a molti

- Per l'associazione, una relazione con lo stesso nome avente per attributi gli attributi dell'associazione e gli identificatori delle entità coinvolte; tali identificatori formano la chiave della relazione.

Se gli attributi originali di entità o associazioni sono opzionali, i corrispondenti attributi di relazione possono assumere valori nulli.

Lo schema relazionale che otteniamo è, quindi, il seguente:

$$\begin{aligned}
 &IMPIEGATO(\underline{Matricola}, Cognome, Stipendio) \\
 &PROGETTO(\underline{Codice}, Nome, Budget) \\
 &PARTECIPAZIONE(\underline{Matricola}, \underline{Codice}, DataInizio)
 \end{aligned}
 \tag{16.1}$$

Per lo schema ottenuto esistono due vincoli d'integrità referenziale tra gli attributi *Matricola* e *Codice* di *PARTECIPAZIONE* e gli omonimi attributi delle entità *IMPIEGATO* e *PROGETTO*.

Possiamo, ovviamente effettuare delle ridenominazioni per rendere lo schema più leggibile, e siamo obbligati a farlo quando abbiamo a che fare con una relazione ricorsiva, come la seguente:

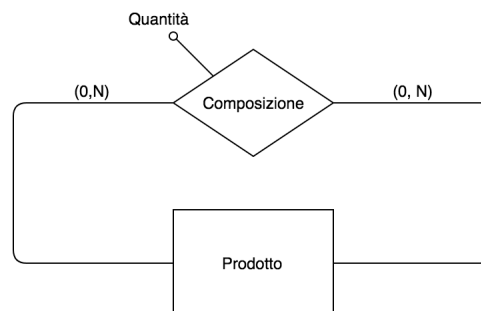


Figura 16.7: Relazione ricorsiva

Questo schema si traduce nelle due relazioni

$$\begin{aligned}
 &PRODOTTO(\underline{Codice}, Nome, Costo) \\
 &COMPOSIZIONE(\underline{Composto}, \underline{Componente}, Quantita)
 \end{aligned}
 \tag{16.2}$$

In questo schema entrambi gli attributi *Composto* e *Componente* contengono codici di prodotti: il primo dei due ha il secondo come componente.

Esiste, quindi, un vincolo d'integrità referenziale tra questi attributi e l'attributo *Codice* della relazione *PRODOTTO*.

Le associazioni con più di due entità partecipanti si traducono in maniera analoga a quanto detto per le associazioni binarie.

Per esempio, consideriamo il seguente schema:

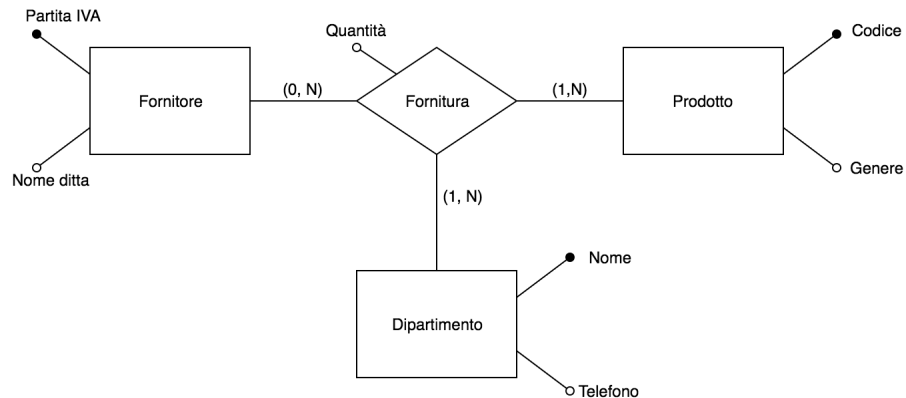


Figura 16.8: Relazione ternaria

Questo schema si traduce nelle seguenti relazioni:

$$\begin{aligned}
 &FORNITORE(\underline{PartitaIVA}, NomeDitta) \\
 &PRODOTTO(\underline{Codice}, Genere) \\
 &DIPARTIMENTO(\underline{Nome}, Telefono) \\
 &FORNITURA(\underline{Fornitore, Prodotto, Dipartimento}, Quantita)
 \end{aligned}
 \tag{16.3}$$

