

Basi di Dati

Appunti delle Lezioni di Basi di Dati

Anno Accademico: 2024/25

Giacomo Sturm

*Dipartimento di Ingegneria Civile, Informatica e delle Tecnologie Aeronautiche
Università degli Studi “Roma Tre”*

Sorgente del file LaTeX disponibile al seguente link:

<https://github.com/00Darxk/Basi-di-Dati/>

Indice

1	Introduzione	1
2	Modello Relazionale	3
2.1	Relazione	3
2.2	Valore Nullo	4
2.3	Vincoli	4
2.3.1	Intra-Relazionali	4
2.3.2	Inter-Relazionali	5
3	Algebra Relazionale	6
4	Introduzione a SQL	7

1 Introduzione

Per definire cosa sono i dati, si utilizza la definizione del Data Governance Act; per “Data” si intende una qualsiasi rappresentazione di informazione. Una base di dati in generale, invece, rappresenta un insieme organizzato di dati utilizzati per il supporto dello svolgimento delle attività. La maggior parte delle attività moderne si basano su una qualche base di dati. Si possono analizzare dal punto di vista metodologico e tecnologico. La definizione dell’Informatica dall’Accademia di Francia si può notare la presenza di queste due anime. L’informatica viene definita come la scienza del trattamento razionale, specialmente per mezzo di macchine, dell’informazione considerata come supporto alla conoscenza umana e della comunicazione.

I dati quindi nei sistemi informatici, e non solo, sono mezzi per poter gestire dell’informazione, rappresentati in modo essenziale. Molto spesso vengono rappresentati in forma di codice numerico. Sono necessari quindi meccanismi di codifica e decodifica dei dati per poterli rappresentare in forma essenziale e quindi in modo da ottenere informazioni. Un dato può essere considerato quindi un elemento di informazione che deve essere ancora elaborato.

Nello specifico una base di dati rappresenta un insieme organizzato di dati, gestiti da un DBMS. Un “DataBase Management System” rappresenta un sistema che gestisce collezioni di dati, grandi, persistenti e condivise. Si tratta di insiemi grandi, molto di più della memoria fisica dei dispositivi centrali di calcolo, rappresentano il loro limite fisico, e poiché la richiesta di immagazzinare dati è sempre maggiore, si vuole aumentare la disponibilità di memorizzare più dati. I DBMS sono persistenti, poiché le informazioni salvate al loro interno non svaniscono nel tempo, sono importanti vengono salvati su memorie secondarie non volatili. Sono accessibili perché sono condivise e permettono accessi in remoto, essendo generalmente salvati nel cloud.

I DBMS garantiscono privacy, affidabilità, efficienza ed efficacia. I DBMS sono privati, poiché devono garantire che i dati salvati al loro interno siano privati, e siano accessibili solamente quando vengono richiesti. Devono comprendere meccanismi di autorizzazione per mantenere la privacy. Sono affidabili poiché devono poter resistere a malfunzionamenti o attacchi, di tipo hardware o software. I dati sono risorse pregiate e devono poter essere conservati a lungo termine. Per interfacciarsi con un DBMS una tecnica fondamentale consiste nelle transazioni. Queste sono un insieme di operazioni da considerare indivisibili o atomiche, anche concorrenti e di effetto definitivo. Poiché sono operazioni atomiche, possono essere eseguite solo per intero, quando vengono eseguite. Devono essere concorrenti, poiché accedendo allo stesso DB da remote bisogna che l’effetto delle due transazioni concorrenti sia coerente sul DB, senza recare danni o perdita di informazioni da nessuna delle due. I risultati delle transazioni devono essere permanenti ed il loro termine viene identificato da un “commit”, un impegno che segna una conclusione positiva. Una serie di commit quindi mantiene traccia dei risultati in modo definitivo, anche in presenza di guasti o esecuzioni concorrenti. L’efficacia e l’efficienza del DBMS dipende da sistema a sistema.

I progettisti e realizzatori di un DBMS compiono un ruolo diverso dai futuri utilizzatori del DBMS. I primi creano un sistema di gestione, mentre la creazione della base di dati è affidata ad altri progettisti. Questa base di dati verrà utilizzata da altri programmatori per realizzare un’applicazione o programma con cui si potranno interfacciare gli utenti. Gli utenti finali si distinguono in utenti finali, per cui è stata realizzata quella specifica applicazione ed eseguono operazioni predefinite. Mentre utenti casuali eseguono operazioni non previste dal sistema, e possono provocare

errori.

