Basi di dati e web

Simone Petta

Secondo Semestre a.a. 2021/2022

Contents

1 Introduzione **13**
1.1 Programma . . . 14
1.2 Informazione, dati e loro gestione . . . 14
1.2.1 Informatica . . . 14
1.2.2 Gestione informazioni . . . 14
1.2.3 Sistema Informativo . . . 15
1.2.4 Informazione e dati . . . 15
1.2.5 Organizzazione delle informazioni . . . 16
1.2.6 Perche' i dati? . . . 16
1.2.7 Base di dati . . . 16
1.3 I DBMS . . . 16
1.3.1 I DBMS oggi . . . 17
1.3.2 Le basi di dati sono... grandi . . . 17
1.3.3 Le basi di dati sono ... persistenti . . . 17
1.3.4 Le basi di dati sono... condivise . . . 18
1.3.5 I DBMS garantiscono... affidabilita' . . . 18
1.3.6 Transazione . . . 18
1.3.7 I DBMS devono essere efficienti . . . 19
1.3.8 I DBMS devono essere efficaci . . . 19
1.3.9 DBMS vs File System . . . 19
1.3.10 Descrizione dei dai . . . 19
1.4 I modelli dei dati . . . 20
1.4.1 Descrizione dei dati nei DBMS . . . 20
1.4.2 Modello dei dati . . . 20
1.4.3 Schema e istanza . . . 21
1.4.4 Due tipi (principali) di modello . . . 22
1.5 Architettura (semplificata) di un DBMS . . . 22
1.5.1 Architettura a tre livelli per DBMS . . . 23
1.5.2 Vista . . . 23
1.5.3 Indipendenza dei dati . . . 24
1.6 Linguaggi . . . 25
1.6.1 Linguaggi per basi di dati . . . 25
1.6.2 Linguaggio SQL standard . . . 25
1.6.3 SQL immerso in un linguaggio ad hoc . . . 26

1.6.4 Due tipi di linguaggi 26

2 Architettura Web con i DataBase 29
2.1 Modello client-server 30
2.1.1 Architettura DBMS 30
2.1.2 Architettura logica client-server per accesso al DMBS . . 30
2.1.3 Diversi tipi di accesso al DBMS 31
2.1.4 Client-Server nel web 32
2.1.5 Caricamento di una pagina statica 32
2.1.6 Programmare per il web 33
2.1.7 Riassunto I/O del web server 35
2.2 Architetture WEB-DB multi-tier (multi-livello) 35
2.2.1 Architettura a 3 livelli 35
2.2.2 Caricamento di una pagina dinamica 36
2.2.3 Esempio di PHP per interrogare PostgreSQL 37
2.2.4 Architetture Web-DB con load balancing, cachine e replicazione 37
2.3 Web application e DBMS 38
2.3.1 Web application e DBMS 38
2.4 Mobile app e DBMS 39
2.4.1 Mobile app e DBMS 40

3 Modello relazionale 41
3.1 I modelli logici dei dati 42
3.1.1 Caratteristiche 42
3.2 Modello relazionale 42
3.3 Relazione 43
3.3.1 Relazione matematica, esempio 43
3.3.2 Struttura non posizionale 45
3.3.3 Tabelle e relazioni 45
3.3.4 Il modello e' basato su valori 45
3.3.5 Alternativa 46
3.3.6 Vantaggi struttura basata su valori 47
3.3.7 Definizioni 47
3.3.8 Relazioni sui singoli attributi 47
3.4 Strutture nidificate 48
3.5 Informazione incompleta 50
3.5.1 Tipi di valore nullo 50
3.6 Vincolo di integrita' 51
3.6.1 Tipi di vincoli 52
3.6.2 Sintassi 52
3.6.3 Vincoli di ennupla, altro esempio 53
3.6.4 Identificazione delle ennuple 53
3.6.5 Chiave 54
3.6.6 Vincoli, schemi e istanze 54
3.6.7 Esistenza delle chiavi 54

3.6.8 Importanza delle chiavi 55
3.6.9 Chiavi e valori nulli 55
3.6.10 Chiave primaria 55
3.7 Integrita' referenziale 56
3.7.1 Vincolo identita' referenziale: definizione formale 57
3.7.2 Con i valori nulli 58

4 Progettazione di basi di dati 61
4.1 Perche' preoccuparci 62
4.2 Progettazione di basi di dati 62
4.3 Due tipi di modelli 63
4.4 Modelli concettuali 63
4.5 Modello Entity-Relationship (Entita'-Associazione (ER)) 64
4.6 Costrutti del modello E-R 64
4.6.1 Entita' 64
4.6.2 Associazioni 65
4.6.3 Graficamente 65
4.6.4 Entita': schema e istanza 65
4.7 Associazione 66
4.8 Occorrenze di associazioni 66
4.9 Promuovere l'associazione 67
4.9.1 Due associazioni sulle stesse entita' 67
4.9.2 Associazione n-aria 67
4.10 Associazione ricorsiva 67
4.10.1 Associazione ternaria ricorsiva 68
4.11 Attributo 69
4.11.1 Rappresentazione grafica 69
4.11.2 Attributo composti 69
4.12 Altri costrutti del modello E-R 70
4.13 Cardinalita' di associazione 70
4.14 Tipi di associazioni binarie 71
4.14.1 Associazioni "molti a molti" 71
4.14.2 Associazioni "uno a molti" 71
4.14.3 Associazioni uno a uno 72
4.14.4 Cardinalita' di attributi 72
4.15 Identificatore di una entita' 72
4.15.1 Identificatori interni 73
4.15.2 Identificate esterno 73
4.15.3 Osservazioni 74
4.16 Generalizzazione 74
4.16.1 Rappresentazione grafica 75
4.16.2 Proprieta' delle generalizzazioni 75
4.16.3 Tipi generalizzazione 76
4.16.4 Altre proprieta' generalizzazione 77
4.17 Documentazione associata agli schemi concettuali 77

5 Patterns progettazione E-R **79**
5.1 Requisiti: documentazione descrittiva 80
5.1.1 Requisiti: organizzazione di termini e concetti 80
5.2 Esempio: biblioteche 80
5.2.1 Specifiche 81
5.2.2 Identificare e risolvere le ambiguita' 81
5.2.3 Separare le frasi 81
5.3 Requisiti discorsivi e schemi concettuali 82
5.4 Design Pattern 82
5.4.1 Part-of 82
5.4.2 Istance-of 83
5.4.3 Reificazione di associazione binaria 83
5.4.4 Caso particolare 84
5.4.5 Storicizzazione di concetto 84
5.4.6 Associazione ternaria 86
5.5 Strategie di progetto 87
5.5.1 Top-Dowm 87
5.5.2 Esempio biblioteche 88
5.5.3 Bottom-up 89
5.5.4 Inside-out 90

6 Progettazione logica **91**
6.1 Dati in ingresso 92
6.2 Ristrutturazione dello schema E-R 92
6.3 Prestazioni 92
6.4 Tavola dei volumi 93
6.4.1 Esempio di valutazione di costo 93
6.5 Attivita' della ristrutturazione 94
6.5.1 Analisi delle ridondanze 94
6.5.2 Analisi di una ridondanza 96
6.5.3 Eliminazione delle generalizzazioni 98
6.5.4 Partizionamento delle entita' 102
6.5.5 Partizionamento delle associazioni 104
6.5.6 Scelta degli identificatori principali 104
6.6 Traduzione nel modello relazionale 105
6.6.1 Entita' e associazioni molti a molti 105
6.6.2 Associazioni ricorsive 106
6.6.3 Associazioni n-arie 106
6.6.4 Associazioni uno a molti 107
6.6.5 Entita' con identificazione esterna 108
6.6.6 Associazioni uno a uno 108
6.6.7 Associazioni uno a uno con due partecipazioni opzionali . 109
6.6.8 Associazioni uno a uno con due partecipazioni opzionali . 109
6.7 Esempio 110
6.8 Strumenti di supporto 110

7 Algebra relazionale **111**
7.1 Operatori . . . 112
7.1.1 Unione . . . 112
7.1.2 Intersezione . . . 113
7.1.3 Differenza . . . 113
7.1.4 Unione sensata ma impossibile . . . 113
7.1.5 Ridenominazione . . . 114
7.1.6 Selezione . . . 115
7.1.7 Proiezione . . . 116
7.1.8 Join . . . 118
7.1.9 Cardinalità del join . . . 119
7.1.10 Join esterno . . . 120
7.1.11 Semi-Join . . . 121
7.1.12 Join ennario . . . 122
7.1.13 Join e proiezioni . . . 122
7.1.14 Prodotto cartesiano . . . 123
7.1.15 equi-join . . . 124
7.2 Esempi . . . 124
7.2.1 1 . . . 124
7.2.2 2 . . . 125
7.2.3 3 . . . 125
7.2.4 4 . . . 125
7.2.5 5 . . . 126
7.2.6 6 . . . 126
7.3 Equivalenza di espressioni . . . 127
7.3.1 Equivalenza importante . . . 127
7.4 Selezioni con i valori nulli . . . 127
7.5 Viste . . . 128
7.5.1 Viste virtuali e materializzate . . . 128
7.5.2 Motivazioni . . . 128
7.6 Convenzione e notazione alternativa per i join . . . 129

8 Calcolo relazionale e datalog **131**
8.1 Calcolo relazionale su Domini . . . 132
8.1.1 Esempio 0 . . . 132
8.1.2 Esempio 0b . . . 133
8.1.3 Esempio 1 . . . 133
8.1.4 Esempio 2 . . . 134
8.1.5 Esempio 3 . . . 134
8.1.6 Esempio 4 . . . 135
8.1.7 Esempio 5 . . . 135
8.2 Quantificatori . . . 135
8.2.1 Espressioni equivalenti . . . 136
8.2.2 Riflessioni . . . 136
8.3 Calcolo relazionale sulle tuple . . . 136
8.3.1 Esempio 0a . . . 137

8.3.2 Esempio 0b . . . 137
8.3.3 Esempio 1 . . . 137
8.3.4 Esempio 2 . . . 138
8.3.5 Esempio 3 . . . 138
8.3.6 Esempio 4 . . . 139
8.3.7 Esempio 5 . . . 139
8.3.8 Espressioni safe' . . . 139
8.4 Calcolo e algebra . . . 140
8.4.1 Limiti . . . 140
8.5 Datalog . . . 140
8.5.1 Sintassi . . . 141
8.5.2 Vincoli . . . 141
8.5.3 Esempio -1 . . . 141
8.5.4 Esempio 0a . . . 142
8.5.5 Esempio 0b . . . 142
8.5.6 Esempio 2 . . . 142
8.5.7 Datalog con negazione . . . 142
8.5.8 Esempio 5 . . . 143
8.5.9 Esempio 6: Ricorsione . . . 143
8.5.10 Espressitiva' . . . 143

9 SQL 145
9.1 Definizione dei dati in SQL . . . 146
9.2 Domini . . . 146
9.2.1 Domini elementari . . . 146
9.2.2 Definizione dei domini . . . 147
9.3 Vincoli intrarelazionali . . . 147
9.4 Unique e primary key . . . 147
9.5 Vincoli interrelazionali . . . 147
9.5.1 Syntax of FK in PGSQL . . . 149
9.6 Modifiche degli schemi . . . 149
9.6.1 ALTER TABLE . . . 149
9.7 Definizione degli indici . . . 150
9.8 DDL, in pratica . . . 150
9.9 SQL, operazioni sui dati . . . 150
9.9.1 Istruzione SELECT (versione base) . . . 150
9.9.2 Selezione e proiezione . . . 151
9.9.3 SELECT, abbreviazioni . . . 151
9.9.4 Selezione, senza proiezione . . . 152
9.9.5 Proiezione, senza selezione . . . 152
9.9.6 SELECT, abbreviazioni . . . 152
9.10 Espressioni nella targhet list . . . 153
9.10.1 Condizione complessa . . . 153
9.10.2 Condizione LIKE . . . 153
9.11 Gestione dei valori nulli . . . 153
9.12 Proiezione in SQL . . . 154

9.13 Selezione, proiezione e join 154
9.14 SQL e algebra relazionale 155
9.15 SQL: esecuzione delle interrogazioni 155
9.15.1 Selezione, proiezione e join 155
9.15.2 2 156
9.15.3 SELECT, con ridenominazione del risultato 156
9.16 JOIN esplicito 156
9.16.1 Esempio 157
9.17 JOIN naturale 157
9.18 JOIN esterno: "outer join" 157
9.19 Ordinamento del risultato 158
9.20 Unione, intersezione e differenza 159
9.20.1 Unione con duplicati usando ALL 159
9.20.2 Notazione posizionale 159
9.20.3 Differenza 160
9.20.4 Intersezione 160
9.21 Interrogazioni nidificate 161
9.21.1 Operatori ANY, ALL, IN 162
9.21.2 Interrogazioni nidificate: commenti 162
9.21.3 Regole visibilita 163
9.21.4 Quantificatore esistenziale 163
9.21.5 Semantica delle espressioni correlate 164
9.21.6 Visibilita sbagliata 164
9.21.7 Differenza e nidificazione 165
9.22 Operatori aggregati 165
9.22.1 COUNT 166
9.22.2 Altri operatori aggregati 167
9.22.3 COUNT e valori nulli 167
9.22.4 Operatori aggregati e valori nulli 167
9.22.5 Operatori aggregati e target list 168
9.22.6 Massimo e nidificazione 168
9.22.7 Operatori aggregati e raggruppamenti 169
9.22.8 Raggruppamenti e target list 169
9.22.9 Condizione sui gruppi 170
9.22.10 WHERE o HAVING 170
9.22.11 HAVING con subquery 171
9.22.12 GROUP BY e valori NULL 171
9.22.13 Riassunto regole su valori nulli 172
9.23 Operazioni di aggiornamento 172
9.23.1 Inserimento 172
9.23.2 Cancellazione 173
9.23.3 Modifica di ennuple 174

10 SQL: caratteristiche evolute **175**
10.1 Transazioni SQL . . . 176
10.1.1 Atomicità . . . 176
10.1.2 Consistenza . . . 176
10.1.3 Isolation . . . 176
10.1.4 Durability . . . 177
10.1.5 Transazione in SQL . . . 177
10.1.6 Savepoints in PostgreSQL . . . 177
10.2 Viste in SQL . . . 178
10.2.1 Aggiornamento sulle viste . . . 178
10.2.2 Interrogazioni sulle viste . . . 179
10.2.3 Interrogazioni non standard . . . 180
10.2.4 Uso delle viste per evitare la nidificazione . . . 180
10.3 Viste ricorsive . . . 181
10.4 Controllo dell'accesso in SQL . . . 182
10.4.1 Concessione di privilegi . . . 183
10.4.2 Tipi di privilegi . . . 183
10.4.3 Revoca di privilegi . . . 183
10.4.4 Autorizzazioni . . . 184
10.4.5 Vincoli di integrità generici: CHECK . . . 184
10.5 Trigger in SQL . . . 185
10.5.1 Sintassi . . . 186
10.5.2 Tipi di eventi . . . 186
10.5.3 Granularità degli eventi . . . 187
10.5.4 Esecuzione Trigger in conflitto . . . 187

11 Normalizzazione **189**
11.1 Dipendenza funzionale . . . 190
11.1.1 Proprieta' . . . 191
11.1.2 Banali . . . 191
11.2 Forma normale di Boyce e Codd (BCNF) . . . 191
11.3 Terza forma normale (3NF) . . . 194
11.3.1 Decomposizione in terza forma normale . . . 194

12 Non standard DBMS **199**
12.1 Basi di dati multimediali . . . 200
12.1.1 Interrogazioni . . . 200
12.1.2 Ricerca in documenti . . . 201
12.2 Gestione di dati spaziali e spazio-temporali . . . 201
12.3 Gestione di dati spaziali e spazio-temporali . . . 201
12.3.1 Estensioni spaziali dei RDBMS . . . 201
12.3.2 Estensione temporale per i RDBMS . . . 202
12.3.3 Tempo di transazione . . . 203
12.4 NoSQL DBMS . . . 204
12.4.1 Esempio: Web loggin . . . 204
12.4.2 Nuovi modelli dei dati . . . 204

12.4.3 Scalabilità 205
12.4.4 Famiglie 206
12.4.5 Query language 208

Chapter 1

Introduzione

1.1 Programma

- Introduzione ai DBMS: sistemi di gestione delle basi di dati
- Architetture Web-DS: le basi di dati di usano in concomitanza con un sistema web per esporre i dati che chiediamo interrogando la base di dati
- Il modello relazionale: la parte fondamentale del corso, eisstono dei modelli astratti per relazionare i dati
- Progettazione concettuale (lui ritiene la parte piu' difficile del corso): sembra facile ma e' la parte del corso in cui dobbiamo usare piu' intelligenza e ragionamento.
- Progettazione logica. Regole per passare da progettazione concettuale e schemi relazionali (come fossero tante tabelle), in parte anche automatizzata
- Algebra relazionale, e' uno strumento formale e fondamentale che serve a formulare delle interrogazioni sui dati
- Calcolo relazionale
- Il linguaggio SQL: e' il linguaggio usato da moltissimi anni per interrogare le basi di dati
- La normalizzazione: vedremo soltanto superficialmente, potremo trovare un cliente che vuole un data base con delle garanzie di qualita'
- Introduzione a nuovi modelli e sistemi per la gestione dei dati

LEZIONI: 8.45 - 10.15, VUOLE TUTTI IN AULA ALLE 8.40, il giovedi' con 3 ore facciamo l'intervallo in mezzo

1.2 Informazione, dati e loro gestione

1.2.1 Informatica

La scienza che studia i processi di rappresentazione, gestione e trasformazione dell'informazione e la loro automazione.

1.2.2 Gestione informazioni

- Raccolta, acquisizione
- Archiviazione, conservazione
- Elaborazione, trasformazione, produzione
- Distribuzione, comunicazione, scambio

Nelle nostre attivita' umane gestiamo le informazioni in forme diverse:

- Idee informali, un idea comunicata a qualcun'altro porta delle informazioni e la comunichiamo tramite il linguaggio naturale
- Disegni, grafici schemi, li usiamo come supporto per comunicare le informazioni
- Numeri e codici

e vari supporti:

- Mente umana, carta, dispositivi elettronici

1.2.3 Sistema Informativo

Componente di una organizzazione che gestisce le informazioni di interesse (cioe' utilizzate per il perseguimento degli scopi dell'organizzazione). E' l'ente che gestisce i dati dell'azienda. Ogni organizzazione ha un sistema informativo (automatizzato o meno).

1.2.4 Informazione e dati

Nei sistemi informatici le <u>informazioni</u> vengono spesso rappresentate in modo essenziale, compatto, attraverso i <u>dati</u>. L'informazione possiamo pensarla come la semantica dell'idea e per rappresentarla, in modo conciso le diamo una forma tramite i dati. Spesso lasciando implicita o ambigua la loro interpretazione, difficile dai dati tornare all'idea. Definizioni del vocabolario;

- Informazione:
  - Notizia, dato o elemento che consente di avere conoscenza piu' o meno esatta di fatti, situazioni, modi di essere
- Dato:
  - cio' che e' immediatamente presente alla conoscenza, prima di ogni elaborazione; (in informatica) elementi di informazione costituiti da simboli che debbono essere elaborati

Abbiamo dei dati con delle rappresentazioni diverse, ma a prima vista e' difficile capire cosa significano. Quando ci aggiungo una rappresentazione grafica del cartello sappiamo ora che in Finlandia sono degli orari, ma ancora non sappiamo la differenza tra i simboli, ad esempio le parentesi o il rosso. Perche' questo esempio? perche' i numeri sono dati e l'interpretazione e' quella che ci ha dato lui, ovvero divieto di sosta il sabato. L'informazione e' il dato + la sua interpretazione, senza interpretazione abbiamo ben poche informazioni.

Figure 1.1: Dati e informazioni

1.2.5 Organizzazione delle informazioni

Questo corso serve a gestire i dati ed implicitamente anche le informazioni. Che e' un task importante per l'informatica gestire i dati e di conseguenza le informazioni. I dati sono spesso il risultato di forme di organizzazione e codifica delle informazioni, spesso i dati si organizzano in modo strutturato, ad esempio, nei servizi anagrafici e nel riferimento a persone ci sono vari dati:

- descrizioni discorsive
- nome e cognome
- estremi anagrafici
- codice fiscale

tutta una serie di dati che corrispondono ad informazioni specifiche.

1.2.6 Perche' i dati?

La rappresentazione precisa di forme piu' ricche di informazione e conoscenza e' difficile. I dati sono una risorsa strategica, perche' piu' stabili nel tempo di altre componenti (processi, tecnologie, ruoli umani).

1.2.7 Base di dati

(accezione generica, metodologica) Una base di dati e' un insieme organizzato di dati utilizzato per il supporto allo svolgimento delle attivita' di un ente (azienda, ufficio, persona)

(accezione specifica) E' un insieme di dati che facciamo gestire ad un sistema di gestione dei dati DBMS (data base management system)

1.3 I DBMS

Cosa sono i sistemi di gestione dei dati, perche' dobbiamo studiarli e perche' non possiamo solo usare dei file. In qualche modo anche i file system sono un modo di gestione dei file. Una cosa che non sappiamo e' che in ogni smartphone c'e' un DBMS.

un DataBase Management System e' un sistema che gestise collezioni di dati:

- Grandi
- Persistenti: non vogliamo perdere i dati
- Condivise: in molti casi sono dati condivisi tra piu' utenti

Un DBMS garantisce:

- Privatezza
- Affidabilita'
- Efficienza
- Efficacia

1.3.1 I DBMS oggi

- Sistemi basati sul modello ralazionale (DB2, Oracle, MySQL, PostgreSQL)
- Sistemi emergenti per applicazioni Internet (no-SQL DBMS) basati su modelli non relazionali come chiave-valore, singola (MongoDB) colonna, grafo. offerte dai servitori di cloud, fanno dei database particolari perche' le loro infrastrutture sono sistemi distribuiti e quindi le tecnologie che ci offrono sono ottimizzate per distribuire il carico ed i dati
- Sistemi relazionali distribuiti altamente scalabili (Google Spanner)

1.3.2 Le basi di dati sono... grandi

Dimensioni (molto) maggiori della memoria centrale dei sistemi di calcolo utilizzati. Ad esempio la marea di dati che i social network acquisiscono ogni giorno, ma non sono quelli che generano piu' dati, ad esempio i sistemi di acquisizione dati di dati scientifici e per ultimo i dati dalla sensoristica (industria 4.0, mira ad integrare nei processi l'utilizzo di sensori)

1.3.3 Le basi di dati sono ... persistenti

Tempo di vita piu' lungo e indipendente dalle singole esecuzioni dei programmi che lo utilizzano. Se veniamo chiamati a fare un'applicazione, noi potremmo raccogliere i dati, utilizzarli e finita l'applicazione buttare via tutto. Spesso pero' i dati non vengono buttati quando ho finito un'applicazione, ma servono per tante altre applicazioni.

Problemi

- Ridondanza: informazioni ripetute
- Rischio di Incoerenza: se manteniamo le informazioni in piu' posti se quando vengono aggiornate, non vengono aggiornate tutte abbiamo un problema di incoerenza

Quindi per condividere i dati e' comodo far accedere tutte le applicazioni ad una stessa base di dati.

1.3.4 Le basi di dati sono... condivise

Sono una risorsa integrata a condivisa tra le applicazioni. Sorge il problema della privatezza, deve esserci un meccanismo per capire chi puo' vedere cosa. Bisogna essere autorizzati, autenticarsi via software sul DBMS che decidera' su che dati possiamo accedere. L'altro problema e' il controllo della concorrenza, cosa succede se piu' applicazioni accedono alla stessa base di dati.

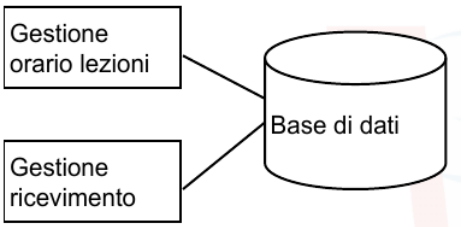


Figure 1.2: DataBase condiviso

1.3.5 I DBMS garantiscono... affidabilita'

Resistenza a malfunzionamenti Hardware e Software. Una base di dati e' una risorsa pregiata e quindi deve essere conservata a lungo termine (journaling per salvare dei log). Tecnica fondamentale: gestione delle transazioni

1.3.6 Transazione

Insieme di operazioni da considerare indivisibile ("Atomico", o riesce a farle tutte o torna allo stato iniziale delle operazioni), corretto anche in presenza di concorrenza e con effetti definitivi.

- Una sequenza di operazioni correlate atomiche: trasferimento da un conto A ad un conto B, ci sono una serie di operazioni sotto, che devono essere fatte tutte insieme, o riesci a farle tutte oppure aspetta, torna indietro

- L'effetto delle transazioni dev'essere coerente: se due assegni emessi sullo stesso conto corrente vengono incassati contemporaneamente. Cosa succede se ci sono due transazioni sullo stesso DB da piu' utenti, il DBMS fa prima tutte quelle di un certo utente e poi faccio quelle dell'altro utente, questo e' un modo inefficiente di fare le cose. Per essere efficiente il DB mischia le operazioni di tutti, e' pericoloso ma aumenta la velocita' complessiva del sistema
- I risultati delle transazioni sono permanenti: la conclusione positiva di una transazione corrisponde ad un impegno a mantenere traccia del risultato in modo definitivo, anche in presenza di guasti, esiste un'istruzione commit che rende stabile il risultato

1.3.7 I DBMS devono essere efficienti

Cercano di utilizzare al meglio le risorse di spazio di memoria e tempo. Ho dei benchmark per capire qual'e' il migliore

1.3.8 I DBMS devono essere efficaci

Cercano di rendere produttive le attivita' dei loro utilizzatori, offrendo funzionalita' articolate, potenti e flessibili. Noi come programmatori come ci troviamo ad utilizzare un certo DBMS rispetto che ad un'altro, questa e' l'efficacia, la comododita' d'uso degli utente

1.3.9 DBMS vs File System

La gestione di insieme di dati grandi e persistenti e' possibile anche attraverso sistemi piu' semplici. I file system prevedono forme rudimentali di condivisione, "tutto o niente", su un certo file io posso dire che l'utente puo' leggerlo, scriverlo, cancellarlo, non posso specificare una singola parte. I DBMS estendono le funzionalita' del File System fornendo piu' servizi ed in maniera integrata, i dati nel DBMS sono strutturati diversamente (non semplice sequenza di byte) e sono anche pensati per essere utilizzati da piu' utenti (gestione della concorrenza) ed in generale offrono piu' servizi ed anche la condivisione a grana piu' fine. Ed anche un modo per andare a pescare i dati in modo articolato.

1.3.10 Descrizione dei dai

Nei programmi tradizionali abbiamo ideato un'applicazione e pensato a nostro uso e consumo come codificare i dati in un file, ognuno di noi ha ideato il modo di organizzare i dati, invece il progettista di una base di dati non lo fa per una specifica applicazione ma cerca di modellare il dominio a servizio di tutte le applicazioni che insisteranno sui dati. Nei DBMS pubblica un catalogo o dizionario che spiega come vengono organizzati i dati, in modo tale che quando arriva una terze parte consulta le informazioni e sa come andare a pescare i dati con il linguaggio che serve per andare ad interrogare il DB.

1.4 I modelli dei dati

1.4.1 Descrizione dei dati nei DBMS

Abbiamo diversi livelli di astrazione, tendenzialmente abbiamo un dominio e noi dobbiamo decidere come memorizzare i dati, non come un'insieme di file ma avendo una struttura logica. Rappresentare le cose con livelli diversi di astrazione e' stato fatto anche in altri settori dell'informatica in cui usare questo approccio e' stato efficace. I programmi fanno riferimento alla struttura a livello piu' alto, e le rappresentazioni sottostanti possono essere modificate senza necessita' di modifica dei programmi: a basso livello il DBMS potrebbe rappresentare i dati a livello particolare sul disco e chi progetta il DBMS si preoccupa di utilizzare strutture efficienti, ma chi scrive programmi (chi scrive il sito web) non sa niente di come i dati vengono memorizzati, sono quindi indipendenti dalla loro struttura fisica (come vengono memorizzati).

1.4.2 Modello dei dati

E' un insieme di concetti utilizzati per organizzare i dati di interesse e descriverne la struttura. E' un livello di astrazione superiore a come i dati vengono strutturati fisicamente. E' fatto da costruttori di tipo (meccanismi di strutturazione): nel modello principale questo costrutto e' la RELAZIONE, possiamo immaginarcela come una tabella, che sara' il modello concettuale (insieme di tabelle), che non ha nulla a che fare con come sono veramente immagazzinati i dati.

Esempio di rappresentazione su web di una tabella, che e' una matrice le cui colonne hanno un'etichetta semantica in alta e ciascuna riga contiene dei dati omogenei, in questo caso le informazioni relative ad ogni insegnamento.

UNIVERSITA' DEGLI STUDI DI CHISSADOVE

Corso di Studi in Ingegneria Informatica

ORARIO DELLE LEZIONI PER L'ANNO ACCADEMICO 1999-2000

INSEGNAMENTO	Docente	Aula	Orario
Analisi matematica I	Luigi Neri	N1	8:00-9:30
Basi di dati	Piero Rossi	N2	9:45-11:15
Chimica	Nicola Mori	N1	9:45-11:30
Fisica I	Mario Bruni	N1	11:45-13:00
Fisica II	Mario Bruni	N3	9:45-11:15
Sistemi informativi	Piero Rossi	N3	8:00-9:30

Figure 1.3: Esempio organizzazione dati su sito Web

Esempio di dati organizzati in forma tabellare, la tabella e' il costruttore

Orario

Insegnamento	Docente	Aula	Ora
Analisi matem. I	Luigi Neri	N1	8:00
Basi di dati	Piero Rossi	N2	9:45
Chimica	Nicola Mori	N1	9:45
Fisica I	Mario Bruni	N1	11:45
Fisica II	Mario Bruni	N3	9:45
Sistemi inform.	Piero Rossi	N3	8:00

ht!]

Figure 1.4: Organizzazione dei dati in forma tabellare

del nostro modello. Quando dice indipendenza dei dati nella rappresentazione fisica, i dati sono organizzati cosi' solo per la parte concettuale, ma fisicamente i dati sono organizzati in modo diverso a basso livello.

1.4.3 Schema e istanza

In questo esempio lo schema, dal punto di vista concettuale, e' quello che mi dice che nella prima colonna mettero' i dati relativi all'insegnamento etc. E' solo la struttura concettuale senza dati. L'istanza sono i dati, quando popolo la tabella diventa un'istanza, ovviamente io posso modificare l'istanza, perche' se cambiano i dati andro' a cambiare l'elemento. Ci sono anche modalita' per cambiare lo schema, posso aggiungere colonne alla tabella, chiaramente fare operazioni sullo schema e' un po' piu' complicato perche' vado a modificare veramente la struttura della base dati. Ci saranno istruzioni diverse per lavorare con uno o con l'altro.

Tabella Orario

Schema della base di dati

Insegnamento	Docente	Aula	Ora
Analisi matem. I	Luigi Neri	N1	8:00
Basi di dati	Piero Rossi	N2	9:45
Chimica	Nicola Mori	N1	9:45
Fisica I	Mario Bruni	N1	11:45
Fisica II	Mario Bruni	N3	9:45
Sistemi inform.	Piero Rossi	N3	8:00

Istanza della base di dati

Figure 1.5: Schema e Istanza

Definiamoli formalmente, in ogni base di dati esistono:

- lo **SCHEMA**, sostanzialmente invariante nel tempo, che ne descrive la struttura (aspetto intensionale), ad esempio le intestazione delle tabelle
- l'**ISTANZA**, i valori attuali, che possono cambiare anche molto rapidamente (aspetto estensionale), ad esempio il corpo di ciascuna tabella

1.4.4 Due tipi (principali) di modello

Un idea del fatto che lavoreremo con il modello relazionale (tabelle) l'abbiamo avuta. Una classe di modelli relazionali si chiamano modelli logici, per fare le interrogazioni sulle basi di dati relazionali andremo a vedere com'e' fatto il modello logico e quello relazionale. Esiste anche una classe di modelli concettuali: sono ad un livello di astrazione piu' alto, e perche' dovrebbe servirci un livello di astrazione piu' alto se tanto quando scriviamo dobbiamo rifarci al modello logico? Per riuscire a dominare la complessita' della realta' che dobbiamo modellare, ovvero avvicinarci di piu' alle idee che abbiamo. Ci serve per passare dal linguaggio naturale a quello tecnico, e per evitare errori di traduzione c'e' uno strumento ovvero usare un modello concettuale dal quale riusciremo a creare lo schema logico che potranno usare tutti i programmatori per usare il DB.

Modelli logici

Utilizzati dai programmi ed indipendenti dalle strutture fisiche. Esempi: relazionale, chiave-valore (estremamente semplice e utilizzato da DB non relazionali che sono molto piu' semplici e sono utili quando i dati non hanno delle relazioni complesse tra loro che dobbiamo sfruttare quando dobbiamo fare delle interrogazioni e sono piu' efficienti quando dobbiamo solo trovare il valore associato ad un'etichetta), a grafo (che sono utilizzati molto nei social network, in quanto i dati sono modellabili facilmente tramite un grafo), a oggetti (si e' pensato anche di realizzare dei DBMS ad oggetti), basato su XML. I DBMS di oggi, anche relazionali, che sono i piu' collaudati in realta' si sono dimostrati efficaci anche oggi e riescono a supportare anche rappresentazioni XML o ad oggetti ma sempre nel modello relazionale.

Modelli concettuali

Ce ne sono anche qui di diverso tipo: Entity-Relationship (UML). Permettono di rappresentare i dati in modo indipendente da ogni sistema, cercano di descrivere i concetti del mondo reale e sono utilizzati nelle fasi preliminari di progettazione.

1.5 Architettura (semplificata) di un DBMS

Vediamo in basso con il simbolo della memoria di massa una base dati, normalmente dato che abbiamo parlato di memorizzazione di grosse quantita' di

dati (Tb o PetaByte) devono finire sulla memoria di massa (non posso tenerli in memoria centrale), la persistenza e' realizzata da una memorizzazione su memoria di massa. Poi abbiamo uno schema interno, strutture dati efficienti per prelevare i dati da disco (bit tree), servono per rappresentare i dati su disco. Sopra abbiamo uno schema logico, l'esempio e' delle relazioni rappresentate come tabelle. L'utente e' il programmatore, e come riferimento ci serve lo schema logico e basta.

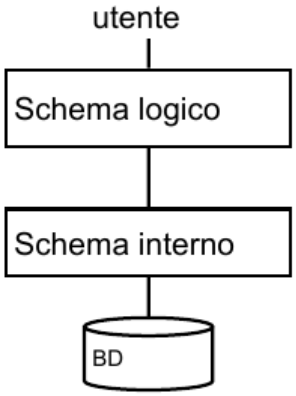


Figure 1.6: Architettura semplificata di un DBMS

1.5.1 Architettura a tre livelli per DBMS

Il programmatore potrebbe anche vedere uno schema logico virtuale, SCHEMA ESTERNO, perche' dovendo fare, ad esempio, il DB di tutta l'universita', ci sono alcune applicazioni che sono specifiche per la segreteria di informatica allora per questi utenti (i programmatori delle applicazioni che accedono solo a dati di informatica) e' inutile fargli vedere lo schema di tutta l'universita' con tutti i dati, allora gli si fa vedere solo un pezzetto dello schema logico o una VISTA particolare ed utilizzeranno quella vista, attenzione che stiamo sempre parlando di schema logico, non siamo arrivati a quello concettuale, e' solo una vista sullo schema logico.

1.5.2 Vista

E' importante fare un esempio di cos'e' una vista. Supponiamo che questi corsi ed aule siano due tabelle dello schema logico (modello logico relazionale) e non vogliamo dare queste due tabelle ad un programmatore, gli vogliamo dare solo una visione parziale che definiamo cosi' (CorsiSedi), in cui vediamo che non ci sono i docenti, le informazioni che ci sono dentro e le etichette nella struttura della tabella derivano dalle tabelle sopra. Questo e' un modo di dare uno schema esterno nascondendo le strutture del DB. Se l'applicazione lavora

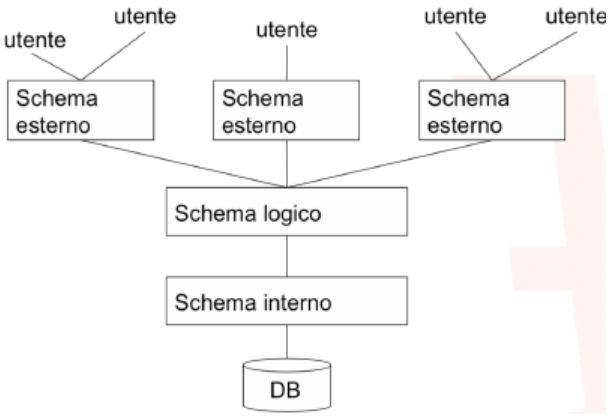


Figure 1.7: Architettura a tre livelli per DBMS

solo sulla tabella sotto, quando vuole inserire una nuova riga (tupla), perche' c'e' un nuovo corso, l'applicazione non ha l'accesso alla tabella corsi, deve inserirlo su CorsiSedi, quindi il DBMS deve vedere questo inserimento e calarlo nel DB vero che ha le due tabelle sopra, il problema in questo caso sarebbe che non abbiamo il nome del docente, esistono dei simboli speciali per dire che quell'informazione non e' conosciuta.

Corsi

Corso	Docente	Aula
Basi di dati	Rossi	DS3
Sistemi	Neri	N3
Reti	Bruni	N3
Controlli	Bruni	G

Aule

Nome	Edificio	Piano
DS1	OMI	Terra
N3	OMI	Terra
G	Pincherle	Primo

CorsiSedi

Corso	Aula	Edificio	Piano
Sistemi	N3	OMI	Terra
Reti	N3	OMI	Terra
Controlli	G	Pincherle	Primo

Figure 1.8: Vista

1.5.3 Indipendenza dei dati

E' una proprieta' desiderabile l'indipendenza dei dati per il DBMS, perche' se devo cambiare l'organizzazione del DBMS mi basta cambiare la parte fisica ma lasciando intatto lo schema logico, quindi non devo modificare le applicazioni

sopra. L'accesso ai dati avviene tramite il livello esterno (o la vista o il livello logico). Questa e' l'indipendenza fisica (cambiare rappresentazione su disco senza cambiare le tabelle (schema logico)). L'indipendenza logica invece ha a che fare con le osservazioni fatte sul modificare le viste. Si ha indipendenza logica quando lo schema logico puo' essere modificato senza modificare gli schemi esterni, e' uguale all'indipendenza fisica ma spostato su di un livello (se io modifico quelle tabelle aule e corsi posso lasciare al programmatore la vista costruita su queste due, vorrei che fosse cosi'), deve pero' limitare delle operazioni: se nell'esempio prima togliessi nella tabella aule una colonna (operazione di schema sullo schema logico) questo ha un impatto sull'applicazione che usa la vista creata da quella tabella. Nella pratica pochi DBMS hanno implementato completamente l'indipendenza logica.

1.6 Linguaggi

1.6.1 Linguaggi per basi di dati

- SQL e' lo standard per i DBMS relazionali
- Non esiste uno standard di linguaggio per i noSQL DBMS: query piu' efficienti se non dobbiamo percorrere relazioni complesse tra i dati

1.6.2 Linguaggio SQL standard

Viene usato in vari modi:

- All'inizio lo utilizzeremo testuale interattivo da terminale, vuol dire che do un istruzione alla volta (do istruzioni ad un server tramite un client)
- Nella maggior parte dei casi utilizzeremo dei comandi SQL immersi all'interno di un linguaggio di programmazione
- SQL immerso in un linguaggio ad hoc con anche altre funzionalita'
- Implicito con interfacce amichevoli

Corsi

Corso	Docente	Aula
Basi di dati	Rossi	DS3
Sistemi	Neri	N3
Reti	Bruni	N3
Controlli	Bruni	G

Aule

Nome	Edificio	Piano
DS1	OMI	Terra
N3	OMI	Terra
G	Pincherle	Primo

Figure 1.9: SQL: esempio

Esempio trovare i corsi tenuti in aule al piano terra:

```
SELECT Corso, Aula, Piano
FROM Aule, Corsi //in che tabelle prendere i dati che mi servono
WHERE Nome = Aula
    AND Piano = 'Terra'
```

La prima cosa che ci consiglia di fare e' capire di quali tabelle ho bisogno, quindi lui comincerebbe a mettere aule di fianco al FROM (da dove peso i dati), ovviamente non posso fare solo con le aule e ci metto di fianco i corsi. La seconda parte piu' importante e' il WHERE (metto le condizioni per selezionare le righe), tra tutti i dati nelle tabelle che ho recuperato con il FROM mi servono solo i dati legati da un vincolo, nome e' il nome di una colonna, capiamo che il programmatore deve avere bene in mente lo schema logico, questo nome = aula e' la cosa fondamentale per incrociare i dati di queste due tabelle, perche' devo capire se le aule coincidono, sto dicendo che di tutti i dati di queste tabelle prendi le righe in cui nome = aula. AND dice che dev'essere vera Nome = Aula e che Piano = Terra, perche' siccome stiamo cercando il valore terra ci interessano solo le righe in cui piano = terra. Identifico tutte le righe che rispettano le condizioni, di cui mi interessa solo il corso, che e' quello che metto in SELECT, nel select devo restituire i corsi, quindi ci metterebbe corso, quindi andrebbe bene mettere solo corso.

SQL immerso in un linguaggio ospite

Dall'esempio e' chiaro che e' un linguaggio di programmazione, ci interessa vedere com'e' possibile inserire SQL nel linguaggio di programmazione (SELECT, FROM, WHERE). Ci da l'idea che possiamo usare delle variabili e fare dei cicli per fare le interrogazioni.

1.6.3 SQL immerso in un linguaggio ad hoc

1.6.4 Due tipi di linguaggi

- Data Manipulation Language (DML): a livello astratto si parla di DML che e' un linguaggio che fornisce istruzioni per interrogare ed aggiornare istanze di basi di dati, manipola le istanza, manipolare significa inserire, cancellare, modificare, leggere dati (istanze)

- Data Definition Language (DDL): e' un linguaggio che ci permette di lavorare sugli schemi dalla base di dati (aggiungere, togliere una colonna)

Un'operazione di DDL

CHAR(N) vuol dire che nella colonna mi aspetto una stringa di N caratteri.

```
CREATE TABLE orario (
insegnamento    CHAR(20),
docente         CHAR(20),
aula            CHAR(4),
```

```
write('nome della citta''?'); readln(citta);
EXEC SQL DECLARE P CURSOR FOR
     SELECT NOME, REDDITO
     FROM PERSONE
     WHERE CITTA = :citta ;
EXEC SQL OPEN P ;
EXEC SQL FETCH P INTO :nome, :reddito ;
while SQLCODE = 0 do begin
    write('nome della persona:', nome, 'aumento?');
    readln(aumento);
    EXEC SQL UPDATE PERSONE
                SET REDDITO = REDDITO + :aumento
                WHERE CURRENT OF P
    EXEC SQL FETCH P INTO :nome, :reddito
  end;
EXEC SQL CLOSE CURSOR P
```

Figure 1.10: SQL immerso in un linguaggio di programmazione

```
declare  Stip number;
begin
    SELECT STIPENDIO INTO STIP FROM IMPIEGATO
    WHERE MATRICOLA = '575488' FOR UPDATE OF STIPENDIO;
    if Stip > 30 then
      UPDATE IMPIEGATO SET STIPENDIO = STIPENDIO * 1.1
            WHERE MATRICOLA = '575488';
    else
      UPDATE IMPIEGATO SET STIPENDIO = STIPENDIO * 1.15
            WHERE MATRICOLA = '575488';
    end if;
    commit;
  exception
    when no_data_found then
     INSERT INTO ERRORI
            VALUES('MATRICOLA INESISTENTE',SYSDATE);
  end;
```

Figure 1.11: Oracle PL/SQL

ora CHAR(5))

Chapter 2

Architettura Web con i DataBase

2.1 Modello client-server

2.1.1 Architettura DBMS

Siccome i DBMS devono garantire la persistenza si appoggiano alla memoria di massa per garantirla, quindi vediamo nel disegno nella parta bassa che abbiamo due tipi di informazioni persistenti su memoria di massa:

- Stored Database: i dati veri e propri
- Stored Database Definiton (Meta-Data): ovvero informazioni su come sono strutturati i dati

Per quanto riguarda il programma che viene lanciato nel server DBMS ha due moduli:

- Software to Access Stored Data: il modulo che deve interfacciarsi con la memoria di massa
- Software to Precesso Queries/Programs: il modulo che serve per fare le interrogazioni al DBMS

Dopo di che in alto abbiamo gli utenti che formulano le query attraverso applicativi (PHP) e mandano le query in esecuzione.