Per lo schema così ottenuto esistono i vincoli d'integrità referenziale tra gli attributi *Fornitore*, *Prodotto* e *Dipartimento* della relazione *FORNITURA* e, rispettivamente, l'attributo *PartitaIVA* della relazione *FORNITORE*, l'attributo *Codice* della relazione *PRODOTTO* e l'attributo *Nome* della relazione *DIPARTIMENTO*.

In quest'ultimo tipo di traduzione è necessario prestare attenzione ad alcuni casi particolari nei quali l'insieme delle chiavi delle relazioni che rappresentano le entità coinvolte costituisce in realtà una **superchiave** ridondante della relazione che rappresenta l'associazione dello schema E-R (esiste cioè un suo sottoinsieme proprio che è una chiave).

Questo potrebbe accadere se, per esempio, nel caso della relazione ternaria appena vista, ci fosse un solo fornitore che fornisce un certo prodotto a un dipartimento.

Si noti che le cardinalità sono ancora valide, perché tale fornitore può fornire diversi prodotti a questo o altri dipartimenti. In questo caso la chiave della relazione *FORNITURA*

sarebbe costituita dai soli attributi *Prodotto* e *Dipartimento*, perché dato un prodotto e un dipartimento, il fornitore è univocamente determinato.

16.4.2 Associazioni uno a molti

Consideriamo il seguente schema:

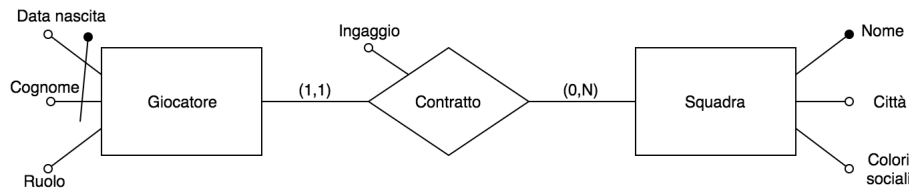


Figura 16.9: Relazione uno a molti

Secondo la regola vista per le associazioni molti a molti, la traduzione di questo schema dovrebbe essere la seguente:

$$\begin{aligned}
 &GIOCATORE(\underline{Cognome}, \underline{DataNascita}, Ruolo) \\
 &SQUADRA(\underline{Nome}, \underline{Citta}, \underline{ColoriSociali}) \\
 &CONTRATTO(\underline{Giocatore}, \underline{DataNascitaGiocatore}, \underline{NomeSquadra}, Ingaggio)
 \end{aligned}
 \tag{16.4}$$

Notiamo che nella relazione *CONTRATTO* la chiave è costituita solo dall'identificatore di *GIOCATORE* perché le cardinalità indicano che ogni giocatore ha un contratto con una sola squadra.

A questo punto le relazioni *GIOCATORE* e *CONTRATTO* hanno ancora la stessa chiave ed è ancora possibile fonderle in un'unica relazione (perché esiste una corrispondenza biunivoca tra le rispettive occorrenze). È quindi preferibile la traduzione che segue:

$$\begin{aligned}
 &GIOCATORE(\underline{Cognome}, \underline{DataNascita}, \underline{Ruolo}, \underline{NomeSquadra}, \underline{Ingaggio}) \\
 &SQUADRA(\underline{Nome}, \underline{Citta}, \underline{ColoriSociali})
 \end{aligned}
 \tag{16.5}$$

In questo schema, ovviamente, esiste il vincolo d'integrità referenziale tra l'attributo *NomeSquadra* della relazione *GIOCATORE* e l'attributo *Nome* della relazione *SQUADRA*.

