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

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 solleverebbe 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.

3.1 Esercitazione 18/10/24

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.

Si considera il seguente database:

```
CREATE TABLE Compositori (codice integer NOT NULL PRIMARY KEY,
                    cognome text ,
```

```
                        nome text);

CREATE TABLE Concerti (codice integer NOT NULL PRIMARY KEY,
                        titolo text ,
                        descrizione text);

CREATE TABLE Pezzi (codice  integer NOT NULL PRIMARY KEY,
                        titolo text,
                        autore integer NOT NULL REFERENCES Compositori,
                        durata integer);

CREATE TABLE Programmazione ( pezzo integer NOT NULL REFERENCES Pezzi,
                        concerto integer NOT NULL REFERENCES Concerti,
                        posizione integer,
                        PRIMARY KEY(pezzo, concerto));
```

Popolato dai seguenti valori:

```
insert into compositori values(1, 'Mozart', 'Wolfgang Amadeus');
insert into compositori values(2, 'Bach', 'Johann Sebastian');
insert into compositori values(3, 'Beethoven', 'Ludwig van');

insert into concerti values(1, 'Concerto di Febbraio', 'Selezione di musica Barocca');
insert into concerti values(2, 'Concerto di Marzo', 'Estratti di belle sinfonie');
insert into concerti values(3, 'Concerto di Giugno', 'Concerto a Villa Ada');

insert into pezzi values(1, 'Variazioni Goldberg', 2, 32);
insert into pezzi values(2, 'L''arte della fuga', 2, 38);
insert into pezzi values(3, 'Il clavicembalo ben temperato', 2, 85);
insert into pezzi values(4, 'Il flauto magico', 1, 95);
insert into pezzi values(5, 'Serenata in do minore k 388', 1, 45);
insert into pezzi values(6, 'Requiem', 1, 87);
insert into pezzi values(7, 'Sinfonia n. 6 in fa maggiore op. 68', 3, 91);
insert into pezzi values(8, 'Sinfonia n. 9 in re minore', 3, 91);
insert into pezzi values(9, 'Trio d''archi in mi bemolle maggiore op. 3', 3, 52);

insert into programmazione values(1, 1,1);
insert into programmazione values(2, 1,2);
insert into programmazione values(3, 2,2);
insert into programmazione values(4, 3,1);
insert into programmazione values(5, 2,3);
insert into programmazione values(5, 3,2);
insert into programmazione values(7, 2,1);
```

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

```
PRAGMA foreign_keys=on
```

Vengono proposti una serie di esercizi su questo database:
Determinare il titolo dei pezzi che hanno durata compresa tra 40 e 60 minuti. Per effettuare quest'operazione è sufficiente operare sulla singola tabella `Pezzi`:

```
SELECT titolo FROM Pezzi
WHERE durata>40 AND durata<60
```

Determinare il titolo, nome e cognome dell'autore dei pezzi che hanno durata compresa tra 40 e 60 minuti. Oltre alla tabella `Pezzi`, è necessario ottenere i campi `nome` e `cognome` della tabella `Compositori`. Si effettua mediante un join, specificando come vengono associate le due relazioni, infatti il campo `autore` di `Pezzi` si riferisce alla chiave primaria della relazione `Compositori`:

```
SELECT titolo, nome, cognome
FROM Pezzi P JOIN Compositori C ON P.autore=C.codice
WHERE durata>40 and durata<60
```

Determinare il nome e cognome dei compositori dei pezzi presenti nel "Concerto di Giugno". Per ottenere questo si utilizza una vista. Nei database industriali la vista p contenuta nella cache. La vista può essere utilizzata come fosse un'altra tabella nel database, ed è possibile utilizzarla per effettuare altre interrogazioni nella stessa query. Si crea quindi una vista di tutti i pezzi presenti nel concerto di Giugno:

```
CREATE VIEW pezzi_giugno AS Select *
FROM Concerti JOIN Programmazione ON Concerti.codice=Programmazione.concerto
WHERE Concerti.titolo='Concerto di Giugno'
```

Per ottenere il nome ed il cognome dei compositori si effettua una join su questa vista, prima con i pezzi, e poi con la relazione compositori, selezionando solo gli attributi richiesti:

```
SELECT DISTINCT nome, cognome
FROM pezzi_giugno JOIN Pezzi ON pezzi_giugno.pezzo=Pezzi.codice
JOIN Compositori ON Compositori.codice=autore
```

Determinare il nome e cognome dei compositori che non hanno pezzi nel "Concerto di Giugno".
Determinare il titolo e descrizione dei concerti in cui sono presenti pezzi di Mozart:

```
SELECT DISTINCT C.titolo, descrizione
FROM Concerti C join Programmazione P1 ON C.codice=concerto
JOIN Pezzi P ON pezzo=P.codice
JOIN Compositori C1 ON autore=C1.codice
WHERE cognome='Mozart'
```

Determinare il titolo e descrizione dei concerti in cui non sono presenti pezzi di Beethoven:

codice e titolo dei pezzi che non sono presenti in nessun concerto. Ordinare per codice i pezzi che compaiono in ultima posizione in almeno un concerto. Mostrare codice, titolo, cognome dell'autore e durata del pezzo. Ordinare per codice i pezzi che compaiono in ultima posizione in TUTTI i concerti. Mostrare codice, titolo, cognome dell'autore e durata del pezzo. Ordinare per codice coppie di pezzi con lo stesso titolo. Mostrare il titolo e i due codici (ordinando il risultato sul titolo). Nota bene: ogni coppia va mostrata una sola volta (ad esempio, se i pezzi 3 e 5 hanno stesso titolo, va mostrata solo la coppia 3,5 e non la coppia 5,3) per ogni concerto, la durata totale (somma delle durate dei pezzi). Supporre che per tutti i concerti ci sia almeno un pezzo. Mostrare codice e titolo del concerto e durata totale. Ordinare per codice. i concerti che hanno durata totale minore di 90 minuti; per ogni concerto, mostrare codice e durata totale. Ordinare per codice. codice, nome e cognome dei compositori che sono presenti in tutti i concerti. Ordinare per codice