Database Relazionali (SQL)

Alberto Filosa

30/12/2019

Indice

1 Introduzione ai DBMS:

Un modello di dati è un insieme di concetti utilizzati per organizzare i dati di interesse e descriverne la struttura in modo da risultare comprensibile a tutti. Il modello relazionale dei dati, attualmente il più diffuso, permette di definire tipi per mezzo della relazione, che consente di organizzare i dati in insiemi di record e struttura fissa.

I DataBase relazionali organizzano dati in tabelle e successivamente queste possono essere unite tra loro tramite delle chiavi primarie (ID Primary Key). I dati al loro interno sono organizzati in colonne e ogni riga indica un singolo elemento di informazione. L'obiettivo principale dei DBMS è quella di trasformare i dati in informazione.

Una *chiave primaria* nel modello relazionale delle basi di dati è un insieme di attributi che permette di individuare univocamente un record in una tabella. Una *chiave esterna*, invece, è un insieme di attributi che fa riferimento a una chiave di un'altra tabella per unire tramite una join due tabelle diverse. Una tabella deve obbligatoriamente possedere una e una sola chiave primaria, e nessun record nella tabella può lo stesso valore di un altro record: questo vincolo è chiamato vincolo di unicità.

Talvolta è possibile che siano presenti dei valori mancanti, chiamati `NA`. Essi possono definire un valore inesistente o un valore sconosciuto. Inoltre, possono essere anche presenti delle informazioni inserite nei DataBase in modo sbagliato.

Una interrogazione di una tabella SQL avviene nel seguente modo:

```
select <attributo/i>
from <tabella/e>
where <condizione/i>
group by <attributo/i>
order by <attributo/i> ASC/DESC;
```

Il comando *select* ha il compito di selezionare le colonne di una o più colonne all'interno di un DataBase. Se sono presenti due colonne da tabelle diverse, è necessario specificare da quale tabella è stata presa. Il comando *where* specifica una condizione di ricerca per filtrare le righe selezionate nella *select*: tramite operatori logici è possibile utilizzare più condizioni. Il comando *group by* è utilizzato assieme alla select per raggruppare le righe di una tabella. Se si vogliono fare delle condizioni sul raggruppamento, si utilizza il comando *having*.

I comandi ed i valori sono *Case Sensitive*. Inoltre, si usano i singoli apici (`'`) durante la costruzione della *query*, non i doppi apici (`"`).

Per non avere ripetizioni nelle osservazioni, si usa il comando `distinct` dopo la *select*. Inoltre, le condizioni nella *where* possono essere complesse; per questo, è possibile combinare questi comandi con `and`, `or` oppure `nor`. Per lavorare sulle stringhe, si usa il comando `like` nella condizione *where*, con le seguenti condizioni:

- `%` per ricercare qualsiasi stringa (es: trovare gli autori con S le iniziali del nome);

  ```
  select *
  from Authors
  where first_name like 'S%';
  ```

- `_` per ricercare qualsiasi carattere (es: trovare gli autori con S le iniziali del nome e la terza lettera a).

  ```
  select *
  from Authors
  where first_name like 'S_a%';
  ```

1.1 Raggruppamento:

Esistono anche dei comandi di raggruppamento:

- **count**: conta il numero di righe;

```
select count(*)
from Authors
where income >= 65000;
```

- **avg**: seleziona la media della colonna selezionata;

```
select avg(income)
from Authors
where income >= 65000;
```

- **sum**: seleziona la somma della colonna selezionata;

```
select sum(income)
from Authors
where income >= 65000;
```

- **min**: seleziona il valore minimo della colonna selezionata;

```
select min(income)
from Authors
where income >= 65000;
```

- **max**: seleziona il valore massimo della colonna selezionata.

```
select max(income)
from Authors
where income >= 65000;
```

Per ogni attributo, il comando `group by` calcola le righe selezionate. La sua presenza porta la *select* ad agire sul gruppo, **non** più sulle righe. In particolare, nella *select* è possibile inserire operatori aggregati o attributi presenti nella `group by`.

```
select genre, last_name, avg ( income )
from Authors
group by genre;
```

2 Unione di Tabelle:

Per selezionare due informazioni da due tabelle diverse si usa il comando `join`. Per unirle, è necessario che le tabelle abbiano una chiave in comune. Le tabelle non vengono unite in modo definitivo, **ma** solo temporaneamente.

```
select first_name, last_name
from Authors as a, BooksAuthors as ba
where ba.author_id = a.author_id
  and book_id = 1
```

Se si vogliono unire solamente due tabele, si può utilizzare il comando `join`:

```
select first_name, last_name
from Authors as a JOIN BooksAuthors as ba
  on ba.author_id = a.author_id
where book_id = 1;
```

In SQL è possibile solamente unire solamente le tabelle da sinistra, chiamate *left join*. Il comando prende in considerazione tutte le righe della tabella di sinistra e unisce le colonne selezionate.

Se un attributo della riga viene modificata, di conseguenza si verificano 3 situazioni:

1. si modifica o cancella il resto;
2. si termina l'operazione;
3. si ottiene `NULL` nel resto.

2.1 Set Operation:

Nei DataBase relazionali è possibile compiere delle operazioni come l'intersezione, l'unione, ecc di unione di due tabelle.

- **Intersect**: si compie una intersezione tra due tabelle Nella realtà, questo comando non è quasi mai utilizzato, dato che può essere sostituito con una condizione *where* più complessa;

```
select last_name, first_name
from Authors
where Income > 90000
INTERSECT
select last_name, first_name
from Authors
where last_name like '%o%';
```

Questa *query* può essere sostituita con la seguente:

```
select last_name , first_name
from Authors
where Income > 90000 AND last_name like '%o%'
```

- **Union**: si compie una unione tra due tabelle. Anche in questo caso, non è quasi mai utilizzato, dato che può essere sostituito con una condizione *where* più complessa;

```
select last_name , first_name
from Authors
where Income > 90000
UNION
select last_name , first_name
from Authors
where last_name LIKE '%o%';
```

Questa *query* può essere sostituita con la seguente:

```
select last_name , first_name
from Authors
where Income > 90000 OR last_name like '%o%'
```

- **Except**: si compie una differenza tra due tabelle. In particolare, stampa solamente le condizioni della prima *where* e non stampa quelle della seconda. In questo caso, il comando è molto utile, ma comunque può essere sostituito con una *query* più complessa (verrà riproposta in seguito);

```
select last_name , first_name
from Authors
where Income > 90000
EXCEPT
select last_name , first_name
from Authors
where last_name like '%o%'
```

3 Nested Query:

Come visto in precedenza, il comando `intersect` serve per compiere più operazioni di selezione. In modo analogo, è possibile sostituirlo con il comando `in`:

```
select last_name, first_name
from Authors
where Income > 90000
INTERSECT
select last_name, first_name
from Authors
where last_name like '%o%';
```

Questa *query* è analoga alla seguente:

```
select last_name, first_name
from Authors
where Income > 90000 and last_name IN
  select last_name
  from Authors
  where last_name like '%O%'
```

Il comando `except`, spiegato in precedenza, è possibile sostituirlo con il comando `not in`:

```
select last_name, first_name
from Authors
where Income > 90000 and id NOT IN
  select id
  from Authors
  where last_name like '%O%'
```