SQL: Query annidate e altro …

Prof. Viglietti Francesco

Www.in4matika.altervista.org

Scrittura di interrogazioni annidate

Un’interrogazione annidata non è altro che un’interrogazione all’interno di un’altra interrogazione, cioè un’interrogazione il cui risultato viene riutilizzato per calcolare un’altra interrogazione. A volte le interrogazioni annidate vengono chiamate anche sotto-interrogazioni o sottoquery o sub-query.

Non aggiungono nuove funzioni, ma rendono le interrogazioni più leggibili rispetto a quelle create usando un insieme complesso di join.

MySQL offre due tipi di interrogazioni annidate:

- **interrogazioni annidate per derivare una tabella**
- **interrogazioni annidate come espressioni**, (compaiono all’interno della condizione WHERE di un comando SELECT).
  - che restituiscono un singolo valore o una riga
  - usate per verificare un espressione booleana.

Interrogazioni annidate per derivare una tabella

Le interrogazioni annidate per derivare una tabella consentono di scrivere un'interrogazione all'interno del comando FROM di un'altra interrogazione. Ciò permette di creare una tabella temporanea e aggiungerla all'interrogazione da calcolare.

employeeID	name	job	departmentID
6651	Ajay Patel	Programmer	128
7513	Nora Edwards	Programmer	128
9006	Candy Burnett	Systems Adminis	128
9842	Ben Smith	DBA	42

Ad esempio, consideriamo:

```
SELECT IDimpiegato, nome
FROM impiegato
WHERE mansione='Programmatore';
```

E' ovvio che si ottengono i nomi e gli ID di tutti i programmatori. Si può usare quest'interrogazione all'interno di un'altra per ottenere un altro risultato utile:

```
SELECT program, nome
FROM (SELECT Idimpiegato, nome FROM impiegato
WHERE mansione = 'Programmatore') AS program, assegnamento
WHERE program.IDimpiegato = assegnamento.IDimpiegato;
```

Interrogazioni annidate per derivare una tabella

In questo caso è stata usata un'interrogazione annidata

(SELECT IDimpiegato, nome, FROM impiegato WHERE mansione = 'Programmatore')

employeeID	name
6651	Ajay Patel
7513	Nora Edwards

per creare una tabella temporanea che contiene solo le colonne IDimpiegato e nome, ed è stata chiamata **program** grazie all'alias.
Si può adesso interrogarla come si farebbe per qualsiasi altra tabella, con la tabella **assegnamento**

clientID	employeeID	workdate	hours
1	7513	2003-01-20	8.5

In questo esempio, è stata utilizzata per trovare quali programmatori sono stati assegnati a un progetto esterno ottenendo il seguente risultato:

```
+--------------+
| nome         |
+--------------+
| Nora Edwards |
+--------------+
```

Interrogazioni annidate che restituisco un singolo valore o una riga

Come nella sezione precedente, iniziate con un'interrogazione semplice:

SELECT max(oreEffet) FROM assegnamento;

Quest'interrogazione restituisce un solo valore, che rappresenta il numero massimo di ore che un impiegato dell'azienda ha dedicato a un progetto. Si sta usando una funzione di MySQL mai vista prima: **max( ),** che trova il valore massimo in una colonna.

L'uso del risultato di questo tipo di funzione è un caso molto diffuso di utilizzo di interrogazione annidate a valore singolo.

Come prima, è possibile fare questa interrogazione all'interno di un'altra. Le interrogazioni annidate a singolo valore restituiscono un solo valore di una colonna e vengono comunemente usate per i confronti.

clientID	employeeID	workdate	hours
1	7513	2003-01-20	8.5

Interrogazioni annidate che restituisco un singolo valore o una riga

Ad esempio, si consideri la seguente interrogazione:

employeeID	name	job	departmentID
6651	Ajay Patel	Programmer	128
7513	Nora Edwards	Programmer	128
9006	Candy Burnett	Systems Adminis	128
9842	Ben Smith	DBA	42

```
SELECT e.IDimpiegato, e.nome
FROM impiegato AS e, assegnamento AS a
WHERE e.IDimpiegato = a.IDimpiegato
AND a.oreEffet = (SELECT max(oreEffet) FROM assegnamento );
```

In quest'interrogazione, si sta cercando chi potrebbe essere definito l'impiegato che lavora di più all'interno dell'azienda: chi è l'impiegato che ha svolto il numero maggiore di ore su un progetto in un giorno?

Il risultato è il seguente:

IDimpiegato	nome
7513	Nora Edwards

Si può anche scrivere un'interrogazione che restituisca una riga piuttosto che un singolo valore, ma questo caso ha un'utilità abbastanza limitata.

Interrogazioni annidate per verificare un'espressione booleana

Le interrogazioni annidate che restituiscono un'espressione booleana vengono usate per verificare l'interrogazione rispetto ad alcune funzioni speciali che restituiscono un valore booleano. Queste funzioni speciali sono ***IN, EXISTS, ALL, ANY.***

- La parola chiave **IN** permette di fare il confronto tra un valore o un campo all'interno di un insieme.

es. ***SELECT * FROM libro WHERE paginetotali IN (250, 220, 170)***;

- **ANY** viene utilizzato in una clausola WHERE in espressioni del tipo: ***x > ANY Elenco***. Il predicato ANY è vero se il confronto è vero per almeno uno dei valori dell'elenco. La condizione di ricerca è falsa se la sottoquery restituisce un insieme vuoto oppure se il confronto è falso per ciascuno dei valori restituiti dalla sottoquery.
- **ALL** viene utilizzato in una clausola Where in espressioni del tipo: ***x <= ALL Elenco***. Il predicato ALL restituisce vero se il confronto è vero per ciascuno dei valori in Elenco. La condizione di ricerca è falsa se il confronto è falso per almeno uno tra i valori dell'elenco restituito dalla sottoquery.