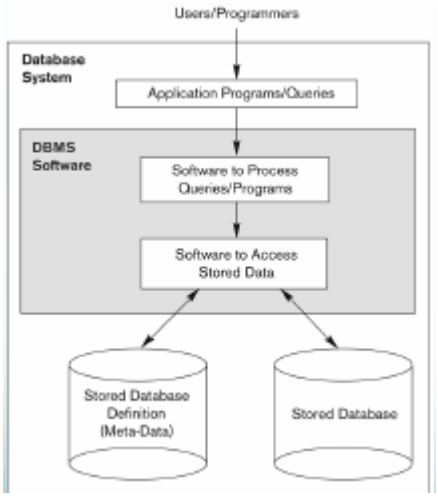


Figure 2.1: Architettura semplificata di un DBMS

2.1.2 Architettura logica client-server per accesso al DMBS

I DBMS nel 99% dei casi vengono utilizzati in modalita' client-server, noi abbiamo un software che si comporta da server ed esegue il DBMS ed un software

che fa da client che richiede le funzionalita' del DBMS. Cosi' come se avessimo un print server che gestisce la coda delle stampe abbiamo per un DBMS server che gestisce la coda di richieste di vari client connessi in rete.

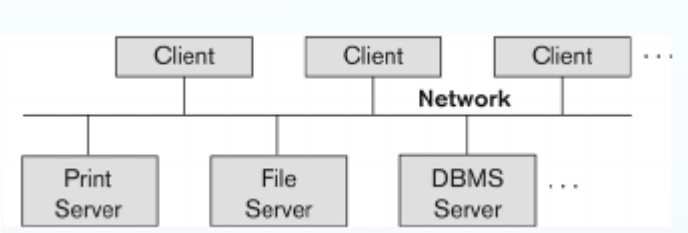


Figure 2.2: Architettura logica client-server per accesso a DBMS

2.1.3 Diversi tipi di accesso al DBMS

Se so che c'e' un certo DBMS che hanno sviluppato altri e se devo fare un programma (fare l'orario delle lezioni di informatica) posso accedere al batabase pero' l'anno prossimo prevedono di cambiare DB, allora se sono sicuro che il DBMS e' uno solo e voglio fare una cosa rapidamente probabilmente uso una libreria (come faccio ad usare delle istruzioni che chiedono di collegarsi a quel tipi di DBMS e mandano istruzioni a quel DBMS). Ci sono varie alternative per interagire con un DBMS:

- Il linguaggio di programmazione che sto usando ha delle istruzioni per usare quel DB li' ed uso queste, quindi le applicazioni usano API per accedere al DBMS.

- In realta' non sappiamo che DBMS il cliente utilizza, oppure si prevede di cambiarlo ma non si sa ancora' qual'e'. Dobbiamo quindi fare un software che funzioni con diversi DBMS, non possiamo quindi cercare che istruzioni ci sono per uno specifico, dobbiamo usare un interfaccia che funzioni su tanti DBMS. Ad esempio si potrebbe usare ODBC che funziona per tutti i DBMS che implementano questa interfaccia.

ODBC e JDBC sono degli standard per connettersi ad un DBMS senza specificare un particolare DBMS, quando poi si va ad implementare il programma veramente dobbiamo preoccuparci che il DBMS abbia un driver ad esempio JDBC.

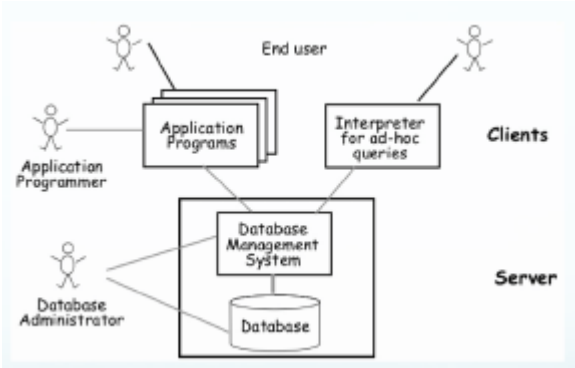


Figure 2.3: Diversi modi per accedere ad un DBMS

2.1.4 Client-Server nel web

Io posso fare una richiesta di connessione attraverso internet ad un database semplicemente tramite un'interfaccia di rete, non centra nulla con il web. Noi quindi se abbiamo la nostra applicazione da sviluppare e vogliamo collegarci al DBMS non per forza dobbiamo farlo tramite web, ma possiamo usare il servizio TCP/IP e la porta su cui risponde il DBMS. Ovviamente siccome il web si e' diffuso moltissimo e ci sono applicazioni che sfruttano non solo TCP/IP ma anche il livello sopra HTTP/HTTPS esiste un modo per interagire con il DBMS tramite WEB. Prima cosa l'interazione classica tra un client ed un web server e' fatta da una richiesta HTTP ed una risposta, e nel caso piu' banale e' la richiesta di un file HTML, il server va a cercare il file e lo restituisce.

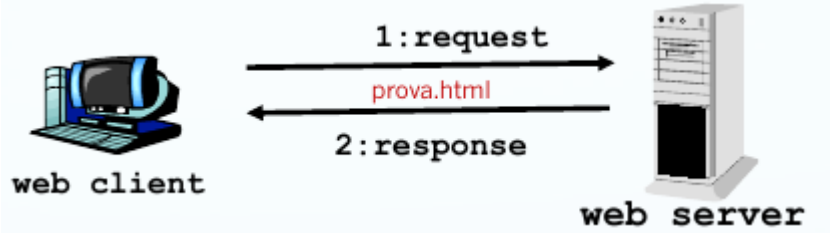


Figure 2.4: Interazione client-server nel web

2.1.5 Caricamento di una pagina statica

Il client fa una richiesta HTTP al server di una pagina HTML al web server, il quale cerca la pagina e la restituisce come la trova al client.

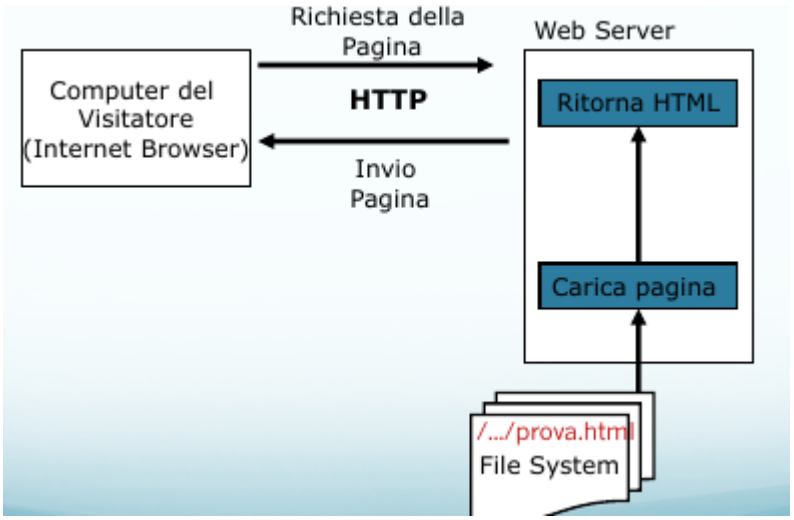


Figure 2.5: Caricamento di una pagina statica

2.1.6 Programmare per il web

La maggior parte dei siti ha dei comportamenti molto piu' complessi rispetto a restituire pagine HTML, la differenza e' che viene anche eseguito del codice, sul server o sul client. Questo codice si chiama script e viene quindi chiamato scripting lato client (JavaScript) e scripting lato server.

JavaScript

Alcune pagine possono contenere del codice che viene eseguito sul client. Quando il browser riceve la pagina:

- Esegue il codice (mediante lo "script engine")
- Visualizza la pagina

Il browser fa una richiesta HTTP ed il server restituisce HTML + pezzi di codice, dall'altra parte il processo browser sul client vede che c'e' qualcosa di piu' di HTML e chiama un suo modulo (script engine) che capisce il JavaScript che fa visualizzare la pagina correttamente.

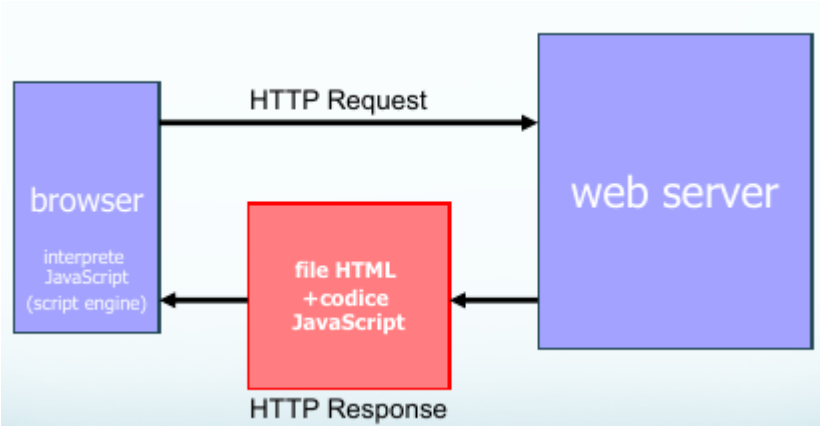


Figure 2.6: JavaScript

Script lato server e pagine web dinamiche

Per ogni pagina il web server memorizza un file che contiene:

- Marcatura (markup) HTML che e' lo scheletro della pagina
- Del codice di script

Vuol dire che scriviamo del codice PHP il quale ha dentro delle istruzioni HTML ed anche delle istruzioni per interrogare il database, pero' questo PHP non puo' essere spedito direttamente al client, quindi quando la pagina viene richiesta, il server, ad esemepio APACHE, vede che e' un file PHP e chiama un modulo che interpreta il PHP che restituisce ad APACHE un file HTML. Quindi quando la pagina viene richiesta, il server:

- Esegue il codice di script, che genera altre informazioni di marcatura
- Invia il risultato al client
- Il browser mostra la pagina

L'interprete PHP scorre le istruzioni e quando trova un pezzo che fa una query al database si interrompe e passando le credenziali fa la richiesta, il DBMS fa le sue interrogazione e restituisce al server web il quale formatta le informazioni in HTML.

Ad esempio Prova.PHP e' un file memorizzato su un server, che e' apparentemente un codice HTML:

```
<html>
    <head>
        <title>Test PHP</title>
    </head>
```

```
    <body>
        <?php echo "<p>Hello World</p>"; ?>
    </body>
</html>
```

L'interprete php esegue il codice php che contiene e restituisce un file HTML al client:

```
<html>
    <head>
        <title>Test PHP</title>
    </head>
    <body>
        <p>Hello World</p>
    </body>
</html>
```

2.1.7 Riassunto I/O del web server

		Scripting lato client	
		NO	SI'
Scripting lato server	NO	**Input**: indirizzo pagina **Output**: HTML	**Input**: indirizzo pagina **Output**: HTML + codice di scripting (eseguito dal client)
	SI'	**Input**: indirizzo pagina (+ eventuali parametri) **Output**: HTML	**Input**: indirizzo pagina (+ eventuali parametri) **Output**: HTML + codice di scripting (eseguito dal client)

2.2 Architetture WEB-DB multi-tier (multi-livello)

2.2.1 Architettura a 3 livelli

Da browser facciamo una richiesta al server web, di un file che avra' dentro del codice, questo codice include una richiesta ad un DBMS il quale fornira' una risposta. Si parla di tre livelli perche' spesso questa architettura implica anche ci siano 3 computer diversi, con diverse funzionalita':

- Presentazione della pagina
- Recupero della logica applicativa
- Gestione dei dati

In questa architettura la parte delle esecuzioni PHP viene fatta nel web server come un suo modulo. In altre architetture cosi' dette a microservizi (con container, docker etc) sono un evoluzione di questo, se io identifico che le funzionalita' del software sono piu' di 3 che ha diverse funzioni che voglio isolare, potrei dedicare un livello a seconda delle funzionalita' fino ad andare ad una granularita' molto fine. Il vantaggio e' che potrei fare si che ciascuna funzinalita' sia gestita da un gruppo di programmatori diversi che usano linguaggi di programmazione diversi.

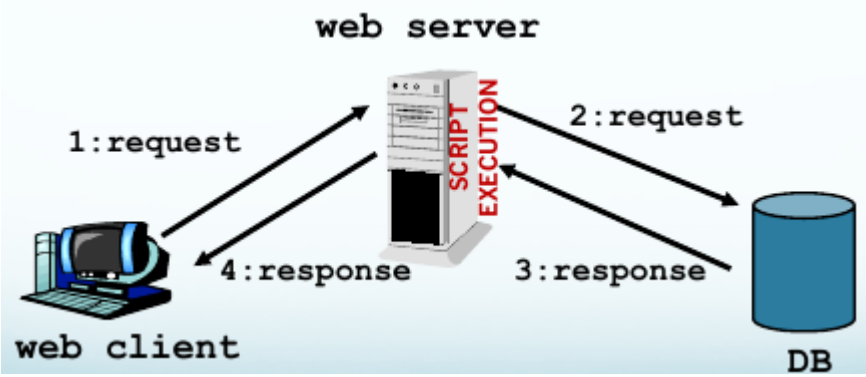


Figure 2.7: Architettura 3-Tier

2.2.2 Caricamento di una pagina dinamica

Cosa succede piu' nel dettaglio, il php dentro ai tag ha le connessioni al DB, e qui ci agganciamo al discorso di prima sullo scegliere il modo per interfacciarsi al DBMS e come visto precedentemente possiamo farlo in modo da poterci interfacciare con qualsiasi DBMS, ad esempio tramite JDBC, oppure usare le librerie di un linguaggio di programmazione per specifici DBMS.

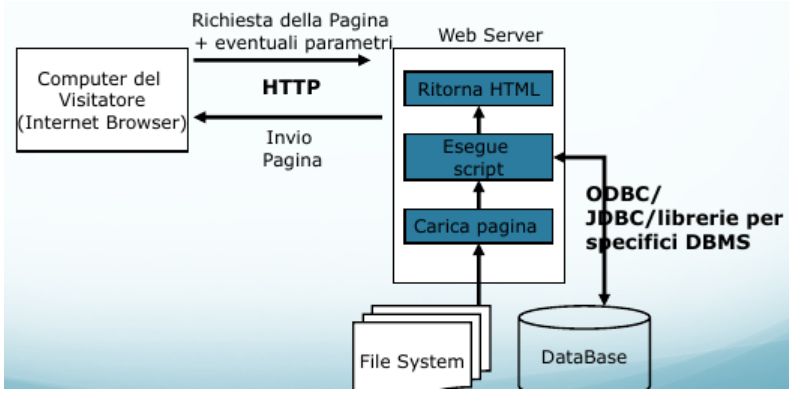


Figure 2.8: Caricamento di una pagina dinamica

2.2.3 Esempio di PHP per interrogare PostgreSQL

```
<?php
//connecting, selecting database

$dbconn = pg_connect("host=localhost dbname=aziendaDB user=WWW password=foo")
            or die('Could not connect:' .pg_last_error())
//performing SQL query

$query = 'SELECT * FROM impeiegati';
//Significa prendi tutto quello che c'e' nella tabella impiegati

$result = pg_query($query) or die('Query failed: '.pg_last_error());

...

?>
```

Tutte le istruzioni pg_ sono quelle che mi servono per interagire con il DBMS. Vediamo anche che la query la mettiamo in una variabile che poi usiamo con pg_query.

2.2.4 Architetture Web-DB con load balancing, cachine e replicazione

Questo solo per farci capire che l'architettura a tre livelli diventa piu' complicata nella realta' perche' c'e' una certa distribuzione orizzontale, intuitivamente capiamo che non puo' esistere un solo server Web per Google. App 1 e App2 sono le applicazioni che eseguono il codice lato server, e viene replicato anche nel database (in questo caso non simmetrica perche' c'e' un master ed uno slave).

E' chiaro quindi che non puo' esserci nella realta' un solo processo che riceve e gestisce le richieste di connessioni . Esiste anche tutto il sistema di caching, se mi e' appena stata chiesta una cosa l'ho memorizzata in cache e la restituisco direttamente all'utente, la cache puo' anche essere piu' vicina all'utente a livello geografico.

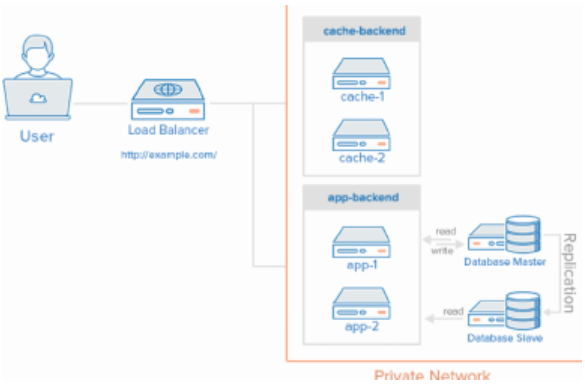


Figure 2.9: Architetture Web-DB replicate

2.3 Web application e DBMS

E' un'applicazione eseguita dentro un Internet Browser:

- Segue il paradigma client/server appoggiandosi ad un server Web ed eseguendo script in locale
- Per migliorare i tempi di risposta aggiorna indipendentemente parti diverse di una pagina
- Permette una limitata memorizzazione di dati in locale (ma solitamente i dati sono gestiti lato server)
- Non necessita installazione (viene caricata come una pagina Web)
- Altamente portabile e adattabile a dispositivi client diversi

L'esempio classico e' google docs che utilizziamo da browser ma hanno funzionalita' simili ad office classico.

2.3.1 Web application e DBMS

Cerchiamo di capire cosa centra la web application con i DBMS. Abbiamo sul browser delle porzioni della pagina, le richieste che partono dal browser al server sono quindi richieste di porzioni di pagina, come se fossero delle chiamate ad una funzione del programma, chi risponde dal server e' sempre il nostro APACHE

(server Web) che al suo interno ha dei moduli che lo aiutano ad interpretare il codice lato server. Le risposte non sono per forza dei file HTML ma potrebbero anche solo essere dei JSON o del JavaScript. Dall'altra parte (server) ci puo' essere dentro una richiesta di interrogazione ad un DBMS. Cambia poco, perche' molte di queste Web app non usano una base di dati relazionali ma usano quelli non relazionali (MongoDB, noSQL), in questo caso nel codice eseguito dal modulo lato server ci sono delle istruzioni che non sono una query SQL ma ci saranno altri modi di accedere al database e non utilizza la sintassi SQL ma una sintassi piu' semplice, che si paga perche' se devo fare delle interrogazioni piu' complesse devo fare attraverso le strutture normali di un linguaggio di programmazione.

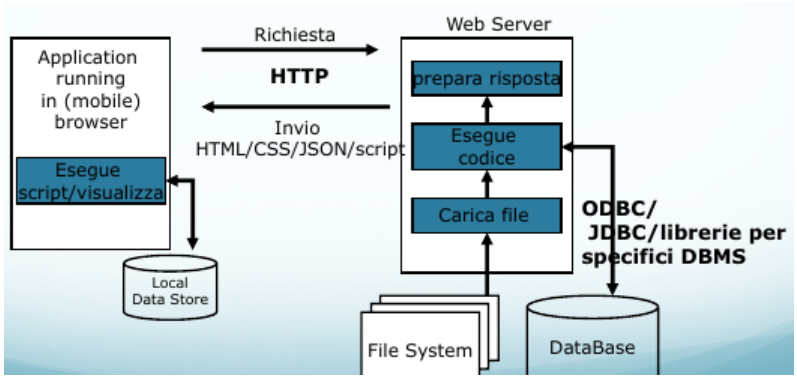


Figure 2.10: Web application e DBMS

2.4 Mobile app e DBMS

E' un'applicazione scritta in codice nativo per una certa piattaforma (Android, IOS). Rispetto ad una Web app:

- "look and feel" tipico della piattaforma (UI migliore)
- maggior reattivita', perche' non c'e' interpretazione
- accesso a componenti hardware del dispositivo, e' la cosa che e' piu' interessante, una delle difficolta' quando sto a livello di web app e' accedere a componenti hardware specifici (sensori) che fanno parte del dispositivo
- accesso completo ad un DBMS/datastore locale, come gia' abbiamo detto tutti gli smartphone hanno un piccolo DBMS a bordo e per gestirlo come DBMS dobbiamo avere del codice nativo che si interfaccia

La maggior parte delle app si appoggia a servizi esterni via HTTP

2.4.1 Mobile app e DBMS

Vediamo come potrebbe interagire una mobile app con un DBMS. Lo puo' fare sia con il suo DBMS locale che si chiama SQLite DB, che e' uno dei pochi DBMS che non funziona con l'approccio client-server perche' e' estremamente semplificato e dedicato ad un dispositivo mono-utente e funziona come una libreria che puo' essere chiamata all'interno del codice. Oppure farlo lato server, che funziona come abbiamo visto in altri casi, nell'applicazione nativa e' possibile fare delle richieste a dei servizi Web, Web service REST, ci sono delle richieste HTTP ed ovviamente non vedremo in risposta dell'HTML perche' non e' un browser. Potrebbe anche chiamare un DBMS direttamente con una connessione TCP/IP, ma quello che si fa al solito e' mandare una richiesta HTTP ad un server web REST il quale esegue del codice esterno che va ad interrogare un database esterno. La risposta e' un JSON o un modo per formattare i dati per farli vedere all'utente.

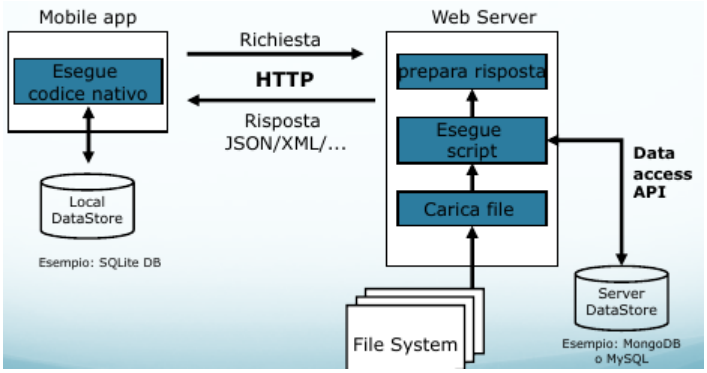


Figure 2.11: Mobile app e DBMS

Chapter 3

Modello relazionale

3.1 I modelli logici dei dati

Il modello relazionale e' un modello logico (non concettuale), quello su cui i programmatori si basano per scrivere le relazioni. I principali modelli logici sono:

- Gerarchico
- Reticolare
- Relazionale

Ne esistono in realta' anche altri:

- Ad oggetti
- A grafo
- chiave-valore (semplicissimi usati da i NO-SQL database)
- Basato su XML

3.1.1 Caratteristiche

Perche' il modello relazionale e' il piu' usato? Perche gli altri due usano dei riferimenti espliciti tra i record logici mentre il relazionale e' basato su valori, quindi anche i riferimenti fra dati in strutture diverse (le tabelle) sono rappresentati per mezzo dei valori stessi, mentre negli altri due non ci sono le tabelle ma i riferimenti sono tramite puntatori.

Per avere un'idea di cosa vuol dire riferimento per valore guardiamo l'esempio. Come facciamo a cercare i voti che ha preso Mario Rossi? Tutti lo sanno fare anche senza avere idea delle base di dati, troviamo la matricola e guardiamo nella tabella esami che voti ha preso. Il modello relazionale basato su valori significa che confrontiamo valori in colonne che semanticamente capiamo essere corrispondenti, quindi da una correlazione semantica delle etichette delle tabelle capiamo di che righe confrontare i valori.

3.2 Modello relazionale

Con quale idea si e' inventato il modello relazionale? Per favorire l'indipendenza dei dati. E' stato proposto da Codd nel 1970 princpipalemente per farvorire l'indipendenza dei dati. Da quando se l'e' inventato a quando ne abbiamo visto uno sono passati 10 anni, perche' non e' facile da implementare (1981). Codd e' partito dalla matematica e dalla teoria degli insiemi, (quindi una delle cose fondamentali che impariamo e' quella di <u>relazione</u>), nel nostro caso e' una relazione nei database e siccome deriva dalla matematica faremo un breve ripasso delle relazioni matematiche. Ci servira' capire qual'e' una relazione e quale no per capire se una tabella e' relazionale o no.

studenti

Matricola	Cognome	Nome	Data di nascita
6554	Rossi	Mario	05/12/1978
8765	Neri	Paolo	03/11/1976
9283	Verdi	Luisa	12/11/1979
3456	Rossi	Maria	01/02/1978

esami

Studente	Voto	Corso
3456	30	04
3456	24	02
9283	28	01
6554	26	01

corsi

Codice	Titolo	Docente
01	Analisi	Mario
02	Chimica	Bruni
04	Chimica	Verdi

Figure 3.1: Un esempio di modello relazionale

3.3 Relazione

Ci sono 3 significati quando si parla di relazione:

- Relazione matematica: come teoria degli insiemi
- Relazione secondo il modello relazionale dei dati
- Associazione: utilizzeremo quando parleremo di modello concettuale. Quindi anche nel modello concettuale c'e' l'idea di mettere in relazione due classi di oggetti, che noi chiameremo associazione per non confonderci.

3.3.1 Relazione matematica, esempio

Prendiamo due domini, che di fatto sono insiemi di simboli:

$$\begin{aligned} D_1 &= \{a, b\} \\ D_2 &= \{x, y, z\} \end{aligned} \tag{3.1}$$

Il prodotto cartesiano dei due insiemi sono tutte le coppie di elementi possibili, prendendo come primo elemento appartiene al primo dominio ed il secondo al secondo dominio:

$$D_1 X D_2 = \{(a, x); (a, y); (a, z); (b, x); (b, y); (b, z)\} \tag{3.2}$$

Che se rappresentiamo in forma tabellare:

a	x
a	y
a	z
b	x
b	y
b	z

Una relazione e' quindi un sottoinsieme del prodotto cartesiano.

$$r \subseteq D_1 X D_2 \tag{3.3}$$

In forma tabellare un esempio di relazione potrebbe essere:

a	x
a	z
b	y

E' chiaro che posso generalizzare ed invece che avere due domini averne N. Di conseguenza il prodotto cartesiano e' l'insieme di tutte le n-uple (d_1 ... d_n) tali che $d_1 \in D_1, ..., d_n \in D_n$ (le n-uple sono le righe della tabella). Mentre una relazione matematica su $D_1, ..., D_n$ e' un sottoinsieme del prodotto cartesiano tra tutti gli insiemi.

Proprieta'

Una relazione matematica e' un insieme di n-uple ordinate (all'interno della tupla, il primo elemento appartiene al primo dominio), ricordiamoci sempre che una relazione e' un insieme (dato che e' un sottoinsieme del prodotto cartesiano) e ricordiamoci anche che non c'e' un ordinamento tra le tuple (esternamente). Inoltre le tuple sono distinte, che significa che nella stessa tabella non ci possono essere due volte gli stessi elementi.

Esempio relazione matematica

Juve	Lazio	3	1
Lazio	Milan	2	0
Juve	Roma	0	2
Roma	Milan	0	1

E' interessante vedere che i domini (4 in totale) si possono ripetere, vorra' dire che avro' una tupla proveniente dal medesimo dominio, ma non significa che sia uguale ad un'altra. A livello matematico se ci dice che questa e' una relazione dobbiamo verificare che le proprieta' di prima siano verificate.

Ciascuno dei domini ha dei ruoli diversi distinguibili dalla posizione (anticipiamo che la differenza tra relazione matematica e modello relazionale e' che

quest'ultimo non e' posizionale), si dice che e' posizionale perche' ho specificato i domini ma non esplicitando un'etichetta semantica si capisce lo stesso il significato degli elementi all'interno della tabella, la squadra di casa e quella ospite le definiamo in base alla loro posizione nella tabella.

3.3.2 Struttura non posizionale

Vogliamo dare una semantica piu' esplicita in modo che se anche scambio la due colonne il significato non cambia. Quindi a ciascun dominio si associa un nome unico nella tabella (attributo), che ne descrive il "ruolo". Stiamo parlando della prima riga della tabella dove ci sono le etichette. Nel nostro esempio:

Casa	Fuori	RetiCasa	RetiFuori
Juve	Lazio	3	1
Lazio	Milan	2	0
Juve	Roma	0	2
Roma	Milan	0	1

3.3.3 Tabelle e relazioni

In una tabella che rappresenta una relazione:

- l'ordinamento tra le righe e' irrilevante: perche' dipende dalla relazione
- l'ordinamento tra le colonne e' irrilevante: perche' e' una struttura posizionale

Una tabella rappresenta una relazione se:

- le righe sono diverse tra loro
- le intestazioni delle colonne sono diverse tra loro
- i valori di ogni colonna sono fra loro omogenei: vuol dire che appartengono tutti allo stesso dominio (interi, stringhe...)

3.3.4 Il modello e' basato su valori

La cosa tra le piu' difficili nel fare le interrogazioni in SQL e' capire quali colonne confrontare, ci sara' un operazione JOINT che ci permettera' di confrontare i valori delle tabelle e dobbiamo capire cosa mettere insieme.

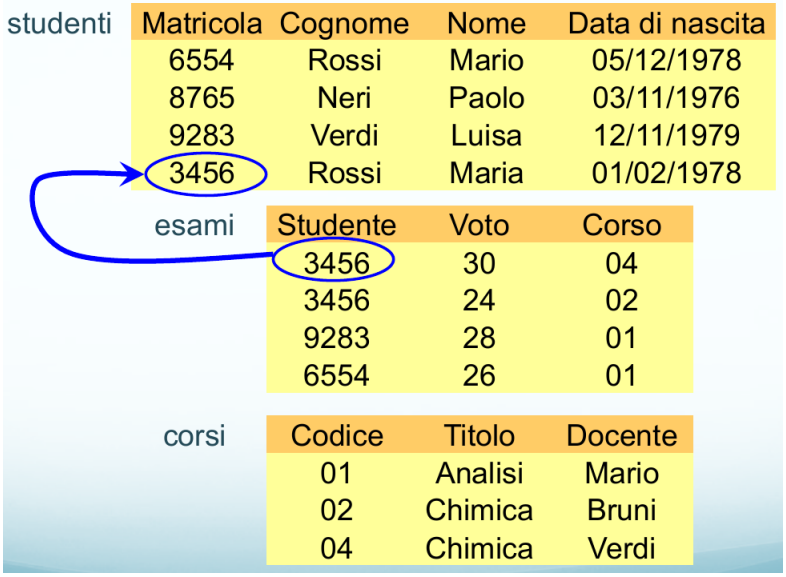


Figure 3.2: Modello basato su valori

3.3.5 Alternativa

Altri modelli (reticolare, gerarchico e ad oggetti prevedono riferimenti espliciti gestiti dal sistema). Nel senso che potrei pensarlo come una struttura dati ed invece che mettere il valore posso mettere dei puntatori ad altre tuple.

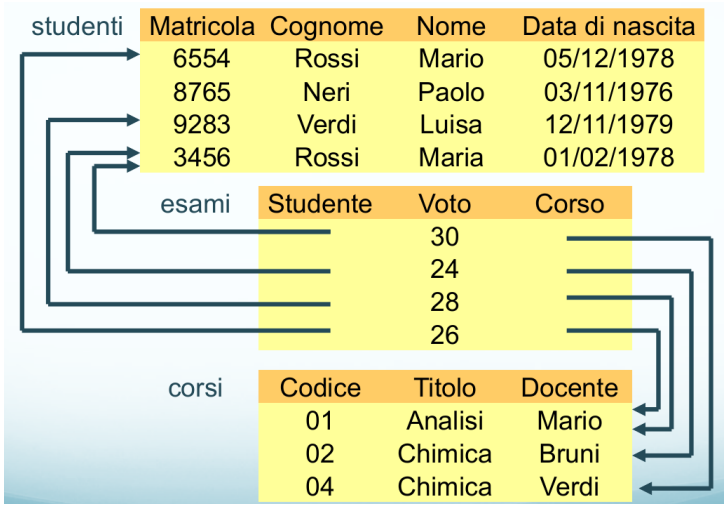


Figure 3.3: Alternativa al modello relazionale con i puntatori

3.3.6 Vantaggi struttura basata su valori

- Indipendenza dalle strutture fisiche che possono cambiare dinamicamente, il discorso e' che quei puntatori potrebbero andare bene per rappresentare fisicamente i dati, pero' dal punto di vista logico e' piu' intuitivo il modello relazionale ed anche se e' ridondante memorizzare piu' volte i valori nascondiamo la struttura dei puntatori al programmatore.
- Si rappresenta solo cio' che e' rilevante dal punto di vista dell'applicazione
- l'utente finale vede gli stessi dati dei programmatori
- i dati sono portabili piu' facilmente da un sistema ad un altro
- si evita il problema che i puntatori sono direzionali

3.3.7 Definizioni

Diamo ora delle definizioni importanti per il proseguimento del corso. Schema di relazione e' dato da un nome R con un insieme di attributi A_1, A_n che ci permettono di avere la struttura non posizionale:

$$R(A_1, ..., A_n) \tag{3.4}$$

Schema relazionale di una base di dati: ci sara' un nome dello schema relazionale(universita') ed all'interno tutti gli schemi di relazioni che appartengono alla base dati (studenti, corsi, professori...).

$$R = \{R_1(X_1), ..., R_k(X_k)\} \tag{3.5}$$

Una ennupla su un insieme di attributi X e' una funzione che associa ciascun attributo A in X un valore del dominio di A. X e' l'insieme degli attributi, la ennupla e' una funzione significa che alla riga N della tabella associo ad esempio ad uno studente tutti i suoi dati.

t[A] denota il valore della ennupla t sull'attributo A, ad esempio voglio solo il nome di uno studente.

Istanza di relazione su uno schema $R(X)$ (X e' un insieme di attributi): e' un insieme r di ennuple sugli attributi X (e' il contenuto delle tabelle).

Istanza di base di dati su uno schema ($R = \{R_1(X_1), ..., R_n(X_n)\}$) e' l'insieme delle istanze delle relazione della base di dati ($r = \{r_1, ..., r_n\}$

3.3.8 Relazioni sui singoli attributi

E' possibile avere una relazione che corrisponde ad una tabella con una sola colonna. Ad esempio posso avere una tabella generale studenti e poi avere una tabella studenti lavoratori in cui non ho bisogno di ripetere tutti i dati, perche' e' un sottoinsieme dell'altra.

studenti

Matricola	Cognome	Nome	Data di nascita
6554	Rossi	Mario	05/12/1978
8765	Neri	Paolo	03/11/1976
9283	Verdi	Luisa	12/11/1979
3456	Rossi	Maria	01/02/1978

studenti lavoratori

Matricola
6554
3456

Figure 3.4: Esempio di relazioni con singoli attributi

3.4 Strutture nidificate

E' veramente possibile lavorare solo con un insieme di tabelle per rappresentare tutti i domini che ci servono? Ci sono degli strumenti dal punto di vista della potenza espressiva dimostrati piu' forti di quello che puo' fare il modello a tabelle (ci sono strumenti che si rifanno alla logica matematica che sono piu' espressivi) il modello a tabelle pero' e' comunque molto piu' espressivo di quello che ci si puo' aspettare.

Da Filippo Via Roma 2, Roma		
Ricevuta Fiscale 1235 del 12/10/2002		
3	Coperti	3,00
2	Antipasti	6,20
3	Primi	12,00
2	Bistecche	18,00
	Totale	39,20

Da Filippo Via Roma 2, Roma		
Ricevuta Fiscale 1240 del 13/10/2002		
2	Coperti	2,00
2	Antipasti	7,00
2	Primi	8,00
2	Orate	20,00
2	Caffè	2,00
	Totale	39,00

Figure 3.5: Esempio di strutture nidificate

Nell esempio in figura 3.5 notiamo che ci sono informazioni che rimangono sempre statiche, ed altre che sono piu' dinamiche di altre, non e' chiarissimo come rappresentare queste informazioni in tabelle mantenendo queste caratteristiche. Quello che viene ordinato dal cliente non e' facilmente dimensionabile

in termini di righe di cui ho bisogno, perche' logicamente quello che ordinano i vari clienti cambia. Una delle soluzioni per rappresentare questo dominio di informazioni potrebbe essere mettendo numero di ricevuta, la data e poi nidificato dentro c'e' una struttura secondaria (che e' quello che hanno ordinato con quella ricevuta).

Questo pero' non si puo' fare, quindi faccio due tabelle invece di una e le incrocio. In una tabella mettiamo la parte statica e nell'altra la parte dinamica che cambia con un solo riferimento.

Ricevute

Numero	Data	Totale
1235	12/10/2002	39,20
1240	13/10/2002	39,00

Dettaglio

Numero	Qtà	Descrizione	Importo
1235	3	Coperti	3,00
1235	2	Antipasti	6,20
1235	3	Primi	12,00
1235	2	Bistecche	18,00
1240	2	Coperti	2,00
...	...	...	...

Figure 3.6: Soluzione sbagliata per strutture nidificate

C'e' un problema pero', perche' non potremmo inserire altre tuple identiche a quelle gia' presenti nelle tabelle e nel caso in cui un cliente voglia ordinare due volte le stesse cose il sistema non gli permetterebbe di farlo. Quindi sono possibili soluzioni diverse, potremmo ad esempio inserire noi un ordinamento:

Ricevute

Numero	Data	Totale
1235	12/10/2002	39,20
1240	13/10/2002	39,00

Dettaglio

Numero	Riga	Qtà	Descrizione	Importo
1235	1	3	Coperti	3,00
1235	2	2	Antipasti	6,20
1235	3	3	Primi	12,00
1235	4	2	Bistecche	18,00
1240	1	2	Coperti	2,00
...	...	...	...	...

Figure 3.7: Ordinamento nella struttura nidificata

3.5 Informazione incompleta

Un altro problema che ci capitera' e' legato al fatto che nella realta' molte informazioni non ci sono, quindi l'informazione nel mondo reale e' incompleta di natura. il tema che si pone quindi e': come facciamo a rappresentare informazione ignota nella base dati?

Se lo risolvessimo noi questo problema e non tramite DBMS probabilmente lo risolveremmo usando un valore speciale, in realta' questa non e' una buona soluzione. Si usa un simbolo speciale che tutti sanno interpretare dando il corretto significato (ovvero che non c'e' l'informazione): il valore nullo (che ricordiamoci non fa parte del dominio).

3.5.1 Tipi di valore nullo

Quando manca un valore, guardiamo la tabella e vediamo NULL. L'interpretazione di questo NULL rimane ambigua, non c'e' una specificazione, vediamo quindi casi specifici per cui potrebbe essere inserito un valore NULL:

- Il valore e' sconosciuto (sappiamo che esiste ma non lo conosciamo, ad esempio la residenza)
- valore inesistente (sappiamo che per quel record non esiste)
- valore senza informazione (non sappiamo se esiste e non lo conosciamo)

Questi NULL ci daranno gran fastidio, perche' negli argomenti successivi del corso dovremo sempre considerare se ci sono i valori NULL come funziona la determinata cosa che stiamo studiando, sono dei casi particolari con i quali dobbiamo convivere.

studenti

Matricola	Cognome	Nome	Data di nascita
6554	Rossi	Mario	05/12/1978
9283	Verdi	Luisa	12/11/1979
NULL	Rossi	Maria	01/02/1978

esami

Studente	Voto	Corso
NULL	30	*NULL*
NULL	24	02
9283	28	01

corsi

Codice	Titolo	Docente
01	Analisi	Mario
02	*NULL*	*NULL*
04	Chimica	Verdi

Figure 3.8: Esempio di troppi valori nulli in una base di dati

Un DBMS ben progettato non dovrebbe permettere di inserire questi valori NULL nell'immagine 3.8 (ad esempio un professore non puo' inserire un voto senza specificare di che corso e di che studente).

3.6 Vincolo di integrita'

(In questa lezione dobbiamo imparare il concetto di Super Chiave e chiave di una relazione ed il concetto di vincolo di inegrita' referenziale)

Esistono delle istanze di basi di dati che, pur sintatticamente corrette, non rappresentano informazioni possibil per l'applicazione di interesse.

Esami

Studente	Voto	Lode	Corso
276545	32		01
276545	30	e lode	02
787643	27	e lode	03
739430	24		04

Studenti

Matricola	Cognome	Nome
276545	Rossi	Mario
787643	Neri	Piero
787643	Bianchi	Luca

Figure 3.9: Esempio di istanza scorretta

Nell'esempio in figura 3.9 vediamo come nonostante le tabelle siano sintatticamente corrette (sono delle relazioni), dalla nostra esperienza sappiamo che ci sono vari errori:

- Non possono esistere due studenti con lo stesso numero di matricola
- Non esiste il 27 e lode
- Ci sono due rappresentazioni del 30 e lode
- Non esiste il numero di matricola nella tabella Studenti di uno studente registrato nella tabella Esami

Dobbiamo quindi evitare queste problematiche.

Perche' mettere dei vincoli in una base di dati? Ci permetteranno di fare l'incrocio dei dati tra le tabelle, il vincolo di super chiave e chiave di una relazione ci permette di fare come si deve l'incrocio tra i dati, e' quindi fondamentale nel modello relazionale.

Il vincolo di integrita' e' una proprieta' che deve essere soddisfatta dalle istanze che rappresentano informazioni correte per l'applicazione. Un vincolo e' una funzione booleana, che se dev'essere soddisfatta deve valere vero per tutte le istanze.

Riassumendo perche' dovremmo usare dei vincoli:

- Ci permettono di fare una descrizione piu' accurata della realta'
- Contribuiscono alla "qualita' dei dati"
- Sono utili nella progettazione
- Sono usati dai DBMS nella esecuzione delle interrogazioni

Alcuni tipi di vincoli non sono "supportati" dal DBMS, possiamo quindi specificare vincoli di tali tipi nella nostra base di dati e il DBMS ne impedisce la violazione. Per i vincoli che non riusciamo ad esprimere nel linguaggio SQL li teniamo in un report e la responsabilita' della verifica e' dell'utente o del programmatore, che gestiranno loro queste informazioni nella logica applicativa.

3.6.1 Tipi di vincoli

- **Intrarelazionali**: noi guardiamo una singola tabella e diciamo che ci sono dei vincoli tra i valori delle varie colonne.
  - Vincoli su valori (o di dominio)
  - Vincoli di ennupla
- **Interrelazionali**: c'e' una relazione tra varie tabelle

Nell'esempio in figura 3.9, abbiamo delle tabelle senza vincoli e a causa di questa cosa sorgono vari problemi. Ad esempio 32 non puo' esistere come voto, e lode solo sul 30, non e' possibile che ci siano due matricole uguali (questa cosa la sappiamo noi, ma se fosse generale dovrebbe esserci un'etichetta). Nonostante questi errori rimane una relazione, perche' le tuple sono tutte diverse e gli attributi sono omogenei. Un problema intrarelazoinale e' che non posso sapere chi ha preso 24, perche' lo studente con quella matricola non esiste, e' uno dei casi in cui vorremmo che il DBMS bloccasse l'inserimento dei dati.

3.6.2 Sintassi

Una possibile sintassi con cui possiamo specificare i vincoli che abbiamo visto prima: (e' un espressione booleana di atomi che confrontano valori di attributo o espressioni aritmetiche su di essi):

```
(Voto >= 18) AND (Voto <=30)

(Voto - 30) OR NOT (Lode = "e lode")
```

Implicazioni logiche

Se hai bisogno di specificare dei vincoli che sono delle implicazioni logiche (SE piove ALLORA prendo l'ombrello), posso trasformare questa implicazione in algebra booleana con not A or B (NOT piove OR prendo l'ombrello).