2 Modello Relazionale

Per organizzare i dati all'interno di una base di dati si possono utilizzare diversi modelli o astrazioni dei dati. Il modello dei dati rappresenta un insieme di costrutti attraverso i quali i dati di interesse vengono organizzati ed utilizzati. Il modello relazionale prevede la costruzione di una tabella, ovvero una relazione, che permette di definire insiemi di record o n-uple composte da unità atomiche, chiamate attributi, omogenee.

Questo rappresenta un modello logico dei dati tradizionale, mentre altri modelli più recenti ad oggetti, XML e "NoSQL". Il modello relazionale è stato proposto da E. F. Codd nel 1970 per favorire l'indipendenza dei dati. Questo modello fu implementato in DBMS già nel 1981, poiché non è facile implementare l'indipendenza dei dati con efficienza ed affidabilità. Si basa sul concetto matematico di relazione che trova una naturale rappresentazione per mezzo di tabelle.

2.1 Relazione

Una relazione ρ rappresenta un sottoinsieme del prodotto cartesiano tra due o più domini D_1 e D_2 :

$$\rho \subseteq D_1 \times D_2$$

Dati n insiemi D_i , il loro prodotto cartesiano $D_1 \times \dots \times D_n$ rappresenta l'insieme di tutte le n-uple (d_1, \dots, d_n) tali che per ogni $i = 1, \dots, n$ si ha $d_i \in D_i$. Poiché è un insieme non sono presenti ennuple uguali. Una relazione ρ su questi n insiemi, chiamati domini, rappresenta un sottoinsieme di questo prodotto cartesiano:

$$\rho \subseteq D_1 \times \dots \times D_n \quad (2.1.1)$$

La struttura così definita non è posizionale, poiché a ciascun dominio si associa un nome, attributo o colonna, nell'intestazione della tabella. Le tabelle che rappresentano una relazione l'ordinamento tra le righe è irrilevante, così come l'ordinamento tra le colonne. Una tabella rappresenta una relazione se tutte le righe sono diverse fra di loro e rappresentano una ennupla distinta, le intestazioni delle colonne sono diverse fra di loro, ed i valori di ogni colonna sono omogenei fra di loro. Nelle tabelle ci sono solo valori, si indica anche come modello basato sui valori. I riferimenti fra dati in relazioni diverse sono rappresentati tramite i valori dei domini nelle ennuple.

Una base di dati rappresenta un'insieme di relazioni, dove il suo schema è costituito da tutte le intestazioni, mentre i valori contenuti quindi tutte le righe contenute nelle tabelle rappresentano un'istanza della base di dati.

Per gestire le relazioni useremo il linguaggio SQL, inizialmente acronimo per "Structured English Query Language", ma poi estesa a "Structured Query Language", senza appoggiarsi principalmente sul linguaggio inglese. Rappresenta una lingua di alto livello, che tratteremo approfonditamente in sezioni successive.

Lo schema di una relazione è composto da un nome R ed un insieme X di n attributi A_i :

$$R(A_1, \dots, A_n) \quad (2.1.2)$$

Lo schema di una base di dati R è costituito da uno schema di relazioni R_i :

$$R = \{R_1(X_1), \dots, R_k(X_k)\} \quad (2.1.3)$$

Un'ennupla su un insieme di attributi X viene definita come una funzione che associa a ciascun attributo A in X un valore del dominio di A . Il valore di una singola ennupla t su un attributo A si indica con $t[A]$. Un'istanza di una relazione ρ su uno schema $R(X)$ rappresenta un insieme di ennuple su X . Un'istanza di una base di dati su uno schema $R(X) = \{R_1(X_1), \dots, R_k(X_k)\}$ si definisce come un insieme di relazioni $\rho = \{\rho_1, \dots, \rho_k\}$, dove ogni ρ_i rappresenta un'istanza di una relazione sullo schema di una relazione $R_i(X_i)$.