Le associazioni *n*-arie sono quasi sempre molti a molti. Nel caso in cui un'entità partecipi a una relazione ternaria con cardinalità massima pari a uno, la traduzione dell'associazione viene fatta in modo analogo a un'associazione binaria tra le altre entità.

L'entità che partecipa alla relazione ternaria con cardinalità massima pari a uno, viene infatti tradotta in una relazione che contiene anche gli identificatori delle altre entità coinvolte nell'associazione (più eventuali attributi dell'associazione stessa) e non è più necessario rappresentare esplicitamente l'associazione di partenza.

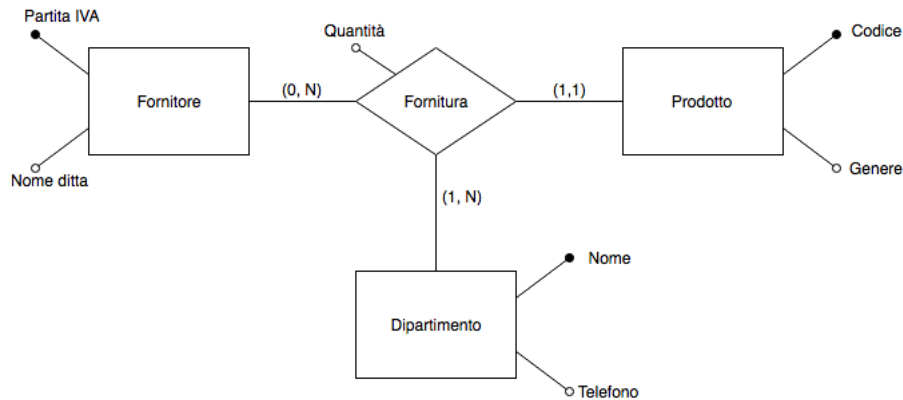


Figura 16.10: Relazione uno a molti

Consideriamo, per esempio, il seguente schema:

Lo schema si traduce nelle seguenti relazioni:

$$\begin{aligned}
 &FORNITORE(\underline{PartitaIVA}, NomeDitta) \\
 &DIPARTIMENTO(\underline{Nome}, Telefono) \\
 &PRODOTTO(\underline{Codice}, Genere, Fornitore, Dipartimento, Quantita)
 \end{aligned} \tag{16.6}$$

In tale schema relazionale esistono i vincoli d'integrità referenziale tra l'attributo *Fornitore* della relazione *PRODOTTO* e l'attributo *PartitaIVA* della relazione *FORNITORE* e tra l'attributo *Dipartimento* della relazione *PRODOTTO* e l'attributo *Nome* della relazione *DIPARTIMENTO*.

16.4.3 Entità con identificatore esterno

Le entità con identificatori esterni danno luogo a relazioni con chiavi che includono gli identificatori delle entità "identificanti".

Consideriamo il seguente schema:

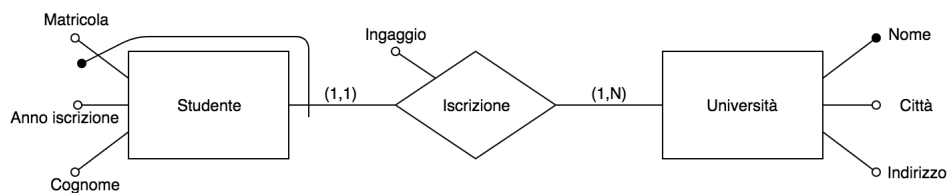


Figura 16.11: Relazione con identificatore esterno

Lo schema relazionale corrispondente a questo schema è il seguente:

$$\begin{array}{l} \textit{STUDENTE}(\textit{Matricola}, \textit{NomeUniversita}, \textit{Cognome}, \textit{AnnoIscrizione}) \\ \textit{UNIVERSITA}(\textit{Nome}, \textit{Citta}, \textit{Indirizzo}) \end{array} \quad (16.7)$$

In questo schema relazionale esiste il vincolo d'integrità referenziale tra l'attributo *NomeUniversita* della relazione *STUDENTE* e l'attributo *Nome* della relazione *UNIVERSITA*.