3.6.3 Vincoli di ennupla, altro esempio

Stipendi	Impiegato	Lordo	Ritenute	Netto
	Rossi	55.000	12.500	42.500
	Neri	45.000	10.000	35.000
	Bruni	47.000	11.000	36.000

Figure 3.10: Un'altro esempio di vincoli di ennupla

Per ogni impiegato vogliamo sapere quel'e' lo stipendio lordo, le ritenute ed il netto, vogliamo accedere velocemente ai dati, ovvero che sono esplicitamete calcolati. Sappiamo pero' che il netto e' il lordo meno le ritenute, quindi non dobbiamo permettere di inserire qualsiasi valore all'interno della tabella. Dobbiamo inserire i vincoli, che in questo caso e' netto = lordo - ritenute. Per evitare che si verifichino problemi come in figura 3.11

Stipendi	Impiegato	Lordo	Ritenute	Netto
	Rossi	55.000	12.500	42.500
	Neri	45.000	10.000	35.000
	Bruni	50.000	11.000	36.000

Figure 3.11: Violazione vincoli di ennupla

3.6.4 Identificazione delle ennuple

Matricola	Cognome	Nome	Corso	Nascita
27655	Rossi	Mario	Ing Inf	5/12/78
78763	Rossi	Mario	Ing Inf	3/11/76
65432	Neri	Piero	Ing Mecc	10/7/79
87654	Neri	Mario	Ing Inf	3/11/76
67653	Rossi	Piero	Ing Mecc	5/12/78

Figure 3.12: Identificazione delle ennuple

Il vincolo di chiave serve anche ad identificare l'intera tupla. Qui non ci sono due ennuple con lo stesso valore sull'attributo matricola, quindi per identificare una specifica tupla non ho bisogno di mettere tutti i dati dello studente, mi basta mettere la matricola e sono sicuro che ho identificato una tupla univocamente.

C'e' un altro modo per identificare lo studente se non mi ricordo la matricola? (esiste una sola chiave nella relazione?) osservando questa tabella (che ricordiamo essere l'istanza di una relazione) potrei provare ad identificare un'altro

modo, noto che non esistono due ennuple uguali di tutti e tre gli attributi (Cognome, Nome e Data di Nascita). Sara' una scelta nostra dichiarare cos'e' una chiave, che come abbiamo intuito adesso puo' non essere una sola colonna, ma puo' essere un insieme di attributi.

3.6.5 Chiave

E' un insieme e di attributi che identificano le tuple (la distinguono da ogni altra). Prima abbiamo menzionato la super chiave, ed e' importante, e' quella che ci permette di distinguere tra le tuple, se pero' prendo tutto lo schema e' una super chiave? si, e quindi se per essere relazione non devono esserci due tuple uguali dev'essere soddisfatta la relazione di super chiave per tutti gli attributi dello schema. Allora automaticamente capiamo che esiste sempre una superchiave per la relazione (ed e' sempre l'insieme di tutti gli argomenti dello schema). La chiave e' una superchiave minimale, dice innanzitutto che dev'essere una superchiave e poi dev'essere minimale. Ci possono essere piu' chiavi per un'istanza (nel nostro caso: matricola e nome cognome data), e noi dovremo sceglierne una primaria.

Formalmente:

- Un insieme K di attributi (cognome, nome, nascita) e' superchiave per r se r non contiene due ennuple distinte t_1 e t_2 con $t_1[K] = t_2[K]$. Se io prendo qualsiasi coppia di tuple se prendo gli attributi devono essere diversi. La relazione non contiene due tuple diverse che hanno gli stessi valori per la relazione e' una super chiave.

- K e' una chiave per r (una relazione) se e' una superchiave minimale per r (cioe' non contiene un'altra superchiave).

C'e' sempre una chiave? nel caso piu' sfortunato non posso togliere nessun attributo alla superchiave composta da tutte le etichette dello schema, ma non e' che non c'e' una chiave, la chiave minimale e' la superchiave originale. Quindi c'e' sempre una chiave in una relazione.

3.6.6 Vincoli, schemi e istanze

I vincoli corrispondono a proprieta' del mondo reale modellato dalla base di dati, interessano a livello di schema (con riferimento cioe' a tutte le istanze). Ad uno schema associamo un insieme di vincoli e consideriamo corrette (valide, ammissibili) le istanze che soddisfano tutti i vincoli. Un istanza puo' soddisfare altri vincoli ("per caso")

3.6.7 Esistenza delle chiavi

Una relazione non puo' contentere ennuple distinte ma uguali, ogni relazione ha come superchiave l'insieme degli attributi su cui e' definita e quindi ha (almeno) una chiave.

3.6.8 Importanza delle chiavi

L'esistenza delle chiavi garantisce l'accessibilita' a ciascun dato della base di dati. Le chiavi permettono di correlare i dati in relazioni diverse, il modello relazionale e' basato su valori e' quindi quando devo confrontare tuple in tabelle diverse il fatto di sapere che una tupla e' unicamente identificata dal valore di un attributo (o un gruppo) e' fondamentale per poter fare questo passaggio. La tabella 3.9 con il problema che c'era un voto con due studenti con la stessa matricola associati, ovvero due valori uguali per la chiave della relazione, se io questa cosa la vieto nessuno potra' inserire nel database due matricole per lo stesso voto.

3.6.9 Chiavi e valori nulli

In presenza di valori nulli, i valori della chiave non permettono:

- Di identificare le ennuple
- Di realizzare facilmente i riferimenti da altre azioni

Se uno studente ha un NULL come matricola come faccio a capire che voti ha preso? non posso.

Matricola	Cognome	Nome	Corso	Nascita
NULL	NULL	Mario	Ing Inf	5/12/78
78763	Rossi	Mario	Ing Civile	3/11/76
65432	Neri	Piero	Ing Mecc	10/7/79
87654	Neri	Mario	Ing Inf	NULL
NULL	Neri	Mario	NULL	5/12/78

Figure 3.13: Esempio di chiavi con valori NULL

Dato che abbiamo fatto l'ipotesi che ci siano due chiavi (matricola, cognome nome nascita) non possono esserci i NULL nei valori delle chiavi. Ci sono problemi perche' se ad esempio non riesco a distinguere il primo e l'ultimo record. Devono quindi limitare la presenza di valori nulli nelle chiavi.

3.6.10 Chiave primaria

Dobbiamo analizzare bene il dominio e cercare di capire per ogni relazione quali sono le chiavi e poi decidiamo qual'e' la chiave primaria. Dobbiamo quindi vietare i valori nulli per questa chiave. La notazione sara' la sottolineatura. (la chiave primaria non dev'essere per forza fatta da un solo attributo).

Matricola	Cognome	Nome	Corso	Nascita
86765	NULL	Mario	Ing Inf	5/12/78
78763	Rossi	Mario	Ing Civile	3/11/76
65432	Neri	Piero	Ing Mecc	10/7/79
87654	Neri	Mario	Ing Inf	NULL
43289	Neri	Mario	NULL	5/12/78

Figure 3.14: Chiave primaria

3.7 Integrita' referenziale

Sono vincoli interrelazionali. Informazioni in relazioni diverse sono correlati attraverso valori comuni, in particolare prendiamo i valori delle chiavi primarie. Le correlazioni debbono essere coerenti.

Infrazioni

Codice	Data	Vigile	Prov	Numero
34321	1/2/95	3987	MI	39548K
53524	4/3/95	3295	TO	E39548
64521	5/4/96	3295	PR	839548
73321	5/2/98	9345	PR	839548

Vigili

Matricola	Cognome	Nome
3987	Rossi	Luca
3295	Neri	Piero
9345	Neri	Mario
7543	Mori	Gino

Figure 3.15: Esempio di vincoli di integrita'

Abbiamo un contesto diverso, la citta' e la polizia municipale, c'e' un database che memorizza i vari agenti della municipale ed anche qui si e' deciso di usare la matricola come chiave primaria. Anche per quanto riguarda la tabella infrazioni si e' deciso di assegnare un codice all'infrazione che e' univoco ed e' usato come chiave primaria. Come integrita' referenziale ci piacerebbe avere correlazione tra vigile e matricola, quindi se progettiamo questo database dovremmo identificare un vincolo che dice che non e' possibile inserire una multa con un vigile che non

esiste. Sara' opportuno quindi stabilire un vincolo tra la prima e la seconda tabella, che dira' che i valori dentro la colonna vigile della prima tabella devono esistere nella tabella dei vigili, in particolare esistere nella chiave (matricola) della tabella dei vigili.

Incidenti

Codice	Data	ProvA	NumeroA	ProvB	NumeroB
34321	1/2/95	TO	E39548	MI	39548K
64521	5/4/96	PR	839548	TO	E39548

Auto

Prov	Numero	Cognome	Nome
MI	39548K	Rossi	Mario
TO	E39548	Rossi	Mario
PR	839548	Neri	Luca

Figure 3.16: Esempio di vincoli con piu' attributi

Qui si complica un pochino il vincolo referenziale, perche' in questo caso la chiave e' formata da due colonne, che in questo caso sono la colonna provincia e la colonna numero. Qui quello che dobbiamo dire e' che quando il vigile scrive il numero di targa il sistema, quando inserisce questo record, verifica che quella targa sia la targa di una macchina almeno esistente, quindi dobbiamo specificare questo come vincolo (attenzione la coppia degli attributi, anche la provincia!).

L'errore che facciamo solitamente negli esercizi e' che non c'e' il vincolo solo tra numero e numero, ma tra prov numero e prov numero, perche' la stessa targa puo' esserci in province diverse. La coppia deve apparire sulla stessa riga.

3.7.1 Vincolo identita' referenziale: definizione formale

Lo troveremo anche chiamato come chiave esterna (foreign key). Un vincolo di integrita' referenziale fra gli attributi X di una relazione R_1 e un'altra relazione R_2 impone ai valori su X di R_1 di comparire come valori della chiave primaria di $R-2$. Attenzione che X non e' un attributo, ma un insieme di attributi. Lo si fa sulla chiave primaria perche' mi garantisce di identificare un record preciso.

Quindi rispetto a quello che abbiamo visto prima quando dobbiamo progettare uno schema logico dobbiamo dire lo schema di ciascuna relazione (definire le strutture delle tabelle) ed i vincoli (chiavi e chiavi primarie e vincoli di integrita' referenziali).

Infrazioni

Codice	Data	Vigile	Prov	Numero
34321	1/2/95	3987	MI	39548K
53524	4/3/95	3295	TO	E39548
64521	5/4/96	3295	PR	839548
73321	5/2/98	9345	PR	839548

Auto

Prov	Numero	Cognome	Nome
MI	E39548	Rossi	Mario
TO	F34268	Rossi	Mario
PR	839548	Neri	Luca

Figure 3.17: Esempio di violazione di vincolo di integrita' referenziale

3.7.2 Con i valori nulli

Qui possono succedere delle cose molto importanti che non ci aspettiamo come progettisti di un database. Esempio: hanno deciso di fare un database per memorizzare i seminari e le lezioni che venivano fatti in Universita' ed in quale aula veniva fatto. C'era un vincolo di integrita' referenziale (il professore veniva messo in un aula esistente). C'e' stato il problema che ad un certo punto un'aula non esiste piu', quindi l'idea e' aggiornare il database eliminando l'aula ma non lo faceva fare ed il motivo e' che c'era un vincolo di integrita' referenziale, quindi tutti i seminari fatti in quell'aula finirebbero in un'aula che non esiste piu'. Quindi il comportamento standard di un DBMS e' di impedirci di fare operazioni di cancellazione sulla tabella.

Impiegati

Matricola	Cognome	Progetto
34321	Rossi	IDEA
53524	Neri	XYZ
64521	Verdi	NULL
73032	Bianchi	IDEA

Progetti

Codice	Inizio	Durata	Costo
IDEA	01/2000	36	200
XYZ	07/2001	24	120
BOH	09/2001	24	150

Figure 3.18: Esempio di vincolo di integrita' relazionale con valore NULL

Il DBMS se non specifichiamo nulla ci impedisce di cancellare perche' c'e'

qualcosa da qualche parte, potremmo fare operazioni aggiuntive, ad esempio mettere un NULL al posto dell'aula che e' stata cancellata (nella tabella dove tengo traccia dei seminari, non quella in cui il nome dell'aula e' una chiave). Ci sono azioni compensative, che sono cancellazione a cascata, vuol dire che nel nostro esempio spariscono tutte le tuple in cui compariva il nome dell'aula, quindi sparisce anche il seminario e se quella tabella aveva il vincolo in un'altra tabella ci sara' una cascata di cancellazioni.

Modi di gestirlo:

- Impedire la cancellazione
- Inserire valori nulli
- Eliminazione a cascata

Chapter 4

Progettazione di basi di dati

4.1 Perche' preoccuparci

Dobbiamo creare delle relazioni (tabelle) che riescono a catturare un determinato dominio. Ma la tabella e' una struttura molto rigida, riuscire a capire qual'e' la milgiore scomposizione di un dominio in diverse tabelle non e' semplice. Come facciamo a passare da una descrizione informale all'organizzazione in tabelle di un database. Proviamo a modellare una applicazione definendo direttamente lo schema logico della base di dati:

- Da dove cominciamo?
- Richiamo di perderci subito nei dettagli
- Dobbiamo pensare subito a come correlare le varie tabelle (chiavi etc)
- Il modello relazionale e' rigido

4.2 Progettazione di basi di dati

La fase di progettazione di una base di dati e' parte del processo di sviluppo dei sistemi informativi, quindi va inquadrata nel ciclo di vita dei sistemi informativi:

- Studio di fattibilita': si puo' fare?
- Raccolta e analisi dei requisiti: fase in cui semplicemente avremo un testo in linguaggio naturale che ci spiega le caratteristiche del dominio ed in base al quale faremo la progettazione.
- Progettazione: separiamo i dati dalle applicazioni, perche' un db e' usato in tante applicazioni diverse, e tipicamente dura di piu' nel tempo rispetto alle applicazioni.
- Implementazione
- Validazione e collaudo
- Funzionamento

Notiamo che queste fasi si percorrono anche dal basso verso l'alto, perche' durante una fase e' possibile che si possano riconsiderare le scelte fatte nelle fasi precedenti. Noi ci focalizzeremo sulla fase di progettazione e di raccolta e analisi dei requisiti.

Dove si colloca rispetto alle cose gia' viste e quali sono gli output di ciascuna fase.

Il nostro obbiettivo e' arrivare allo schema logico, vediamo ora pero' come crearlo, partendo dai requisiti, l'output di questa progettazione concettuale sara' uno schema concettuale, che e' uno schema grafico.

La progettazione di un sistema informativo riguarda due aspetti:

- Progettazione dei dati

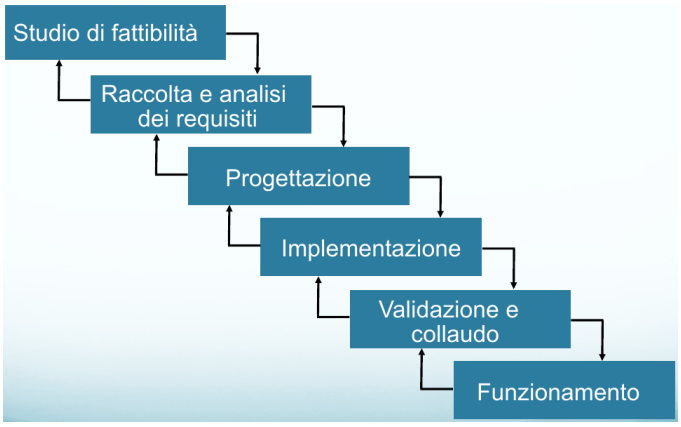


Figure 4.1: Ciclo di vita del software

- Progettazione delle applicazioni

Ma i dati hanno un ruolo centrale, sono piu' stabili, ovvero durano nel tempo, mentre le applicazioni potrebbero cambiare.

Per garantire prodotti di buona qualita' e' opportuno seguire una metodologia di progetto, con:

- Articolazione della attivita' in fasi
- Criteri di scelta
- Modelli di rappresentazione
- Generalita' e facilita' d'uso

Le fasi, del ciclo di vita del software, su cui ci concentreremo noi saranno la raccolta e l'analisi dei requisiti e la progettazione.

4.3 Due tipi di modelli

Come ci sono diversi modelli logici (ad esempio il relazionale) ci sono anche diversi modelli concettuali, quello piu' noto e' il modello Entity-Relationship (ER). Pure essendo molto vecchio e' ancora utilissimo ed utilizzatissimo. Il modello concettuale ci permette di descrivere i concetti del mondo reale e sono utilizzati nelle fasi preliminari di progettazione.

4.4 Modelli concettuali

Servono per ragionare sulla realta' di interesse, indipendentemente dagli aspetti realizzativi. Permettono di rappresentare le classi di oggetti di interesse e le

Figure 4.2: Metodologia di progettazione

loro correlazioni. Prevedono efficaci rappresentazioni grafiche (utili anche per documentazione e comunicazione).

4.5 Modello Entity-Relationship (Entita'-Associazione (ER))

E' il piu' diffuso modello concettuale, di cui ne esistono molte versioni.

4.6 Costrutti del modello E-R

- Entita'
- Associazioni (relationships)
- Attributo
- Identificatore
- Generalizzazione

4.6.1 Entita'

E' una classi di oggetti (fatti, persone, cose) della realta' di interesse con proprieta' comuni e con esistenza "autonoma".

4.6.2 Associazioni

Legame logico fra due o piu' entita', rilevante nell'applicazione di interesse. Esempi:

- Residenza(ra persone e citta')
- Esame (fra studente e corso)

4.6.3 Graficamente

I rettangoli sono usati per rappresentare le entita' ed rombi per le associazioni. Nell'esempio in figura il nostro dominio parlera' di studenti, di corsi e li mette in relazione tramite gli esami.

Figure 4.3: Esempio di schema ER

4.6.4 Entita': schema e istanza

- L'entita' e' una classe di oggetti, persone, ... omogenei.
- Le occorrenze (o istanze) di entita' sono elementi della classe(l'oggetto, la persona ... non i dati).

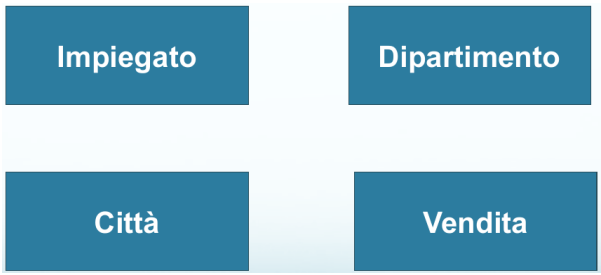


Figure 4.4: Rappresentazione grafica di entita'

Nello schema concettuale rappresentiamo le entita', non le singoli istanze.

Commenti

Ogni entita' ha un nome che la identifica univocamente nello schema, usare nomi espressivi e singolari.

4.7 Associazione

E' un legame logico fra due o piu' entita', rilevante nell'applicazione di interesse. Esempi:

- Residenza (fra persone e citta')
- Esame (fra studente e corso)

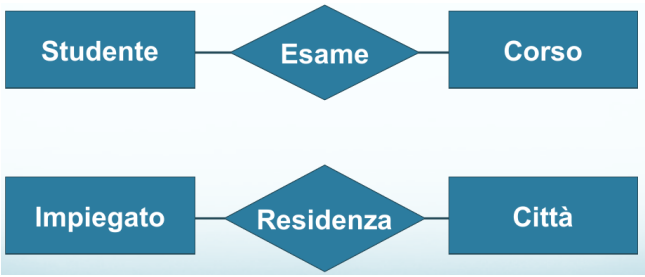


Figure 4.5: Rappresentazione grafica di associazioni

Nomi delle associazioni

Ogni associazione ha un nome che la identifica univocamente nello schema. Cercare comunque di usare nomi espressivi e se possibile non usare un verbo ma meglio usare il sostantivo perche' le associazioni non hanno un verso.

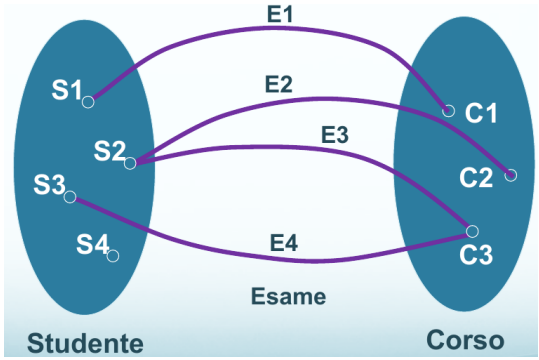


Figure 4.6: Esempio di occorrenze

4.8 Occorrenze di associazioni

Se ci chiedesse cos'e' un'occorrenza di un'associazione? Siccome un'associazione e' un legame logico deve dire cosa sta legando. Nell'esempio in figura 4.6 tutte le coppie s_n, c_n sono delle occorrenze di un'associazione.

Una occorrenza di una associazione binaria e' una coppia di occorrenze di entita', una per ciascuna entita' coinvolta. Una occorrenza di una associazione n-aria e' una n-upla di occorrenze di entita', una per ciascuna entita' coinvolta. Nell'ambito di un'associazione non ci possono essere occorrenze ripetute, significa che io non voglio (nel nostro esempio) studente, corso con l'esame due volte.

4.9 Promuovere l'associazione

Facciamo diventare esame un'entita' e non piu' una associazione. Ovviamente dopo aver promosso un'associazione dobbiamo creare dei legami logici tra la nuova entita' e le entita' che legava prima l'associazione. Puo' anche chiamarsi entita' associativa.

Figure 4.7: Promuovere un'associazione

4.9.1 Due associazioni sulle stesse entita'

E' possibile che le stesse entita' abbiano due associaizoni diverse.

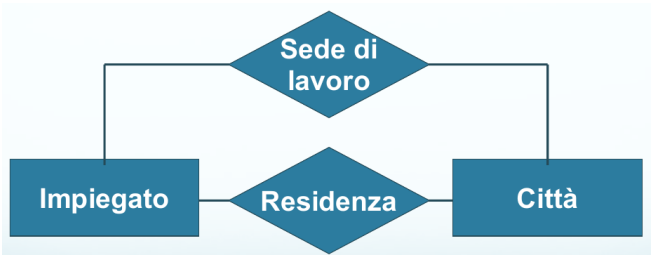


Figure 4.8: Due associazioni sulle stesse entita'

4.9.2 Associazione n-aria

Mettono in relazione piu' di due entita'.

4.10 Associazione ricorsiva

E' possibile avere una relazione che coinvolge due volte la stessa entita'? Ad esempio il legamene della conoscenza o l'amicizia di Facebook.

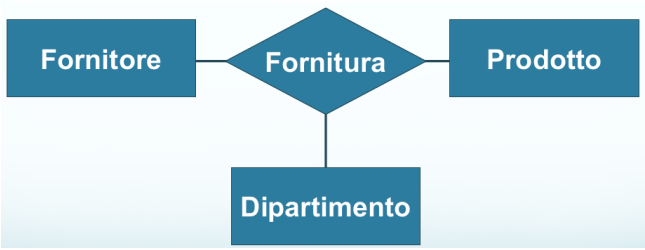


Figure 4.9: Associazione n-aria

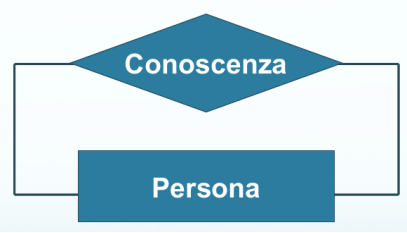


Figure 4.10: Associazione ricorsiva

Associazione ricorsiva con ruoli

Io voglio rappresentare nel database chi succede a chi come ministro delle finanze, allora non hanno lo stesso ruolo il successore e predecessore. Di conseguenza ho un'associazione ricorsiva con dei ruoli

4.10.1 Associazione ternaria ricorsiva

Nell'esempio vogliamo rappresentare in una classifica di tennisti chi e' il migliore, pero' in base alla superficie.

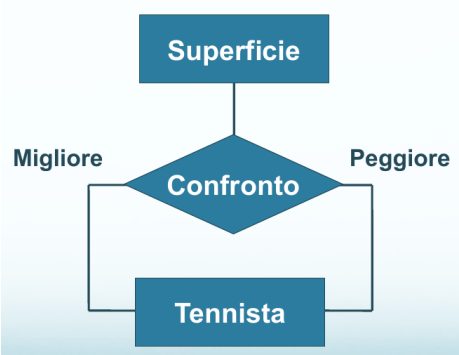


Figure 4.11: Associazione ternaria ricorsiva

4.11 Attributo

E' una proprieta' elementare di un'entita' o di una associazione, di interesse ai fini dell'applicazione. Associa ad ogni occorrenza di entita' o associazione un valore appartenente ad un insieme detto dominio dell'attributo.

4.11.1 Rappresentazione grafica

Matricola associa ad ogni occorrenza un valore di un dominio di numeri. Per l'esame (che e' un associazione) un attributo e' la data o il voto. Il modo con cui lo rappresento e' un pallino che e' collegato all'entita' o all'associane con una linea.

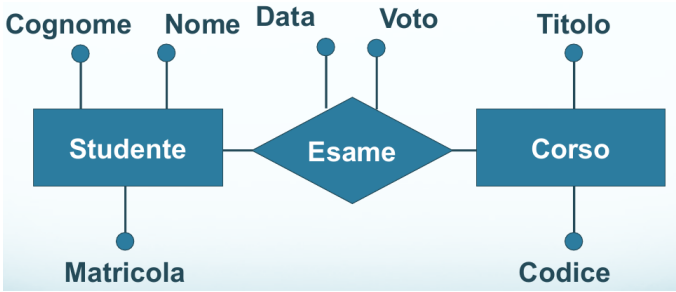


Figure 4.12: Rappresentazione grafica degli attributi

4.11.2 Attributo composti

E' vero che un'attributo e' una proprieta' elementare, ma possiamo anche raggruppare attributi in uno composto, come nel caso di un indirizzo abbiamo la via, il numero ed il cap.

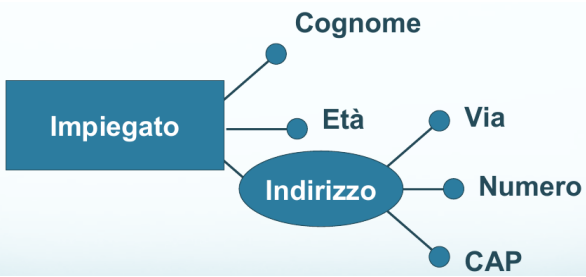


Figure 4.13: Rappresentazione grafica degli attributi composti

4.12 Altri costrutti del modello E-R

Cardinalita':

- Di Associazione
- Di attributo

Identificatore:

- Interno
- Esterno

Generalizzazione

4.13 Cardinalita' di associazione

Coppia di valori che associamo ad ogni entita' che partecipa ad un associazione. Ciascuno di questi valori specifica il numero minimo e massimo di occorrenze dell'associazione cui ciascuna occorrenza di una entita' puo' partecipare.

Figure 4.14: Esempio di cardinalita'

Con un'associazione binaria abbiamo 2 coppie (2 cardinalita') ciascuna vicino alle entita' interessate. Qual'e' la domanda da fare/farci per mettere questi numerini? In questo caso riguardano il ruolo degli impiegati: per ogni impiegato quante volte puo' partecipare all'associazione (quanti incarichi puo' avere minimo un impiegato?) ed e' il primo numero della cardinalita', mentre al massimo un impiegato quanti incarichi puo' avere? ed e' il secondo numero della cardinalita'. A destra invece per ogni incarico che ho in azienda quanti impiegati possono esservi assegnati?

Negli schemi veri e propri non si usano i numeri precisi, ma solamente 3 simboli. Come minimo:

- 0: ci dice che la partecipazione e' opzionale, e' possibile che un elemento dell'entita' non partecipi a quella relazione
- 1: almeno un elemento di quella entita' partecipa a quella relazione (partecipazione obbligatoria)

Per il massimo:

- 1: per ogni elemento c'e' una sola associazione
- N: se non ho limiti di associazioni

4.14 Tipi di associazioni binarie

Con riferimento alle cardinalita' massime, abbiamo associazioni binarie:

- Uno a uno
- Uno a molti
- Molti a molti

4.14.1 Associazioni "molti a molti"

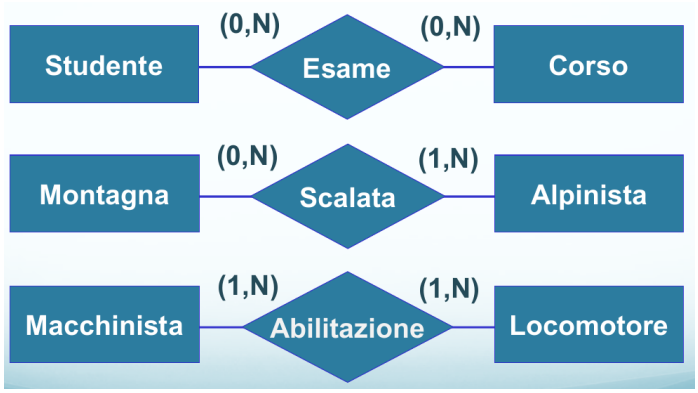


Figure 4.15: Associazioni molti a molti

4.14.2 Associazioni "uno a molti"

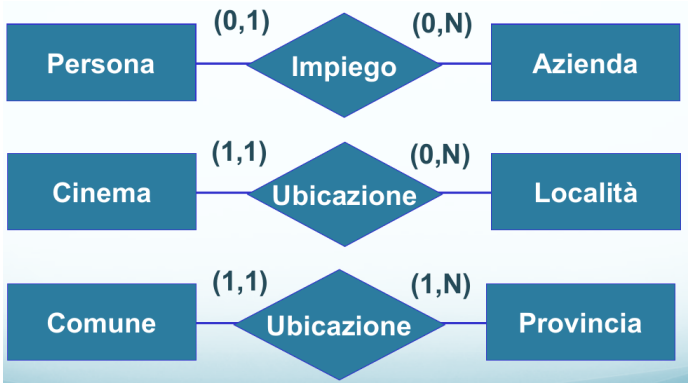


Figure 4.16: Associazioni uno a molti

Attenzione al verso nelle associazioni uno a molti, le associazioni obbligatorie-obbligatorie (1-1) sono molto rare.

4.14.3 Associazioni uno a uno

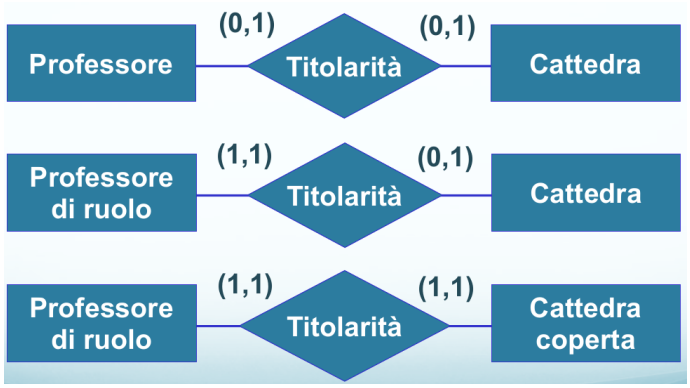


Figure 4.17: Associazioni uno a uno

4.14.4 Cardinalita' di attributi

E' possibile associare delle cardinalita' anche agli attributi con due scopi:

- Indicare opzionalita' ("informazione incompleta", puo' essere che ad esempio una persona non abbia un secondo nome). La domanda da fare e': per ogni elemento dell'entita' questo attributo deve avere per forza un valore oppure no?
- Indicare attributi multivalore

Rappresentazione grafica

La cardinalita' 0,N si chiama attributo multivalore da non confondere con attributo multicomposito (quello con dei sotto attributi), in questo caso potrei avere N numeri di cellulare. Se non scrivo niente non e' che mi sono dimenticato di mettere la cardinalita', ma e' il comportamento standard di un attributi, deve esiste per forza e dev'essere un solo valore (1, 1).

4.15 Identificatore di una entita'

Ho qualcosa di simile ai vincoli che mi permettono di distinguere le occorrenze di un'enetita' (voglio distinguere gli studenti di una classe). Per ogni entita'

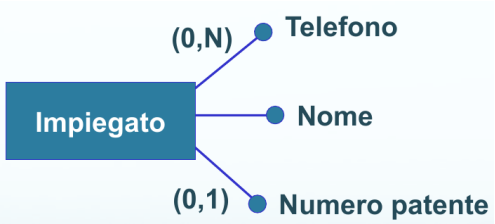


Figure 4.18: Rappresentazione grafica cardinalita' degli attributi

dev'esserci un identificatore di una entita', come faccio a distinguere gli elementi di una classe, e' quindi uno strumento per l'identificazione univoca delle occorrenze di un'entita'.

Costituito da:

- Attributi dell'entita', identificatore interno (la matricola di uno studente)
- (attributi +) entita' esterne attraverso associazione: identificazione interno

4.15.1 Identificatori interni

E' distinto come un cerchio riempito di nero, rispetto agli altri attributi. Per una persona invece posso dire che la identifico con nome, cognome e data di nascita.

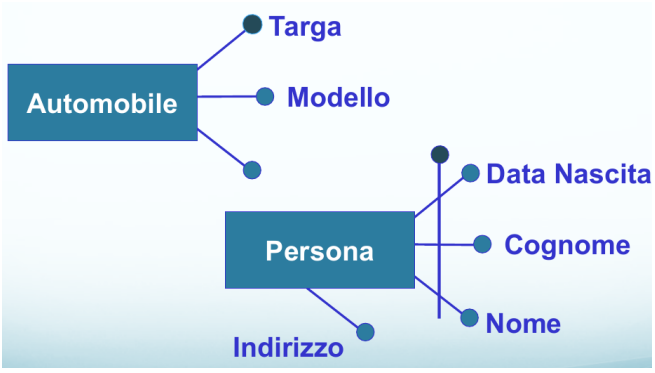


Figure 4.19: Identificatori interni

4.15.2 Identificate esterno

Potremmo modellare un dominio con attributi che da soli potrebbero non definire l'entita'. Nel nostro esempio il proprietario del dominio non e' un solo ateneo,

quindi per identificare uno studente mi serve anche la sua universita' di appartenenza (identificatore esterno). Tutti gli studenti si possono distinguere guardando l'attributo matricola ed anche la partecipazione ad un associazione (l'universita' alla quale sono iscritti). L'entita' studente si dice debole perche' per identificare devo usare anche le associazioni.

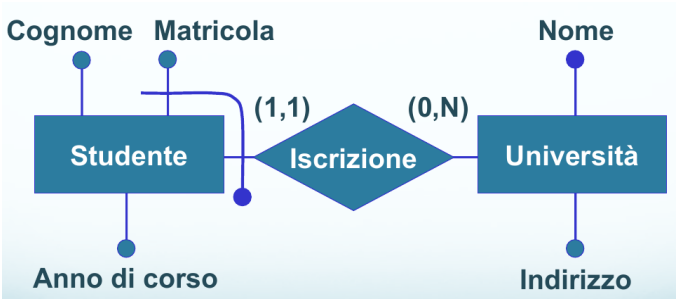


Figure 4.20: Identificatore esterno

4.15.3 Osservazioni

Ogni entita' deve possedere almeno un identificatore, ma puo' averne in generale piu' di uno (ci ricorda le chiavi, perche' possiamo averne piu' di una). Una identificazione esterna e' possibile solo attraverso un associazione a cui l'entita' da identificare partecipa con cardinalita' (1,1). Una identificazione esterna puo' coinvolgere un'entita' che e' a sua volta identificata esternamente, purche' non si generino cicli. Le entita' identificate esternamente vengono anche chiamate entita' deboli.

4.16 Generalizzazione

E' un costrutto che ci aiuta a rappresentare le cose in un modo piu' vicino a quello che siamo abituati a fare noi, che non e' possibile fare con le tabelle. Mette in relazione una o piu' identita' con un identita' particolare che le comprende come casi particolari. Dobbiamo solo immaginare il concetto di classe e sottoclasse.

Costrutto che mette in relazione uno o piu' entita' E1, E2, ... , En. Con una entita' E, che la comprende come casi particolari.

- E e' generalizzazione di E1, E2,..., En
- E1, E2,..., En sono specializzazione (o sottotipi, sottoclassi) di E
- Puo' essere vista come una associazione "speciale" spesso etichettata con "ISA" (E1 ISA E)

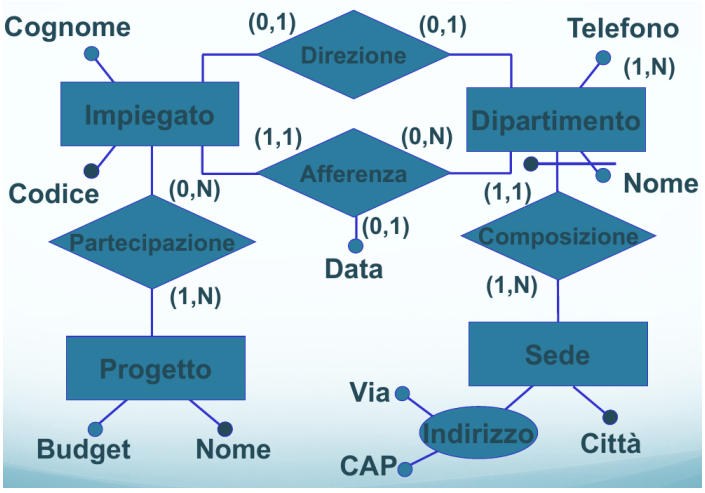


Figure 4.21: Esempio di schema E-R

Quello che succede in pratica e' che quando abbiamo delle entita' con attributi in comune possiamo fare ereditare questi attributi da un entita' piu' generale detta gentiore.

4.16.1 Rappresentazione grafica

E' come se fosse un'associazione particolare. C'e' un freccione che collega le due entita' figlie ed una entita' padre, che ci dice che gli studenti ed i lavoratori sono persone.

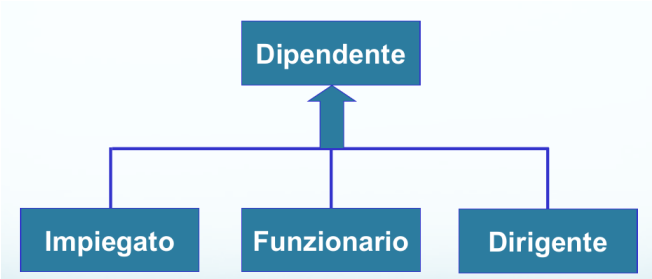


Figure 4.22: Generalizzazione

4.16.2 Proprieta' delle generalizzazioni

Se E (genitore) e' generalizzazione di E1, E2,..., En (figlie):

- ogni proprieta' di E e' significativa per E1, E2,..., En

- ogni occorrenza di E1, E2,..., En e' occorrenza anche di E

Occhio che quando abbiamo una generalizzazione non dobbiamo specificare gli attributi, ma li ereditano dall'entita' genitore.

4.16.3 Tipi generalizzazione

La prima cosa da capire e' la distinzione se la generalizzazione e' totale o parziale

- Totale: se ogni occorrenza dell'entita' genitore e' occorrenza di almeno una degli entita' figlie, altrimenti e' parziale (le entita' della classe genitore deve per forza essere anche entita' delle entita' figlie)

- Esclusiva: se ogni occorrenza dell'entita' genitore e' occorrenza di al piu' una delle entita' figlie, altrimenti e' sovrapposta

Quello che noi faremo e' indicare solo la differenza tra totale e parziale, perche' tutte possono diventare esclusive, basta mettere una classe comune (tipo studenteLavoratore)

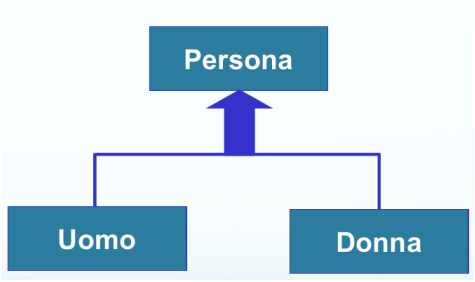


Figure 4.23: Notazione per le generalizzazioni totali

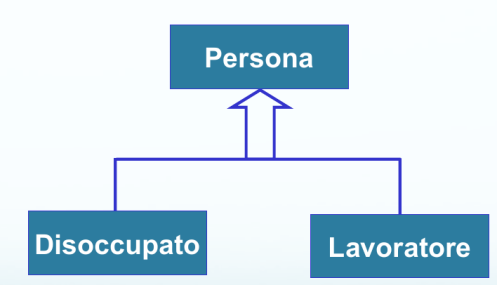


Figure 4.24: Notazione per le generalizzazioni parziali

4.16.4 Altre proprieta' generalizzazione

Possono esistere gerarchie a piu' livelli (gerarchie a piu' livello e multiple generalizzazioni allo stesso livello), un'entita' puo' essere inclusa in piu' gerarchie, come genitore e/o figlia. Se una generalizzazione ha solo un'entita' figlia si parla di sottoinsieme (es: un responsabile di progetto e' uno dei progettisti).

4.17 Documentazione associata agli schemi concettuali

Cosa consegniamo in questa fase di progetto (in azienda, non per l'esame).

- Lo schema E-R deve essere corredato di descrizioni dei concetti rappresentati e vincoli non esprimibili.
- Dizionario dei dati:
  - Entita'
  - Associazioni
- Vincoli non esprimibili

Entità	Descrizione	Attributi	Identificatore
Impiegato	Dipendente dell'azienda	Codice, Cognome, Stipendio	Codice
Progetto	Progetti aziendali	Nome, Budget	Nome
Dipartimento	Struttura aziendale	Nome, Telefono	Nome, Sede
Sede	Sede dell'azienda	Città, Indirizzo	Città

Figure 4.25: Dizionario dei dati: entita'

Relazioni	Descrizione	Componenti	Attributi
Direzione	Direzione di un dipartimento	Impiegato, Dipartimento	
Afferenza	Afferenza a un dipartimento	Impiegato, Dipartimento	Data
Partecipazione	Partecipazione a un progetto	Impiegato, Progetto	
Composizione	Composizione dell'azienda	Dipartimento, Sede	

Figure 4.26: Dizionario dei dati: associazioni

Vincoli di integrità sui dati
(1) Il direttore di un dipartimento deve afferire a tale dipartimento
(2) Un impiegato non deve avere uno stipendio maggiore del direttore del dipartimento al quale afferisce
(3) Un dipartimento con sede a Milano deve essere diretto da un impiegato con più di dieci anni di anzianità
(4) Un impiegato che non afferisce a nessun dipartimento non deve partecipare a nessun progetto

Figure 4.27: Vincoli non esprimibili

Chapter 5

Patterns progettazione E-R

Approfondiamo ancora la metodologia per progettare l'ER. Noi acquisiamo i requisiti con l'analisi dei documenti che ci vengono dati, facciamo la progettazione concettuale che ci da in output lo schema concettuale (anche la documentazione), poi abbiamo la progettazione logica ed otteniamo lo schema logico.

5.1 Requisiti: documentazione descrittiva

Come impostiamo il lavoro quando stiamo leggendo i documenti che il cliente ci ha dato:

- Scegliere il corretto livello di astrazione
- Standardizzare la strutture delle frasi
- Suddividere le frasi articolate
- Separare le frasi sui dati da quelle sulle funzioni

Nel testo saranno mischiate le caratteristiche dei dati e le caratteristiche sull'elaborazione dei dati.

5.1.1 Requisiti: organizzazione di termini e concetti

- Individuare omonimi e sinonimi e unificare i termini
- Rendere esplicito il riferimento fra termini
- Riorganizzare le frasi per concetti
- Costruire un glossario dei termini

5.2 Esempio: biblioteche

Si desidera automatizzare il sistema di prestiti di una biblioteca. Le specifiche del sistema acquisiti attraverso un'intervista con il bibliotecario le analizzeremo dopo. Dobbiamo quindi:

- Analizzare le specifiche
- Filtrare le ambiguita' presenti e raggrupparle in modo omogeneo
- Individuare i collegamenti esistenti tra i vari gruppi di specifiche

5.2.1 Specifiche

I lettori che frequentano la biblioteca hanno una tessera su cui è scritto il nome e l'indirizzo ed effettuano richieste di prestito per i libri che sono catalogati nella biblioteca. I libri hanno un titolo, una lista di autori e possono esistere in diverse copie. Tutti i libri contenuti nella biblioteca sono identificati da un codice. A seguito di una richiesta viene dapprima consultato l'archivio dei libri disponibili (cioè non in prestito). Se il libro è disponibile, si procede alla ricerca del volume negli scaffali; il testo viene poi classificato come in prestito. Acquisito il volume, viene consegnato all'utente, che procede alla consultazione. Terminata la consultazione, il libro viene restituito, reinserito in biblioteca e nuovamente classificato come disponibile. Per un prestito si tiene nota degli orari e delle date di acquisizione e di riconsegna.