2.2 Valore Nullo

Questo modello impone ai dati una struttura rigida, solo alcuni formati sono infatti ammessi, ed esclusivamente rappresentati come ennuple che appartengono un certo schema di relazione. Ma la realtà potrebbe non corrispondere alla struttura attesa, quindi i dati ottenuti dalla realtà potrebbero non rappresentare un'ennupla intera. Per cui quando un'ennupla contiene informazioni incomplete, il valore dell'ennupla per quell'attributo è vuoto nella relazione. Per ovviare a questo problema nel modello relazionale, si utilizza un valore convenzionale per rappresentare questo valore vuoto nell'ennupla, si utilizza un valore diverso dai valori del dominio A , chiamato *null*. L'introduzione di questo altro valore è una soluzione semplice, ma efficace ai fini della base di dati. Tutti i domini A sono in grado di accettare un valore *null* per indicare la mancanza del valore nell'ennupla. Per cui data un'ennupla t il suo valore su di un attributo A può essere:

$$t[A] = \begin{cases} \text{dom}(A) \\ \text{null} \end{cases} \quad (2.2.1)$$

Nonostante la semplicità non è una tecnica perfetta, poiché la perdita di informazioni impedisce di effettuare riferimenti per valori tra le relazioni. Su SQL si può utilizzare il comando `NOT NULL` alla creazione di uno schema di relazione.

2.3 Vincoli

Oltre all'assenza di informazioni, è possibile riscontrare errori interni alla base di dati, delle scorrettezze legate all'integrità dei dati. Si possono introdurre vincoli di integrità per rappresentare istanze ammissibili. I vincoli sono delle funzioni booleane, dei predicati, che associa ad ogni istanza un valore vero o falso.

Solo alcuni tipi di vincoli sono integrati nei DBMS, e questi li verificano e ne impediscono la violazione. Per i vincoli supportati invece la verifica spetta all'utente o al programmatore. Questi vincoli possono essere intra-relazionali o inter-relazionali. I vincoli intra-relazionali coinvolgono solamente i valori di una singola relazione, e possono essere sul valore o di dominio, di ennupla, o di chiave.

2.3.1 Intra-Relazionali

I vincoli di dominio impongono condizioni sull'ammissibilità dei valori di un singolo attributo. Possono utilizzare operatori booleani come AND, OR e NOT, ed operazioni di confronto matematiche con una costante.

In SQL data uno schema, si può aggiungere un vincolo con la sintassi `ADD CONSTRAINT`, seguita dalla funzione booleana che definisce il vincolo. Si possono aggiungere ad uno schema di relazione tramite il comando `ALTER TABLE`. Se il vincolo che si prova a definire è violato, non si può definirlo. Convenzionalmente prima viene definito lo schema con i vincoli poi questi vengono verificati ad ogni modifica, ed in caso rifiutata.

I vincoli possono essere di *ennupla*, quando controllano più valori, appartenenti a più attributi, della stessa *ennupla*. Il vincolo di dominio quindi rappresenta un caso particolare del vincolo di *ennupla*. Rappresenta una combinazione booleana di condizioni semplici sui singoli valori di attributi, due attributi o più valori di essi.

Se un insieme K , dominio della relazione, è una chiave, allora per il vincolo di chiave si impone che non possano esistere due *ennuple* uguali su K . Una chiave essendo univoca permette di identificare le *ennuple*, ed è minimale rispetto a questa proprietà. Ovvero è l'insieme più piccolo possibile per poter identificare univocamente tutte le *ennuple* della relazione. Se una chiave è formata da più di un insieme allora si chiama *superchiave*. Su SQL si indica che un attributo è una chiave attraverso il comando `UNIQUE`. Alcune chiavi possono essere definite su più attributi, ma devono essere univoche. Una chiave è una *superchiave* minimale.

Ogni relazione è un insieme, quindi non può contenere due *ennuple* uguali, e ha come *superchiave* almeno l'insieme degli attributi su cui è stata definita. Ogni relazione ha almeno una chiave. L'esistenza delle chiavi garantisce la possibilità di accedere ad ogni *ennupla* della base di dati e permettono di correlare relazioni diverse. Se nella chiave sono presenti valori nulli, questo non permette di identificare le *ennuple* e di realizzare facilmente riferimenti ad altre relazioni. Per cui la loro presenza nelle chiavi deve essere limitata o almeno controllata.

Una chiave si dice *primaria* se su di essa non sono ammessi valori nulli, indicata nell'intestazione sottolineando l'attributo corrispondente.

In SQL dopo aver definito lo schema della relazione, si inserisce la parola chiave `KEY` o `PRIMARY KEY`, in seguito all'attributo considerato come chiave oppure si inseriscono tra parentesi questi attributi.