Come abbiamo potuto notare nell'esempio, rappresentando l'identificatore esterno si rappresenta direttamente anche l'associazione tra le due entità. Le entità identificate esternamente partecipano all'associazione sempre con una cardinalità minima e massima pari a uno.

Questo tipo di traduzione è valido indipendentemente dalla cardinalità con cui l'altra entità partecipa all'associazione.

16.4.4 Associazioni uno a uno

Per le associazioni uno a uno ci sono, in genere, diverse possibilità di traduzione. Consideriamo il seguente schema, in cui la partecipazione di entrambe le entità è obbligatoria:

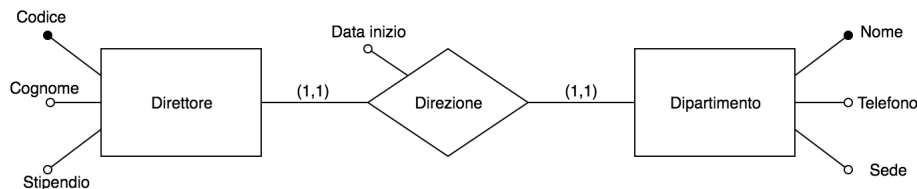


Figura 16.12: Relazione uno a uno

Un possibile schema relazionale corrispondente a questo schema è il seguente:

$$\begin{array}{l} \textit{DIRETTORE}(\textit{Codice}, \textit{Cognome}, \textit{Stipendio}, \textit{DipartimentoDiretto}, \textit{InizioDirezione}) \\ \textit{DIPARTIMENTO}(\textit{Nome}, \textit{Telefono}, \textit{Sede}) \end{array} \quad (16.8)$$

In questo schema esiste il vincolo d'integrità referenziale tra l'attributo *DipartimentoDiretto* della relazione *DIRETTORE* e l'attributo *Nome* della relazione *DIPARTIMENTO*, oppure:

$$\begin{array}{l} \textit{DIRETTORE}(\textit{Codice}, \textit{Cognome}, \textit{Stipendio}) \\ \textit{DIPARTIMENTO}(\textit{Nome}, \textit{Telefono}, \textit{Sede}, \textit{Direttore}, \textit{InizioDirezione}) \end{array} \quad (16.9)$$

In questo secondo schema esiste il vincolo d'integrità referenziale tra l'attributo *Direttore* della relazione *DIPARTIMENTO* e l'attributo *Codice* della relazione *DIRETTORE*.

È possibile, quindi, rappresentare l'associazione in una qualunque delle relazioni che rappresentano le due entità.

Trattandosi di una relazione biunivoca tra le occorrenze delle entità, sembrerebbe possibile un'ulteriore alternativa nella quale si rappresentano tutti i concetti in un'unica relazione contenente tutti gli attributi in gioco.

Questa alternativa è, però, da escludere perché non dobbiamo dimenticare che lo schema E-R che stiamo traducendo è il risultato di una fase di ristrutturazione nella quale sono state effettuate precise scelte anche riguardo l'accorpamento e il partizionamento di entità.

Questo significa che, se nello schema E-R ristrutturato abbiamo due entità collegate da una relazione uno a uno, vuol dire che abbiamo ritenuto conveniente tenere separati i due concetti ed è quindi inopportuno fonderli in sede di traduzione verso il modello relazionale.

Consideriamo ora il seguente schema, in cui soltanto un'unità ha partecipazione opzionale:

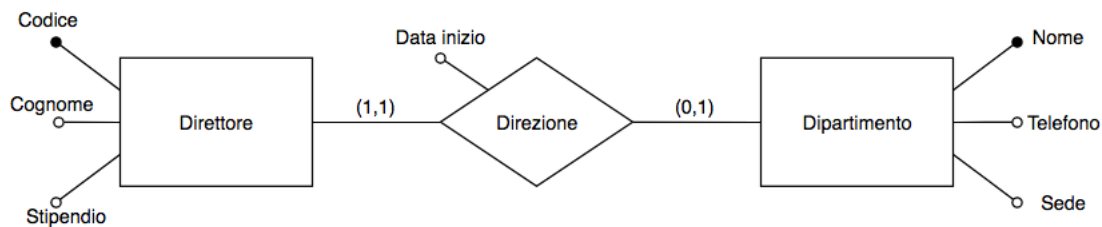


Figura 16.13: Relazione uno a uno

In questo caso abbiamo una soluzione preferibile rispetto alle altre:

$$\begin{aligned} &IMPIEGATO(\underline{Codice}, Cognome, Stipendio) \\ &DIPARTIMENTO(\underline{Nome}, Telefono, Sede, Direttore, InizioDirezio) \end{aligned} \quad (16.10)$$

Per lo schema ottenuto esiste il vincolo d'integrità referenziale tra l'attributo *Direttore* della relazione *DIPARTIMENTO* e l'attributo *Codice* della relazione *IMPIEGATO*. Questa alternativa è preferibile rispetto a quella in cui l'associazione viene rappresentata nella relazione *IMPIEGATO* mediante il nome del dipartimento diretto perché avremo, per questo attributo, possibili valori nulli.

Consideriamo ora il caso in cui entrambe le entità hanno partecipazione opzionale, come nel seguente schema:

In questo caso abbiamo una soluzione preferibile rispetto alle altre:

$$\begin{aligned} &IMPIEGATO(\underline{Codice}, Cognome, Stipendio) \\ &DIPARTIMENTO(\underline{Nome}, Telefono, Sede) \\ &DIREZIONE(\underline{Direttore}, Dipartimento, DataInizioDirezio) \end{aligned} \quad (16.11)$$

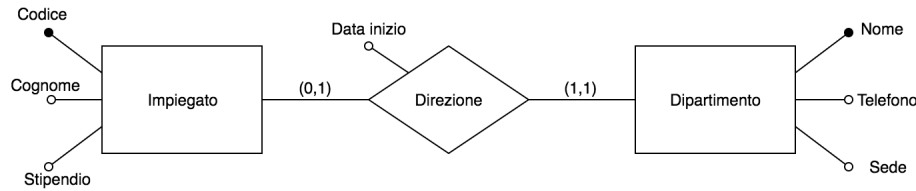


Figura 16.14: Relazione uno a uno

Su questo schema abbiamo due vincoli d'integrità referenziale: uno tra l'attributo *Direttore* della relazione *DIREZIONE* e l'attributo *Codice* della relazione *IMPIEGATO* e l'altro tra l'attributo *Dipartimento* della relazione *DIREZIONE* e l'attributo *Nome* della relazione *DIPARTIMENTO*.

Questa soluzione ha il vantaggio, rispetto a quella in cui si accorpa la relazione *DIREZIONE* in una delle altre due relazioni (come nel caso precedente), di non presentare mai valori nulli sugli attributi che rappresentano l'associazione. Per contro, abbiamo bisogno di una relazione in più con un conseguente aumento della complessità della base di dati.

Diciamo quindi che la soluzione con tre relazioni è da prendere in considerazione solo se il numero di occorrenze dell'associazione è molto basso rispetto alle occorrenze delle entità che partecipano all'associazione. In questo caso, c'è infatti il vantaggio di evitare la presenza di molti valori nulli.

16.5 Documentazione di schemi logici

Come nel caso della progettazione concettuale, il risultato della progettazione logica non è costituito solo da un semplice schema di una base di dati ma anche da una documentazione a esso associata.

Innanzitutto, buona parte della documentazione dello schema concettuale in ingresso alla fase di progettazione logica può essere ereditata dallo schema logico ottenuto come risultato di questa fase. In particolare, se i nomi dei concetti dello schema E-R sono stati riutilizzati per costruire lo schema relazionale, le regole aziendali precedentemente definite possono essere usate per documentare anche quest'ultimo.