5.2.2 Identificare e risolvere le ambiguita'

- Omonimi:
  - Libro
- Sinonimi:
  - Copia-Libro-Testo-Volume

5.2.3 Separare le frasi

- LETTORI: I lettori che frequentano la biblioteca hanno una tessera su cui è scritto il nome e l'indirizzo ed effettuano richieste di prestito per i libri che sono catalogati nella biblioteca.
- LIBRI: I libri hanno un titolo, una lista di autori e possono esistere in diverse copie.
- COPIA: Tutti i libri contenuti nella biblioteca sono identificati da un codice. A seguito di una richiesta viene dapprima consultato l'archivio dei libri disponibili (cioè non in prestito). Se il libro è disponibile, si procede alla ricerca del volume negli scaffali;
- PRESTITO: Il testo viene poi classificato come in prestito. Acquisito il volume, viene consegnato all'utente, che procede alla consultazione. Terminata la consultazione, il libro viene restituito, reinserito in biblioteca e nuovamente classificato come disponibile. Per un prestito si tiene nota degli orari e delle date di acquisizione e di riconsegna.

Termine	Descrizione	Sinonimo	Collegamenti
Lettore	Una persona che prende in prestito libri dalla biblioteca	Utente	Copia, Prestito
Libro	Tipo di libro presente in biblioteca. La biblioteca ha una o più copie di uno stesso libro.		Copia
Copia	Ogni copia di un libro presente in biblioteca. Può essere prestato a un lettore.	Libro, Testo, Volume	Libro, Lettore, Prestito
Prestito	Un prestito fatto a un lettore: ogni prestito si riferisce ad una copia di un libro.		Lettore, Copia

Figure 5.1: Definire un glossario

5.3 Requisiti discorsivi e schemi concettuali

Quale costrutto E-R va utilizzato per rappresentare un concetto presente nelle specifiche? Bisogna basarsi sulle definizioni dei costrutti del modelo E-R:

- Entita': se ha proprieta' significative e descrive oggetti con esistenza autonoma
- Attributo: se e' semplice e non ha proprieta'
- Associazione: se correla due o piu' concetti
- Generalizzazione: se e' un caso particolare di un altro

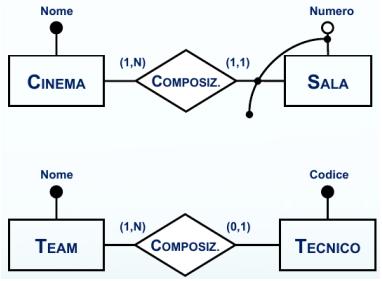
5.4 Design Pattern

Nella vita professionale di un professionista si ripetono delle situazioni, quindi con l'esperienza ci si puo' ricondurre ad esperienze passate riapplicando soluzioni di successo. Sono quindi soluzioni progettuali a problemi comuni, largamente usati nell'ingegneria del software.

5.4.1 Part-of

Tutte le volte che ci troviamo nella situazione in cui c'e' un concetto che dice parte ti qualcos'altro c'e' il design pattern part-of. Ad esempio una sala cinematografica e' parte di un cinema, ed un tecnico e' parte di un team. La soluzione e' l'associazione 1 a molti, dove l'uno sta dalla parte della parte (tecnico), mentre il molti e' dalla parte del concetto che costituisce le parti (team).

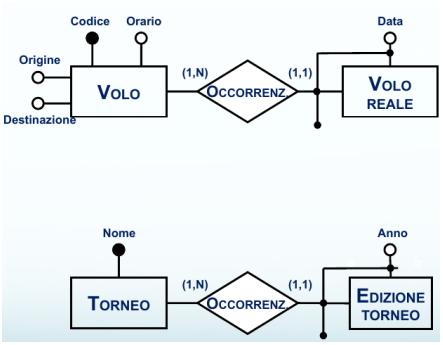
L'unica cosa che dobbiamo verificare e' se mettere 1 o 0 dalla parte minima del concetto parte. E' possibile che un tecnico non faccia parte di nessun team? in questo caso si quindi ha cardinalita' minima 0. Ma una sala cinematografica senza cinema non puo' esistere, quindi come cardinalita' minima metto 1. Nell'esempio del cinema possiamo usare un identificatore esterno per



identificare la sala con il nome del cinema perche' ho relazione 1/1, mentre nel caso del tecnico non possiamo usare un identificatore esterno, quindi dobbiamo inserire un codice per identificare il tecnico..

5.4.2 Istance-of

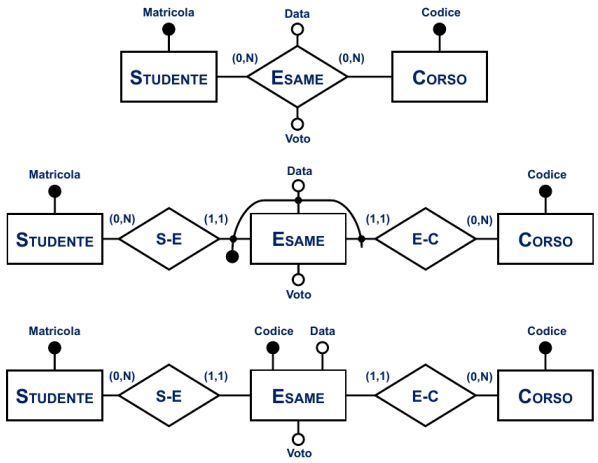
E' un pattern che utilizziamo quando dobbiamo specificare un caso specifico di un entita' piu' generale che potrebbe ripetersi nel tempo. Come un volo specifico o una particolare edizione di un torneo.



5.4.3 Reificazione di associazione binaria

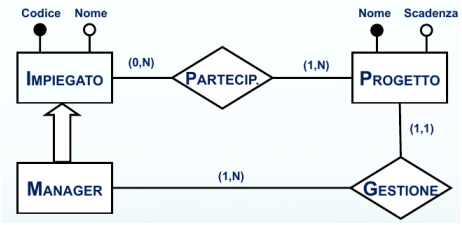
Noi abbiamo pensato di usare l'esame come associazione tra studenti ed un corso, poi ci ragioniamo un po' e capiamo che e' meglio usare un entita'. Quindi tutte le volte che vogliamo trasformare un associazione in entita', devo inventarmi due associazioni di fianco all associazione che e' diventata un'entita'. Ma anche alle cardinalita' non dobbiamo pensarci, e' tutto stabilito. Dalla parte in cui leghiamo il concetto appena creato con le associazioni nuove ci mettiamo 1,1. Dall'altra parte ci mettiamo quello che avevamo prima. Gli attributi rimangono nella nuova entita'. Notiamo che e' importante mettere un identificatore per la nuova entita' creata (ogni entita' deve avere un identificatore). In questo caso possiamo usare le relazioni nuove come identificatore esterno., oppure come

nell'esempio 3 utilizzare un codice per identificare un'esame. Notiamo che in questo caso non sarebbe possibile inserire due voti per lo stesso studente e corso.



5.4.4 Caso particolare

In questo caso abbiamo un entita' (manager) che e' un sottoinsieme particolare di impiegato, quindi lo esprimiamo come generalizzazione inserendo anche un'associazione (gestione). Guardando le cardinalita' notiamo che un manager puo' gestire 1 o piu' progetti.



5.4.5 Storicizzazione di concetto

In un caso reale dobbiamo chiederci se il cliente vuole tenere traccia del passato oppure gli interessano solo le informazioni attuali. Nell'esempio il cliente vorrebbe tenere traccia di quanti impieghi ha avuto una persona in una azienda:

Figure 5.2: Problema

Notiamo che in questa prima soluzione non e' possibile rappresentare piu' impieghi della stessa persona, in periodo diversi, presso la stessa azienda.

La prima soluzione che potrebbe venirci in mente e' quella di usare due associazioni, un IMPIEGO CORRENTE ed un IMPIEGO PASSATO. Ma anche in questo caso avremmo lo stesso problema di prima, in quanto non sarebbe possibile immagazzinare piu' impieghi passati nella stessa azienda (ricordiamo che non e' possibile avere piu' ennuple uguali).

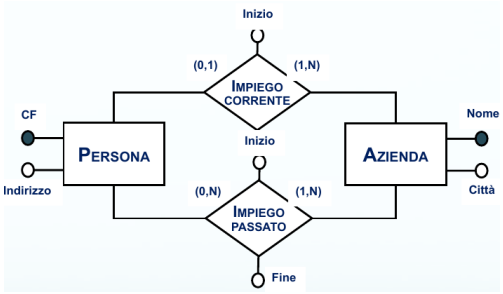


Figure 5.3: Soluzione sbagliata

Quindi una soluzione corretta sarebbe fare diventare un entita' l'associazione IMPIEGO usando le reificazione (vista in precedenza), utilizzando anche la generalizzazione creando due entita' figlie: IMPIEGO CORRENTE e IMPIEGO PASSATO. Per utilizzare questa soluzione dobbiamo imporre anche il vincolo che ogni occorrenza della relazione tra PERSONA e IMPIEGO CORRENTE dovra' comparire anche nella relazione tra PERSONA ed IMPIEGO. Guardiamo le cardinalita', notiamo che ci puo' essere al massimo un impiego corrente per ogni persona (1,1). Ora guardiamo l'identificazione di IMPIEGO (ricordando che una nuova entita' deve sempre essere identificata), viene identificato tramite un identificatore esterno che prendeL

- Il codice fiscale di una persona
- Il nome dell'azienda
- la data di inizio dell'impiego

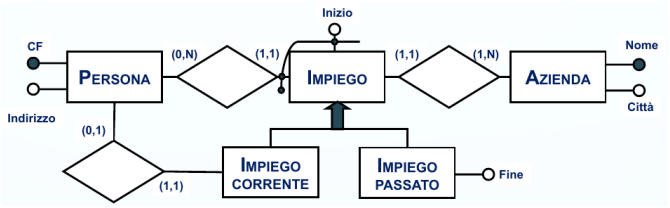


Figure 5.4: Soluzione 1

Un'alternativa potrebbe venirci in mente se pensiamo di togliere la relazione tra persona ed impiego, in quanto esistono solo impieghi correnti e passati.

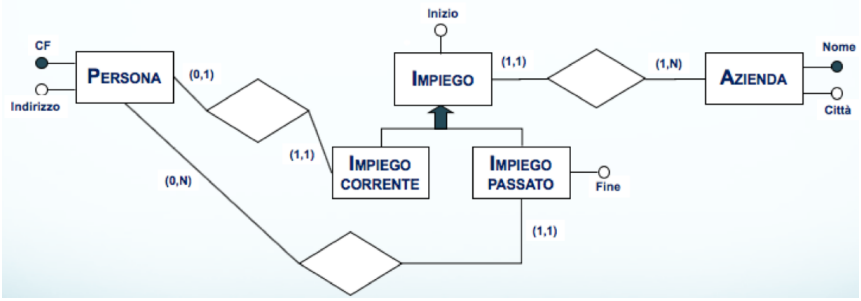


Figure 5.5: Soluzione 2

Una terza alternativa potrebbe esistere se permettessimo di inserire numerosi impieghi correnti, quindi non distinguendo piu' tra impiego presente e passato, in quanto a livello applicativo sarebbe molto facile distinguere tra impiego corrente e passato (controllando la data di fine). Facciamo questo semplicemente reificando l'associazione IMPIEGO senza pero' inserire la generalizzazione totale.

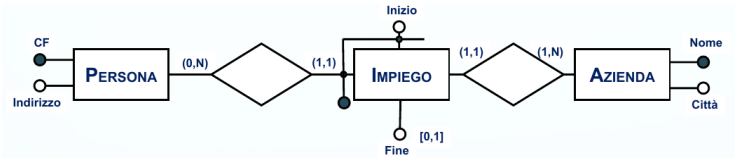


Figure 5.6: Soluzione 3

5.4.6 Associazione ternaria

E' un modo per eliminare un'associazione ternaria, in quanto sono molto complicate da gestire.

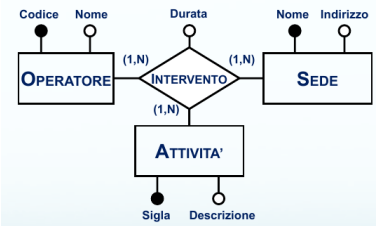


Figure 5.7: Problema: associazione ternaria

La soluzione che troviamo e' reificare l'associazione INTERVENTO, rendendolo quindi un entita' ed inserendo 3 associazioni ausiliare che collegano INTERVENTO a OPERATORE, SEDE e ATTIVITA'. Le cardinalita' saranno in ogni caso 1,1 dalla parte dell'entita' creata e dalla parte delle vecchie entita' rimarranno come prima. L'identificazione della nuova entita' e' un identificazione esterna che prende tutte le entita' vecchie, in questo caso:

- Il codice dell'operatore
- Il nome della sede
- La sigla dell'attivita'

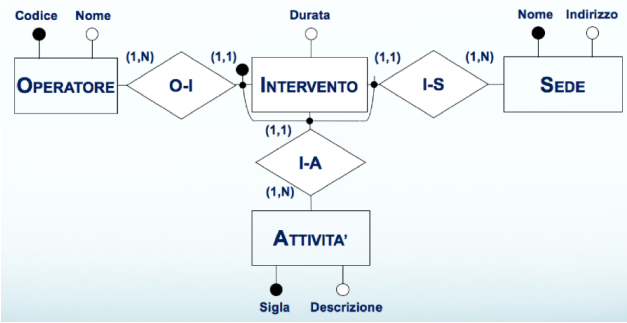


Figure 5.8: Reificazione associazione ternaria

5.5 Strategie di progetto

Quando il progetto e' complesso ci sono delle operazioni piu' generali di quelle viste prima:

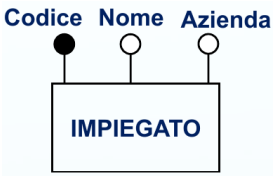
- top-dowm
- bottom-up
- inside-out: variante del bottom-up

5.5.1 Top-Dowm

Si parte dalle specifiche poi si crea uno schema iniziale contenente le entita' principali e piano piano si raffinano fino ad arrivare allo schema finale utilizzando delle primitive di raffinamento.

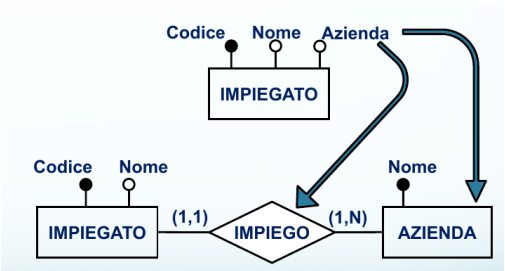
Definizione di attributi di un'entita' o di una associazione

Una volta che abbiamo le entita' e le associazioni principali inseriamo gli attributi che le caratterizzano.



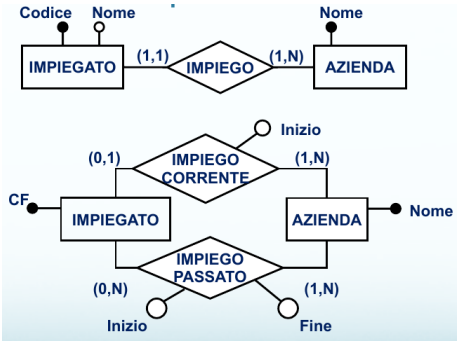
Reificazione di un attributo o di una associazione

Se ci rendiamo conto che un'associazione e' caratterizzante e conviene trasformarla in entita' va fatto (e' una cosa molto comune)



Decomposizione di una associazione in due associazioni distinte

Anche in questo caso se ci rendiamo conto che un'associazione e' espressa meglio se la dividiamo in due, va fatto.



Trasformazione di un'entita' in una gerarchia di generalizzazione

5.5.2 Esempio biblioteche

Riprendiamo l'esempio delle biblioteche di prima e facciamone lo schema ER utilizzando l'approccio top-dowm.

Inizialmente mettiamo solo le entita' e le associazioni base, successivamente ci accorgiamo dei problemi e lo raffiniamo utilizzando le primitive:

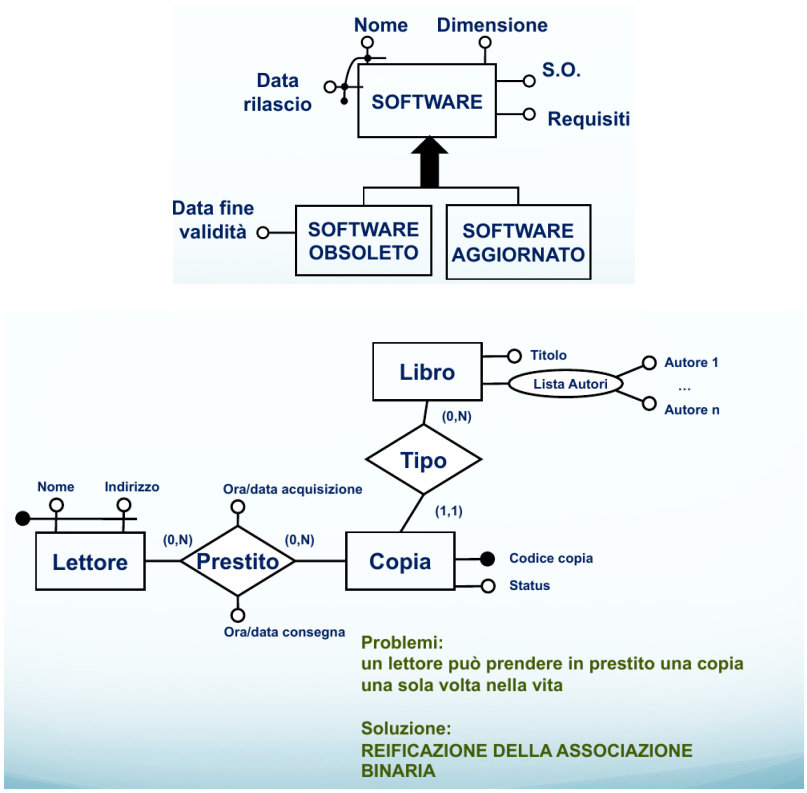


Figure 5.9: Primo schema ER

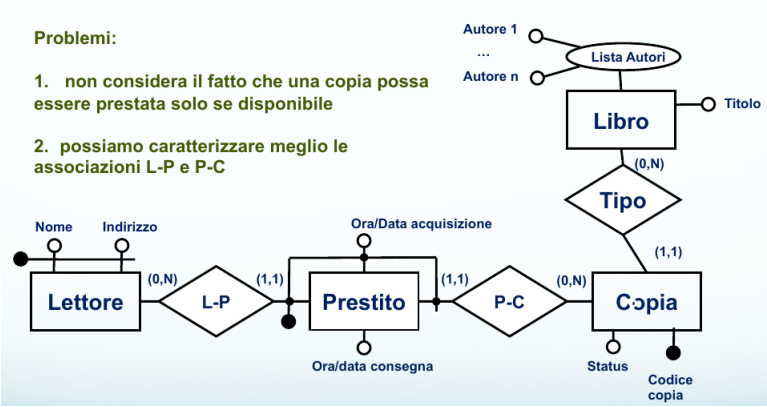


Figure 5.10: Secondo schema ER

5.5.3 Bottom-up

Si creano vari schemi (tendenzialmente da diversi team) e poi si cercano di mettere insieme per formare lo schema finale. Separiamo delle parti del dominio,

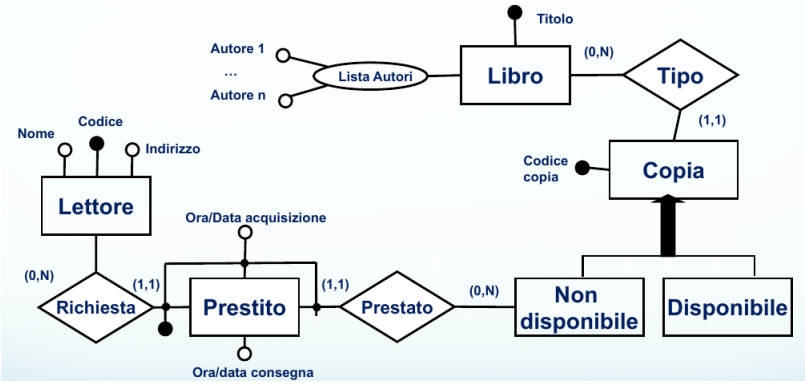


Figure 5.11: Terzo schema ER

perche' potrebbe essere molto complesso, le sviluppiamo completamente e poi facciamo l'unione, vuol dire che le entita' comuni vanno messe insieme.

5.5.4 Inside-out

E' un caso particolare della strategia bottom-up, si decompone il problema in concetti elementari. A partire da un concetto si modellano gli altri a macchia d'olio, si rappresentano prima i concetti iniziali e poi si naviga tra la specifiche allontanandosi.

Chapter 6

Progettazione logica

L'obbiettivo della progettazione logica e' "tradurre" lo schema concettuale in uno schema logico che rappresenti gli stessi dati in maniera corretta ed efficiente.

6.1 Dati in ingresso

Dobbiamo produrre uno schema logico e la documentazione associativa. Come input abbiamo:

- schema concettuale
- informazioni sul carico applicativo
- modello logico (se relazionale o di altro genere)

Non si tratta di una pura e semplice automazione perche' dovremmo anche considerare questioni di prestazioni per interrogare il database. Le due cose che dobbiamo fare sono:

- Considerare il carico applicativo e quindi modificare un pochino lo schema ER
- Traduzione vera a propria

6.2 Ristrutturazione dello schema E-R

Per semplificare la traduzione (la generalizzazione e' difficile da tradurre) ed ottimizzare le prestazioni dovremo ristrutturare lo schema E-R. La parte concettuale e' gia' finita, uno schema ristrutturato non e' piu' uno schema concettuale nel senso stretto del termine. Modificandolo vuol dire che lo schema concettuale e' quello che abbiamo gia' consegnato ma noi potremmo farne una ristrutturazione diversa per i vari usi applicativi.

6.3 Prestazioni

In questo corso come prestazioni intenderemo ad esempio andare a vedere un'informazione sia modificare dei dati.

Le prestazioni non sono valutabili con precisione su uno schema concettuale, bisogna capire come fare, faremo quindi alcune considerazioni di alto livello. Considereremo dei parametri che regolano le prestazioni, in informatica si distinguono sempre in spazio (la memoria occupata) ed il tempo (quanto tempo ci metto a trovare un elemento), il tempo e' influenzato dal numero di occorrenze che vengono visitate durante un'operazione.

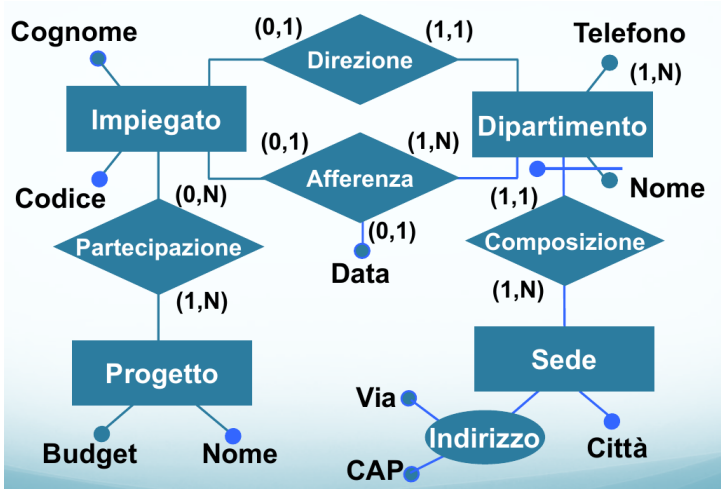


Figure 6.1: Esempio 1

6.4 Tavola dei volumi

Chiediamo di darci un idea di quante occorrenze ci sono per ogni entita' ed associazioni (quanti impiegati avete nel dipartimento?). Noi da queste info ci costruiamo una tavola dei volumi, che e' molto approssimata, vogliamo solo avere un'idea al momento quant'e' il volume attuale del database. Per quanto riguarda le associazioni ci chiediamo quante coppie associa ogni associazione, ma lo capiamo dalle associazioni (tranne nel caso della partecipazione al progetto, ci siamo inventati che mediamente un impiegato partecipa a 3 progetti, da cui 6000).

Concetto	Tipo	Volume
Sede	E	10
Dipartimento	E	80
Impiegato	E	2000
Progetto	E	500
Composizione	R	80
Afferenza	R	1900
Direzione	R	80
Partecipazione	R	6000

6.4.1 Esempio di valutazione di costo

A volte per decidere se nella tabella risultante togliere o meno un attributo ridondante dobbiamo valutare le prestazioni. Vediamo un esempio di valutazione di costo, l'operazione che dobbiamo fare è: trova tutti i dati di un impiegato

del dipartimento al quale lavora ed ai progetti ai quali partecipa. Si costruisce una tavola degli accessi basata su uno schema di navigazione. Siccome abbiamo lo schema E/R immaginiamo cosa dobbiamo fare per andare a prendere le informazioni dell'impiegato: partendo da Impiegato devo muovermi seguendo le associazioni per restituire le informazioni che mi sono richieste.

In base a questa navigazione conto quante volte devo accedere ai dati. Il tipo e' lettura (L) o scrittura (S) in quanto hanno dei costi diversi (la lettura costa un po' meno), gli accessi vuol dire quante volte vado a prendere un'occorrenza.

Concetto	Costrutto	Accessi	Tipo
Impiegato	Entità	1	L
Afferenza	Relationship	1	L
Dipartimento	Entità	1	L
Partecipazione	Relationship	3	L
Progetto	Entità	3	L

6.5 Attivita' della ristrutturazione

- Analisi delle ridondanze, dobbiamo capire se c'e' qualcosa di ridondante e decidere se tenerlo o no
- Eliminazione delle generalizzazioni, facciamo dei ragionamenti su come togliere le generalizzazioni
- Partizionamento/accorpamento di entita' e associazioni
- Scelta degli identificatori primari

6.5.1 Analisi delle ridondanze

Il vantaggio nel tenere le ridondanze e' quello di semplificare le interrogazioni. Gli svantaggi:

- Appesantimento degli aggiornamenti
- Maggiore occupazione di spazio

Forme di ridondanza

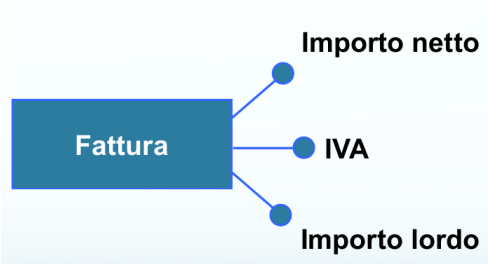
attributi derivabili:

- da altri attributi della stessa entità (o relationship)
- da attributi di altre entità (o relationship)

Oppure relationship derivabili dalla composizione di altre (più in generale: cicli di relationship).

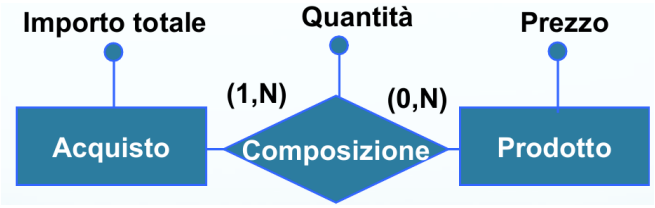
Attributo derivabile

E' un'attributo che si puo' derivare dagli altri attributi.



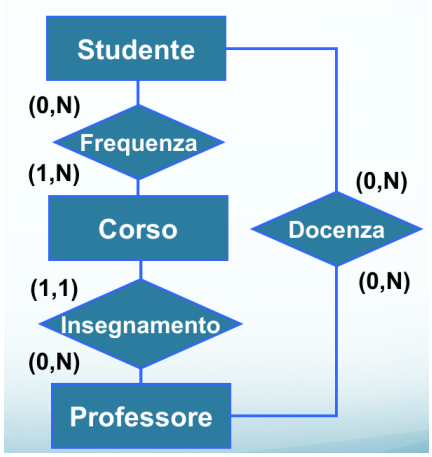
Attributo derivabile da altra entita'

Ci sono delle relazioni tra i tre attributi ed e' possibile eliminare qualcosa di questi, ad esempio per calcolare l'importo totale basta moltiplicare la quantita' per il prezzo.



Ridondanza dovuta ad un ciclo

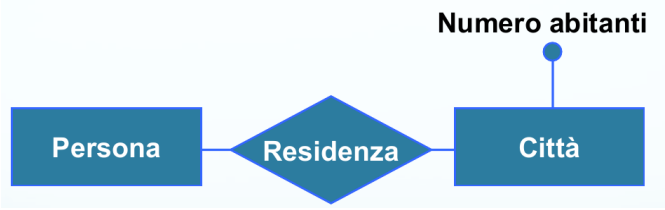
Il fatto che un professore sia il docente di uno studente lo posso derivare dal fatto che la persona che sto considerando frequenta il corso insegnato dal docente, quindi indirettamente e' il docente dello studente (e' l'associazione ad essere ridondante).



6.5.2 Analisi di una ridondanza

Non c'e' una risposta SI o NO per tenere una ridondanza, bisogna fare un'analisi. Possiamo fare un ragionamento intuitivo parlando con l'esperto del dominio, anche se in realta' si dovrebbe seguire un approccio formale.

Nell'esempio abbiamo un associazione residenza tra persona ed abitanti, il numero di abitanti e' il numero di residenti, c'e' ridondanza perche' posso contare le occorrenze e derivare il numero di abitanti. Conviene tenerlo questo attributo?



Facciamo quello che abbiamo fatto prima, andiamo dal nostro esperto e gli chiediamo piu' o meno in questo database quante citta' ci saranno, quante persone e stiliamo la tabella dei volumi.

Concetto	Tipo	Volume
Città	E	200
Persona	E	1000000
Residenza	R	1000000

Dopo che ho fatto questa tabella dei volumi chiediamo quali sono le operazioni principali che coinvolgono questo pezzo di databse, nel nostro esempio:

1. Memorizza una nuova persona con la relativa città di residenza (500 volte al giorno)

2. Stampa tutti i dati di una città (incluso il numero di abitanti) (2 volte al giorno)

Dobbiamo considerare queste operazioni, faccio la tavola degli accessi pensando di tenere la ridondanza numero di residenti:

Operazione 1

Concetto	Costrutto	Accessi	Tipo
Persona	Entità	1	S
Residenza	Relazione	1	S
Città	Entità	1	L
Città	Entità	1	S

Operazione 2

Concetto	Costrutto	Accessi	Tipo
Città	Entità	1	L

l'operazione 1 dice di andare ad inserire una nuova persona con la sua residenza, ci sono quindi anche operazioni di scrittura (S). L'operazione 2 deve solo andare ad accedere citta' e fa un accesso in lettura.

Facciamo ora la tavola degli accessi in assenza di ridondanza, ovvero togliendo l'attributo numero di residenti a citta':

Operazione 1

Concetto	Costrutto	Accessi	Tipo
Persona	Entità	1	S
Residenza	Relazione	1	S

Operazione 2

Concetto	Costrutto	Accessi	Tipo
Città	Entità	1	L
Residenza	Relazione	5000	L

L'operazione 1 in questo caso va a scrivere una nuova persona (un accesso in scrittura all'entita' persona), quando vado ad inserire la residenza non devo piu' aggiornare il numero di residenti, quindi ho un solo accesso in scrittura alla relazione residenza. La seconda operazione e' piu' complicata, perche' devo andare a ricavare il numero di abitanti contando le occorrenze delle associazioni tra della citta', quindi accedo una volta alla citta' e poi devo contare tutte le occorrenze per quella citta', 5000 lo ricaviamo da quello che ci hanno detto prima, cioe' che c'era 1000000 di abitanti nel database e un numero di citta', faccio la media di abitanti per citta'.

A questo punto devo calcolare i costi, prima in presenza di ridondanza:

- Operazione 1: 1500 accessi in scrittura con la presenza di ridondanza (perche' erano 3 accessi per 500 volte al giorno) e 500 accessi in lettura

- L'operazione 2: non e' rilevante

Facciamo che le scritture costano il doppio delle letture, quindi il costo totale e' 3500 accessi al giorno.

senza ridondanza invece:

- Operazione 1: 1000 accessi in scrittura, perche' c'erano 2 accessi per 500 volte al giorno.
- Operazione 2: 10000 accessi al giorno, perche' c'erano 5000 accessi per 2 volte al giorno.

Quindi sempre considerando che le scritture valgono 2 letture, il costo totale e' 12000 accessi in lettura al giorno.

A questo punto capiamo che e' meglio tenere l'attributo abitanti nel progetto (la ridondanza), perche' se lo tengo ho molti meno accessi da fare (quindi costa meno).

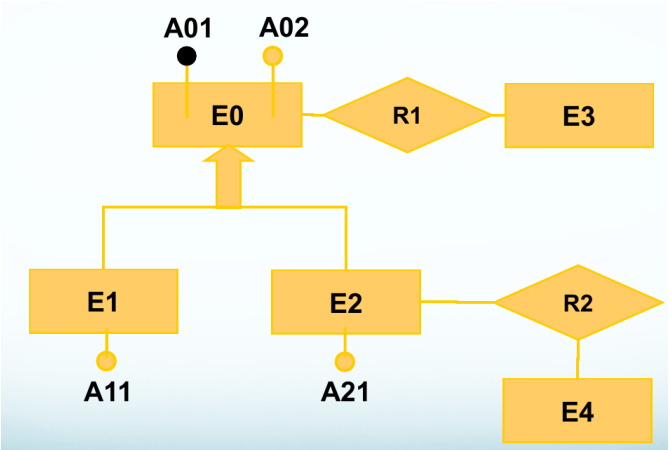
In realta' non va fatto solo per le ridondanze, quando siamo incerti su cosa fare un'analisi sulle operazioni piu' frequenti potrebbe farci decidere tra due schemi che sono entrambi giusti ma optiamo per quello piu' efficiente.

6.5.3 Eliminazione delle generalizzazioni

Non possiamo rappresentare la generalizzazione in tabelle, quindi dobbiamo toglierle. Si eliminano perciò le gerarchie, sostituendole con entità e relationship. Abbiamo tre possibilità per eliminare le generalizzazioni:

- Accorpamento delle figlie di generalizzazione nel genitore
- Accorpamento del genitore della generalizzazione nelle figlie
- Sostituzione della generalizzazione con relationship, invece di fare una generalizzazione li colleghiamo con delle associazioni all'entita' padre

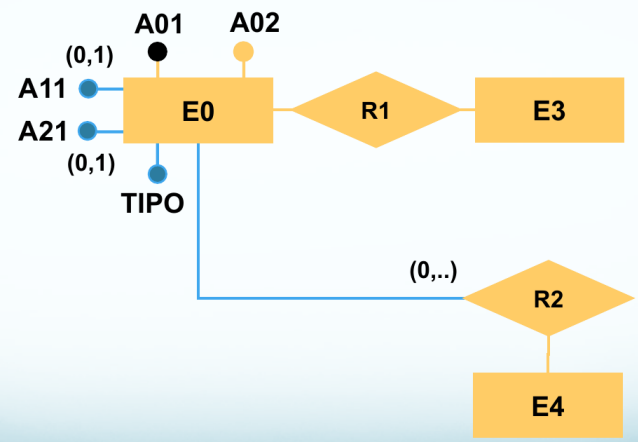
Consideriamo questo schema astratto, ricordiamo che distinguiamo due entita' dei figli dall'attributo A01 del padre.



Accorpamento delle figlie di generalizzazione nel genitore

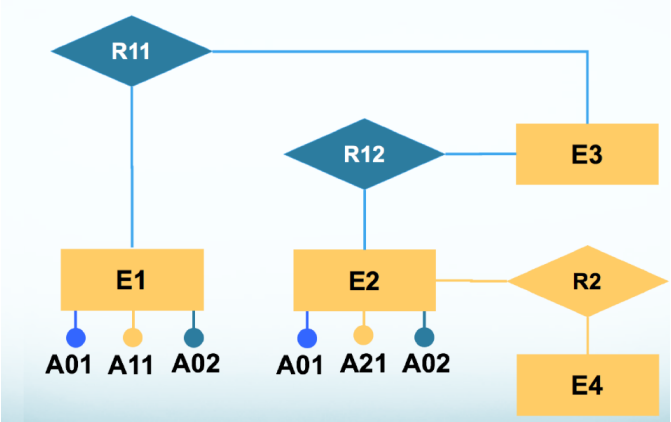
Questa attivita' devo farla cercando di non inficiare la validita' del database risultante, quindi tutti gli elementi che ero in grado di memorizzare prima devo essere in grado di memorizzarli anche ora. Metto tutti gli attributi dei figli nel padre, anche l'associazione R2 che prima era solo del figlio, ovviamente nel fare questo devo riconsiderare le cardinalita', dobbiamo metterli opzionali (0,1) in quanto ci sono ora occorrenze solo per un certo tipo di entita', non per tutti. Inoltre dobbiamo aggiungere un nuovo attributo TIPO che mi serve per distinguere i due figli.

Il problema di questa soluzione e' che perdiamo dei vincoli, tipo la cardinalita' 1,1 di alcuni attributi, devo quindi tramite la documentazione dire questi vincoli che devono esserci ma ho perso agli sviluppatori delle applicazioni.



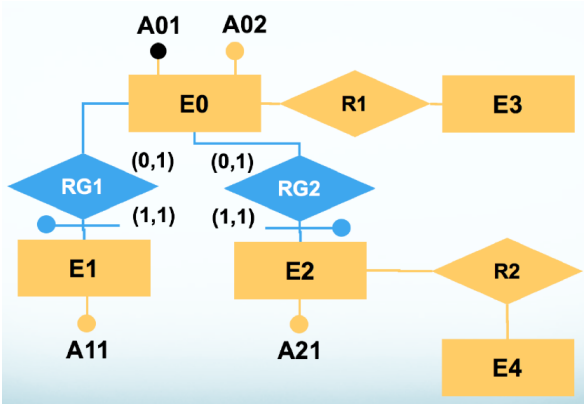
Accorpamento della generalizzazione nelle figlie

Se la generalizzazione e' parziale vuol dire che c'e' un elemento nel padre che non appartiene a nessuna delle sottoclassi non posso fare questo accorpamento, in quanto perderei quelle entita' e diminuiremmo la capacita' espressiva del database. L'unico modo per accorpare in questo modo e' che la generalizzazione sia TOTALE. Succede che abbiamo bisogno dell'identificatore che era nel padre per ciascuno dei figli (se c'era codice fiscale ora c'e' codice fiscale in tutti i figli), hanno anche ereditato un nuovo attributo che e' ridondato in tutti e due A02. Allo stesso modo ho dovuto usare due associazioni, perche' l'entita' padre era in associazione con E3 e dato che non esiste piu' ho dovuto passare questa associazione anche ai figli.



Sostituzione della generalizzazione con delle relazioni

Siccome so che l'attributo A01 funzionava bene come identificatore per le classi figlie, uso una generalizzazione esterna per evitare di usare un identificatore specifico per tutte le classi figlie. La cardinalita' e' 0,1 perche' non posso obbligare ogni occorrenza del padre E0 a partecipare all'occorrenza con E1, e' sbagliato mettere 1,1 anche se la generalizzazione fosse totale sovrapposta, perche' significherebbe che ogni elemento di E0 dev'essere per forza in relazione con E1 ed E2, ma non e' cosi'. Quello che perdo e' che non capisco piu' se la generalizzazione e' totale o parziale, devo scrivermi a parte questo vincolo, perche' anche se fosse stata totale ora possono esistere entita' di E0 che non partecipano ne ad E1 ne a E2.



Scelta tra alternative

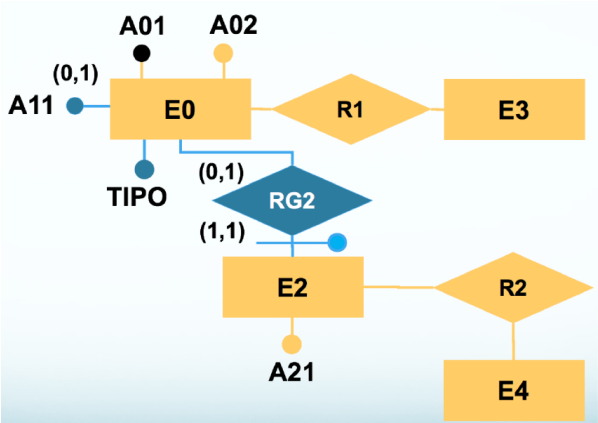
La scelta fra le alternative si puo' fare con metodo simile a quello viso per l'analisi delle ridondanze considerando il costo delle operazioni (tavola degli accessi) e occupazione di memoria. Facciamo quello fatto prima per eliminare le ridondanze per capire quale schema e' piu conveniente. Esistono anche delle regole generali:

- La prima soluzione conviene se gli accessi al padre e alle figlie sono contestuali. Se si parla di accessi vuol dire che abbiamo chiesto all'esperto del dominio. Accessi contestuali vuol dire che quando chiedo le informazioni delle entita' padre, in realta' chiedo anche gli attributi del figlio. Se invece mi dicono che quando fanno l'accesso chiedono solo gli attributi del padre e raramente accedono a quelle dei figli, sono accessi non contestuali. Ovviamente se fanno accessi contestuali ha senso logicamente mettere tutti gli attributi insieme

- La seconda necessita di generalizzazione totale e conviene se gli accessi alle figlie sono disgiunti, vuol dire che faccio delle interrogazioni per una singola figlia, ma mai insieme

- La terza conviene per generalizzazione parziale e se gli accessi alle entita' figlie sono disgiunte agli accessi al padre. Questa e' utile nel caso in cui gli accessi sono disgiunti, non chiediamo tutti gli attributi quando facciamo un accesso.

Si possono anche usare delle soluzioni ibride, nell'esempio per una delle figlie abbiamo usato il terzo metodo e per l'altro il primo metodo. Questi ragionamenti li facciamo in base all'utilizzo del database considerando le operazioni frequenti, valutando cosa puo' essere piu' efficiente per il nostro cliente.



6.5.4 Partizionamento delle entita'

Gli accessi si riducono:

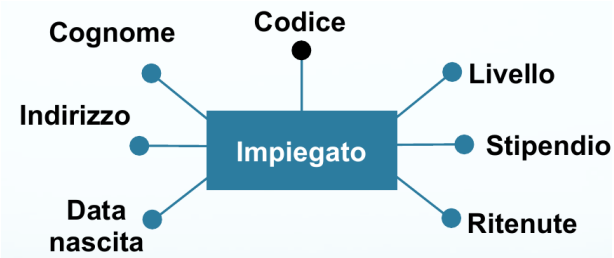
- Separando attributi di un concetto che vengono acceduti separatamente
- Raggruppando attributi di concetti diversi acceduti insieme

I casi principali in cui devo fare una ristrutturazione sono:

- Partizionamento di entità
- Partizionamento di relationship
- Eliminazione di attributi multivalore
- Accorpamento di entità/relationship

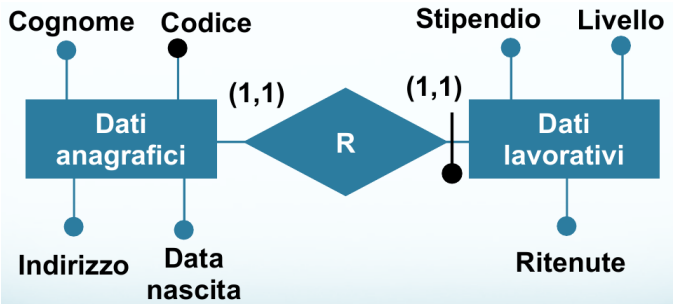
Partizionamento di entità

Abbiamo la nostra entita' con tutti gli attributi che ci servono, pero' quando chiediamo all'esperto del dominio ci dice che accedono raramente ad alcuni attributi rispetto ad altri.



