

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	Algebra Relazionale	2
3	Introduzione a SQL	3

1 Introduzione

2 Algebra Relazionale

3 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**.