2.3.2 Inter-Relazionali

Un vincolo di integrità referenziale, utilizza delle chiavi esterne "foreign key" X per collegare due relazioni R_1 e R_2 . Questo vincolo impone ai valori su X in R_1 di comparire come valori della chiave primaria in R_2 .

In SQL si definiscono dopo le colonne degli attributi tramite la parola chiave `FOREIGN KEY` specificando tra parentesi gli attributi da considerare. Si inserisce in seguito `REFERENCES` seguito dal nome della relazione R_1 e tra parentesi gli attributi di questa relazione da utilizzare.

3 Algebra Relazionale

4 Introduzione a SQL

SQL è indifferente tra maiuscolo e minuscolo, ma è preferibile essere coerente con le scelte di sintassi effettuate. Inoltre è indifferente dall'indentazione, ma si preferisce inserire parentesi oppure si va a capo per aumentare la leggibilità dell'interrogazione. Il linguaggio lo interpreta cercando il nome della parola chiave, e gli argomenti dell'interrogazione.

Le istruzioni che coinvolgono più relazioni, nel formato base, consiste in una parola chiave **SELECT** seguita la lista dove operare, seguita dalla clausola **FROM** ed in caso una condizione introdotta con **WHERE**.

```
SELECT ListaAttributi
FROM ListaTabelle
WHERE Condizione
```

Essenzialmente realizza un prodotto cartesiano delle relazioni specificate nella clausola **FROM**, in seguito viene effettuata un'operazione di selezione in base alla condizione specificata dalla parola chiave **WHERE**, se si cercano solo certi attributi, si può effettuare una proiezione specificando la lista di attributi dopo la parola chiave **SELECT**. In SQL bisogna specificare con la parola chiave **DISTINCT** che si stanno cercando solo gli attributi diversi.

In algebra relazionale è possibile scrivere interrogazioni equivalenti in modi diversi, in cui ci sono variazioni di efficienza, l'algebra è procedurale. In SQL invece il sistema si preoccupa dell'efficienza delle operazioni, è almeno in parte dichiarativo dove le interrogazioni possono essere scritte in modi diversi, ma alcune differenze presenti in algebra non emergono.

Il sistema esegue selezione join ed un ulteriore proiezione, nella versione base di SQL. Qualche anno dopo venne introdotto il join esplicito, introducendo la possibilità di specificare i join nella clausola **FROM** specificando l'argomento al posto di una lista di attributi, una lista di join effettuati su attributi, specificando la condizione di join dopo la clausola **ON**.

Date due relazioni contenente un attributo in comune, per realizzare una relazione di join su questo attributo in comune si specifica nella clausola **SELECT** l'attributo in notazione puntata, altrimenti sollevarebbe un errore poiché rappresenta un nome ambiguo. Anche nella condizione di join nella clausola **FROM** bisogna specificare a chi appartiene l'attributo indicato, utilizzando la notazione puntata:

```
SELECT Attributo1, Attributo2, Lista1.AttributoComune
--oppure anche Lista2.AttributoComune
FROM Lista1 JOIN Lista2 ON Lista1.AttributoComune = Lista2.AttributoComune
```

In SQL esiste il modo per effettuare join naturali specificando il nome dell'attributo non in notazione puntata, utilizzando la clausola **USING**, ma è preferibile non usarlo per favorire la comprensione:

```
SELECT AttributoComune, Attributo1, Attributo2
FROM Lista1 JOIN Lista2 USING AttributoComune
```

Inoltre è possibile utilizzare più volte la stessa relazione in un'interrogazione, utilizzando un nome diverso, chiamati alias in SQL, specificando dopo il nome l'alias, oppure utilizzando la clausola **AS**. In questo modo è possibile eliminare le ambiguità generate effettuando diversi join sulle stesse relazioni.

Nome **AS** N

Questo è utile per visualizzare campi dallo stesso nome, ridenominando gli attributi del risultato, oppure per facilitare la scrittura evitando nomi di relazioni molto lunghi.

Esiste in SQL il join esterno, dove alcuni degli operandi partecipano solamente in parte, tramite la clausola **LEFT**, **RIGHT** o **FULL** seguito da **JOIN**. Inoltre è possibile inserire **OUTER** per realizzare join equivalenti, ma queste funzioni non sono presenti nel servizio web SQLite, dove è possibile utilizzare solamente **LEFT JOIN**.