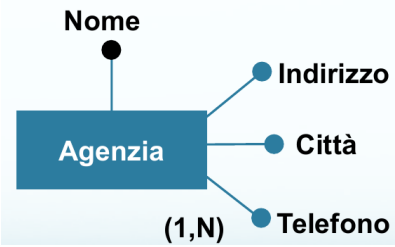
Potrei quindi decidere di partizionare le entita', perche' se vengono acceduti degli attributi in modo diverso posso spezzare le due entita', sempre tenendole in

relazione, nell'esempio divido i dati anagrafici dai dati lavorativi, la cardinalita' in questo caso e' 1,1 perche' stiamo parlando della stessa persona.

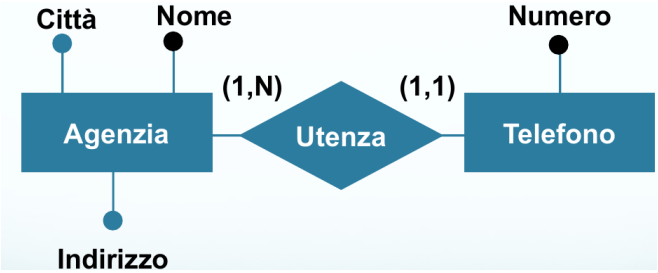


Eliminazione di attributi multi-valore

Non sara' possibile rappresentare in tabelle un attributo con piu' valori (una cella con piu' valori). Quindi lo trasformo in un entita'.



Da questo esempio notiamo che un agenzia piu' avere piu' numeri di telefono. Le cardinalita' sono 1,N in questo caso ma potrei mettere nello specifico quanti numeri di telefono ci sono.



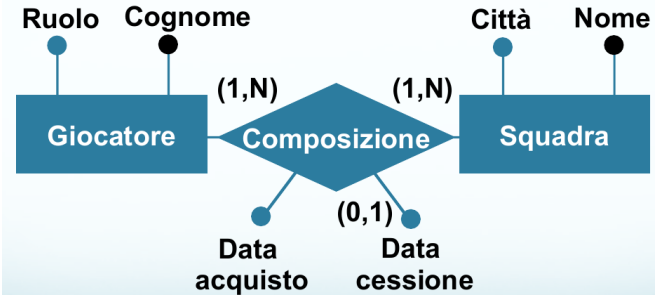
Accorpamento di entita'

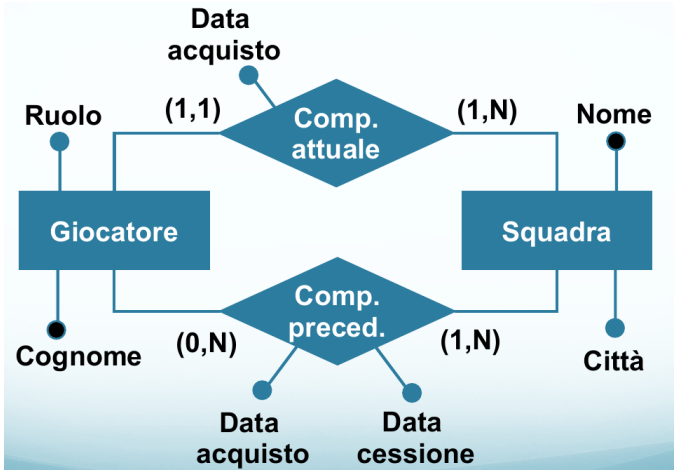
Ci sono dei casi in cui ho progettato due entita' e ne vale la pena farne una sola per questioni di prestazioni. Quando le operazioni piu' frequenti di un

entita' ne coinvolgono un'altra potrebbe convenire accorparle. Nell'esempio ci viene detto che tutte le volte che cerca i dati della persona vuole anche i dati dell'appartamento che e' intestato, allora possiamo evitare di percorrere un'associazione tutte le volte e portiamo gli attributi di un entita' nell'altra miglioro l'efficienza. Devo stare attento a mettere gli attributi opzionali (0,1) perche' c'era una partecipazione opzionale da mantenere negli attributi.

6.5.5 Partizionamento delle associazioni

Se si osservasse che le operazioni piu' frequenti accedono in modo disgiunto alle occorrenze dell'associazione attuale, in modo disgiunto vuol dire che se il mio cliente mi dice che guarda quasi sempre la composizioni attuale della squadra e non la storia, allora potrei decidere di dividere la composizione attuale da quella precedente.





6.5.6 Scelta degli identificatori principali

In reltà abbiamo già messo gli attributi con il pallino pieno o esterni.

Con che criteri scegliamo:

- Assenza di opzionalita': non puo' essere opzionale un identificatore, le chiavi primarie non possono essere nulle
- Semplicita': meglio un identificare fatto da un solo attributo piuttosto che da molti
- Utilizzo nelle operazioni piu' frequenti: se un attributo e' usato molto piu' di un altro e' da preferire rispetto ad un attributo usato meno.

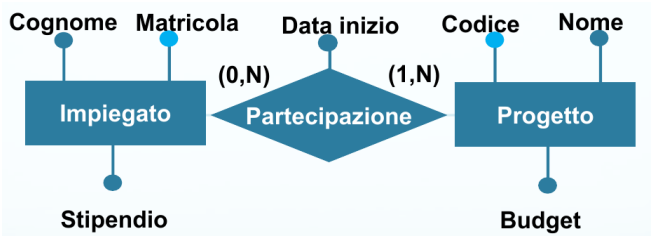
Se nessuno degli identificatori soddisfa i requisiti visti si introducono nuovi attributi (codici) contenenti valori speciali generati appositamente per questo scopo.

6.6 Traduzione nel modello relazionale

Supponiamo di avere ristrutturato lo schema secondo i criteri visti ed adesso dobbiamo trovare lo schema delle tabelle, c'e' un'idea di base:

- Le entita' diventano relazioni sugli stessi attributi
- Le associazioni diventano relazioni sugli identificatori delle entita' coinvolte (piu' gli attributi propri)

6.6.1 Entita' e associazioni molti a molti



Abbiamo un pezzo di uno schema. Allora noi qui procediamo secondo l'idea base: abbiamo detto che impiegato e progetto diventano due relazioni (si da un nome che nel nostro caso e' quello dell'entita', poi si apre una parentesi e si mettono i nomi degli attributi separati da virgola, uno di questi e' l'identificatore principale e lo sottolineiamo):

$$Impiegato(\underline{Matricola}, Cognome, Stipendio)$$
$$Progetto(\underline{Codice}, Nome, Budget)$$
$$Partecipazione(\underline{Matricola}, \underline{Codice}, DataInizio)$$

L'associazione se la facciamo nel modo banale scritto prima lo facciamo come abbiamo scritto impiegato e progetto, identificata con gli identificatori delle due

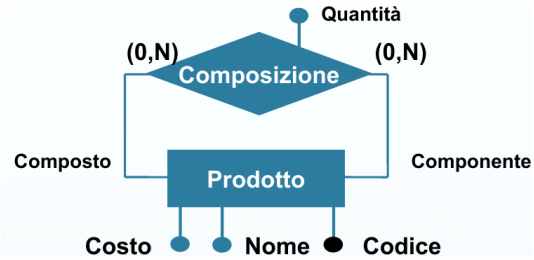
partecipanti, ovviamente qui non sono obbligato nella traduzione a mantenere gli stessi nomi: Partecipazione(Matricola, Codice, DataInizio).

Fare sempre attenzione ai vincoli di integrita' referenziale, matricola deve avere un vincolo di integrita' referenziale con Impiegato ed il codice del progetto deve avere vincolo di integrita' referenziale con Progetto (devono esistere).

Conviene ovviamente usare dei nomi piu' espressivi per gli attributi della chiave della relazione che rappresenta l'associazione.

La traduzione non riesce a tenere conto della cardinalita' minime delle associazioni molti a molti (se non con vincoli di CHECK complessi e poco usati). Vuol dire l'opzionalita', quindi non riusciamo a tradurre l'opzionalita'.

6.6.2 Associazioni ricorsive



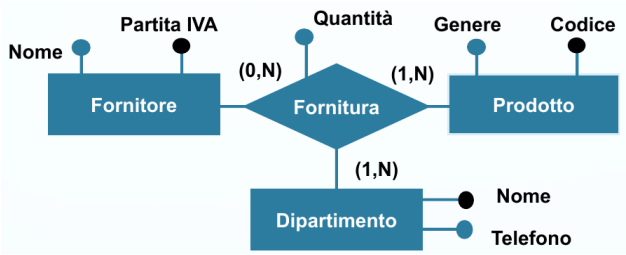
Quando ho un'associazione ricorsiva, significa che mette in relazione un prodotto con un altro prodotto (i componenti di un prodotto sono a loro volta prodotti). Quando abbiamo una cosa di questo genere e' meglio scriversi delle etichette, uno ha un ruolo da componente ed uno da composto (in questo caso). Utilizzeremo queste etichette nella traduzione:

$$Prodotto(\underline{Codice}, Nome, Costo)$$
$$Composizione(\underline{Composto}, \underline{Componente}, Quantita')$$

Il vincolo di integrita' del componente dev'essere Codice, cosi' come il Composto. Quindi ci vanno due codici ed una quantita', per dire che non ci va un codice a caso ma ci va un codice di prodotto usiamo le etichette.

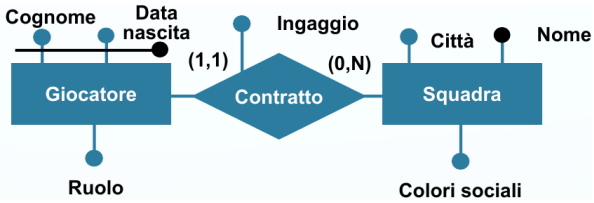
6.6.3 Associazioni n-arie

Non c'e' nessuna differenza con quello fatto prima. L'associazione e' esattamente come prima, prende le tre entita' che sono messe in relazione dall'associazione, subito pensiamo ai vincoli di integrita' referenziale, e poi il suo attributo quantita'.



$$Fornitore(\underline{PartitaIVA}, Nome)$$
$$Prodotto(\underline{Codice}, Genere)$$
$$Dipartimento(\underline{Nome}, Telefono)$$
$$Fornitura(\underline{Fornitore}, \underline{Prodotto}, \underline{Dipartimento}, Quantita)$$

6.6.4 Associazioni uno a molti



Qui abbiamo un giocatore che ha uno ed un solo contratto con una squadra. In questa prima versione abbiamo usato lo stesso criterio di prima, quello generale, mettendo l'identificatore dell'entita' giocatore e poi abbiamo messo l'identificatore dell'altra entita' che partecipa:

$$Giocatore(\underline{Cognome, DataNascita}, Ruolo)$$
$$Contratto(\underline{CognGiocatore, DataNascG, Squadra}, Ingaggio)$$
$$Squadra(\underline{Nome}, Citta, ColoriSociali)$$

Ma e' corretto? Dobbiamo chiederci se la chiave di Contratto e' veramente una chiave? Ovvero e' una superchiave minimale, posso togliere uno degli identificatori ed essere ancora in grado di distinguere due contratti? Siccome c'e' un 1,1 per ogni giocatore ci puo' essere un solo contratto con le squadre, allora se io ho due contratti posso distinguerli solamente avendo il nome del giocatore, quindi posso togliere il nomeSquadra. Il motivo e' solo che la cardinalita' e' 1,1 non c'e' bisogno neanche di pensarci.

Quindi una soluzione piu' compatta la ottengo togliendo squadra dalla chiave di contratto, tuttavia le relazioni giocatore e contratto hanno la stessa chiave,

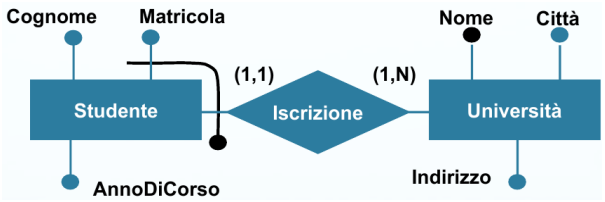
allora potrei anche pensare un database piu' semplice se per ogni giocatore c'e' sempre un solo contratto e la chiave e' la stessa. Possiamo mettere dentro giocatore gli attributi di contratto e fare un'unica tabella:

$$Giocatore(\underline{Cogome, DataNascita}, Ruolo, Squadra, Ingaggio)$$
$$Squadra(\underline{Nome}, Citta', ColoriSociali)$$

Con vincolo di integrita' referenziale fra Squadra in Giocatore e la chiave di Squadra. Se la cardinalita' dell'associazione dalla parte di giocatore fosse (0,1), allora Squadra e Ingaggio in Giocatore dovrebbero ammettere valore nullo.

Quindi in generale quando c'e' un associazione uno a molti si fa assorbire l'associazione dalla parte dell'1,1.

6.6.5 Entita' con identificazione esterna



Basta che ci ricordiamo quando si puo' fare l'identificazione esterna, ho bisogno di avere cardinalita' 1,1 dell'associazione che partecipa. Allora probabilmente la traduciamo allo stesso modo di come abbiamo fatto adesso. Facciamo prima le due entita' e l'associazione non le mettiamo, semplicemente ci mettiamo l'attributo esterno di Universita'.

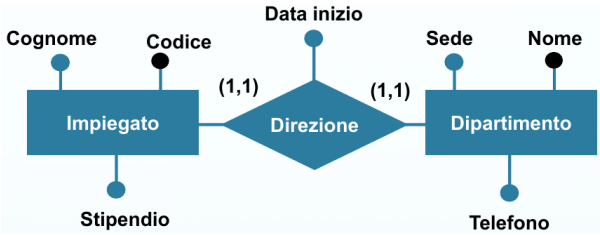
$$Studente(\underline{Matricola, Universita}, Cognome, AnnoDiCorso)$$
$$Universita(\underline{Nome}, Citta, Indirizzo)$$

Il vincolo di integrita' e' tra Universita' in studente ed il Nome dell' Universita'.

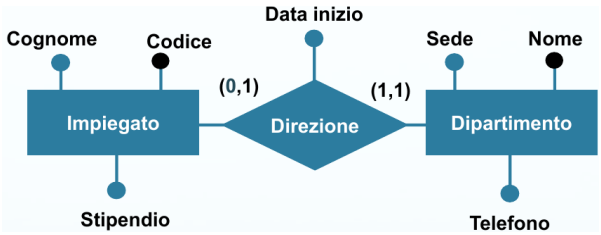
6.6.6 Associazioni uno a uno

Come visto prima potremmo fare assorbire l'associazione in una delle due parti 1,1. Ma c'e' anche un'altra possibilita' che potrebbe venirci in mente, mettere tutto insieme. Si puo' fare anche questo ma l'osservazione e' che dopo aver valutato se accorpare o separare le entita' se le abbiamo messe cosi' significa che ci conviene tenerle separate, quindi non fondiamo tutto in una sola tabella ma decidiamo se mettere l'associazione in una delle due tabelle.

Se da una parte c'e' 0,1 preferiamo sempre la fusione dove ci sono meno valori nulli.



6.6.7 Associazioni uno a uno con due partecipazioni opzionali

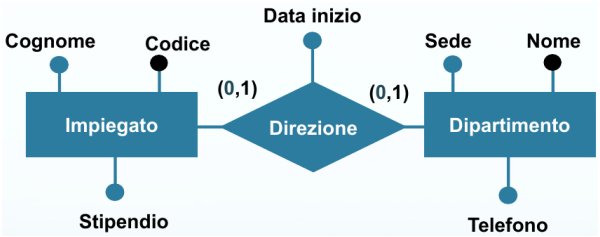


Se 1,1 non ce l'ho da nessuna delle due parti si preferisce la traduzione in tre relazioni distinte (con due vincoli di integrita' referenziale), faccio una tabella anche per la relazione. Il motivo di questa scelta e' che non ci piacciono i valori nulli.

$$Impiegato(\underline{Codice}, Cognome, Stipendio)$$
$$Dipartimento(\underline{Nome}, Sede, Telefono, Direttore, InizioD)$$

Con vincolo di integrità referenziale, senza valori nulli.

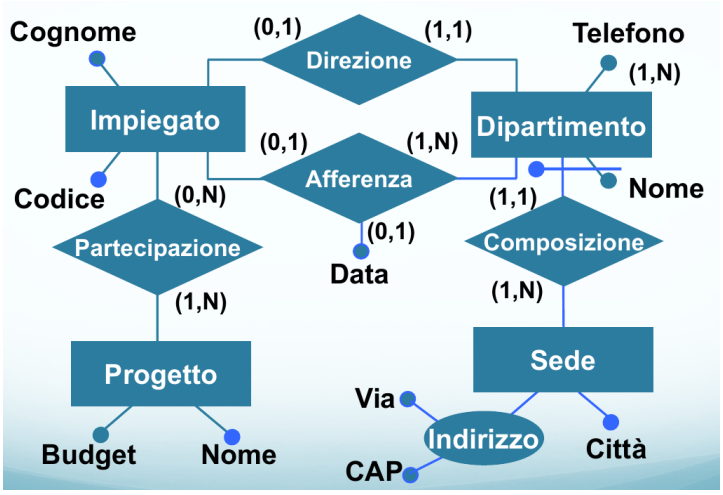
6.6.8 Associazioni uno a uno con due partecipazioni opzionali



Si preferisce la traduzione in tre relazioni distinte (con due vincoli di integrità referenziale).

6.7 Esempio

Proviamo a tradurre lo schema in uno schema relazionale.



Cominciamo delle entita', creiamo una relazione per ciascuna con il nome delle entita' ed i loro attributi. Poi guardiamo le cardinalita' delle relazioni e dove troviamo 1,1 facciamo assorbire la relazione da quell'entita' (ad esempio Afferenza la facciamo assorbire da Impiegato).

Mancano i vincoli di integrita' referenziale. Direttore e' il codice dell'impiegato che e' il Direttore, quindi ogni volta che faccio assorbire devo fare attenzione che c'e' un vincolo di integrita' referenziale.

$$Impiegato(\underline{Codice}, Cognome, Dipartimento, Sede, Data)$$
$$Dipartimento(\underline{Nome}, \underline{Citta}, Telefono, Direttore)$$
$$Sede(\underline{Citta}, Via, CAP)$$
$$Progetto(\underline{Nome}, Budget)$$
$$Partecipazione(\underline{Impiegato}, \underline{Progetto})$$

6.8 Strumenti di supporto

Esistono sul mercato prodotti (CASE) che forniscono un supporto a tutte le fasi della progettazione di basi di dati.

Ci possiamo gia' esercitare con un compito d'esame aperto su exam.net con uno strumento di disegno che dovremo utilizzare all'esame. Nella nostra vita professionale e' meglio usare un tool che ci aiuta a fare la traduzione in schema logico.

Chapter 7

Algebra relazionale

Passiamo a capire come si puo' interrogare una base di dati, parleremo quindi di linguaggi ma inizialmente dovremo partire dalle operazioni di base (teoria) dei linguaggi. Queste operazioni le faremo sulle relazioni, andremo quindi a manipolare le relazioni con degli operatori. Ricordiamo che le operazioni sullo schema si fanno tramite DDL (data definition language), mentre le operazioni sui dati si fanno tramite DML (data manipulation language), interrogazione (query) e aggiornamento. I linguaggi per la base di dati quindi ci aiutano a cambiare lo schema e ci permettono di fare le operazioni sui dati delle relazioni.

L'algebra relazionale è un insieme di operatori che operano sulle relazioni ed e' obbligatorio che il risultato sia una relazione, questo vuol dire che li posso comporre.

7.1 Operatori

- unione, intersezione, differenza
- ridenominazione
- selezione
- proiezione
- join (join naturale, prodotto cartesiano, theta-join)

I primi tre sono operazioni sugli insiemi, dato che le relazioni sono degli insiemi, è possibile applicare unione, intersezione e differenza solo a relazioni definite sugli stessi attributi. Selezione, proiezione e join sono molto importanti. (ridenominazione e' solo cambiare dei nomi). La selezione vuol dire che di una tabella selezioniamo alcune righe, dato che non le usiamo tutte. La proiezione e' il duale della selezione, mi interessano solo alcune colonne. Il Join e' la piu' difficile, ci permette di incrociare i dati da piu' tabelle, se devo capire dov'e' residente uno studente e se la tabella residente e' diversa da quella degli studenti dobbiamo per forza usare l'operazione di join per trovare delle informazioni in piu' tabelle.

7.1.1 Unione

Ci sono due tabelle in cui ci sono degli studenti con lo stesso schema (etichette), voglio fare l'unione, ma ricordando che non ci possono essere due tuple uguali in una relazione, di conseguenza le tuple risultanti dalla relazione sarebbero 4 (non mettiamo quelle che si ripetono).

Laureati

Matricola	Nome	Età
7274	Rossi	42
7432	Neri	54
9824	Verdi	45

Specialisti

Matricola	Nome	Età
9297	Neri	33
7432	Neri	54
9824	Verdi	45

Laureati ∪ Specialisti

Matricola	Nome	Età
7274	Rossi	42
7432	Neri	54
9824	Verdi	45
9297	Neri	33

7.1.2 Intersezione

Ho sempre le stesse tabelle ed intuitivamente tengo solo le tuple che stanno in entrambe.

Laureati

Matricola	Nome	Età
7274	Rossi	42
7432	Neri	54
9824	Verdi	45

Specialisti

Matricola	Nome	Età
9297	Neri	33
7432	Neri	54
9824	Verdi	45

Laureati ∩ Specialisti

Matricola	Nome	Età
7432	Neri	54
9824	Verdi	45

7.1.3 Differenza

Ho sempre le stesse tabelle, se voglio fare A-B prendo tutte le tuple di A e gli tolgo quelle che stanno in B.

Laureati

Matricola	Nome	Età
7274	Rossi	42
7432	Neri	54
9824	Verdi	45

Specialisti

Matricola	Nome	Età
9297	Neri	33
7432	Neri	54
9824	Verdi	45

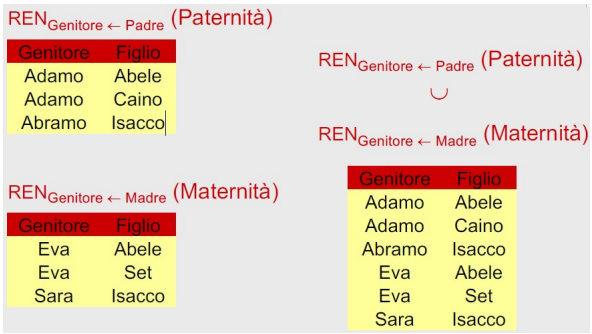
Laureati – Specialisti

Matricola	Nome	Età
7274	Rossi	42

7.1.4 Unione sensata ma impossibile

Cosa succede se vogliamo fare un'operazione che coinvolge due schemi che non sono uguali (le etichette delle colonne sono diverse), l'algebra non ci permette di farlo, vedremo poi che nei linguaggi pratici sara' possibile farlo, basta che siano tutte stringhe ad esempio. Qui ci viene in aiuto l'operatore di ridenominazione,

perche' possiamo pensare di cambiare il nome della colonna temporaneamente, in questo caso potrei cambiare il nome della colonna padre e madre in genitore, cosi' abbiamo lo stesso schema e possiamo fare l'unione.



7.1.5 Ridenominazione

Ci permette di cambiare temporaneamente il nome di una colonna, vuol dire che non sto andando a cambiare il database ma lo faccio soltanto per fare l'interrogazione. E' un operatore monadico (con un singolo argomento). La sintassi:

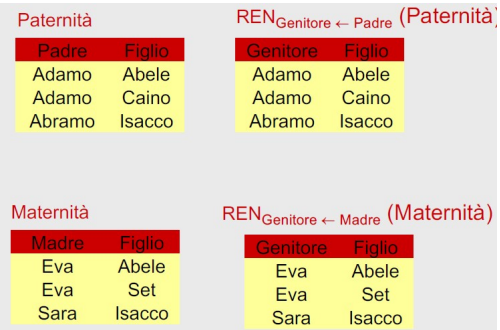
$$REN_{ListaAttributiNew \leftarrow ListaAttributi}(Operando)$$

$$Oppure:$$

$$\rho_{ListaAttributiNew \leftarrow ListaAttributi}(Operando)$$

Semantica: sostituisce il nome di ogni attributo della relazione Operando in ListaAttributi con il corrispondente in ListaAttributiNew.

Notiamo che posso cambiare anche piu' nomi contemporaneamente basta che siano separati da virgole.



7.1.6 Selezione

Vuol dire selezionare qualche riga, e' un operatore monadico (vuol dire che agisce su una sola cosa, ha un solo operando). Produce un risultato che ha lo stesso schema dell'operando, perche' non stiamo toccando le colonne, contiene un sottoinsieme delle ennuple dell'operando, quelle che soddisfano una condizione.

Sintassi:

$$SEL_{Condizione}(Operando)$$

La condizione è un espressione booleana (come quelle dei vincoli di ennupla). Il risultato contiene le ennuple dell'operando che soddisfano la condizione.

Impiegati

Matricola	Cognome	Filiale	Stipendio
7309	Rossi	Roma	55
5998	Neri	Milano	64
9553	Milano	Milano	44
5698	Neri	Napoli	64

Partendo dalla tabella Impiegati vediamo ora varie condizioni di selezione:

- impiegati che guadagnano piu' di 50, stipendio > 50
- impiegati che guadagnano piu' di 50 e lavorano a Milano, Stipendio > 50 AND Filiale = 'Milano'
- impiegati che hanno lo stesso nome della filiale presso cui lavorano, Cognome = Filiale

$SEL_{Stipendio > 50}$ (Impiegati)

Matricola	Cognome	Filiale	Stipendio
7309	Rossi	Roma	55
5998	Neri	Milano	64
5698	Neri	Napoli	64

$SEL_{Stipendio > 50 \text{ AND } Filiale = 'Milano'}$ (Impiegati)

Matricola	Cognome	Filiale	Stipendio
5998	Neri	Milano	64

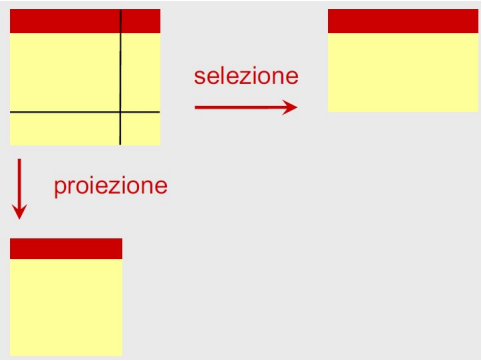
SEL $_{\text{Cognome = Filiale}}$(Impiegati)

Matricola	Cognome	Filiale	Stipendio
9553	Milano	Milano	44

L'operazione mi deve dare un sottoinsieme. I vari esempi ci dicono che l'espressione puo' essere lunga quanto vogliamo, basta che sia booleana. Inoltre non per forza devono esserci costanti da confrontare, ma come nell'ultima selezione posso confrontare due relazioni.

Selezione e proiezione

Sono operatori ortogonali tra loro. La selezione è la decomposizione orizzontale mentre la proiezione è la decomposizione verticale.



7.1.7 Proiezione

E' un operatore monadico. Produce un risultato che ha parte degli attributi dell'operando, puo' avere uno schema diverso da quello originale. Contiene ennuple cui contribuiscono tutte le ennuple dell'operando.

Sintassi:

$$PROJ_{ListaAttributi}(Operando) \text{ oppure } \pi_{ListaAttributi}(Operando)$$

Il risultato contiene le ennuple ottenute da tutte le ennuple dell'operando ristrette agli attributi nella lista.

Sempre considerando la tabella Impiegati di prima, per tutti gli impiegati voglio sapere:

- matricola e cognome
- cognome e filiale

A pedice mettiamo la lista delle colonne che vogliamo ottenere.

Matricola	Cognome
7309	Neri
5998	Neri
9553	Rossi
5698	Rossi

PROJ $_{\text{Matricola, Cognome}}$ (Impiegati)

Cognome	Filiale
Neri	Napoli
Neri	Milano
Rossi	Roma

PROJ $_{\text{Cognome, Filiale}}$ (Impiegati)

Cardinalità delle proiezioni

Se applico una proiezione quante tuple mi trovo nel risultato? intuitivamente dato che stiamo lavorando con le colonne rimangono le stesse, il problema nasce quando troviamo due tuple duplicate facendo la proiezione in alcuni casi. Allora l'algebra non puo' accettare una cosa del genere li toglie e lascia un solo elemento. Quindi la risposta e' che il numero di tuple sicuramente non puo' essere maggiore, ma potrebbe anche essere minore se dovessimo togliere alcune tuple duplicate nel risultato.

Quindi ricapitolando una proiezione:

- Contiene al più tante ennuple quante l'operando
- Può contenerne di meno

Inoltre sappiamo che se X è una superchiave di R, allora $PROJ_X(R)$ contiene esattamente tante ennuple quante R (fare la proiezione su una chiave ci restituisce sempre lo stesso numero di ennuple dello schema originale).

Selezione e proiezione

Combinando selezione e proiezione, possiamo estrarre interessanti informazioni da una relazione. Ad esempio vogliamo sapere matricola e congome degli impiegati che guadagnano più di 50:

Matricola	Cognome
7309	Rossi
5998	Neri
5698	Neri

PROJ$_{\text{Matricola,Cognome}}$ (SEL$_{\text{Stipendio > 50}}$ (Impiegati))

Combinando selezione e proiezione possiamo estrarre informazioni da una relazione. Non possiamo però correlare informazioni presente in relazioni diverse, nè informazioni in ennuple diverse di una stessa relazione. Ci puo capitare di provare a confrontare valori su tuple diverse, è quindi una selezione in un'unica tabella, non possiamo farlo solo con la selezione e la proiezione. Se vogliamo confrontare tuple diverse della stessa tabella abbiamo bisogno del join, in questo caso della stessa tabella con se stessa.

7.1.8 Join

Il join è l'operatore più interessante dell'algebra relazionele poichè permette di correlare dati in relazioni diverse. Supponendo che il nostro database sia di due tabelle voto e candidato e l'interrogazione è di combinare le due tabelle, per farlo combino le tuple che hanno lo stesso numero.

Numero	Voto
1	25
2	13
3	27
4	28

Numero	Candidato
1	Mario Rossi
2	Nicola Russo
3	Mario Bianchi
4	Remo Neri

Numero	Candidato	Voto
1	Mario Rossi	25
2	Nicola Russo	13
3	Mario Bianchi	27
4	Remo Neri	28

Join Naturale

Ci sono in reltà diversi tipi di join, che non è più un operatore monadico ma binario, il primo tipo si chiama join naturale, perchè fa quello che ci è venuto naturale, guardato una colonna che ha lo stesso nome di attributo e confrontato i valori con la colonna con lo stesso nome nell'altra tabella. E' generalizzabile perchè ne posso prendere anche più di 2 tabelle e fare il join tutto insieme, ma vedremo che è equivalente a farne due alla volta (per la proprietà associativa). Lo schema risultante è l'unione degli attributi dei due insiemi operandi, le tuple risultanti sono costruite a partire da una ennupla di uno.

Dare $R_1(X_1), R_2(X_2)$ $R_1 JOIN R_2$ è una relazione su X_1, X_2 così definita:

$$\{ \text{ t su } X_1X_2 | t[X_1] = t_1 \text{ e } t[X_2] = t_2 \}$$

Un join è completo quando tutte le tuple partecipano a costruire il risultato.

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

Reparto	Capo
A	Mori
B	Bruni

Impiegato	Reparto	Capo
Rossi	A	Mori
Neri	B	Bruni
Bianchi	B	Bruni

Figure 7.1: Join completo

E' possibile avere un join incompleto quando ci sono tuple che non partecipano.

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

Reparto	Capo
B	Mori
C	Bruni

Impiegato	Reparto	Capo
Neri	B	Mori
Bianchi	B	Mori

Figure 7.2: Join non completo

E' inoltre possibile avere un join vuoto, ovvero che nessuna tupla partecipa, quindi la cardinalità minima puo essere 0.

La cardinalità massima è nxm. Quindi se le tabelle hanno molte righe puo essere molto costoso.

7.1.9 Cardinalità del join

Se il join coinvolge (gli attributi in comune tra le due tabelle contengono una chiave di R_2) può essere 0 come minimo o il massimo di tuple che ci sono nella prima tabella (la cardinalità di $|R_1|$).

Se facciamo un join e quello su cui lo facciamo è una chiave di R2 sappiamo già che le tuple al massimo non possono essere più di R1 ma se so anche che c'è un vincolo di integrità relazionale tra la colonna della prima e la colonna della seconda relazione allora posso dire che sono esattamente R1. Significa che per ogni tupla ho la certezza che si combina con qualche cosa e quella cosa è una chiave ho come risultato il numero di tuple che c'è nella prima tabella.

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

Reparto	Capo
B	Mori
C	Bruni

Impiegato	Reparto	Capo
Neri	B	Mori
Bianchi	B	Mori

7.1.10 Join esterno

Nel risultato le tuple che non partecipano vengono perse. In qualche caso potrebbe non andare bene, quindi si introduce una nozione. Il join esterno estende con valori nulli le tuple che verrebbero escluse con un join interno. Esistono tre tipi di join esterno:

- Sinistro, $JOIN_{LEFT}$: tengo le tuple della tabella di sinistra ma ignoro quelle della tabella di destra
- Destra $JOIN_{RIGHT}$: tengo le tuple della tabella di destra (anche mettendo i valori null) ma ignoro quella di sinistra
- Completo $JOIN_{vull}$: tengo le tupe (con i valori null) di tutte e due le tabelle

Impiegati

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

Reparti

Reparto	Capo
B	Mori
C	Bruni

Impiegati $\text{JOIN}_{\text{LEFT}}$ Reparti

Impiegato	Reparto	Capo
Neri	B	Mori
Bianchi	B	Mori
Rossi	A	NULL

Figure 7.3: Join Sinistro

Impiegati

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

Reparti

Reparto	Capo
B	Mori
C	Bruni

Impiegati $\text{JOIN}_{\text{RIGHT}}$ Reparti

Impiegato	Reparto	Capo
Neri	B	Mori
Bianchi	B	Mori
NULL	C	Bruni

Figure 7.4: Join Destro

Impiegati

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

Reparti

Reparto	Capo
B	Mori
C	Bruni

Impiegati $\text{JOIN}_{\text{FULL}}$ Reparti

Impiegato	Reparto	Capo
Neri	B	Mori
Bianchi	B	Mori
Rossi	A	NULL
NULL	C	Bruni

Figure 7.5: Join Completo

7.1.11 Semi-Join

Non ci chiederà di farlo, ma per completezza. Restituisce le tuple di R1 che parteciperebbero al join naturale di R2. E' come se facessi il join naturale e poi proietto su R1 e tengo solo le tuple che parteciperebbero.

E' quindi una relazione su X_1, X_2 così definita:

$$\{ \text{ t su } X_1X_2 | t \in R_1 \text{ e siste } t_2 \in R_2 \text{ con } t[X_1 \cap X_2] = t_2[X_1 \cap X_2]\}$$

Impiegati

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

Reparti

Reparto	Capo
B	Mori
C	Bruni

Impiegati SEMIJOIN Reparti

Impiegato	Reparto
Neri	B
Bianchi	B

7.1.12 Join ennario

Avendo la proprietà associativa possiamo fare il join tra i primi due e poi fare il join tra il risultato ed R3, quindi non c'è bisogno di imparare join diversi da quelli binari.

7.1.13 Join e proiezioni

Riflettiamo un attimo su quello che succede quando facciamo delle proiezioni e dei join. Facendo prima il join naturale e poi facendo la proiezione sulle tabelle otteniamo un sottoinsieme delle tabelle originali. Avviene questo perchè abbiamo escluso delle tuple (il join non era completo).

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

Reparto	Capo
B	Mori
C	Bruni

Impiegato	Reparto	Capo
Neri	B	Mori
Bianchi	B	Mori

Impiegato	Reparto
Neri	B
Bianchi	B

Reparto	Capo
B	Mori

Se adesso partiamo da una tupla che potrebbe essere il risultato di un join tra due tabelle e consideriamo la sua proiezione su ImpiegatoReparto e poi su RepartoCapo, se adesso faccio il join delle proiezioni, notiamo che la cardinalità della relazione che risulta dal join di queste proiezioni è maggiore di quella dei partenza. In generale dobbiamo sapere che non è così facile se partiamo da una tabella e facciamo delle proiezioni ricostruire l'informazione originale con un join.

Impiegato	Reparto	Capo
Neri	B	Mori
Bianchi	B	Bruni
Verdi	A	Bini

Impiegato	Reparto
Neri	B
Bianchi	B
Verdi	A

Reparto	Capo
B	Mori
B	Bruni
A	Bini

Impiegato	Reparto	Capo
Neri	B	Mori
Neri	B	Bruni
Bianchi	B	Mori
Bianchi	B	Bruni
Verdi	A	Bini

7.1.14 Prodotto cartesiano

Lo otteniamo se facciamo un join su relazioni senza attributi in comune. Come fare il join se non c'è lo stesso nome tra due colonne.

Contiene sempre lo stesso numero di tule pari al prodotto delle cardinalità degli operandi (le ennuple sono tutte combinabili, costa sempre nxm)

Impiegati

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

Reparti

Codice	Capo
A	Mori
B	Bruni

Impiegati JOIN Reparti

Impiegato	Reparto	Codice	Capo
Rossi	A	A	Mori
Rossi	A	B	Bruni
Neri	B	A	Mori
Neri	B	B	Bruni
Bianchi	B	A	Mori
Bianchi	B	B	Bruni

Siccome gli attributi sono disgiunti, prendo la prima tupla e la metto con tutte le altre (dell'altra tabella) senza controllare valori eventuali valori uguali.

Il prodotto cartesiano nella pratica ha senso solo se seguito da una selezione (dobbiamo scremare e tenere solo le tuple che ci interessano), nel nostro esempio ci potrebbero interessare solo le tuple in cui Reparto = Codice.

$$SEL_{Condizione}(R_1 JOIN R_2)$$

Questa operazione di selezione e prodotto cartesiano si sostituisce con un operatore specifico il theta-join (si chiama theta perchè posso metterci operazioni diverse dall'uguaglianza, ma posso metterci qualsiasi equazione booleana).

$$R_1 JOIN_{Condizione} R_2$$

7.1.15 equi-join

E' il join con l'uguaglianza. Diciamo equi-join quell'operatore tra due relazioni con un uguaglianza tra due attributi delle due tabelle. Capiamo quindi che la scelta dei nomi nella progettazione concettuale è fondamentale per chi fa le interrogazioni per capire su cosa fare i join in modo che sia semanticamente corretto.

Impiegati

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

Reparti

Codice	Capo
A	Mori
B	Bruni

Impiegati $JOIN_{Reparto=Codice}$ Reparti

Impiegato	Reparto	Codice	Capo
Rossi	A	A	Mori
Neri	B	B	Bruni
Bianchi	B	B	Bruni

7.2 Esempi

Vediamo degli esempi di equazioni di algebra relazionale a partire da queste tabelle Impiegati e Supervisione

Impiegati

Matricola	Nome	Età	Stipendio
7309	Rossi	34	45
5998	Bianchi	37	38
9553	Neri	42	35
5698	Bruni	43	42
4076	Mori	45	50
8123	Lupi	46	60

Supervisione

Impiegato	Capo
7309	5698
5998	5698
9553	4076
5698	4076
4076	8123

7.2.1 1

Trovare matricola, nome, età e stipendio degli impiegati maggiore di 40.

In questo caso semplicemente dobbiamo selezionare dalla tabella Impiegati chi ha lo stipendio maggiore di 40.

$$SEL_{Stipendio>40}(Impiegati)$$

7.2.2 2

Trovare matricola, nome ed età degli impiegati che guadagnano più di 40. Si tratta inizialmente di selezionare dalla tabella impiegati gli impiegati che hanno stipendio maggiore di quaranta e poi di fare delle proiezioni sugli attributi che ci interessano.

$$PROJ_{Matricola,Nome,Eta}(SEL_{Stipendio>40}(Impiegati))$$

7.2.3 3

Trovare i capi degli impiegati che guadagnano più di 40. La prima cosa da fare è capire quali tabelle sono coinvolte, dato che si parla dei capi ci sarà la tabella Supervisione e dato che ci serve lo stipendio ci serve anche la prima tabella Impiegati. Dato che dobbiamo confrontare i valori in due tabelle sappiamo quindi che dobbiamo fare un join, su cosa fare un join però. Sarà un equi-join e la condizione di uguaglianza che vogliamo fare, dato che non ci sono colonne con nome in comune, dobbiamo sicuramente fare un join matricola = o impiegato o capo. Siccome mi si chiede di cercare i capi farò impiegato = matricola. Quindi dice questo join, di combinare le tuple della seconda relazione, che si combinano con la matricola di quelli che abbiamo già selezionato. Notiamo che abbiamo fatto nel join impiegato=Matricola perchè nella tabella risultante vogliamo le informazioni dell'impiegato, perchè sto selezionando gli impiegati che guadagnano più di 40, non i capi. Ottengo dopo i join una tabella di 6 colonne, ma dato che mi interessano solo i capi faccio una proiezione su Capo.

$$PROJ_{Capo}(SupervisioneJOIN_{Impiegato=Matricola}(SEL_{Stipendio>40}(Impiegati)))$$

7.2.4 4

Trovare nome e stipendio dei capi degli impiegati che guadagnano più di 40. Chiede la stessa cosa di prima ma oltre alla matricola vuole nome e stipendio dei capi. Devo in qualche modo andare a leggere i dati della tabella impiegati corrispondenti a quelle due matricole (dei capi) che sono il risultato di una tabella. Per trovare i dati dei capi (dopo aver trovato la loro matricola dalla tabella risultante dalla proiezione capo) devo fare un join con impiegato di quella tabella della matricola del capo e successivamente fare una proiezione, in quanto ci interessano solo nome e stipendio.

$$PROJ_{Nome,Stipendio}$$
$$(ImpiegatiJOIN_{Matricola=Capo}$$
$$PROJ_{Capo}(SupervisioneJOIN_{Impiegato=Matricola}(SEL_{Stipendio>40}(Impiegati))))$$

7.2.5 5

Trovare gli impiegati che guadagnano più del proprio capo, mostrando matricola, nome e stipendio dell'impiegato e del capo. Alla fine ci sarà quindi una tabella con 6 colonne. Qui vediamo un esempio in cui dovremo confrontare due valori di stipendio che stanno in due tuple diverse della stessa tabella, sappiamo subito che dovremo fare un join e per confrontare due tuple nella stessa tabella dovremo fare un join con la tabella stessa. Devo inoltre devo confrontare lo stipendio di un impiegato con lo stipendio specifico del proprio capo.

L'ultima sotto espressione cerca di capire chi è il capo dell'impiegato, creo quindi una tabella in cui ci sono tutti i dati di ciascun impiegato e nella stessa riga c'è anche la matricola del suo capo. A questo punto dato che ho capito chi è il capo devo confrontare lo stipendio di uno e dell'altro, ma per farlo devo fare un join con la tabella Impiegati stessi, in quanto non mi basta la matricola del capo, ma devo trovare lo stipendio del capo, andando a cercare la matricola del capo. Qui vediamo l'operatore di ridenominazione, perché usando gli stessi nomi mi incasino, quando trovo i dati dei capi meglio cambiare il nome delle colonne. La tabella dopo la ridenominazione è una tabellona di 10 colonne, in cui ci sono tutti i dati di un impiegato e del suo capo. Dopo di che seleziono da questa tabella solo le tuple in cui lo stipendio dell'impiegato è maggiore dello stipendio del proprio capo. Ed alla fine ritorno la tabella facendo la proiezione delle colonne che mi interessano.

$$
\begin{aligned}
&PROJ_{Matricola,Stip,MatrC,NomeC,StipC}\\
&(SEL_{Stipendio>StipC}\\
&(REN_{MatrC,NomeC,EtaC,StipC\leftarrow Matr,Nome,Eta,Stip}(Impiegati)\\
&(SupervisioneJOIN_{Impiegato=Matricola}Impiegati)))
\end{aligned}
$$

In generale ricordiamoci che quando usiamo un join immaginiamoci il risultato, facendo quindi un esempio e vediamo subito che schema esce dal join, per controllare che stiamo facendo l'operazione corretta. Poi si va per raffinazioni successive.