A questa documentazione ne va aggiunta però dell'altra, in grado di descrivere i vincoli d'integrità referenziale introdotti dalla traduzione.

A tale riguardo, è possibile adottare un semplice formalismo grafico che ci permette di rappresentare sia le relazioni con i relativi attributi che i vincoli d'integrità referenziale esistenti tra le varie relazioni.

Un esempio di tale rappresentazione, di facile comprensione, è dato dal seguente schema: In questi diagrammi le chiavi delle relazioni sono rappresentate in grassetto, le frecce indicano vincoli d'integrità referenziale e la presenza di asterischi sui nomi di attributo indica la possibilità di avere valori nulli.

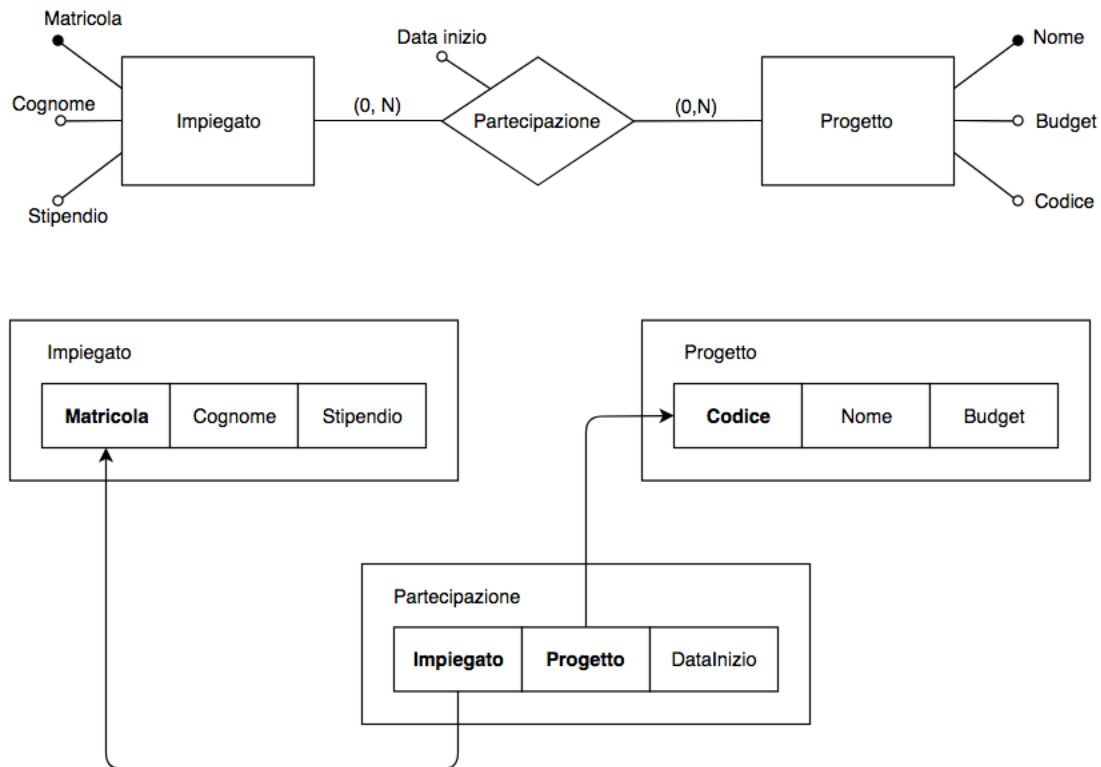


Figura 16.15: Vincoli d'integrità

Osserviamo che, con questo formalismo, si riesce a mantenere traccia delle associazioni dello schema E-R originale. Questo può risultare utile per individuare, in maniera immediata, i **cammini di join**, ovvero le operazioni di join necessarie per ricostruire l'informazione rappresentata dalle associazioni originarie, nel caso dell'esempio, le informazioni sui progetti ai quali gli impiegati partecipano, attraverso il join tra *IMPIEGATO*, *PARTECIPAZIONE* e *PROGETTO*.