L'ordinamento del risultato è un altro fattore determinante, in base a cui si distinguono due ennuple o soluzioni tra di loro. Si può effettuare operazioni sulla target list e si può utilizzare la condizione **LIKE** per identificare espressioni regolari, dove **_** identifica un qualsiasi carattere e **%** per qualsiasi sequenza di carattere, inseriti tra doppi apici " . . . ".

Il contenuto delle basi di dati viene spesso aggregato, ma questo non è possibile in algebra relazionale. SQL prevede la possibilità di calcolare piccole elaborazioni a partire da insiemi di ennuple, di conteggio, minimo, massimo, media o totale.

Queste operazioni vengono svolte da operatori aggregati quali **COUNT** per contare tutti le righe in una relazione. Le funzioni aggregative lavorano anche con valori nulli. Bisogna specificare l'attributo di cui contare tutte le righe nella relazione tra parentesi tonde:

COUNT(Attributo)

Altri operatori aggregati sono **SUM**, **AVG**, **MAX** e **MIN**. Si nota un'ulteriore utilità del valore nullo, poiché il sistema li riconosce come un valore non reale e non lo utilizza nel calcolo, al contrario di un sistema dove i valori nulli vengono codificati con 0 o -1.

Esiste un'altra clausola **GROUP BY**, insieme alle funzione aggregate, divide le ennuple di una relazione sulla base dell'attributo specificato, questo attributo deve essere presente anche nella target list. Ma in questo modo nel raggruppamento sotto l'attributo raggruppato, se è presente più di uno, sarà scelto uno a caso da visualizzare, su SQLite.

Esiste un'altra clausola **HAVING** per definire una condizione su raggruppamenti, mentre la **WHERE** si usa sulle singole ennuple.

In SQLite online l'inserimento di una ennupla in una tabella non controlla se gli attributi che si vogliono aggiungere sono presenti nella tabella, infatti potrebbe causare dei danni all'intera tabella. Questa reference dovrebbe essere controllata ad ogni inserimento.

Su SQLite si può attivare il controllo della chiave esterna con il comando:

PRAGMA foreign_keys=on

La funzioni di aggregazione effettuano l'operazione sulla target list, raggruppando i campi, quindi in alcuni casi non permettono di scegliere di quale ennupla mostrare l'attributo. Si applicano al gruppo di ennuple che soddisfano la condizione imposta dal **GROUP BY**. Per risolvere questo problema

e non perdere informazioni sulle relazioni utilizzati, si può utilizzare una vista, oppure interrogazioni nidificate. Le viste sono interrogazioni, calcolata per mezzo di un'espressione, utilizzabile come fosse una relazione. Prima di creare una vista è consigliabile effettuare l'interrogazione per osservare il suo risultato. In SQL non esistono valori, ma relazioni di singolo attributi su una singola ennupla.

All'interno di un'interrogazione è possibile scrivere altre interrogazioni, in molti modi diversi nella versioni recenti di SQL. Si può inserire nei comandi **WHERE**, **FROM**, **SELECT**, etc. Esiste anche con i tipi, per esempio con valori booleani con **EXISTS**.

Se nel **WHERE**, invece di restituire una ennupla con un singolo valore dalla sotto-interrogazione, vengono restituite diverse ennupla, in SQLite questo non genera un errore, ma rappresenta un comportamento errato della piattaforma. Su PostgreSQL infatti questo solleva un errore, poiché non può utilizzare le ennuple fornite nel confronto. In questo modo è come se l'interrogazione interna nella **WHERE**, utilizzando attributi dell'interrogazione esterna, venga eseguita ogni volta per ogni ennupla della interrogazione esterna.

Le interrogazioni nidificate erano nella versione base di SQL, fin dalla sua nascita, poiché non si credeva di poter utilizzare la join. Quindi ogni interrogazioni veniva realizzata con valori distinti, ma si capì molto presto la necessità di introdurre la join per effettuare interrogazioni più semplicemente.

In SQL si possono scrivere operazioni di aggiornamento del database, tramite il comando **INSERT INTO**, allo stesso modo si possono eliminare dei valori con **REMOVE FROM**. Per modificare una tabella già creata, si può utilizzare il comando **ALTER TABLE**.