7.2.6 6

Trovare le matricole dei capi i cui impiegati guadagnano TUTTI più di 40. Risolveremo questa interrogazione usando l'operatore di differenza insiemistica.

$$
\begin{aligned}
&PROJ_{Capo}(Supervisione)-\\
&PROJ_{Capo}(SupervisioneJOIN_{Impiegato=Matricola}\\
&(SEL_{Stipendio\leq 40}(Impiegati)))
\end{aligned}
$$

Quel tutti è importante perchè un capo potrebbe avere un impiegato che guadagna più di 40 ma anche uno che guadagna meno di 40. L'idea è di ragionare al contrario, questo tutti equivale ad una quantificazione universale e si

può risolvere ragionando al contrario, se trovo capi che non soddisfano questa condizione? Se un capo ha anche un solo impiegato che guadagna meno di 40 non deve stare nel risultato. Quindi prendo tutti i capi e gli devo togliere quelli che hanno almeno un impiegato che guadagna meno di 40. La seconda parte di fatto l'abbiamo già fatta, dobbiamo solo cambiare la condizione di selezione, internamente dalla tabella impiegati toglie tutte le tuple di quelli che guadagnano meno di 40, poi faccio il join con supervisione per capire quali sono i loro capi, poi, dato che la differenza vuole due tabelle con lo stesso schema da una parte e dall'altra, quindi faccio una proiezione su capo (prendo quindi solo la colonna dei capi) e li tolgo da tutti i capi. In questo modo mi rimane una tabella in cui ci sono i capi i cui impiegati guadagnano tutti più di 40.

C'è un problema in questa soluzione nel caso in cui ci siano dei NULL negli stipendi, dovrei fare un OR nella select, OR stipendio=NULL.

7.3 Equivalenza di espressioni

In generale ci sono delle espressioni algebriche equivalenti se producono lo stesso risultato qualunque sia l'istanza attuale della base di dati. L'equivalenza è importante in pratica perchè i DBMS cercano di eseguire espressioni equivalenti a quelle date, ma meno costose.

7.3.1 Equivalenza importante

Supponiamo che A sia un attributo di R2 e qui stiamo facendo un join tra la seconda relazione ed un altra, ma poi mi interessano solo le tuple in cui A=10. In pratica vado a togliere tutti gli attributi dal join che non soddisfano la condizione. Ed è equivalente che selezionare prima tutte le tuple che mi servono. Se facciamo prima il join combiniamo prima le tabelle e può essere molto costoso se le tabelle sono grosse, ma se facciamo prima la selezione riduciamo la dimensione della taballa intermedia. Quindi quando si può facciamo prima la selezione e poi il join.

$$SEL_{A=10}(R_1 JOIN R_2)$$
$$=$$
$$R_1 JOIN SEL_{A=10}(R_2)$$

7.4 Selezioni con i valori nulli

Se faccio una selezione su impiegati sull'attributo età che potrebbe avere valori nulli, mi trovo solo come risultato gli impiegati che hanno un attributo per il valore età. Nei miei ragionamenti potrei assumere che se facciamo l'unione degli impiegati che hanno età maggiore di 30 con quelli minore uguali a 30 non trovo la stessa tabella di partenza, perchè comunque non trovo quelli con età = NULL.

Nonostante facciamo un'unica espressione booleana non troveremmo la stessa tabella, perchè sono operazioni atomiche che vengono valutate una alla volta.

Per prendere anche le tuple con valore nullo dovremmo anche mettere IS NULL.

Impiegati

Matricola	Cognome	Filiale	Età
7309	Rossi	Roma	32
5998	Neri	Milano	45
9553	Bruni	Milano	NULL

$SEL_{Età > 40}$ (Impiegati)

7.5 Viste

Sono delle rappresentazioni diverse per gli stessi dati (schema esterno).

Relazioni di base sono quelle che effettivamente stanno nel database. Le relazioni derivate sono relazioni il cui contenuto è funzione del contenuto di altre relazioni (definito per mezzo di interrogazioni) Da relazioni derivate posso crearne altre derivate.

Afferenza

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

Direzione

Reparto	Capo
A	Mori
B	Bruni

7.5.1 Viste virtuali e materializzate

Nella maggior parte dei casi queste viste non sono memorizzate, e si chiamano relazioni virtuali. Quelle che invece vengono memorizzate si chiamano viste materializzate.

Supervisione materializzata vuol dire che pre-calcolo il risultato dell'operazione (la tabella risultante) e la memorizzo nel database. Se invece uso la supervisione come vista virtuale quando qualcuno usa la tabella supervisione calcolo il risultato dell'espressione.

In quelle memorizzate ho i valori già lì e risparmio di doverli calcolare. Lo svantaggio è che se cambio spesso i valori nella vista tutte le volte che cambio i valori devo aggiornare la tabella. Alcuni DBMS supportano viste materializzate, ma la maggior parte quando parlano di viste sono virtuali.

7.5.2 Motivazioni

Quando programmiamo non è tanto semplice scrivere delle espressioni lunghe, ma è comodo usare delle viste in modo che le interrogazioni diventi più semplice.

Definiamo una sottoespressione con un nome (una vista) e poi usiamo quel nome in tutte le altre interrogazioni.

Il problema delle viste è l'aggiornamento.

7.6 Convenzione e notazione alternativa per i join

Rispetto alla notazione usata fino ad adesso ce n'è un'altra che sarà più comoda in SQL. Una possibilità è fare assegnamento Capi := Imp, uso un nuovo nome per una nuova tabella, che è però una copia della tabella Impiegati, in questo modo farò il join tra le tabelle capi e la seconda senza confondermi. Gli attributi diventano Capi.matricola (Capi.nomeAttributo), quello che faremo in SQL Impiegati AS Capi.

Chapter 8

Calcolo relazionale e datalog

Una famiglia di linguaggi dichiarativi, basati sul calcolo dei predicati del primo ordine. Diverse versioni:

- Calcolo relazionale su domini (Domain Relational Calculus - DRC)
- calcolo su ennuple con dichiarazioni di range (Tple Relational Calculus)

E' un altro linguaggio che sta alla base dell'SQL (all'esame non avremo interrogazioni con il calcolo relazionale, ma c'e' sempre una domanda nei quiz). Mentre l'algebra e' un linguaggio procedurale il calcolo fa parte di una serie di linguaggi che si chiamano dichiarativi, dichiarativo vuol dire che non dobbiamo spiegare esattamente come si raggiunge il risultato ma bensi' dobbiamo descrivere la funzione da calcolare piuttosto che come calcolarla.

8.1 Calcolo relazionale su Domini

Le espressioni hanno forma:

$$\{A_1 : x_1, ..., A_k : x_k | f\}$$

- f e' una formula (con connettivi booleani AND, OR, NOT e quantificatori esistenziali e universali).
- La prima parte si chiama target list, di fatto qui mettiamo il nome degli attributi delle colonne che vogliamo come risultato.
  - A_1, ..., A_k sono attributi distinti (anche non nella base di dati)
  - x_1, ..., x_k sono variabili distinte.
- Il risultato e' una relazione su A_1, ..., A_k che contiene ennuple per x_1, ..., x_k che rendono vera la formula f. L'insieme e' quello che mettiamo nella target list, che contiene tuple di valori per x1, xk che rendono vera la formula f

Utilizzeremo la seguente base di dati per vedere alcuni esempi:

$$Impiegati(\underline{Matricola}, Nome, Eta, Stipendio)$$
$$Supervisione(Capo, \underline{Impiegato})$$

8.1.1 Esempio 0

Trovare matricola, nome, eta' e stipendio degli impiegati che guadagnano piu' di 40.

La soluzione in algebra relazionale:

$$SEL_{Stipendio>40}(Impiegati)$$

Mentre la soluzione in calcolo relazionale:

$$\{Matricola : m, Nome : nEta : e, Stipendio : s|$$
$$Impiegati(Matricola : m, Nome : n, Eta : e, Stipendio : s)ANDs > 40.\}$$

8.1.2 Esempio 0b

Trovare matricola, nome ed età di tutti gli impiegati.

$$PROJ_{Matricola,\ Nome,\ Età}(Impiegati)$$

{ Matricola: m, Nome: n, Età: e |
EXISTS s (Impiegati(Matricola: m, Nome: n, Età: e, Stipendio: s)}

{ Matricola: m, Nome: n, Età: e |
Impiegati(Matricola: m, Nome: n, Età: e, Stipendio: s)}

8.1.3 Esempio 1

Trovare matricole, nome ed eta' degli impiegati che guadagnano piu' di 40:

$$PROJ_{Matricola,\ Nome,\ Età}(SEL_{Stipendio>40}(Impiegati))$$

{ Matricola: m, Nome: n, Età: e |
Impiegati(Matricola: m, Nome: n, Età: e, Stipendio: s) AND s > 40 }

8.1.4 Esempio 2

Trovare le matricole dei capi degli impiegati che guadagnano piu' di 40:

$$\mathrm{PROJ}_{\mathrm{Capo}}\,(\mathrm{Supervisione\ JOIN}_{\mathrm{Impiegato=Matricola}}\,(\mathrm{SEL}_{\mathrm{Stipendio>40}}(\mathrm{Impiegati})))$$

{ Capo: c | Supervisione(Capo:c,Impiegato:m) AND
Impiegati(Matricola: m, Nome: n, Età: e, Stipendio: s) AND s > 40 }

8.1.5 Esempio 3

Trovare nome e stipendio dei capi degli impiegati che guadagnano piu' di 40:

$$\mathrm{PROJ}_{\mathrm{NomeC,StipC}}\,(\mathrm{REN}_{\mathrm{MatrC,NomeC,StipC,EtàC\leftarrow Matr,Nome,Stip,Età}}(\mathrm{Impiegati})\ \mathrm{JOIN}_{\mathrm{MatrC=Capo}}\ (\mathrm{Supervisione\ JOIN}_{\mathrm{Impiegato=Matricola}}\,(\mathrm{SEL}_{\mathrm{Stipendio>40}}(\mathrm{Impiegati}))))$$

{ NomeC: nc, StipC: sc |
Impiegati(Matricola: m, Nome: n, Età: e, Stipendio: s) AND
s > 40 AND
Supervisione(Capo:c,Impiegato:m) AND
Impiegati(Matricola:c, Nome:nc, Età:ec, Stipendio: sc) }

8.1.6 Esempio 4

Trovare gli impiegati che guadagnano piu' del rispettivo capo, mostrando matricola, nome e stipendio di ciascuno di essi e del capo:

$$\text{PROJ}_{\text{Matr,Nome,Stip,MatrC,NomeC,StipC}}$$
$$(\text{SEL}_{\text{Stipendio>StipC}}(\text{REN}_{\text{MatrC,NomeC,StipC,EtàC} \leftarrow \text{Matr,Nome,Stip,Età}}(\text{Impiegati})$$
$$\text{JOIN}_{\text{MatrC=Capo}}$$
$$(\text{Supervisione JOIN}_{\text{Impiegato=Matricola}} (\ (\text{Impiegati}))))$$

$$\{\ \text{Matr: m, Nome: n, Stip: s, MatrC: c, NomeC: nc, StipC: sc} \mid$$
$$\text{Impiegati(Matricola: m, Nome: n, Età: e, Stipendio: s) AND}$$
$$\text{Supervisione(Capo:c,Impiegato:m) AND}$$
$$\text{Impiegati(Matricola: c, Nome: nc, Età: ec, Stipendio: sc) AND s > sc}\}$$

8.1.7 Esempio 5

Trovare matricola e nome dei capi i cui impiegati guadagnano tutti piu' di 40:

$$\text{PROJ}_{\text{Matricola,Nome}}\ (\text{Impiegati JOIN}_{\text{Matricola=Capo}}$$
$$(\text{PROJ}_{\text{Capo}}\ (\text{Supervisione}) -$$
$$\text{PROJ}_{\text{Capo}}\ (\text{Supervisione JOIN}_{\text{Impiegato=Matricola}}\ ($$
$$\text{SEL}_{\text{Stipendio} \leq 40}(\text{Impiegati}))))$$

$$\{\text{Matricola: c, Nome: n} \mid$$
$$\text{Impiegati(Matricola: c, Nome: n, Età: e, Stipendio: s) AND}$$
$$\text{Supervisione(Capo:c, Impiegato:m) AND}$$
$$\text{NOT EXISTS m'(EXISTS n'(EXISTS e' (EXISTS s' (Impiegati(Matr: m', Nome:}$$
$$\text{n', Età: e', Stip: s') AND}$$
$$\text{Supervisione(Capo:c, Impiegato:m') AND s'} \leq 40))))\}$$

8.2 Quantificatori

Una formula logica con FORALL si puo' tradurre in una equivalente che utilizza NOT e EXISTS (e viceversa). Queste regole ci spiegano che in un linguaggio non e' necessario averli tutti e due come nativi, ma basta averne uno e gli altri si derivano.

$$FORALLx(P(x)) = NOTEXISTSx(NOT(P(x))$$
$$EXISTSx(P(x)) = NOTFORALLx(NOT(P(x))$$

8.2.1 Espressioni equivalenti

```
{Matricola: c, Nome: n |
Impiegati(Matricola: c, Nome: n, Età: e, Stipendio: s) AND
Supervisione(Capo:c, Impiegato:m) AND
NOT EXISTS m'(EXISTS n'(EXISTS e'(EXISTS s'(Impiegati(Matr: m', Nome: n', Età: e', Stip: s')
AND
Supervisione(Capo:c, Impiegato:m') AND s' ≤ 40))))}

{Matricola: c, Nome: n |
Impiegati(Matricola: c, Nome: n, Età: e, Stipendio: s) AND
Supervisione(Capo:c, Impiegato:m) AND
FORALL m'(FORALL n'(FORALL e'(FORALL s'(NOT(Impiegati(Matr:m', Nome:n', Età:e', Stip:s')
AND
Supervisione(Capo:c, Impiegato:m')) ∨ s' > 40))))}
```

8.2.2 Riflessioni

I pregi del calcolo sui domini sono la dichiarativita', i difetti sono la verbosita' e la presenza di espressioni che dipendono dal dominio, ad esempio tutti i nomi che non sono di santi, dipende da che religione stiamo considerando. Quindi queste espressioni non ci piacciono. Quindi vogliamo qualcosa che abbia meno variabili e poi possibilmente vogliamo restringere il dominio.

8.3 Calcolo relazionale sulle tuple

L'idea e' di mettere una variabile per ogni tupla, con espressione del tipo $\{t|Formula\}$. Le variabili sono di tupla e non di attributo, quindi uso una sola variabile per la tupla e per riferirmi agli attributi si usa la notazione puntata, t.Stipendio.

L'unione nel calcolo delle tuple e':

$$\{t|R_1(t)ORR_2(t)\}$$

8.3.1 Esempio 0a

Trovare matricola, nome, eta' e stipendio degli impiegati che guadagnano piu' di 40:

$$SEL_{Stipendio>40}(Impiegati)$$

{ Matricola: m, Nome: n, Età: e, Stipendio: s |
Impiegati(Matricola: m, Nome: n, Età: e, Stipendio: s) AND s > 40 }

{ t | Impiegati(t) AND t.Stipendio > 40 }

8.3.2 Esempio 0b

Trovare matricola, nome ed eta' di tutti gli impiegati

$$PROJ_{Matricola,\ Nome,\ Età}(Impiegati)$$

{ Matricola: m, Nome: n, Età: e |
Impiegati(Matricola: m, Nome: n, Età: e, Stipendio: s)}

{ t.Matricola, t.Nome, t.Età | Impiegati(t)}

8.3.3 Esempio 1

Trovare matricola, nome ed eta' degli impiegati che guadagnano piu' di 40:

$$PROJ_{Matricola,\ Nome,\ Età}(SEL_{Stipendio>40}(Impiegati))$$

{ Matricola: m, Nome: n, Età: e |
Impiegati(Matricola: m, Nome: n, Età: e, Stipendio: s) AND s > 40 }

{t.Matricola, t.Nome, t.Età | Impiegati (t) AND t.Stipendio > 40 }

8.3.4 Esempio 2

Trovare le matricole dei capi degli impiegati che guadagnano piu' di 40:

```
{ Capo: c | Supervisione(Capo:c,Impiegato:m) AND
Impiegati(Matricola: m, Nome: n, Età: e, Stipendio: s) AND s > 40 }

{ s.Capo | Impiegati(i) AND Supervisione(s) AND
i.Matricola=s.Impiegato AND i.Stipendio > 40 }
```

8.3.5 Esempio 3

Trovare nome e stipendio dei capi degli impiegati che guadagnano piu' di 40:

```
{ NomeC: nc, StipC: sc |
Impiegati(Matricola: m, Nome: n, Età: e, Stipendio: s) AND s > 40  AND
Supervisione(Capo:c,Impiegato:m) AND
Impiegati(Matricola:c, Nome:nc, Età:ec, Stipendio:sc) }

{ t.NomeC, t.StipC | t.NomeC=i'.Nome AND t.StipC=i'.Stipendio AND
Impiegati(i') AND Supervisione(s) AND Impiegati(i) AND
i'.Matricola=s.Capo AND i.Matricola=s.Impiegato AND i.Stipendio > 40 }
```

8.3.6 Esempio 4

Trovare gli impiegati che guadagnano piu' del rispettivo capo, mostrando matricola, nome e stipendio di ciascuno di essi e del capo:

```
{ Matr: m, Nome: n, Stip: s, NomeC: nc, StipC: sc |
Impiegati(Matricola: m, Nome: n, Età: e, Stipendio: s) AND
Supervisione(Capo:c,Impiegato:m) AND
Impiegati(Matricola: c, Nome: nc, Età: ec, Stipendio: sc) AND s > sc}

{ t.Nome, t.Matr, t.Stipendio, t.NomeC, t.MatrC, t.StipC |
t.Nome=i.Nome AND t.Matr=i.Matr AND t.Stip=i.Stipendio AND
t.NomeC=i'.Nome AND t.MatrC=i'.Matr AND t.StipC=i'.Stipendio AND
Impiegati(i') AND Supervisione(s) AND Impiegati(i) AND
i'.Matricola=s.Capo AND i.Matricola=s.Impiegato AND i.Stipendio > i'.Stipendio }
```

8.3.7 Esempio 5

Trovare matricola e nome dei capi i cui impiegati guadagnano tutti piu' di 40:

```
{Matricola: c, Nome: n |
Impiegati(Matricola: c, Nome: n, Età: e, Stipendio: s) AND
Supervisione(Capo:c, Impiegato:m) AND
NOT EXISTS m',n',e',s' (Impiegati(Matr: m', Nome: n', Età: e', Stip: s') AND
Supervisione(Capo:c, Impiegato:m') AND s' ≤ 40}

{ i.Matricola, i.Nome) | Supervisione(s) AND Impiegati(i) AND
i.Matricola=s.Capo AND NOT (EXISTS i',s' (Impiegati(i') AND Supervisione(s')
AND s.Capo=s'.Capo AND s'.Impiegato=i'.Matricola AND i'.Stipendio ≤ 40)}
```

8.3.8 Espressioni safe'

Un'espressione e' safe se garantisce di restituire un numero finito di tuple, altrimenti e' unsafe. Ad esempio $t|NOTImpiegati(t)$ e' unsafe, perche' prene tutte le tuple che non stanno negli impiegati di t, quindi potenzialmente sono infinite.

8.4 Calcolo e algebra

Sono linguaggi che ci preparano a fare interrogazioni SQL. Ma se dovessimo confrontarli per capire quali sono i piu' espressivi bisogna mettere delle assunzioni. Se consideriamo solo le espressioni safe/indipendenti dal dominio, calcolo e algebra sono equivalenti, il che vuol dire che per ogni espressione del calcolo relazionale esiste un'espressione dell'algebra equivalente, vale il duale.

8.4.1 Limiti

Tutti i valori che restituiamo sono valori che stanno nel database, in realta' puo' essere utile calcolare valori, ad esempio potrei volere l'eta' media degli impiegati o lo stipendio massimo (saranno possibili in SQL). Inoltre non possiamo fare interrogazioni inerentemente ricorsive, come la chiusura transitiva.

Chiusura transitiva

Abbiamo la relazione supervisione, di fatto se voglio trovare tutti i superiori (il capo del capo). Come risultato dell'interrogazione devi darmi tutti i superiori di Rossi. Siccome devo confrontare delle tuple diverse della stessa tabella devo usare dei Join. Non esiste in algebra relazionale un modo per fare questa operazione.

Impiegato	Capo
Rossi	Lupi
Neri	Bruni
Lupi	Falchi

Impiegato	Superiore
Rossi	Lupi
Neri	Bruni
Lupi	Falchi
Rossi	Falchi

Impiegato	Capo
Rossi	Lupi
Neri	Bruni
Lupi	Falchi
Falchi	Leoni

Impiegato	Superiore
Rossi	Lupi
Neri	Bruni
Lupi	Falchi
Falchi	Leoni
Rossi	Falchi
Lupi	Leoni
Rossi	Leoni

Non esiste in algebra e calcolo relazionale la possibilta' di esprimere l'interrogazione che, per ogni relazione binaria, ne calcoli la chiusura transitiva. Per ciascuna relazione, e' possibile calcolare la ciusura transitiva, ma con un'espressione ogni volta diversa, quanti join servono? non c'e' un limite.

8.5 Datalog

E' un linguaggio di programmazione logica per basi di dati derivato dal Prolog. Che e' un linguaggio di interrogazione basato sulla programmazione logica. Usa predicati di due tipi:

- Estensionali: relazioni della base di dati
- Intensionali: corrispondono alle viste

Il linguaggio e' basato su regole utilizzate per "definire" i predicati intensionali.

8.5.1 Sintassi

Un'interrogazione e' fatta da piu' espressioni. Che sono fatte cosi': testa $\leftarrow$ corpo.

- Testa e' un atomo (ristretto a predicato intensionale)
- Corpo e' una lista di atomi interpretata come AND logico

Un atomo e' un predicato (intensionale o estensionale) con variabili o costanti. Le interrogazioni sono specificate per mezzo di predicati atomici (convenzionalmente preceduti da "?").

Per interpretare la regola, la testa della regola e' vera per i valori assunti dalle variabili che vi compaiono SE esistono valori per le ALTRE variabili che compaiono nel corpo che rendono 'vero' ciascun elemento (sottogol) che appare nel corpo della regola.

8.5.2 Vincoli

I predicati estensionali (relazioni del DB) sono nel corpo della regola. Se una variabile compare nella testa, allora deve comparire anche nel corpo, $P(x) \leftarrow Q(y)$ viola questo vincolo. Se una variabile appare in un confronto, allora deve comparire anche in un atomo nel corpo della stessa regola, $P(x) \leftarrow Q(x,'2')ANDx < y$ viola questo vincolo.

8.5.3 Esempio -1

Trovare matricola, nome, eta' e stipendio degli impiegati che hanno 30 anni:

```
{ Matricola: m, Nome: n, Età: e, Stipendio: s |
Impiegati(Matricola: m, Nome: n, Età: e, Stipendio: s) AND e = 30 }

? Impiegati(Matricola: m, Nome: n, Età: 30, Stipendio: s)
```

8.5.4 Esempio 0a

Trovare matricola, nome, eta' e stipendio degli impiegati che guadagnano piu' di 40. Serve un predicato intensionale:

```
ImpRicchi(Matricola: m, Nome: n, Età: e, Stipendio: s) <-
                    Impiegati(Matricola: m, Nome: n, Età: e, Stipendio: s) , s >40

? ImpRicchi(Matricola: m, Nome: n, Età: e, Stipendio: s)
```

8.5.5 Esempio 0b

Trovare matricola, nome ed eta' di tutti gli impiegato:

$$PROJ_{Matricola,\ Nome,\ Età}(Impiegati)$$

```
{ Matricola: m, Nome: n, Età: e |
Impiegati(Matricola: m, Nome: n, Età: e, Stipendio: s)}

InfoPubbliche(Matricola: m, Nome: n, Età: e)
   <- Impiegati(Matricola: m, Nome: n, Età: e, Stipendio: s)

? InfoPubbliche(Matricola: m, Nome: n, Età: e)
```

8.5.6 Esempio 2

Trovare le matricole dei capi degli impiegati che guadagnano piu' di 40:

```
{ Capo: c | Supervisione(Capo:c,Impiegato:m) AND
Impiegati(Matricola: m, Nome: n, Età: e, Stipendio: s) AND s > 40 }

CapiDeiRicchi (Capo:c) <-
    ImpRicchi(Matricola: m, Nome: n, Età: e, Stipendio: s), Supervisione(Capo:c,Impiegato:m)

? CapiDeiRicchi (Capo:c)
```

8.5.7 Datalog con negazione

Il Datalog con negazione ammette l'operatore NOT davanti ad un atomo nel corpo della regola. Esempio:

$$P(x) \leftarrow Q(x, y)$$
$$S(x, z) \leftarrow NOTP(x), R(z)$$

8.5.8 Esempio 5

Trovare matricola e nome dei capi i cui impiegati guadagnano tutti piu' di 40. Serve la negazione:

```
CapiDiNonRicchi (Capo:c) <-
        Supervisione (Capo:c,Impiegato:m),
        Impiegati (Matricola: m, Nome: n, Età: e, Stipendio: s) ,
        s ≤ 40
CapiSoloDiRicchi (Matricola: c, Nome: n) <-
        Impiegati (Matricola: c, Nome: n, Età: e, Stipendio: s) ,
        Supervisione (Capo:c,Impiegato:m),
        NOT CapiDiNonRicchi (Capo:c)

? CapiSoloDiRicchi (Matricola: c, Nome: n)
```

8.5.9 Esempio 6: Ricorsione

Per ogni impiegato, trovare tutti i superiori. Serve la ricorsione (lo stesso atomo appare in testa e corpo nella seconda regola):

```
Superiore (Impiegato: i, SuperCapo: c) ←
    Supervisione (Impiegato: i, Capo: c)

Superiore (Impiegato: i, SuperCapo: c) ←
    Supervisione (Impiegato: i, Capo: c'),
    Superiore (Impiegato: c', SuperCapo: c)
```

8.5.10 Espressitiva'

La definizione della semantica delle regole ricorsive e' delicata (in particolare con la negazione). Potere espressivo:

- Datalog non ricorsivo con negazione e' equivalente al calcolo e all'algebra
- Datalog non ricorsivo senza negazione e' equivalente al calcolo senza negazione e senza quantificatore universale

- Datalog ricorsivo senza negazione e calcolo sono incompatibili
- Datalog ricorsivo con negazoine e' piu' espressivo di calcolo e algebra

Chapter 9

SQL

E' un linguaggio che ha radici storiche ma e' utilizzatissimo ancora. SQL contiene sia una parte di DDL (data definition language) e DML (data manipulation language). Esistono tante versioni di SQL ed e' un acronimo di "Structured Query Language".

9.1 Definizione dei dati in SQL

L'istruzione CREATE TABLE ci permette di inserire una tabella nel database. Definisce uno schema di relazione e ne crea un'istanza vuota. Specifica attributi, domini e vincoli.

Vediamone un esempio:

```
CREATE TABLE Impiegato(
    Matricola CHAR(6) PRIMARY KEY,
    Nome CHAR(20) NOT NULL,
    Cognome CHAR(20) NOT NULL,
    Dipart CHAR(15),
    Stipendio NUMERIC(9) DEFAULT 0,
    FOREIGN KEY(Dipart) REFERENCES
        Dipartimento(NomeDip),
    UNIQUE (Cognome,Nome)
)
```

Ci sono una lista di attributi, il dominio (CHAR(6) sono tutte stringhe di 6 caratteri), ci sono i vincoli (PRIMARY KEY, FOREIGN KEY, NOT NULL). Notiamo che in SQL se vogliamo che l'attributo abbia per forza un valore va messo NOT NULL che e' l'equivalente della cardinalita' (1,1) altrimenti se non si mette e' un valore opzionale (0,N). DEFAULT 0 vuol dire che non mette niente il valore specificato e' quello dopo il DEFAULT. UNIQUE invece dice quali sono delle altre chiavi oltre a quella primaria.

9.2 Domini

Esistono dei domini elementari, detti Data Types ed inoltre possiamo anche definire noi i domini.

9.2.1 Domini elementari

- Carattere: singoli caratteri o stringhe, anche di lunghezza variabile
- Numerici, esatti e approssimati
- Data, ora, intervalli di tempo
- Itrodotti in SQL nel 1999:
  - Boolean

– BLOB, CLOB (binary/character large object): per grandi immagini e testi

9.2.2 Definizione dei domini

Potrebbe avere senso definire un nuovo dominio per fare dei controlli particolari sul dominio, ad esempio se definiamo un dominio per i voti validi diciamo che i numeri devono essere compresi tra 18 e 30:

```
CREATE DOMAIN Voto
    AS SMALLINT DEFAULT NOTNUL
    CHECK (VALUE >= 18
           AND VALUE <= 30)
```

9.3 Vincoli intrarelazionali

- NOT NULL
- UNIQUE: definisce chiavi
- PRIMARY KEY: chiave primaria (una sola, implica NOT NULL)
- CHECK, vedremo piu' avanti

9.4 Unique e primary key

Ci sono due forme per definire una chiave, o con una PRIMARY KEY se è una sola stringa, altrimenti se sono più elementi con l'UNIQUE. In generale UNIQUE dato che è una chiave ma non primaria ammette valori nulli, a meno che non sia esplicitato il NOT NULL ed in quel caso non ammette valori nulli.

9.5 Vincoli interrelazionali

Sono quelli di integrità referenziale. REFERENCES e FOREIGN KEY permettono di definire vincoli di integrità referenziale.

Nel primo caso dato che il vincolo referenziale comprende un solo attributo possiamo mettere la dichiarazione di chiave esterna sulla stessa riga.

Infrazioni

Codice	Data	Vigile	Prov	Numero
34321	1/2/95	3987	MI	39548K
53524	4/3/95	3295	TO	E39548
64521	5/4/96	3295	PR	839548
73321	5/2/98	9345	PR	839548

Vigili

Matricola	Cognome	Nome
3987	Rossi	Luca
3295	Neri	Piero
9345	Neri	Mario
7543	Mori	Gino

Infrazioni

Codice	Data	Vigile	Prov	Numero
34321	1/2/95	3987	MI	39548K
53524	4/3/95	3295	TO	E39548
64521	5/4/96	3295	PR	839548
73321	5/2/98	9345	PR	839548

Auto

Prov	Numero	Cognome	Nome
MI	39548K	Rossi	Mario
TO	E39548	Rossi	Mario
PR	839548	Neri	Luca

```
CREATE TABLE Infrazioni(
    Codice CHAR(6) PRIMARY KEY,
    Data DATE NOT NULL,
    Vigile INTEGER NOT NULL
        REFERENCES Vigili(Matricola),
    Provincia CHAR(2),
    Numero CHAR(6),
    FOREIGN KEY(Provincia, Numero),
        REFERENCES Auto(Provincia, Numero)
)
```

Nella seconda dichiarazione di vincolo interrelazionale nella prima tabella sono coinvolti due attributi del vincolo, quindi non posso farlo sulla stessa riga di creazione, ma devo metterlo infondo.

Se lo scrivo sulla stessa riga scrivo REFERENCES, invece se sono coinvolti più di un attributo scrivo FOREIGN KEY la lista degli attributi coinvolti, poi REFERENCES la tabella di destinazione ed i due campi che devono corrispondere. Se non mi ricordo la prima forma posso anche scrivere con il FOREIGN KEY un attributo solo.

9.5.1 Syntax of FK in PGSQL

CONSTRAINT e' opzionale e serve al DBMS per identificare il vincolo che abbiamo esplicitato

```
[CONSTRAINT fk_name]
    FOREIGN KEY(fk_columns)
    REFERENCES parent_table(parent_key_columns)
     [ON DELETE delete_action]
     [ON UPDATE update_action]
```

ON DELETE e' anch'esso opzionale e se non lo mettiamo il sistema assume il delete_action assume il valore no action, inoltre possiamo mettere delle opzioni che possiamo dire al sistema di fare quando qualcuno cancella i dati:

- Mettere i valori a nullo: set null
- Posso mettere cascade come delete action e mi cancella tutta la tupla (e tutte le tuple che hanno FK con l'elemento che sto cancellando) dell'elemento che voglio cancellare

9.6 Modifiche degli schemi

- ALTER DOMAIN
- ALTER TABLE
- DROP DOMAIN
- DROP TABLE

Sono tutte istruzioni per eliminare le tabelle o i domini

9.6.1 ALTER TABLE

Invece di eliminare una tabella (con il DROP) la vorremo modificare, questa modifica. A volte ci viene chiesto di modificare le tabelle per modificare i dati da memorizzare.

```
ALTER TABLE Impiegati
    ADD COLUMN Eta NUMERIC(2)
```

Ricordiamo che uno schema di un database è l'unione di schemi di relazione, quindi modificare uno schema di un database significa modificare uno schema di relazione, quindi significa modificare una colonna della tabella.

Se faccio un ALTER TABLE su una tabella già popolata nella colonna nuova troveremo tutti NULL, a meno che non ci sia un DEFAULT nella dichiarazione della nuova colonna.

9.7 Definizione degli indici

La primary key mi permette di distinguere le tuple le une dalle altre, in realtà anche le altre chiavi mi permettono di fare questa cosa, ma la primary mi garantisce che i valori sono presenti e non NULL. Poi il database ottimizza la ricerca di queste tuple secondo la colonna di primary key, se ho un'altra colonna particolarmente utile nelle interrogazioni, si puo' usare l'istruzione CRETE INDEX per una particolare colonna ed il DBMS si costruisce una struttura dati particolarmente efficiente (bit tree) che permette di cercare molto rapidamente tra tutte le tuple in base all'attributo che abbiamo scelto (la colonna).

9.8 DDL, in pratica

In molti sistemi si utilizzano strumenti diversi dal codice SQL per definire lo schema della base di dati. Esistono dei tool che dalla progettazione logica ci generano direttamente le tabelle nel database.

9.9 SQL, operazioni sui dati

- Interrogazione:
  - SELECT
- Modifica:
  - INSERT, DELETE, UPDATE

La cosa principale di questo argomento è la corrispondenza tra le operazioni viste nell'algebra e quelle di un interrogazione base in SQL.

Quello che troviamo in SQL come SELECT non è la selezione, ma è la proiezione.

9.9.1 Istruzione SELECT (versione base)

Nel calcolo la prima parte dice quali saranno le colonne della tabella risultante (targhet list) in questo caso nella SELECT mettiamo la targhet list. Il FROM ci dice da dove prendo i dati. Il WHERE non è obbligatorio e sono le condizioni della SELECT.

```
SELECT ListaAttributi
FROM ListaTabelle
 [WHERE Condizione]
```

Maternità

Madre	Figlio
Luisa	Maria
Luisa	Luigi
Anna	Olga
Anna	Filippo
Maria	Andrea
Maria	Aldo

Paternità

Padre	Figlio
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo

Persone

Nome	Età	Reddito
Andrea	27	21
Aldo	25	15
Maria	55	42
Anna	50	35
Filippo	26	30
Luigi	50	40
Franco	60	20
Olga	30	41
Sergio	85	35
Luisa	75	87

9.9.2 Selezione e proiezione

Trovare nome e reddito delle persone con meno di trenta anni.

$$PROJ_{Nome,Reddito}(SEL_{Eta>30}(Persone))$$

Nella SELECT metto subito gli attributi che mi interessano, poi capisco da che tabella devo prendere i dati (Persone), dopo di che vado a vedere le condizioni e le metto nel WHERE.

```
SELECT nome, reddito
FROM persone
WHERE eta < 30;
```

9.9.3 SELECT, abbreviazioni

In questo esempio vediamo che possiamo usare delle variabili. Persone AS p è una variabile che va sulle tuple di persone, quando dichiamo p.eta scorre le tuple di persone e per ogni tupla controlla se l'eta è minore di 30. AS viene usato anche nella targhet list, perchè come vedremo potremmo voler cambiare il nome della colonna risultante delle tabelle risultanti dalle interrogazioni, quindi in pratica AS lo useremo anche come ridenominazione.

AS è anche utile perchè se voglio usare due volte la stessa tabella posso cambiare il nome.

```
SELECT nome, reddito
FROM persone
WHERE eta < 30;
```

```
SELECT p.nome AS nome,
       p.reddito AS reddito
FROM persone AS p
WHERE p.eta < 30
```

9.9.4 Selezione, senza proiezione

Se vogliamo come risultato tutti gli attributi è inutile che li elenchiamo tutti, mettiamo l'asterisco star. Ad esempio vogliamo trovare il nome, l'eta' ed il reddito delle persone con meno di trenta anni:

$$SEL_{Eta>30}(Persone)$$

```
SELECT *
FROM persone
WHERE eta < 30;
```

Sarebbe uguale a:

```
SELECT nome,eta,reddito
FROM persone
WHERE eta < 30;
```

9.9.5 Proiezione, senza selezione

Se non ho una condizione il WHERE è opzionale. Se voglio mettere per forza il WHERE cosa scrivo? scrivo WHILE true, dato che l'argomento del WHERE è una condizione booleana se ne metto una che sara' sempre vera e' sempre verificata e non ho problemi.

Ad esempio vogliamo trovare nome e reddito di tutte le persone:

$$PROJ_{Nome,Reddito}(Persone)$$

```
SELECT nome, reddito
FROM persone
```

9.9.6 SELECT, abbreviazioni

$$R(A,B)SELECT * FROMR$$

Intuitivamente equivale a :

```
SELECT X.A AS A, X.B AS B
FROM R X
WHERE true
```

9.10 Espressioni nella targhet list

Nell'SQL possiamo generare valori che non stanno nel database.

```
SELECT Reddito/2 AS redditoSemestrale
FROM Persone
WHERE Nome='Luigi'
```

9.10.1 Condizione complessa

```
SELECT *
FROM persone
WHERE reddito > 25 AND (eta < 30 OR eta > 60)
```

9.10.2 Condizione LIKE

Il simbolo _ vuol dire qualsiasi carattere ed il % vuol dire qualsiasi stringa. Quindi se vogliamo trovare le persone che hanno un nome che inizia per 'A' e ha una 'd' come terza lettera.

```
SELECT *
FROM persone
WHERE nome LIKE 'A_d%'
```

9.11 Gestione dei valori nulli

Impiegati

Matricola	Cognome	Filiale	Età
5998	Neri	Milano	45
9553	Bruni	Milano	NULL

Se chiedo gli impiegati che hanno più di 40 anni e poi cerco di capire se vogliono anche quelli che potrebbero avere più di 40 anni allora non possiamo più fare la query WHERE eta ¿ 40, ma se vogliamo anche le tuple di Impiegati con valore eta NULL, bisogna mettere nel WHERE or is null. Vogliamo trovare gli impiegato la cui eta' e' o potrebbe essere magiore di 40:

$$SEL_{(Eta>40)OR(EtaISNULL)}(Impiegati)$$

```
SELECT *
FROM impiegati
WHERE eta > 40 OR eta IS NULL
```

9.12 Proiezione in SQL

Fino adesso abbiamo visto che la selezione nell'algebra si risolve nella clausola WHERE, la proiezione abbiamo intuito che è un po' come la targhet list che va nella SELECT, ma non è esattamente così. La proiezione nell'algebra elimina le tuple uguali, in SQL non ottengo lo stesso risultato se scrivo SELECT Cognome, Filiale FROM Impiegati. Perchè in SQL si è deciso che come standard non toglie i duplicati, però è anche possibile eliminarli. Quindi una domanda a quale spesso sbagliamo è qual'è l'istruzione SQL equivalente alla proiezione, che è SELECT DISTINCT, la quale elimina i duplicati. Ci sono due motivi:

- A volte è utile non eliminare i duplicati, quando ad esempio vogliamo contare le occorrenze.
- Eliminare i duplicati è un'operazione in più, c'è un algoritmo che lo fa, quindi costa.

Cognome	Filiale
Neri	Napoli
Neri	Milano
Rossi	Roma

Trovare cognome e filiale di tutti gli impiegati:

$$PROJ_{Cognome,Filiale}(Impiegati)$$

```
SELECT
    cognome, filiale
FROM impiegati
```

```
SELECT DISTINCT
    cognome, filiale
FROM impiegati
```

Cognome	Filiale
Neri	Napoli
Neri	Milano
Rossi	Roma
Rossi	Roma

Cognome	Filiale
Neri	Napoli
Neri	Milano
Rossi	Roma

9.13 Selezione, proiezione e join

Istruzioni SELECT permettono di realizzare con una sula relazione nella clausola FORM: selezioni, proiezioni e ridenominazioni. Con piu' relazione nella FROM ottengo join e prodotti cartesiani.

9.14 SQL e algebra relazionale

```
R1(A1,A2) R2(A3,A4)
SELECT DISCTINCT R1.A1, R2.A4
FROM R1, R2
WHERE R1.A2 = R2.A3
```

Il prodotto cartesiano riusciamo a farlo nel FROM. Il JOIN è equivalente ad una selezione sul prodotto cartesiano di due tabelle.

$$PROJ_{A1,A4}(SEL_{A2=A3}(R1JOINR2))$$

Possono essere necessarie ridenominazioni nel prodotto cartesiano e nella targhet list (quando una tabella è usate due volte):

```
SELECT x.A1 AS B1,...
FROM R1 X, R2 Y,R1 Z
WHERE X.A2 = Y.A3 AND...
```

```
SELECT X.A1 AS B1, Y.A4 AS B2
FROM R1 X, R2 Y, R1 Z
WHERE X.A2 = Y.A3 AND Y.A4 = Z.A1
```

$$\begin{aligned}&REN_{B1,B2\leftarrow A1,A4}(\\&\quad PROJ_{A1,A4}(SEL_{A2=A3 \text{ AND } A4=C1}(R1 \text{ JOIN}\\&\quad R2 \text{ JOIN REN }_{C1,C2\leftarrow A1,A2}(R1))))\end{aligned}$$

9.15 SQL: esecuzione delle interrogazioni

Le espressioni SQL sono dichiarative e noi ne stiamo vedendo la semantica. In pratica, i DBMS eseguono le operazioni in modo efficiente, ad esempio:

- Eseguono le selezioni al più presto
- Se possibile, eseguono join e non prodotti cartesiani

Queste operazioni sono di query optimization.

9.15.1 Selezione, proiezione e join

Trovare i padri di persone che guadagnano più di 20.

$$PROJ_{Padre}(PaternitaJOIN_{Figlio=Nome}SEL_{Reddito>20}(persone))$$

```
SELECT DISTINCT padre
FROM persone, paternita
WHERE figlio = nome AND reddito > 20
```

Maternità

Madre	Figlio
Luisa	Maria
Luisa	Luigi
Anna	Olga
Anna	Filippo
Maria	Andrea
Maria	Aldo

Paternità

Padre	Figlio
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo

Persone

Nome	Età	Reddito
Andrea	27	21
Aldo	25	15
Maria	55	42
Anna	50	35
Filippo	26	30
Luigi	50	40
Franco	60	20
Olga	30	41
Sergio	85	35
Luisa	75	87

9.15.2 2

Le persone che guadagnono piu' dei rispettivi padri; mostrare nome, reddito e reddito del padre.

```
SELECT f.nome, f.reddito, p.reddito
FROM persone p, paternita, persone f
WHERE p.nome = padre AND figlio = f.nome AND f.reddito > p.reddito
```

Nel FROM possiamo mettere piu' tabelle separandole da virgola ed opzionalmente (meglio farlo per chiarezza) possiamo definire una variabile per riferirsi alle tabelle persone p. Nel WHERE metto tutte le condizioni che metto nell'equi JOIN. Riesco a fare il controllo f.reddito ¿ p.reddito perche' ho messo sulla stessa riga che esce dal prodotto cartesiano tutti i dati del padre e tutti quelli del figlio.

9.15.3 SELECT, con ridenominazione del risultato

Si usa AS per fare la ridenominazione. Ad esempio se abbiamo f.reddito non e' chiaro cosa il sistema deve mettere come nome della colonna quindi decidiamo di chiamarla reddito semplicemente.

```
SELECT figlio, f.reddito AS reddito, p.reddito AS redditoPadre
FROM persone p, paternita, persone f
WHERE p.nome = padre AND figlio = f.nome AND f.reddito > p.reddito
```

9.16 JOIN esplicito

Se non vogliamo solo fare il prodotto cartesiano (creando una tabellona con dentro tutte le informazioni che dobbiamo prendere), ma vogliamo direttamente implementare un equi JOIN dell'algebra, lo facciamo mettendo nella clausola

FROM il JOIN seguito dalla parola chiave ON dove mettiamo la condizione dell'equi JOIN.

Le due espressioni di seguito sono equivalenti, solo che la prima e' fatta con il prodotto cartesiano e la seconda con il JOIN esplicito (o inner JOIN):

```
SELECT paternita.figlio, padre, madre
FROM maternita, paternita
WHERE paternita.figlio = maternita.figlio

SELECT madre, paternita.figlio, padre
FROM maternita JOIN paternita ON paternita.figlio = maternita.figlio
```

9.16.1 Esempio

Le presone che guadagnano piu' dei rispettivi padri; mostrare nome, reddito e reddito del padre.

L'espressione sopra e' gia' vista fatta con un prodotto cartesiano tra tre tabelle, mentre sotto faccio la stessa cosa ma usiamo il JOIN ON. Qui la seconda e' piu' esplicita, facile da capire e da scrivere.

```
SELECT f.nome, f.reddito, p.reddito
FROM persone p, paternita, persone f
WHERE p.nome = padre AND figlio = f.nome AND f.reddito > p.reddito

SELECT f.nome, f.reddito, p.reddito
FROM (persone p JOIN paternita ON p.nome = padre)
    JOIN persone f ON figlio = f.nome
WHERE r.reddito > p.reddito
```

9.17 JOIN naturale

Il natural JOIN prendeva due tabelle e se avevano colonne con lo stesso nome faceva il JOIN, quindi anche in SQL ci da' l'istruzione per farlo:

```
SELECT madre, paternita.figlio, padre
FROM maternita JOIN paternita ON paternita.figlio = maternita.figlio

SELECT madre, figlio, padre
FROM maternita NATURAL JOIN paternita
```

9.18 JOIN esterno: "outer join"

Il JOIN esterno serve se voglio fare comparire lo stesso una tupla con una tupla di un'altra tabella nonostante non si sia combinata con niente e metto valori nulli. Inoltre ricordiamo che abbiamo il JOIN esterno sinistro se vogliamo tenere

i valori che non si combinano della tabella di sinistra, il JOIN esterno destro per tenere le tuple che non si combinano della tabella di destra ed il FULL JOIN per tenere tutte le tuple di entrambe le tabelle.

```
SELECT paternita.figlio, padre, madre
FROM paternita LEFT JOIN maternita ON paternita.figlio = maternita.figlio
```

```
SELECT paternita.figlio, padre, madre
FROM paternita LEFT OUTER JOIN maternita ON paternita.figlio = maternita.fig
```

```
SELECT paternita.figlio, padre, madre
FROM paternita FULL OUTER JOIN maternita ON paternita.figlio = maternita.fig
```

OUTER e' opzionale.

9.19 Ordinamento del risultato

Vediamo un'altra cosa che distingue l'SQL dal linguaggio puro dell'algebra. Abbiamo detto tante volte che l'ordinamento delle tuple nel modello relazionale non e' importante. Mentre vediamo che nell'SQL per praticita' ci permette di ordinare le tuple.

Per prima cosa guardiamo il FROM (da che tabelle stanno arrivando i dati), la condizione nel WHERE, cosa abbiamo nel risultato SELECT e solo alla fine andiamo a vedere come vengono ordinate le tuple nell'ORDER BY

```
SELECT nome, reddito
FROM persone
WHERE eta < 30
ORDER BY nome
```

Persone

Nome	Reddito
Andrea	21
Aldo	15
Filippo	30

Persone

Nome	Reddito
Aldo	15
Andrea	21
Filippo	30

Per avere ordinamenti particolari devo mettere delle parole chiavi, se non dico niente l'ordinamento implicito l'ordinamento e' ascendente (ASC) se invece li voglio discendenti scrivero' DESC. Posso anche fare piu' criteri di ordinamento, per avere ad esempio un ordinamento ulteriore sia sul cognome che sul nome.

Se avessi i null? cambia da DBMS a DBMS lo standard. Pero' lo standard ci permette di imporre un ordine per i valori null (ad esempio mettere tutti i null infondo) aggiungiamo null las in ORDER BY (o null first per metterli incima).

9.20 Unione, intersezione e differenza

Abbiamo visto che le operazioni insiemistiche erano utili nell'algebra e di conseguenza le vogliamo anche in SQL. L'SQL ci permette di usare gli operatori insiemistici definendo le parole chiavi corrispondenti.

Possiamo selezionare solo delle colonne dall'unione con la SELECT

SELECT ...
UNION [**ALL**]
SELECT ...

Per le operazioni insiemistiche i duplicati vengono eliminati a meno che si usi la parola chiave ALL. Anche dalle proiezioni (in senso algebrico) quindi se per caso dalla tabella che uso per farle le operazioni insiemistiche c'erano gia' duplicati vengono eliminati lo stesso.

9.20.1 Unione con duplicati usando ALL

```
SELECT A, B
FROM R
UNION
SELECT A, B
FROM S
```

```
SELECT A, B
FROM R
UNION ALL
SELECT A, B
FROM S
```

9.20.2 Notazione posizionale

In algebra non possiamo fare l'unione (operazioni insiemistiche) su tabelle con schema diverso. Ma l'SQL ci permette di farlo se i domini sono compatibili, ma non e' una buona pratica.

```
SELECT padre, figlio
FROM paternita
UNION
SELECT madre, figlio
FROM maternita
```

In questo caso avremo due colonne come risultato, ma come nome della seconda colonna non si sa cosa succede dipende da DBMS (perche' i nomi sono diversi) oppure non scrive niente come nome della colonna.

Quello che dovremmo fare e' rinominare, usando l'AS nella targhet list.

	Figlio
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo
Luisa	Maria
Luisa	Luigi
Anna	Olga
Anna	Filippo
Maria	Andrea
Maria	Aldo

Padre	Figlio
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo
Luisa	Maria
Luisa	Luigi
Anna	Olga
Anna	Filippo
Maria	Andrea
Maria	Aldo

```
SELECT padre AS genitore, figlio
FROM paternita
UNION
SELECT madre AS genitore, figlio
FROM maternita
```

Attenzione che l'ordine degli attributi che metto nelle SELECT dev'essere coerente, e' importante!

9.20.3 Differenza

Per la differenza si usa l'operatore EXCEPT:

```
SELECT Nome
FROM Impiegato
EXCEPT
SELECT Cognome AS Nome
FROM Impiegato
```

Non stiamo confrontando valori della sulla stessa tupla, ma stiamo confrontando i nomi e cognomi di persone diverse. Questa interrogazione restituisce i nomi degli impiegati che non sono anche cognomi degli impiegati.

9.20.4 Intersezione

```
SELECT Nome
FROM Impiegato
INTERSECT
SELECT Cognome AS Nome
FROM Impiegato
```

equivale a

```
SELECT I.Nome
FROM Impiegato I, Impiegato J
WHERE I.Nome = J.Cognome
```

Quindi l'intersezione mi permette di confrontare valori su tuple diverse della stessa tabella.

9.21 Interrogazioni nidificate

Le condizioni atomiche (quello che mettiamo nel WHERE) permettono anche il confronto fra un attributo (o piu') e il risultato di una sottointerrogazione. Potrebbe essere che nel WHERE ci troviamo una sottointerrogazione, un' altra SELECT FROM WHERE.

In alcuni casi e' piu' intuitivo esprimere l'interrogazione cosi', ma dobbiamo fare attenzione a come viene interpretata l'interrogazione. Nel calcolo usavamo exists e for all.

Trovare nome e reddito del padre di Franco:

```
SELECT Nome, Reddito
FROM Persone
WHERE Nome = (SELECT Padre
              FROM Paternita
              WHERE Figlio = 'Franco')
```

Uno potrebbe pensare a questa interrogazione cercando prima il padre di Franco e poi cercare il nome ed il reddito. Notiamo che nel WHERE comunque abbiamo una condizione booleana perche' c'e' un uguale mi restituira' quindi un true o false.

La forma nidificata e' meno dichiarativa (perche' seguo un ragionamento procedurale, prima cerco il papa' di franco e poi cerco i suoi dati). La forma piana (senza nidificazione) e quella nidificata possono essere combinate, ci possono essere notazioni che usano in parte il JOIN ed in parte usano la nidificazione. Le sottointerrogazioni non possono contenere operatori insiemistici (l'idea' e' che l'unione si fa a livello esterno).

Trovare nome e reddito dei padri di persone che guadagnano piu' di 20:

```
SELECT DISTINCT P.Nome, P.Reddito, P.Reddito
FROM Persone P, Paternita, Persone F
WHERE P.Nome = Padre AND Figlio = F.Nome
    AND F.Reddito > 20
```

```
SELECT Nome, Reddito
FROM Persone
WHERE Nome IN(SELECT Padre
              FROM Paternita
              WHERE Figlio = ANY(SELECT Nome
                                 FROM Persone
                                 WHERE Reddito > 20)
```

IN (=ANY) ci dice che il valore di nome dev'essere uguale ad uno di quelli restituiti dal SELECT successivo.

Quando guardiamo una query nidificata cominciamo dalla parte piu' interna per capirla.

9.21.1 Operatori ANY, ALL, IN

Nelle query nidificate e' utile confrontare valori con insiemi di valori risultanti dalla subquery. L'operatore ANY si combina con tutti quelli matematici (diverso ANY vuol dire un valore diverso tra tutti quelli risultati dell'interrogazione). Esiste anche ALL

Riscriviamo l'interrogazione di prima reddito maggiore di 20 con un solo livello di nidificazione:

```
SELECT Nome, Reddito
FROM Persone
WHERE Nome IN (SELECT Padre
        FROM Paternita, Persone
        WHERE Figlio = Nome AND Reddito > 20
```

9.21.2 Interrogazioni nidificate: commenti

La prima versione di SQL prevedeva solo la forma nidificata (o strutturata), con una solo relazione in ogni clausola FROM. Ma era insoddisfacente in quanto la dichiarativita' era limitata e non si potevano includere nella target list attributi di relazioni nei blocchi interni.

Trovare nome e reddito dei padri di persone che guadagnano piu' di 20, con indicazione del reddito del figlio (eliminando i duplicati se ci sono):

```
SELECT DISTINCT P.Nome, P.Reddito, F.Reddito
FROM Persone P, Paternita, Persone F
WHERE P.Nome = Padre AND Figlio = F.Nome
    AND F.Reddito > 20
```

Senza usare la forma nidificata non ho problemi perche' dichiaro una variabile e ritorno quella.

```
SELECT Nome, Reddito, ????
FROM Persone
WHERE Nome in (SELECT Padre
                FROM Paternita
                WHERE Figlio = ANY(SELECT Nome
                                   FROM Persone
                                   WHERE Reddito > 20))
```

Con la forma nidificata ho un problema nel select esterno nell'identificare il reddito dei figli.

```
SELECT DISTINCT P.Nome, P.Reddito, F.Reddito
FROM Persone P, Persone F
WHERE F.Reddito > 20 AND
        P.Nome IN
        (SELECT Padre FROM Paternita WHERE Figlio = F.Nome);
```

Potrei decidere di combinare la forma piana con la nidificata ma tirando (facendo un prodotto cartesiano nel from) fuori la seconda istanza di persone in modo tale che nel primo from posso usare le due variabili p ed f. Quindi stiamo cercando di capire come usare le variabili quando abbiamo query nidificate.

9.21.3 Regole visibilita

Una variabile e' utilizzabile solo nell'ambito della query in cui e' definita o nell'ambito di una nidificata interna (qualsiasi livello). In un blocco si puo' fare riferimento a variabili definite in blocchi piu' esterni.

9.21.4 Quantificatore esistenziale

EXISTS(sottoespressione). Si usa nelle interrogazioni nidificate, WHERE EXISTS (se esiste sara' true se non esiste sara' false). Esiste almeno una tupla nella tabella?

Ritornare le persone che hanno almeno un figlio:

```
SELECT *
FROM Persone
WHERE EXISTS (SELECT *
              FROM Paternita
              WHERE Padre = Nome)
      OR
      EXISTS (SELECT *
              FROM Maternita
              WHERE Madre = Nome)
```

Sto prima scorrendo tutte le coppie della tabella Paternita' per vedere se esiste un padre uguale al nome che cerchiamo, stessa cosa nel secondo exists. Mettendoli in or basta che uno sia true per far diventare true il WHERE esterno. Faccio questa cosa per tutte le persone della tabella Persone, che e' quello che mi dice il SELECT FROM esterno come se fosse un ciclo for sulla tabella Persone.

Notiamo che Nome e' un attributo che e' definito fuori, quindi per capire le interrogazioni nidificate dobbiamo partire dall'esterno.

Per trovare tutte le persone che hanno nel database sia padre che madre devo mettere in AND le due interrogazioni nidificate, inoltre devo cambiare le condizioni dei WHERE nidificati con WHERE Figlio = Nome.

Ora trovare i padri i cui figli guadagnano tutti piu' di 20:

Ricordiamo che tutti piu' di 20 posso calcolare quelli che hanno i padri che hanno almeno un figlio che guadagna meno di 20 e fare la differenza quindi usare gli operatori insiemistici.

```
SELECT DISTINCT Padre
FROM Paternita Z
WHERE NOT EXISTS (
            SELECT *
            FROM Paternita W, Persone
            WHERE W.Padre = Z.Padre
                AND W.Figlio = Nome
                AND Reddito <= 20)
```

Guardo i candidati papa' uno ad uno, ed uso una variabile Z, che assume il valore di una tupla per ogni ciclo. Nella nidificata voglio controllare tutti i figli di Mario (un padre) per capire se esistono alcuni che guadagnano meno uguale di 20, quindi l'interrogazione interna fa' un prodotto cartesiano tra Paternita' ed un'altra istanza di Persone (per prendere il reddito), dopo aver fatto questa unione tra tabelle devo ancora scorrere la tabella Paternita ma devo dire che mi interessano solo le tuple in cui Mario (Z.Padre) e' padre, guardo che di tutte le tuple che trovo in cui Mario e' papa' guardiamo se il figlio guadagna minore uguale 20.

Occhio che se non scorriamo tutti i figli del padre che stiamo selezionando volta per volta otteniamo anche i padri che hanno solo un figlio che guadagna piu' di 20:

```
SELECT DISTINCT Padre
FROM Paternita
WHERE NOT EXISTS (
            SELECT *
            FROM Persone
            WHERE Figlio = Nome
                AND Reddito <= 20)
```

E' sbagliata! Perche' non consideriamo tutti i figli di un padre, ma teniamo anche i padri che hanno almeno un figlio che guadagna piu' di 20.

9.21.5 Semantica delle espressioni correlate

L'interrogazione interna viene eseguita una volta per ciascuna ennupla dell'interrogazione esterna. Quindi quando guardiamo queste interrogazioni dobbiamo partire dall'esterno capendo cosa restituisce il ciclo esterno e guardare cosa succede dentro partendo da un istanza d'esempio.

9.21.6 Visibilita sbagliata

```
SELECT *
FROM Impiegato
WHERE Dipart IN(SELECT Nome
                FROM Dipartimento D1
                WHERE Nome = 'Produzione')
```

```
OR
Dipart IN (SELECT Nome
           FROM Dipartimento D2
           WHERE D2.Citta = D1.Citta)
```

Possiamo usare una variabile nella query in cui e' stata definita oppure possiamo usarne una che e' stata definita ad un livello piu' fuori. Quindi in questo caso non possiamo usare D1 nella seconda query dell' OR, in quanto e' stata definta in una query dello stesso livello.

9.21.7 Differenza e nidificazione

```
SELECT Nome FROM Impiegato
EXCEPT
SELECT Cognome AS Nome FROM Impiegato
```

Mi da tutta la lista dei nomi, quella sotto mi da tutta la lista dei cognomi e mi restituisce tutti i nomi degli impiegati tranne quelli che sono uguali a qualche cognome di impiegato.

Si puo' scrivere anche cosi':

```
SELECT I.Nome
FROM Impiegato I
WHERE NOT EXISTS (SELECT *
                  FROM Impiegato
                  WHERE Cognome != I.Nome)
```

Dammi il nome di tutti gli impiegati, ma facendo attenzione che non esista uno con il cognome uguale al nome che stiamo considerando. Attenzione che lui consiglia di utilizzare sempre le variabili (Impiegato Z e poi Z.Cognome) in modo che sia tutto piu' chiaro. Questa roba qua la leggo ancora una volta dall'esterno, scorro tutti i nomi di impiegato e per ciascuno valuto la query dentro, ad esempio per Franco vado a veder se c'e' qualcuno che ha cognome Franco, se lo trovo l'esiste vale true ed il non esiste vale false (quindi non lo metto nella tabella risultante).

9.22 Operatori aggregati

L'ultima cosa che ci rimane da vedere ma che useremo tantissimo (anche all'esame) e' l'uso degli operatori aggregati. Nelle espressioni della target list possiamo avere anche espressioni che calcolano valori a partire da insiemi di ennuple:

- Conteggio, minimo, massimo, media, totale

Abbiamo gia' visto che in SQL possiamo inventarci dei valori che non esistono nel database.

La sintassi base (semplificata):

```
Funzione([DISTINCT]*)
Funzione([DISTINCT] Attributo)
```

La semantica delle funzioni e' ovvia, ma creano dei problemi nell'utilizzo quando ad esempio ci sono valori NULL.

9.22.1 COUNT

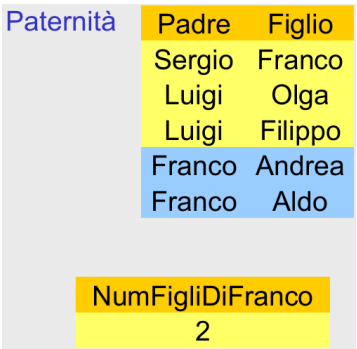
Il numero di figli di Franco:

```
SELECT COUNT(*) AS NumFigliDiFranco
FROM Paternita
WHERE Padre = 'Franco'
```

COUNT(*) conta le tuple nella tabella. L'AS e' per dire al DBMS come chiamare la colonna risultante dall'interrogazione.

L'operatore aggregato viene applicato al risultato dell'operazione, quindi nel nostro caso il FROM WHERE prende le tuple che mi servono e poi il COUNT le conta.

Come argomento del COUNT posso metterci un attributi, una lista di attributi o l'*.



COUNT DISTINCT

Se conto le tuple dato che non siamo nell'algebra non so se ci sono tuple duplicate (soprattutto se faccio una proiezione).

Con il DISTINCT reddito mi torna solo il numero di valori diversi:

```
SELECT COUNT(DISTINCT reddito) FROM persone
```

Persone	Nome	Età	Reddito
	Andrea	27	21
	Aldo	25	35
	Maria	55	21
	Anna	50	35

9.22.2 Altri operatori aggregati

SUM, AVG, MAX, MIN.

Trovare la media dei redditi dei figli di Franco:

```
SELECT AVG(reddito)
FROM persone JOIN paternita ON nome = figlio
WHERE padre = 'Franco'
```

Nel FROM sto mettendo sulla stessa riga il nome del papa ed i dati del figlio (solo di padre = Franco). A questo punto che ho questa tabella con tutti i figli di Franco applico l'operatore di Media a tutte le righe di Reddito della tabella.

9.22.3 COUNT e valori nulli

Persone	Nome	Età	Reddito
	Andrea	27	21
	Aldo	25	NULL
	Maria	55	21
	Anna	50	35

Il NULL non viene considerato, solo se sto considerando un solo valore, ma non ignoro tutta la tupla solo perche' c'e' un valore NULL.

```
SELECT COUNT(*) FROM persone
```

Restituisce 4.

```
SELECT COUNT(reddito) FROM persone
```

Restituisce 3.

```
SELECT COUNT DISTINCT(reddito) FROM persone
```

Restituisce 2.

9.22.4 Operatori aggregati e valori nulli

I valori NULL, come nel COUNT, vengono ignorati.

```
SELECT AVG(reddito) AS redditomedio
FROM persone
```

9.22.5 Operatori aggregati e target list

Nella target list possiamo metterci degli operatori di aggregazione, pero' normalmente nella target list abbiamo tante cose. Quindi come puo' essere la target list se c'e' anche un operatore di aggregazione.

Un'interrogazione scorretta per dire il nome di chi ha il reddito piu' alto di tutti:

```
SELECT nome, MAX(reddito)
FROM persone
```

Di chi sarebbe il nome? la target list dev'essere omogenea, quindi posso mettere piu' funzioni di aggregazione ma non posso mettere cose miste, ad esempio un valore di attributo insieme ad un valore aggregato.

9.22.6 Massimo e nidificazione

Un modo per trovare le persone con reddito piu' alto:

```
SELECT *
FROM persone
WHERE reddito = (SELECT MAX(reddito)
                 FROM persone)
```

Qui scorro tutte le persone e riporto la persona soltanto se il suo reddito e' il massimo di quelle che ci sono nella tabella. Quindi ricordiamoci che ci capitera' molte volte di dover fare un'interrogazione in termini di un operatore di aggregazione.

Nel caso ci fossero piu' persone che guadagnano la stessa cifra che e' il massimo ritorna tutti e due.

Persone

Nome	Età	Reddito
Andrea	27	21
Aldo	25	15
Maria	55	42
Anna	50	35
Filippo	26	30
Luigi	50	40
Franco	60	20
Olga	30	41
Sergio	85	35
Luisa	75	87

```
(SELECT MAX(reddito)
FROM persone)
```

Nel nostro esempio ritornerebbe una tabella di una sola colonna con 87.

```
SELECT *
FROM persone
WHERE reddito =
      (SELECT MAX(reddito)
      FROM persone)
```

Nel nostro esempio ritornerebbe una tabella con la sola tupla: Luisa,75,87.

9.22.7 Operatori aggregati e raggruppamenti

Stiamo arrivando a vedere come mettere nelle targat list operatori aggregati e valori di attributi.

Vediamo un altra key-word GROUP BY, di fatto che permette di partizionare le tuple di una tabella. Vuol dire che se ho 10 tuple nella tabella persone e le raggruppo in 3 gruppi diversi ho fatto una partizione di queste tuple e non ne ho lasciate fuori nessuna. Prende come argomento una lista di attributi.

Trovare il numero di figli di ciascun padre:

```
SELECT p.padre, COUNT(*) AS NumFigli
FROM paternita p
GROUP BY p.padre
```

Voglio che per ogni papa mi ritorni il numero di figli, notiamo che qui abbiamo sia un nome di attributo che di operatore aggregato, si puo' fare solo quando questo attributo compare nella clausola GROUP BY. In questo caso gli sto dicendo di guardare la tabella paternita', partiziona questa tabella formando dei gruppi mettendo insieme le tuple che hanno lo stesso papa (il valore e' uguale per l'attributo padre). Quando faccio questa partizione l'operatore che scrivo in alto COUNT non e' fatto sull'intera tabella ma e' fatto su ciascun gruppo. Quindi il COUNT (*) significa che conto quante tuple ci sono in ciascun gruppo, quindi adesso ha senso metterci vicino il valore dell'attributo rispetto al quale ho raggruppato, quindi e' come se dicessi Luigi 2 figli, il gruppo che ha il valore uguale per quel gruppo ha come funzione di aggregazione il valore dell'attributo.

paternita

Padre	Figlio
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo

Padre	NumFigli
Sergio	1
Luigi	2
Franco	2

9.22.8 Raggruppamenti e target list

La prima e' scorretta perche' c'e' un reddito, ma e' sbagliato perche' ho raggruppato solo per padre.

```
SELECT p.padre, avg(f.reddito), q.reddito
FROM persone f JOIN paternita p ON p.figlio = f.nome JOIN
    persone q ON p.padre = q.nome
GROUP BY p.padre
```

Mentre se faccio il raggruppamento sia per padre che per reddito posso mettere il reddito nella terget list.

```
SELECT p.padre, avg(f.reddito), q.reddito
FROM persone f JOIN paternita p ON p.figlio = f.nome JOIN
    persone q ON p.padre = q.nome
GROUP BY p.padre, q.reddito
```

9.22.9 Condizione sui gruppi

Trovare i padri i cui figli hanno un reddito medio maggiore di 25; mostrare padre e reddito medio dei figli.

```
SELECT p.padre, AVG(f.reddito)
FROM persone j JOIN paternita p ON p.figlio = f.nome
GROUP BY p.padre
HAVING AVG(f.reddito) > 25
```

Dobbiamo prima trovare le tabelle che mi interessano, poi e' richiesta una media ma e' una media di alcuni figli raggruppati per papa', quindi dovrei intuire che devo raggruppare le tuple che hanno lo stesso papa' e calcolare la media del reddito dei figli che stanno nelle tuple. Sulla riga ho bisogno di mettere insieme con un JOIN i dati della persona che e' figlio di un certo papa' (JOIN tra persone e paternita'). A questo punto dobbiamo raggruppare queste tuple, perche' per ora cisono tutti i papa' con i loro figli e dato che voglio fare la media solo per i figli che hanno lo stesso nome devo raggruppare. Quindi la semantica per questo operatore e': prendiamo la parte di interrogazione FROME WHERE e guardiamo che tabella c'e' e soltanto dopo raggruppiamo . L'HAVING e' come un WHERE (che dice quali tuple vogliamo dalla tabella del FROM che soddisfano una condizione), vuol dire che raggruppo queste tuple ma non tengo tutti i gruppi, ma seleziono solo quelli che soddisfano la condizione dell'HAVING, come se fosse una selezione algebrica sui gruppi e non sulle tuple, nel nostro caso dice che dei gruppi fatti tieni solo quelli la cui media dei redditi dei figli e' maggiore di 25.

9.22.10 WHERE o HAVING

Trovare i padri i cui figli sotto i 30 anni hanno un reddito medio maggiore di 20:

```
SELECT p.padre, AVG(f.reddito)
FROM persone f JOIN paternita p ON p.figlio = f.nome
WHERE f.eta < 30
```

```
GROUP BY p.padre
HAVING AVG(f.reddito) > 20;
```

Non e' molto diverso da prima, qui come al solito metto insieme sulla stessa riga il nome del papa' e tutti i dati del figlio (per tutti i padri e tutti i figli del padre), a questo punto non li tengo tutti, ma tengo solo i figli che hanno meno di 30 anni nel WHERE, questa operazione posso farla globalmente, non ho bisogno di farla in ogni gruppu. A questo punto quando raggruppo per papa' in ogni gruppo non ci saranno piu' quelli piu' vecchi di 30 anni, e tengo solo i gruppi la cui media dei redditi dei figli e' maggiore di 20 (nel HAVING).

Riassumendo il WHERE opera sulla tabella risultante dal FROM tupla per tupla, come se fosse un ciclo for e per ciascuna delle tuple della tabella del FROM applica la condizione, quelle che non soddisfano la condizione se ne vanno.

9.22.11 HAVING con subquery

Selezionare i padri i cui figli sotto i 30 anni hanno un reddito medio maggiore di quello medio dei figli di Franco.

```
SELECT p.padre, AVG(f.reddito)
FROM persone f JOIN paternita p ON p.figlio = f.nome
WHERE f.eta < 30
GROUP BY p.padre
HAVING AVG(f.reddito) >
        (SELECT AVG(q.reddito)
         FROM persone q JOIN paternita r ON r.figlio = q.nome
            WHERE r.padre = 'Franco'
            GROUP BY r.padre)
```

E' la stessa interrogazione di prima, solo che invece che scrivere maggiore di 20 dobbiamo dobbiamo metter il reddito medio maggiore del reddito medio dei figli di Franco. Intuitivamente prima troviamo una query che ci da il reddito medio dei figli di franco. Questa cosa ci fa vedere che l'HAVING, come il WHERE, nella sua condizione potrebbe avere delle query nidificate.

9.22.12 GROUP BY e valori NULL

```
SELECT B, COUNT(*)
FROM R GROUP BY B
```

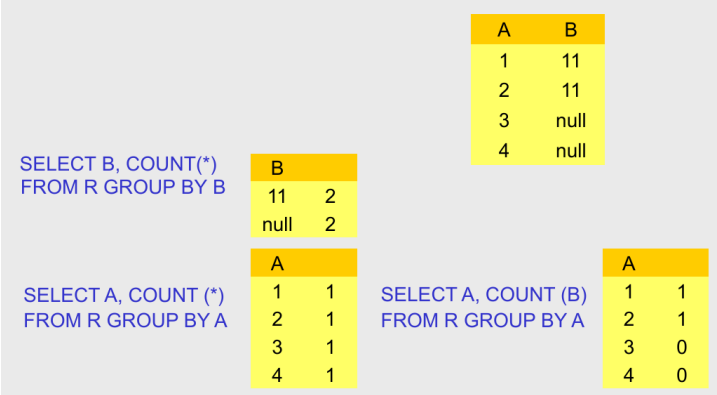
Significa fare il raggruppamento di tutte le tuple che hanno lo stesso valore per B, in caso di valori NULL il GROUP BY fa un gruppo per il NULL.

```
SELECT B, COUNT(*)
FROM R GROUP BY A
```

Stiamo dicendo di raggruppare per A, quindi i gruppi saranno 4 dato che i valori di A sono diversi, COUNT (*) si applica ad ogni gruppo e conta le tuple di ciascun gruppo.

```
SELECT A, COUNT(B)
FROM R GROUP BY B
```

Dato che il COUNT non conta i valori nulli quando li deve contare restituisce 0.



9.22.13 Riassunto regole su valori nulli

I valori NULL vengono ignorati in ogni aggregazione, non contribuiscono a somme, medie, conteggi, ne' possono essere considerati minimo o massimo. NULL viene considerato come un valore nei raggruppamenti, quindi si forma un gruppo dei valori NULL. Il risultato di una aggregazione (eccetto COUNT) su un insieme vuoto di valori restituisce NULL. COUNT restituisce 0.

9.23 Operazioni di aggiornamento

Parliamo sempre di Data Manipulation Language, e tramite queste istruzioni possiamo inserire i valori nelle tabelle.

- INSERT INTO: inserire
- DELETE FROM: cancellare
- UPDATE: modifica

9.23.1 Inserimento

```
INSERT INTO Tabella [(Attributi)]
    VALUES(Valori)
```

oppure

```
INESRT INTO Tabella [(Attributi)]
```

```
INSERT INTO Persone VALUES ('Mario', 25,25);
```

Inserisco nella tabella Persone dei valori, che ovviamente devono rispettare il dominio. L'effetto e' di inserire una tupla nella tabella con quei valori.

```
INSERT INTO Persone (Nome,Eta,Reddito)
VALUES ('Mario', 25,25);
```

Uguale a prima ma ho solo specificato il nome degli attributi.

```
INSERT INTO Persone (Nome,Reddito)
VALUES ('Mario', 25);
```

Non conoscendo il valore dell'eta' inserisco una nuova tupla dicendogli che il 25 e' il reddito, specificando gli attributi.

```
INSERT INTO Persone(Nome)
SELECT Padre
FROM Paternita
WHERE Padre NOT IN
    (SELECT Nome FROM Persone)
```

Inserisce nella tabella Persone una tupla con NomePapa',NULL,NULL solo se e' vera la condizione WHERE, ovvero che NomePapa' non e' presente nella tabella Persone. In pratica lo aggiungo se non c'e' gia'. E' utile perche' se devo inserire tanti tuple nuove lo faccio fare al databse senza farlo a mano.

Commenti

L'inserimento degli attributi (se presente) e dei valori e' significativo. Le due lise debbono avere lo stesso numero di elementi. Se la lista di attributi e' omessa, si fa riferimento a tutti gli attributi della relazione, secondo l'ordine con cui sono stati definiti. Se la lista di attributi non contiene tutti gli attributi della relazione, per gli altri viene inserito un valore nullo (che deve essere permesso) o un valore di default.

9.23.2 Cancellazione

```
DELETE FROM Tabella
[WHERE Condizione];
```

```
DELETE FROM Persone
WHERE Eta < 35
```

```
DELETE FROM Paternita
WHERE Figlio NOT IN
    (SELECT Nome FROM Persone);
```

Toglie tutte le tuple di papa' figlio per cui il figlio non compare nella tabella persone.

```
DELETE FROM Paternita
```

Cancella tutte le tuple della tabella, come se fosse WHERE true, non cancella tutte la tabella.

Commenti

Quando facciamo il DELETE facciamo l'operazione d'esempio della cancellazione dell'alula, se cancelliamo qualcosa e c'erano FK si applicano le politiche che abbiamo definito creando la tabella, ad esempio cascade, se eliminiamo quella tupla con il vincolo cascade si eliminano tutte le altre tuple in cascata.

9.23.3 Modifica di ennuple

Spesso non conviene cancellare e poi reinserire ma modifichiamo il valore dell'eta', con l'UPDATE

```
UPDATE NomeTabella
SET Attributo = <Espressione |
                    SELECT ... |
                    NULL |
                    DEFAULT >
        [WHERE Condizione];

UPDATE Persone SET Reddito = 45
    WHERE Nome = 'Piero'

UPDATE Persone
SET Reddito = Reddito * 1.1
WHERE Eta < 30;
```

Aumenta del 10% il valore dello stipendio degli impiegati giovani.

Chapter 10

SQL: caratteristiche evolute

10.1 Transazioni SQL

Parliamo ora di una parte meno di base dell'SQL. Supportare le transazioni è una parte molto complicata anche per chi sviluppa un DBMS per supportarle efficacemente. Quindi guardiamo cosa sono, dato che sono una caratteristica importante de DBMS anche rispetto ai file system.

Le transazioni sono proprietà "acid" (automicity consistency isolation durability):

- Atomicita'
- Consistenza
- Isolamento
- Durabilita'

10.1.1 Atomicità

Un'operazione atomica è qualcosa che non si può dividere. Quindi la transazione è una sequenza di operazioni sulla base di dati che viene eseguita tutta per intero senza errori, oppure se qualcosa va male devo annullare l'effetto delle operazioni della sequenza che ho già svolto. Lo stato del databse dev'essere identico a prima della sequenza.

Un classico esempio e' il trasferimenti di fondi da un conto A ad un conto B: o si fanno sia il prelevamento da A sia il versamento su B, o nessuno dei due.

10.1.2 Consistenza

Consistente significa che al termine della transazione tutti i vincoli di integrita' definiti sul databse per tutto l'insieme delle operazioni dev'essere rispettato. E' ammesso fare qualcosa che vada contro i vincoli all'interno della transazione ma non alla fine.

10.1.3 Isolation

E' la cosa che mette in crisi i progettisti di DBMS. L'effetto di transazioni concorrenti dev'essere coerente. Le base di dati sono condivise e nello stesso momento ci saranno tanti applicativi che fanno operazioni sullo stesso databse, quindi potrei avere delle sequenze di operazioni atomiche svolte da diverse applicazioni contemporaneamente. L'esecuzione di due operazioni concorrenti dev'essere uguale alle stesse transazioni eseguite in sequenza. Inoltre il disfacimento di una transazione non deve comportare il disfacimento anche di altre transazioni. Quindi in SQL si possono specificare diversi livelli di isolamento, quello con maggiori garanzie si dice SERIALIZABLE, ma questo SERIALIZABLE è costoso e richiede tanto tempo nell'esecuzione delle transazioni, ci sono dei compromessi tra il velocizzare sopportando alcuni problemi che si potrebbero verificare ogni tanto. Generalmente il DBMS fa l'interliving delle operazioni (un

po' per ciascuna), ma deve stare attento, perchè ogni transazione potrebbe modificare temporaneamente dei valori.

10.1.4 Durability

Dobbiamo fare si che quando si arriva infondo all'ultima istruzione si rendano permanenti le modifiche alle tabelle. Generalmente si fa tramite l'operazione di commit.

10.1.5 Transazione in SQL

Nello standard SQL, si usa START TRANSACTION (BEGIN in Postgres). Si può anche chiudere una transazione con il COMMIT, che è la parte di durability e rende stabile nel databse l'effetto del gruppo di transazioni. ROLLBACK: si usa quando dentro la sequenza di istruzioni si controlla se c'è qualche errore ed allora esplicitamente il programmatore se si accorge che c'è qualche valore che non gli piace dice che questo stato non va bene ed allora torna all'inizio, fino all'istruzione prima dello START TRANSACTION.

Anche uno START TRANSACTION potrebbe anche essere implicito, in generale quando inseriamo nel DB le istruzioni di fatto il sistema assume che siano delle transazioni, come se mettessimo uno START TRANSACTION ed il COMMIT. Ma soprattutto il ROLLBACK può essere implicito, possono esserci degli stati raggiunti dal DBMS che non sono degli errori, ma per la semantica del dominio potrebbero non andare bene quindi mettiamo ROLLBACK, ma nel caso in cui si verifichi un errore interno al DB automaticamente mette un ROLLBACK.

```
START TRANSACTION;
UPDATE ContoCorrente
    SET Saldo = Saldo - 10
    WHERE NumeroConto = 12345;
IF errors <> 0 ROLLBACK WORK;
UPDATE ContoCorrente
    SET Saldo = Saldo + 10
    WHERE NumeroConto = 55555;
COMMIT WORK;
END TRANSACTION;
```

10.1.6 Savepoints in PostgreSQL

Se ho tante operazioni e ad un certo punto una mi va male, non vorrei disfare tutte le operazioni buone, quindi indichiamo un punto sicuro in cui fare il ROLLBACK indicandolo con SAVEPOINT:

```
BEGIN;
UPDATE accounts SET balance = balance - 100.00
WHERE name = 'Alice';
```

```
SAVEPOINT my_savepoint;
UPDATE accounts SET balance = balance + 100.00
    WHERE name = 'Bob'
--oops ... forget that and use Wally's account
ROLLBACK TO my_savepoint;
UPDATE accounts SET balance = balance + 100.00
    WHERE name = 'Wally'
COMMIT;
```

10.2 Viste in SQL

Le viste è molto facile doverle usare, magari anche nel progetto. Creiamo una tabella virtuale:

```
CREATE VIEW NomeVista [(ListaAttributi)]
AS SelectSQL
[WITH [LOCAL | CASCADED] CHECK OPTION]
```

Crea questa tabella virtuale come una qualsiasi interrogazioni SELECT FROM WHERE.

Esempio: data la tabella impiegato raccogliamo tutti e soli gli impiegati che lavorano al dipartimento Amministrazione e che guadagnano più di 10 mila euro l'anno:

```
CREATE VIEW ImpiegatiAmmin(Nome, Cognome, Stipendio)
AS SELECT Nome, Cognome, Stipendio
FROM Impiegato
WHERE Dip = 'Amministrazione' AND Stipendio > 10
```

Dobbiamo fare attenzione alla lista di attributi della VIEW, nell'esempio è un caso che abbiamo usato gli stessi nomi, potremmo usare nomi diversi, allora devo stare attento all'ordine in cui metto gli attributi in modo che le colonne della vista prende i valori giusti della SELECT.

10.2.1 Aggiornamento sulle viste

Le viste funzionano perfettamente se dobbiamo leggere dei valori, il problema è quando modifichiamo i valori della vista, perchè non c'è una tabella corrispondente alla vista nel DB, dobbiamo capire dall'interrogazione fatta per creare la vista se possiamo inserire i valori nelle tabelle giuste all'interno del DB.

Molto DBMS non ci permettono di mettere più tabelle all'interno del FROM per creare una vista, perchè facendo dei JOIN non è per niente semplice tornare alla tabella originale. Detto questo quando definiamo le viste è opportuno mettere alcuni controlli, aggiungendo alla dichiarazione della vista WITH CHECK OPTION, vuol dire che permetto di fare modifiche ma prima fai prima questi

controlli, permette quindi modifiche, ma solo a condizione che la ennupla continuo ad appartenere alla vista, ad esempio se prima avevo messo tutti gli impiegati che guadagnano più di 10, non posso metterci poi qualcuno che guadagna meno di 10. Se la vista e' definita in termini di altre viste ci sono due tipi di controlli:

- Local: il controllo è solo sulla vista più esterna, sto guardando solo la vista che stai cercando di modificare, ma in realtà potrei definire una vista a partire da una vista etc.
- Cascaded: il controllo è propagato a cascata su tutte le viste (default)

```
CREATE VIEW ImpiegatiAmmin(Nome, Cognome, Stipendio)
AS SELECT Nome, Cognome, Stipendio
FROM Impiegato
WHERE Dip = 'Amministrazione' AND Stipendio > 10
```

Definiamo una vista a partire da una vista, di tutti gli impiegati dell'amministrazione chiediamo una visione in cui facciamo vedere solo quelli che guadagnano maggiore di 50. Quando definisco questa vista definisco WITH CASCADE CHECK OPTION, quindi diciamo che quando qualcuno modifica questa vista dobbiamo fare dei controlli, a cascata anche alla vista dalla quale abbiamo definito questa:

```
CREATE VIEW ImpieagtiAmminiPoveri as
    SELECT *
    FROM ImpiegatiAmmin
    WHERE Stipendio < 50
    WITH CASCADED CHECK OPTION
```

Successivamente facciamo un'istruzione in cui diciamo che lo stipendio di Paolo viene messo ad 8, cosa succede?

```
UPDATE ImpiegatiAmminPoveri
SET Stipendio = 8
WHERE Nome = 'Paola'
```

Il sistema si rifiuta di farla, perchè avendo definito CASCADE vado a controllare anche nelle viste precedenti se vado a violare le condizioni di quelle viste. Se invece mettessi LOCAL lo farebbe, perchè guarda solo la vista in cui lo stiamo facendo e lo stipendio di Paolo rimarrebbe comunque minore di 50.

10.2.2 Interrogazioni sulle viste

Si può fare riferimento alle viste come relazioni di di base.

```
SELECT * FROM ImpiegatiAmmin
```

equivale a (e viene eseguita come):

```
SELECT Nome, Cognome, Stipendio
FROM Impiegato
WHERE Dipart = 'Amministrazione' AND Stipendio > 10
```

10.2.3 Interrogazioni non standard

Se dobbiamo concatenare in cascata diversi operatori di aggregazione (aggregati). Ad esempio vogliamo il numero medio di uffici che compongono un dipartimento: Interrogazione scorretta:

```
SELECT AVG(COUNT(DISTINCT Ufficio))
FROM Impiegato
GROUP BY Dipart
```

Contare, raggruppando per dipartimento e fare la media non si può fare, uno dei modi per farlo correttamente è definire una vista:

```
CREATE VIEW DipartUffici (NomeDip, NroUffici) AS
    SELECT Dipart, COUNT(DISTINCT Ufficio)
    FROM Impiegato
    GROUP BY Dipart;

SELECT AVG(CAST(NroUffici AS decima(5,2))) AS NumeroMedioUffici
FROM DipartUffici
```

Prima ho definito una vista in cui metto il numero di uffici per ogni dipartimento, facendo un'aggregazione su dipartimento e contando il numero di uffici. Successivamente facciamo la media degli uffici di ogni dipartimento utilizzando la vista che abbiamo definito prima. Il CAST è una funzione, ci serve per cambiare il tipo della variabile, dato che il numero di uffici era un intero e facendo la media potrebbe venire un decimale.

In generale è comodo se abbiamo un'interrogazione complicata dividerla e creare una vista con la prima parte dell'interrogazione, tanto al DBMS non cambia niente.

10.2.4 Uso delle viste per evitare la nidificazione

Esempio: trovare il dipartimento che spende di più in stipendi.

Una prima soluzione che potrebbe venirci in mente:

```
SELECT Dipart
FROM Impiegato
GROUP BY Dipart
HAVING SUM(Stipendio) >= ALL(SELECT SUM(Stipendio)
                             FROM Impiegato
                             GROUP BY Dipart)
```

Non tutti i DBMS ci permettono di fare questa cosa, ci sono dei DBMS che non permettono di fare le nidificazioni nell HAVING. Possiamo aggirare il problema creando una vista

```
CREATE VIEW BudgetStipendi(Dip, TotaleStipendi) AS
SELECT Dipart, SUM(Stipendio)
FROM Impiegato
GROUP BY Dipart
```

Dopo interroghiamo la vista :

```
SELECT Dip
FROM BudgetStipendi
WHERE TotaleStipendi = (SELECT MAX( TotaleStipendi)
                        FROM BudgetStipendi)
```

(C'è sempre una nidificazione, ma le limitazioni stanno nel HAVING e non nel WHERE). Il suggerimento è di provare a cambiare noi le cose che si chiedono cambiando le funzioni di aggregazione e come farle.

10.3 Viste ricorsive

Come si possono fare delle interrogazioni ricorsive in SQL, abbiamo detto che in algebra e nel calcolo non si possono fare (la chiusura ricorsiva), ma nel datalog si può fare.

Dobbiamo ricordarci come si facevano nel datalog, perchè la parte logica è uguale.

Per ogni persona, trovare tutti gli antenati avendo: Paternita(Padre, Figlio).

Serve usare la ricorsione, in datalog, prima scriviamo la base della ricorsione, se ho l'informazione che uno è il papà di qualcun altro posso anche dire che il figlio è il discendente di questo papà. Poi scriviamo la regola ricorsiva in cui c'è discendenza a sinistra della freccia ed a destra, nel nostro caso dice che se Aldo è discendente di Franco potrebbe essere che l'ho appena derivato, se trovo che Sergio è il papà di franco e sono vere tutte e due le condizioni nella regola, allora posso dire che Aldo è un discendente di Sergio. Ma se adesso ho Aldo è discende di Sergio ed in paternità ho anche il padre di Sergio posso andare avanti ad applicare la regola, in questo modo continuo ad aggiungere tuple a discendenza:

$$\begin{aligned} Discendenza(Antenato : p, Discendente : f) \leftarrow \\ Paternita(Padre : p, Figlio : f) \\ Discendenza(Antenato : a, Discendente : f) \leftarrow \\ Paternita(Padre : a, Figlio : f), \\ Discendenza(Antenato : f, Discendente : d) \end{aligned}$$

Questa ricorsione finisce perchè le tuple in paternità sono finite, quindi ho la garanzia che anche discendenza è finita.

Esempio di passo di esecuzione:

$$\begin{aligned} Discendenza(Antenato : Franco, Discendente : Aldo) \leftarrow \\ Paternita(Padre : Franco, Figlio : Aldo) \\ Discendenza(Antenato : Sergio, Discendente : Aldo) \leftarrow \\ Paternita(Padre : Sergio, Figlio : Franco), \\ Discendenza(Antenato : Franco, Discendente : Aldo) \end{aligned}$$

Ora guardiamo la sintassi di SQL:

```
WITH RECURSIVE Discendenza (Antenato, Discendente) AS
(    SELECT Padre, Figlio
     FROM Paternita
UNION
     SELECT D.Antenato, Figlio
     FROM Discendenza D, Paternita
     WHERE D.Discendente = Padre)
SELECT *
FROM Discendenza
```

Si fa con lo statement WITH RECURSIVE, che dice che discendenza è la parte ricorsiva che vogliamo usare, e vediamo che all'interno abbiamo un unione. La parte tra parentesi è la parte dell'espressione ricorsiva. Se notiamo la parte prima dell'unione è il caso base della ricorsione, la parte dopo l'unione è la vera regola ricorsiva. La prima query è la base perchè ,sto aumentando le tuple di discendenza, la prima dice che se c'è qualcosa che compare in paternità (coppia padre figlio) allora mettila in discendenza, poi non mettiamo solo queste tuple che stanno in paternità, ma metti dentro anche (UNION) le tuple della seconda query, di due colonne, coppie in cui faccio un JOIN, paternità con discendenza, il discendente dev'essere il papà. Alla fine il SELECT * from Discendenza dice di buttare fuori tutte le tuple che ha trovato, viene eseguita solo quando la ricorsione non produce più tuple.

Il motivo per cui non è stato introdotto subito in SQL è perchè ci sono dei rischi, perchè se uno non soddisfa dei vincoli si potrebbero creare dei cicli e delle computazioni potenzialmente infinite. (non ci metterà all'esame query ricorsive).

10.4 Controllo dell'accesso in SQL

Dobbiamo decidere chi puo' accedere ai dati delle nostre tabelle. In SQL è possibile dare dei privilegi, un privilegio e' caratterizzato da:

- Risorsa: relazioni (tabelle), viste, domini, colonne di tabelle
- Utente che concede il privilegio
- Utente che riceve il privilegio
- Azione che viene permessa di fare (insert, select, update, etc)
- La trasmissibilità del privilegio, ovvero se chi lo riceve lo può dare a sua volta a qualcun altro.

Esiste un utente predefinito _system (amministratore della base di dati) che ha tutti i privilegi. Il creatore di una risorse ne è in genere il proprietario (owner) e ha tutti i privilegi su di essa (funziona come la creazione dei file in Linux).

10.4.1 Concessione di privilegi

Per concedere un privilegio, dobbiamo usare l'istruzione GRANT:

```
GRANT <privilegi | ALL PRIVILEGES>
 ON risorsa TO utente [WITH GRANT OPTION]
```

La GRANT OPTION specifica se il privilegio può essere trasmesso ad altri utenti:

```
GRANT ALL PRIVILEGES ON persone TO segreteria
```

Questa istruzione dice che una persone a cui ho dato i privilegi può a sua volta trasmettere questi privilegi ad altre persone.

10.4.2 Tipi di privilegi

- insert: permette di inserire nuovi oggetti (ennuple) in tabelle e/o viste
- update: permette di modificare il contenuto di tabelle, viste e attributi
- delete: permette di eliminare oggetti
- select: permette di leggere la risorsa
- usage: è poco usato perchè si applica alla definizione dei domini,da il permesso di usare il dominio che ho specificato a qualcun altro.
- references: ha a che fare con il discorso di cosa succede quando cancello una tupla di una tabella di cui sono proprietario ed il sistema me lo vieta, perchè sto violando un vincolo di identità referenziale (politica no action nella definizione delle tabelle). Quindi devo dare un permesso esplicito di dare vincoli di integrità referenziale alla tabella, altrimenti potrei avere i problemi di essere bloccato in alcune operazioni sulle tabelle che ho generato io.

10.4.3 Revoca di privilegi

Si utilizza REVOKE:

```
REVOKE privilegi ON risorsa
FROM utente [RESTRICT | CASCADE]
```

Anche il privilegio di GRANT può essere revocato (il fatto che l'utente possa concedere il permesso a qualcun altro). Solo chi ha concesso i privilegi può sottrarli.

RESTRICT dice che il comando non viene eseguito se la revoca comporta la rimozione di privilegi concessi ad altro o l'eliminazione di oggetti reati tramite tale privilegio (default). CASCADE invece forza la rimozione del privilegio concesso, di quelli concessi per propagazione e di tutti gli oggetti creati sulla base di tale privilegio "a catena".

10.4.4 Autorizzazioni

Per dare autorizzazione su solo alcune tuple lo faccio combinando la definizione di una vista con la concessione di privilegi su quella vista. La vista può essere complicata quanto vogliamo. Quindi le risorse di cui stiamo parlando non sono solo tabelle.

10.4.5 Vincoli di integrità generici: CHECK

Specifica di vincoli di ennupla e anche vincoli più complessi, non sempre supportati. Un vincolo ulteriore l'abbiamo già visto negli attributi impliciti, in cui dobbiamo fare attenzione che il valore rispetti alcuni valori.

CHECK (Condizione)

Condizione e' una qualisasi condizione che puo' comparire nella clausola where.

Possiamo anche usare il CHECK per definire dei valori di chiave o not null:

```
CREATE TABLE Impiegato(
Matricola CHARACTER(6)                                  Matricola CHARACTER(6) PRIMARY KEY,
        CHECK ( Matricola IS NOT NULL AND
        1 = (SELECT COUNT(*) FROM Impiegato I WHERE Matricola = I.Matricola)),

  Cognome CHARACTER(20)                                 Cognome CHARACTER(20) NOT NULL,
        CHECK (Cognome IS NOT NULL),

Nome CHARACTER(20),
Sesso CHARACTER NOT NULL CHECK (Sesso in ('M','F'))

Dipartimento CHARACTER(15)          Dipartimento CHARACTER(15) REFERENCES Dipartimento(NomeDip)),
        CHECK (Dipartimento IN (SELECT NomeDip FROM Dipartimento))

Stipendio INTEGER,
Ritenute INTEGER,                   ❑ I vincoli predefiniti sono più leggibili
Netto INTEGER,                      ❑ Il sistema riconosce i vincoli e può ottimizzarli
CHECK (Netto = Stipendio -  Ritenute)
```

Nell'ultimo esempio c'è un attributo esplicito ma ridondante, voglio che il netto è uguale al lordo meno le ritenute.

```
CREATE TABLE Impiegato(
    Matricola CHARACTER(6) PRIMARY KEY,
    Cognome CHARACTER(20) NOT NULL,
    Nome CHARACTER(20),
    Sesso CHARACTER NOT NULL CHECK (Sesso IN ('M','F'))
    Stipendio INTEGER,
    Ritenute INTEGER,
    Netto INTEGER,
    CHECK (Netto = Stipendio - Ritenute)
)

INSERT INTO Impiegato VALUES (1 , 'Rossi', 'Mario', '', 100, 20, 80);   No : viola il vincolo M o F
INSERT INTO Impiegato VALUES (2 , 'Neri', 'Mario', 'M', 100, 10, 80);   No : 100 – 10 = 90
INSERT INTO Impiegato VALUES (3 , 'Rossini', 'Luca', 'M', 70, 20, 50);  Sì
```

```
CREATE TABLE Impiegato (
    Matricola INTEGER PRIMARY KEY,
    Cognome CHARACTER(20),
    Nome CHARACTER(20),
    Sesso CHARACTER NOT NULL CHECK (sesso IN ('M','F')) ,
    Stipendio INTEGER CHECK (Stipendio > 0) ,
    Superiore INTEGER,
    CHECK (Stipendio <= (SELECT Stipendio
                         FROM Imp J WHERE Superiore = J.Matricola))
)
```

- La maggior parte dei DBMS non supporta subquery nella clausola CHECK
  - Non ammette il confronto di una ennupla con un insieme di altre ennuple

10.5 Trigger in SQL

E' stato introdotto in standard successivi dell'SQL. I DBMS che usano questi trigger si chiamano basi di dati attive, perchè sono eventi che scatenano delle reazioni. Seguono un paradigma ECA (Evento Condizione Azione), quando un evento si verifica se la condizione è vera allora l'azione è eseguita. Praticamente le azioni sono delle modifiche o cancellazioni di tuple, ad esempio se qualcuno modifica lo stipendio di un impiegato fai queste azioni.

Ogni trigger è caratterizzato da:

- Nome
- target: Tabella che il trigger controlla
- modalità

- evento: inserimento, cancellazione o aggiornamento
- azione
- granularità

10.5.1 Sintassi

Le cose comuni ai DBMS sono create trigger con il nome del trigger, poi dobbiamo dire se valutare le condizioni prima o dopo fare l'aggiornamento.

```
create trigger NomeTrigger
    {before | after}
    {insert | delete | update[of Column]} on TabellaTarget
    [referencing
        {[old table [as] VarTupla]
         [new table [as] VarTuplaNew]}
        {[old [row] [as] VarTupla]
         [new [row] [as] VarTuplaNew]}]
    [for each {row | statement}]
    [when Condizione]
    StatementProcedureleSQL
```

10.5.2 Tipi di eventi

- BEFORE:
  - Il trigger e' considerato e possibilmente eseguito prima dell'evento
  - I Trigger before non possono modificare lo stato del database; possono al piu' condizionare i valori "new" in modalita' row-level
  - Normalmente questa modalita' e' usata quando si vuole verificare una modifica prima che essa avvenga e "modificare la modifica"
- AFTER:
  - Il trigger e' considerato e eseguito dopo l'evento
  - E' la modalita' piu' comune, adatta alla maggior parte delle applicazioni

"Conditioner" (agisce prima dell'update e della verifica di integrita'):

```
create trigger LimitaAumenti
before update of Salario on Impiegato
for each row
when (New.Salario > Old.Salario * 1.2)
set New.Salario = Old.Salario * 1.2
```

"Re-installer" agisce dopo l'update:

```
create trigger LimitaAumenti
after update of Salario on Impiegato
for each row
when (New.Salario > Old.Salario * 1.2)
set New.Salario = Old.Salario * 1.2
```

10.5.3 Granularità degli eventi

Possiamo specificare for each statement (viene eseguito una volta sola per lo statement che lo ha generato, non per tutte le tuple) o for each row.

```
create trigger AccountMonitor
after update on Account
for each row
when new.Totale > old.Totale
insert values
    (new.NumeroConto,
     new.Totale-old.Totale)
     into Pagamenti
```

In questo esempio se il nuoto totale è maggiore del vecchio totale vado ad inserire un nuovo pagamento specificando il numero del conto, e di quale cifra su un'altra tabella. Quindi in base all'azione che succede su qualche tabella possiamo andare a modificare le informazioni di altre tabelle.

```
create trigger ArchiviaFattureCancellate
after delete on Fattura
referencing old_table as VecchieFatture
insert into FattureCancellate
(select *
from VecchieFatture)
```

Delete significa che stiamo monitorando gli eventi di cancellazione su una specifica tabella. References significa che chiamo VecchieFatture la tabella che si chiama old_table, a questo punto metto dentro una tabella FattureCancellate le fatture che ho eliminato. Non sto facendo un trigger che controlla riga per riga ma controllo tutta la tabella.

10.5.4 Esecuzione Trigger in conflitto

Possiamo mettere diversi trigger per uno stesso evento, ma se queste azioni sono in contraddizione? C'è una priorità con cui li controlla:

- prima i before
- poi viene eseguita la modifica e verificati i vincoli di integrità
- Infine sono eseguiti gli after

Se abbiamo due trigger appartengono alla stessa categoria, l'ordine di esecuzione è definito in basse al loro timestamp di creazione (i trigger più vecchi hanno priorità più alta).

Chapter 11

Normalizzazione

E' un processo, un attivita' che dovremo fare come professionisti. E' una specie di certificato di qualita', la base di dati dev'essere in forma normale. Una forma normale e' una proprieta' di una base di dati relazionale che ne garantisce la qualita', cioe' l'assenza di determinati difetti. Quando una relazione non e' normalizzata:

- Presenta ridondanze
- Si presta a comportamenti poco desiderabili durante gli aggiornamenti

E' una procedura che permette di trasformare schemi non normalizzati in schemi che soddisfano una forma normale. Quindi noi vorremmo il database in una forma che evita le ridondanze. La normalizzazione va utilizzata come tecnica di verifica dei risultato della progettazione di una base di dati, non e' una metodologia per la progettazione.

Impiegato	Stipendio	Progetto	Bilancio	Funzione
Rossi	20	Marte	2	tecnico
Verdi	35	Giove	15	progettista
Verdi	35	Venere	15	progettista
Neri	55	Venere	15	direttore
Neri	55	Giove	15	consulente
Neri	55	Marte	2	consulente
Mori	48	Marte	2	direttore
Mori	48	Venere	15	progettista
Bianchi	48	Venere	15	progettista
Bianchi	48	Giove	15	direttore

Anomalie:

- Lo stipendio di ciascun impiegato e' ripetuto in tutte le ennuple relative (ridondanza)
- Se lo stipendio di un impiegato varia, e' necessario andarne a modificare il valore in diverse ennuple (anomalia di aggiornamento)
- Se un impiegato interrompe la partecipazione a tutti i progetti, dobbiamo cancellarlo (anomalia di cancellazione)
- Un nuovo impiegato senza progetto non puo' essere inserito (dipendenza funzionale)

Perche' abbiamo questi fenomeni indesiderabili? abbiamo usato una relazione unica per rappresentare delle informazioni eterogenee, come ad esempio gli impiegati con i relativi stipendi, i progetti con i relativi bilanci e la partecipazione degli impiegati ai progetti con le relative funzioni.

11.1 Dipendenza funzionale

Per studiare in modo sistematico questi aspetti, è necessario introdurre un vincolo di integrità, la dipendenza funzionale. Le proprietà del dominio nel nostro esempio sono:

- Ogni impiegato ha un solo stipendio (anche se partecipa a più progetti)
- Ogni progetto ha un solo bilancio
- Ogni impiegato in ciascun progetto ha una solo funzione (anche se può avere funzioni diverse in progetti diversi)

Se abbiamo una relazione r su uno schema R(x), e prendiamo due sottoinsiemi non vuoti Y e Z di X. Allora esiste in r una dipendenza funzionale (FD) da Y a Z se, per ogni coppia di tuple t_1, t_2 di r con gli stessi valori su Y, risulta che t_1, t_2 hanno gli stessi valori anche su Z.

Notazione:

$$Y \rightarrow Z$$

Nel nostro esempio:

$$\begin{aligned} \text{Impiegato} &\rightarrow \text{Stipendio} \\ \text{Progetto} &\rightarrow \text{Bilancio} \\ \text{Impiegato Progetto} &\rightarrow \text{Funzione} \end{aligned}$$

11.1.1 Proprieta'

Non vale la proprieta' riflessiva ma vale la proprieta' transitiva.

11.1.2 Banali

Y determina Z se Z e' un sottoinsieme di Y allora e' banale.

Se invece c'e' almeno un attributo a destra che non e' uguale anche a sinistra allora non e' banale.

11.2 Forma normale di Boyce e Codd (BCNF)

Una relazione r e' in forma normale di BCNF se, per ogni dipendenza funzionale (non banale) $X \rightarrow Y$ definita su di essa, X e' una superchiave di r. (devo controllare per ciascuna delle dipendenze funzionali che la parte sinistra alla freccia sia una superchiave).

Questa forma normale e' molto forte, ma in alcuni casi potrebbe essere impossibile, quindi esistono altre forme normali che sono un po' piu' deboli, ma abbiamo la garanzia di arrivarci sempre.

Cosa facciamo se quando troviamo una dipendenza funzionale notiamo che la parte a sinistra non e' una superchiave (quindi la relazione non è in forma normale BCNF)? Dobbiamo decomporre le relazioni sulla base delle dipendenze funzionali, al fine di separare i concetti.

Impiegato	Stipendio
Rossi	20
Verdi	35
Neri	55
Mori	48
Bianchi	48

Impiegato	Progetto	Funzione
Rossi	Marte	tecnico
Verdi	Giove	progettista
Verdi	Venere	progettista
Neri	Venere	direttore
Neri	Giove	consulente
Neri	Marte	consulente
Mori	Marte	direttore
Mori	Venere	progettista
Bianchi	Venere	progettista
Bianchi	Giove	direttore

Progetto	Bilancio
Marte	2
Giove	15
Venere	15

Impiegato	Progetto	Sede
Rossi	Marte	Roma
Verdi	Giove	Milano
Verdi	Venere	Milano
Neri	Saturno	Milano
Neri	Venere	Milano

Impiegato	Sede
Rossi	Roma
Verdi	Milano
Neri	Milano

Progetto	Sede
Marte	Roma
Giove	Milano
Saturno	Milano
Venere	Milano

A volte il problema e' che se facciamo la decomposizione e poi con un interrogazione mettiamo insieme le informazioni che abbiamo spezzato (JOIN) troviamo delle tuple che nella relazione di prima non c'era.

Impiegato	Sede
Rossi	Roma
Verdi	Milano
Neri	Milano

Progetto	Sede
Marte	Roma
Giove	Milano
Saturno	Milano
Venere	Milano

Impiegato	Progetto	Sede
Rossi	Marte	Roma
Verdi	Giove	Milano
Verdi	Venere	Milano
Neri	Saturno	Milano
Neri	Venere	Milano
Verdi	Saturno	Milano
Neri	Giove	Milano

Dobbiamo quindi decomporre la relazione senza perdite. Una relazione r si

decompone senza perdita su X_1 e X_2 se il join delle proiezioni di r su X_1 e X_2 è uguale a r stessa (cioè non contiene ennuple spurie). La decomposizione senza perdita è garantita se gli attributi comune contengono una chiave per almeno una delle relazinoi decomposte. Dobbiamo decomporre in modo che l'attributo in comune sia una chiave di almeno una delle relazioni che creiamo.

Impiegato	Progetto	Sede
Rossi	Marte	Roma
Verdi	Giove	Milano
Verdi	Venere	Milano
Neri	Saturno	Milano
Neri	Venere	Milano

Impiegato	Sede
Rossi	Roma
Verdi	Milano
Neri	Milano

Impiegato	Progetto
Rossi	Marte
Verdi	Giove
Verdi	Venere
Neri	Saturno
Neri	Venere

Impiegato → Sede
Progetto → Sede

Non basta fare questa cosa, ma dobbiamo anche conservare le dipendenze funzionali (ad esempio che progetto determina sede), se facciamo la decomposizione come abbiamo fatto non possiamo verificarlo, perche' non c'e' più nessuna tabella che mi parla di progetto e sede insieme. Quindi una decomposizione conserva le dipendenza se ciascuna delle dipendenze funzionali dello schema originario coinvolge attributi che compaiono tutti insieme in uno degli schemi decomposti.

Le due proprietà da verificare:

- Decomposizione senza perdite: Fare un JOIN per vedere se trovo altre tuple non è una soluzione efficace. Ma si fa se gli attributi comuni contengono una chiave di una delle relazioni decomposte
- Conservazione delle dipendenze, che garantisce il mantenimento dei vincoli di integrità originari

Dirigente	Progetto	Sede
Rossi	Marte	Roma
Verdi	Giove	Milano
Verdi	Marte	Milano
Neri	Saturno	Milano
Neri	Venere	Milano

Progetto Sede → Dirigente
Dirigente → Sede

Abbiamo una nuova relazione non normalizzata con due relazioni funzionali, se ho due tuple con progetto e sede allora avranno lo stesso valore di dirigente e che il dirigente implica la sede (un dirigente lavora solo in una sede). Ma possiamo decomporre questa relazione per portarla in forma normale? Se dobbiamo mantenere tutte le dipendenze notiamo che la prima relazione funzionale parla di tutti gli attributi, quindi non posso proprio decomporre questa relazione. Le dipendenze funzionali sono dei veri e propri vincoli dato che la dipendenza funzionale mi dice che per ogni dirigente devo avere la stessa sede, potrei anche definire una chiave dirigente progetto avendo la garanzia, data dai vincoli funzionali, che non possono esistere istanze uguali.

11.3 Terza forma normale (3NF)

Una relazione r è in TERZA FORMA NORMALE se, per ogni FD (non banale) $X \to X$ definita su r, vale:

- X è una superchiave di r OPPURE
- ogni attributo in Y (che non appare in X) è contenuto in almeno una chiave di r

Controlliamo la stessa cosa per la BCNF (che la parte a sinistra sia una superchiave), se non lo è diamo un'occhiata alla parte destra, ogni attributo nella parte destra deve essere contenuto in almeno una chiave della relazione. La 3NF è meno restrittiva della forma normale BCNF, notiamo quindi che se una relazione è in BCNF allora sarà anche in 3NF. La 3NF ha il vantaggio di essere sempre raggiungibile e si può dimostrare che se una ralazione ha una sola chiave, allora essa è in BCNF se e se se è in 3NF.

11.3.1 Decomposizione in terza forma normale

Per verificare se una relazione è in terza forma normale prendiamo sempre una dipendenza funzionale alla volta, prima vediamo se la parte sinistra è una superchiave, se lo è tutto ok, altrimenti guardiamo a destra (della freccia) che dev'essere contenuto in una chiave della relazione.

Se invece la relazione non è in forma normale:

- Si crea una relazione per ogni gruppo di attributi coinvolti in una dipendenza funzionale
- Si verifica che alla fine (almeno) una relazione contenga una chiave della relazione originaria
- Dipende dalle dipendenze individuate

Impiegato	Progetto	Stipendio
Rossi	Marte	30 000
Verdi	Giove	30 000
Verdi	Venere	30 000
Neri	Saturno	40 000
Neri	Venere	40 000

Impiegato → Stipendio

Consideriamo una relazione di partenza (impiegato, progetto, stipendio), non è in BCNF perchè impiegato non è superchiave, perchè la chiave sarà impiegato, progetto e dato che la superchiave è questa, stipendio (la parte a destra della dipendenza funzinale) non fa parte di una chiave e quindi questa relazione non è neanche in terza forma normale. Quindi come decomponiamo, siccome ho una sola dipendenza funzionale mettiamo insieme impiegato stipendio e dall'altra parte mettiamo la chiave della relazione originaria. Ora dobbiamo controllare che abbiano conservato le dipendenze (ovvio c'è una sola tabella con la dipendenza che c'era prima) e che non abbia perdite, per farlo controlliamo che l'attributo in comune sia una chiave di una relazione.

Impiegato	Progetto
Rossi	Marte
Verdi	Giove
Verdi	Venere
Neri	Saturno
Neri	Venere

Impiegato	Stipendio
Rossi	30 000
Verdi	30 000
Neri	40 000

Strategia:

- Se la relazione non è normalizzata si decompone in terza forma normale
- Alla fine si verifica se lo schema ottenuto è anche non BCNF

Dirigente	Progetto	Sede
Rossi	Marte	Roma
Verdi	Giove	Milano
Verdi	Marte	Milano
Neri	Saturno	Milano
Neri	Venere	Milano

Dirigente → Sede
Progetto Sede → Dirigente

Abbiamo già visto che questa relazione non è in forma BCNF ma è in terza forma normale perchè sede fa parte di una chiave. Noi potremmo cercare di cambiare i vincoli del dominio per riorganizzare la relazione introducendo altri attributi in modo da poter poi decomporre la relazione. Nel nostro esempio potremmo aggiungere l'attributo reparto che partiziona le sedi per reparto e dirigente.

Dirigente	Progetto	Sede	Reparto
Rossi	Marte	Roma	1
Verdi	Giove	Milano	1
Verdi	Marte	Milano	1
Neri	Saturno	Milano	2
Neri	Venere	Milano	2

Dirigente → Sede Reparto
Sede Reparto → Dirigente
Progetto Sede → Reparto

A questo punto saltano fuori delle nuove dipendenze funzionali, ad esempio nella prima a destra della freccia ci sono due attributi ci dice che per ogni dirigente c'è una sola sede con uno specifico reparto, la seconda dice che per ogni sede e reparto c'è un solo dirigente; la terza dice che un progetto in una certa sede viene svolto in un solo reparto (questa sarà sempre verificata perchè progetto-sede è una chiave). Per dire che questa relazione è in BCNF devo guardare tutte le dipendenze funzionali, guardiamole una alla volta, la prima dirigente dev'essere una superchiave, non lo è, quindi non è un BCNF. Non abbiamo lo stesso problema di prima che una dipendenza funzionale coinvolge tutti gli attributi. Decomponiamo con la nostra tecnica di farci guidare dalle dipendenze funzionali.

Dirigente	Sede	Reparto
Rossi	Roma	1
Verdi	Milano	1
Neri	Milano	2

Progetto	Sede	Reparto
Marte	Roma	1
Giove	Milano	1
Marte	Milano	1
Saturno	Milano	2
Venere	Milano	2

Nota: Sede,Reparto è un'altra chiave della prima relazione

La decomposizione è senza perdita e conserva le dipendenze

Chapter 12

Non standard DBMS

I database relazionali hanno avuto un grande successo di lunga durata ed hanno ancora un mercato enorme. Garantiscono:

- Linguaggio di query dichiarativo ed espressivo (SQL)
- Integrità dei dati
- Autenticazione e controllo dell'accesso
- Supporto per backup e recovery
- Tutte le proprietà ACID delle transazioni

Però già da un po' di anni sono nati altre esigenze di gestire dei tipi di dati che non sono quelli tabellari classici, ma sono dati multimediali, spazio-temporali (tutti i dati che arrivano dai dispositivi mobili, geolocalizzazione), dati con struttura particolare (chiave-valore, XML,...) o non uniformemente strutturati.

Oltre a diversi tipi di dati è aumentata anche la quantità, big data, ricerche su miliardi di record (social network). Tendenzialmente sono tutte strutture distribuite, ovvero che i db non stanno su un unico server.

12.1 Basi di dati multimediali

Tipi di dati multimediali:

- Immagini
- Audio
- Video
- Documenti
- Annotazioni

12.1.1 Interrogazioni

Possono essere interrogazioni sulla parte strutturata dei dati (i metadati) e quindi come le facciamo adesso con l'SQL. Poi ci sono altre interrogazioni più sofisticate che riguardano condizioni di somiglianza ed i risultati possono essere di tipo probabilistico. Sono utili tecniche adattive, che permettono all'utente di arrivare al risultato.

12.1.2 Ricerca in documenti

Qui stiamo parlando di ricerche all'interno dei documenti testuali. Ci sono quindi delle tecniche per indicizzare i documenti, come ad esempio trovare delle parole chiave che dicono quello che c'è dentro al documento:

- esclusione di parole irrilevanti
- riduzione a parole chiave
- considerare delle frequenze delle parole

12.2 Gestione di dati spaziali e spazio-temporali

Ci sono dei DBMS molto utili per la gestione del territorio, che sono i GIS (Geographical Information System). Di fatto immaginiamo come dei DBMS ma i dati che manipolano sono dati di informazioni geografica, delle rappresentazioni vettoriali delle mappe. Potrei avere una serie di layer per poter progettare meglio le varie strutture. Chi progetta il DBMS deve progettare delle strutture dati ad hoc per rappresentare queste strutture.

12.3 Gestione di dati spaziali e spazio-temporali

Il trattamento di informazione temporale richiede anch'esso supporto nativo dei DBMS. Esempi di interrogazioni interessanti:

- Quando la traiettoria dell'aereo A è arrivata più vicina di 500 mt alla traiettoria dell'aereo B? Dove erano gli aerei?
- Quale tra i miei amici si trovano in questo momento nello stesso quartiere dove sono io?

12.3.1 Estensioni spaziali dei RDBMS

Molti DBMS relazionali offrono estensioni per trattare dati spaziali con supporto efficiente delle relative query.

- Oracle spatial
- DB2 spatial extender
- PostGIS

12.3.2 Estensione temporale per i RDBMS

Siccome è stato riconosciuto che il tempo non è come qualsiasi altro attributo, per i bambini ci vogliono degli anni per capire le convenzioni sociali costruite per capire il tempo e le relazioni temporali. Hanno aggiunto all'SQL la possibilità di rappresentare in un modo esplicito il tempo, è diverso da dire che è semplicemente una data:

```
CREATE TABLE Emp(
    ENo INTEGER,
    EStart DATE,
    EEnd DATE,
    EDept INTEGER,
    PERIOD FOR EPeriod (EStart, EEnd)
)
```

In questo caso stiamo dicendo al DB che c'è una data di inizio ed una data di partenza che sono di tipo data, l'ultima riga dice al DBMS che le righe di prima definiscono un intervallo di tempo, è un concetto, un meta-dato.

Il tempo di validità (tempo di applicazione) è il tempo in cui l'informazione è vera (Claudio ha una macchina rossa dal tal anno al tal anno, indipendentemente da quando ho messo questa informazione nel DB).

Per popolare questa tabella:

```
INSERT INT Emp VALUES
    (22217,
    DATE '2012-01-10',
    DATE '2013-11-12', 3)
```

ENo	EStart	EEnd	EDept
22217	2012-01-01	2013-11-12	3

L'unica cosa a cui dobbiamo stare attenti che adesso il DBMS sa che la data di inizio dev'essere precedente alla data di fine.

Se qui dobbiamo fare delle operazioni di aggiornamento possiamo non preoccuparci più dell'intervallo, ma lasciamo che sia il sistema a gestirli, noi semplicemente gli diciamo di voler aggiornare l'intervallo. Quello che ha dovuto fare è stato spezzare l'intervallo precedente .

```
UPDATE Emp FOR PORTION OF EPeriod
    FROM DATE '2012-02-03'
    TO DATE '2012-09-10'
    SET EDept = 4
    WHERE ENo = 22217
```

ENo	EStart	EEnd	EDept
22217	2012-01-01	2012-02-03	3
22217	2012-02-03	2012-09-10	4
22217	2012-09-10	2013-11-12	3

Sono ammessi anche operatori di confronto tra intervallo di tempo, che non centrano niente con gli operatori insiemistici, c'è un'algebra degli intervalli che dice quali sono tutte le possibili relazioni tra gli intervalli di tempo, ad esempio l'OVERLAP, se ho due intervallo che un pezzo in comune ed uno inizia ad un pezzo dell'altro è un OVERLAP.

```
SELECT EName, EDept FROM Emp
WHERE ENo = 22217 AND
 EPeriod OVERLAPS
     PERIOD(DATE '2012-01-02'
            DATE '2013-01-01')
```

Questa interrogazione chiede di dare il nome degli impiegati e del dipartimento, prendendoli dalla relazione impiegati, solo dell'impiegato 22217, ma nel WHERE c'è una condizione temporale, dice che il periodo di validità che riguarda l'impiegato ed il dipartimento deve intersecare il periodo che mi interessa.

12.3.3 Tempo di transazione

Le tabelle possono essere di system versioned, cioè avere due attributi che indicano per ogni tupla il tempo al quale è stata inserita e il tempo al quale è stata cancellata, mantenendo così una storia completa del BD (come se recuperassi dei file da un backup).

Questa cosa si ottiene con la dichiarazione del meta-attributo PERIOD FOR SYSTEM_TIME (tempo di inizio, tempo di fine) WITH SYSTEM VERSIONING (come opzione del CREATE TABLE).

Una tabella bi-temporale vuol dire che ci sono due tempi ed i due tempi di cui abbiamo parlato sono i tempi di validità e tempi di transazione, significa quindi che in questa tabella ci posso mettere tutte e due queste informazioni. Chiaramente una tabella così sarà molto costosa da tenere perchè così non cancelleremmo nessuna tupla, quindi un DB con solo tabelle così cresce soltanto e non può diminuire di dimensione.

```
CREATE TABLE Emp(
ENo INTEGER,
EStart DATE,  EEnd DATE,
EDept INTEGER,
PERIOD FOR EPeriod (EStart, EEnd),
Sys_start TIMESTAMP(12) GENERATED ALWAYS AS ROW START,
Sys_end TIMESTAMP(12) GENERATED ALWAYS AS ROW END,
EName VARCHAR(30),
PERIOD FOR SYSTEM_TIME (Sys_start, Sys_end)
) WITH SYSTEM VERSIONING
```

12.4 NoSQL DBMS

Parliamo ora di basi di dati non relazionali. NoSQL = "Not Only SQL". Quand'è che possiamo scegliere altro rispetto ai DBMS relazionali:

- ho bisogno di uno schema flessibile, magari abbiamo tanti valori nulli perchè cambia nel tempo. Oppure vogliamo sviluppare un'app con dati non molto strutturati.
- Semplicità e facilità di costruzione di DB
- Alta scalabilità
- Rinunciare alle transazioni
- Rinuncio ad un linguaggio ricco come l'SQL, nei DB che vediamo oggi spesso le interrogazioni le facciamo all'interno del linguaggio di programmazione

12.4.1 Esempio: Web loggin

Queste tecnologie sono state inventate per essere utilizzate nel web. In questa app vogliamo memorizzare un'insieme di recordo che corrispondono a cosa l'utente stava chiedendo sul web, (UserID, URL, timestamp, altraInfo). In questo caso le interrogazioni da fare sono molto semplici, voglio solamente trovare tutti i record di un'entità. Non serve la potenza dell'SQL per query di questo genere. Può essere parallelizzabile, se ho dati semplici e tanti, posso distribuire questi dati su server diversi e fare interrogazione che vanno a prendere questi dati in parallelo (parallelizzo tanti server).

12.4.2 Nuovi modelli dei dati

Oggi ci sono diverse centinaia di DBMS tra i quali possiamo scegliere. Le 3 categorie principali:

- Relazionali

- NoSQL
- newSQL: introdotti da GOOGLE, perchè cercano di prendere la capacità di parallelizzare in modo massivo le interrogazioni ma anche alcune proprietà di garantire le transazioni ed un po' di flessibilità nello schema.

Web 2.0 e mobile computing richiedono operazioni su larga scala (tanti dati), ma queste info sono di insert e read, meno di update e delete, ovviamente è più facile gestire un DB in cui bisogna solo leggere o inserire.

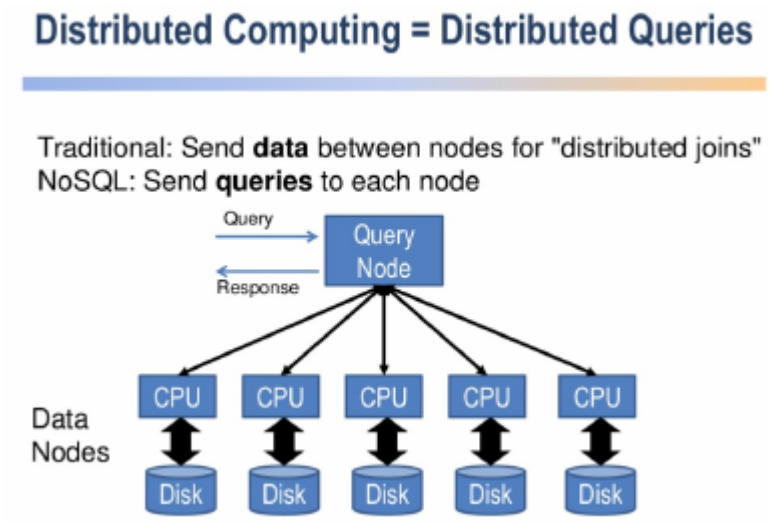
12.4.3 Scalabilità

Aumentano gli utenti e di conseguenza aumentano i numeri delle interrogazioni. Viene ottenuta attraverso architetture con molti nodi (computer), con un approccio share nothing, vuol dire che ciascuno di questi nodi è un computer indipendente (con tutti i componenti) e poi semplicemente comunica con gli altri, è il contrario di avere un pc con più CPU e gli altri componenti in comune. Risultano maggiormente scalabili dei sistemi tradizionali (se i dati sono partizionabili e distibuibili automaticamente in modo da avere accessi bilanciati e le operazioni sono "localizzabili"), non è che per qualsiasi problema informatico sia facile distribuire il calcolo su migliaia di nodi, ci vuole un'impostazione del DB che suggerisce come fare questa cosa.

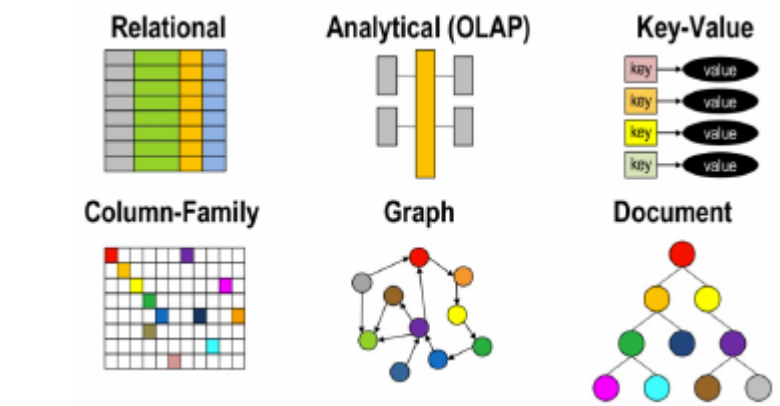
Operazioni locali

La scalabilità di letture è ottenuta (anche) attraverso la replicazione, se ho tanti computer ed ho creato tante copie dei dati per le letture basta avere queste copie per eseguire la lettura in contemporanea. Per le scritture è un problema, quello che si usa è la tecnica di shardgin, big table di google, questa tabella è partizionata e distribuita su tanti nodi del data center con qualche tecnica furba che riesce a prelevare i dati.

I database distribuiti relazionali esistono, ma esistendo l'operatore di JOIN il quale deve prendere tabelle diverse e se dovessimo fare un JOIN su tabelle che sono in computer diversi c'è bisogno che questi computer comunichino. Nel noSQL la stessa interrogazione viene mandata a tutti i nodi, i quali non comunicano tra loro, fanno l'interrogazione e restituiscono, quindi si risparmia molto tempo.



12.4.4 Famiglie



- Relational
- Analytical
- Key-value
- Column-family
- Graph
- Document-Base

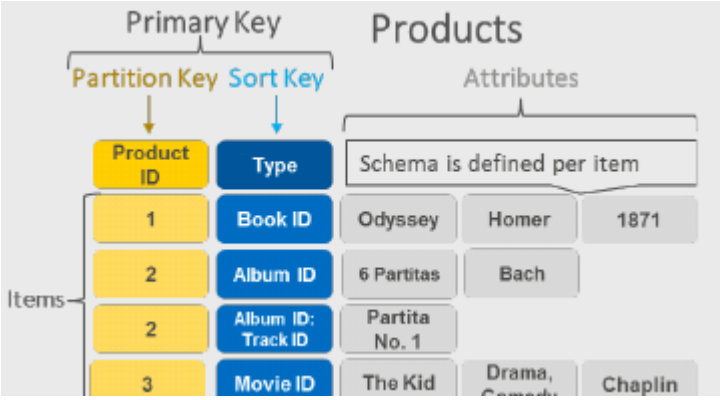
Sono tutti modelli logici.

Key-Value

Modello di dati a coppie (chiave valore). Esempi:

- Amazon DynamoDB (tra i vari software che ci mette a disposizione il cloud di amazon ci sono anche dei DBMS)

Le ricerche si fanno solo per il campo chiave, le operazioni sono INSERT(key, value), FETCH(key), UPDATE(key), DELETE(key). Sia le chiavi che i valori possono non essere oggetti semplici ma strutturati (e i valori possono avere struttura diversa da un elemento all'altro). L'idea alla base di questi DBMS è di rendere facile la vita al programmatore. Questa cosa corrisponde anche alla maggior probabilità di rendere inconsistenti le cose.

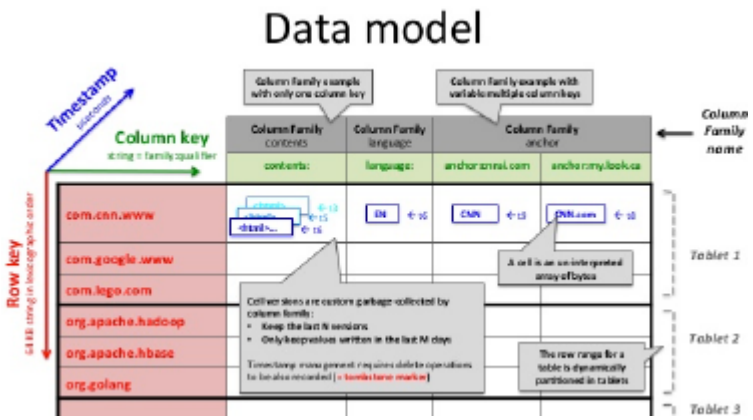


Document-Base

Memorizza gli oggetti in una struttura gerarchica dei dati che strutturiamo noi, spesso JSON, organizzati in collezioni (ma senza vincoli di schema all'interno delle collezioni). (MongoDB)

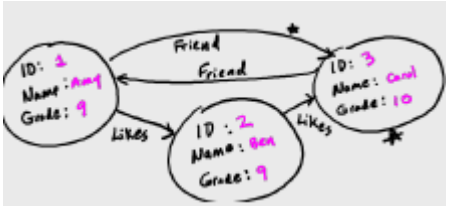
Column-Family

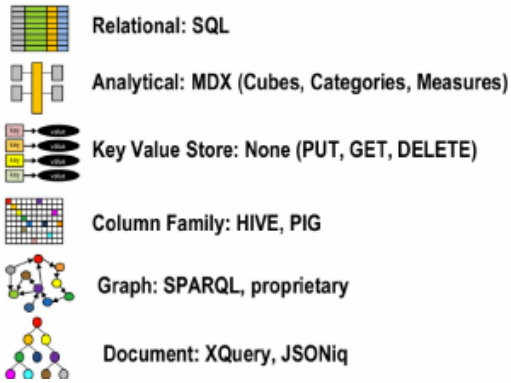
Sono molto simili a quelli chiave valore (big table di google). Il molto di dati è quello di una tabella (non vuol dire che poi faccio il JOIN tra tabelle, le interrogazioni sono fatte su quella tabella, non con altre tabelle) con nome e formato delle colonne che sono variabili per ogni riga. Alcune colonne sono raggruppate in famiglie con valori omogenei di solito utilizzati insieme. Le interrogazioni tipiche sono con chiave di riga o colonna.



Basati sui grafi

Il più conosciuto è Neo4j. La struttura è a grafo sembrerebbe la rappresentazione più naturale per i social network. Memorizza nodi che descrivono gli oggetti (con proprietà, includo ID) ed archi che descrivono relazioni tra oggetti.



12.4.5 Query language

Non ci sono dei linguaggi standard, quando scegliamo un noSQL dobbiamo andare a cercare qual'è il linguaggio da utilizzare per interrogarlo.