Interrogazioni annidate per verificare un'espressione booleana

Si può usare la parola chiave IN per fare un confronto con un insieme di valori. Si consideri l'interrogazione:

SELECT nome FROM impiegato
WHERE IDimpiegato NOT IN
(SELECT IDimpiegato FROM assegnamento);

Questa query restituisce lo stesso risultato di quella vista per il LEFT JOIN. Permette di trovare gli impiegati che non sono tra quelli che hanno svolto un progetto esterno.
Sapendo che IN permette di fare il confronto all'interno di un insieme di valori. Il risultato è:

employeeID	name	job	departmentID
6651	Ajay Patel	Programmer	128
7513	Nora Edwards	Programmer	128
9006	Candy Burnett	Systems Adminis	128
9842	Ben Smith	DBA	42

clientID	employeeID	workdate	hours
1	7513	2003-01-20	8.5

```
+---------------+
| nome          |
+---------------+
| Ajay Patel    |
| Candy Burnett |
| Ben Smith     |
+---------------+
```

Interrogazioni annidate per verificare un'espressione booleana

- **EXISTS** funziona in modo leggermente diverso rispetto a IN. Le interrogazioni con EXISTS usano i dati dell'interrogazione esterna in quella annidata. EXISTS viene utilizzato in una clausola Where in espressioni del tipo: EXISTS Tabella. Il predicato EXISTS controlla se vengono restituite righe dall'esecuzione della sottoquery: la condizione di ricerca è vera se la SELECT nidificata produce almeno una riga, è falsa altrimenti.
  Questa situazione viene indicata come interrogazione annidata correlata. Ad esempio, l'interrogazione:

employeeID	name	job	departmentID
6651	Ajay Patel	Programmer	128
7513	Nora Edwards	Programmer	128
9006	Candy Burnett	Systems Adminis	128
9842	Ben Smith	DBA	42

clientID	employeeID	workdate	hours
1	7513	2003-01-20	8.5

```
SELECT e.nome, e.IDimpiegato
FROM impiegato AS e
WHERE NOT EXISTS
( SELECT *
  FROM assegnamento
  WHERE assegnamento.IDimpiegato = e.IDimpiegato);
```

In questo caso si stanno cercando gli impiegati che non hanno mai lavorato in un progetto esterno.

Uso di interrogazioni annidate per veri un’espressione booleana

Guardando le righe della tabella assegnamento e cercando nell’interrogazione annidata per quali IDimpiegato della tabella assegnamento si ha lo stesso valore in impiegato.IDimpiegato.
Il valore e.IDimpiegato arriva dall’interrogazione esterna. Questo è quello che fa MySQL: per ciascuna riga della tabella impiegato viene calcolato il risultato dell’interrogazione annidata e se non viene trovato nulla (WHERE NOT EXISTS), dettagli dell’impiegato vengono inseriti nel risultato.
Anche se alcuni utenti trovano questa sintassi più semplice da comprendere, si potrebbe ottenere lo stesso risultato usando un LEFT JOIN come è stato fatto in precedenza.
Quest’interrogazione ha esattamente lo stesso risultato:

nome	IDimpiegato
Ajay Patel	6651
Candy Burnett	9006
Ben Smith	9842

Uso di interrogazioni annidate per veri un’espressione booleana

Le parole chiave ALL, ANY .

Si supponga che Nora Edwards (ricordate, la programmatrice che lavora il numero massimo di ore) voglia dimostrare che nessuno lavora di più dei programmatori. Per ottenere quest’informazione Nora scrive la seguente interrogazione:

```
SELECT e.nome
FROM impiegato AS e, assegnamento AS a
WHERE e.IDimpiegato = a.IDimpiegato
AND a.oreEffettuate > ALL
( SELECT a. oreEffettuate
FROM assegnamento.a, impiegato.e
WHERE e.IDimpiegato = a.IDimpiegato
AND e.mansione = ‘Programmatore’);
```

Uso di interrogazioni annidate per veri un’espressione booleana

L’interrogazione annidata trova l’elenco delle ore lavorate sui vari progetti da parte dei programmatori della compagnia. Poi controlla se esiste qualche altro impiegato che abbia svolto su un progetto più ore dei programmatori, usando il controllo a.oreEffettuate > ALL (le ore dei programmatori).

L’interrogazione non restituisce nulla visto che, in effetti, nessuno in questa compagnia lavora più duramente dei programmatori.

Proprietà opzionali del comando SELECT

Vediamo ora la sinassi completa e tutto quello che ancora non si conosce del comando SELECT:

```
SELECT [STRAIGHT_JOIN]
       [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
       [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS] [HIGH_PRIORITY]
       [DISTINCT | DISTINCTROW | ALL]
   espressione select,...
   [INTO {OUTFILE | DUMPFILE} 'nome_file' export_options]
   [FROM tabelle_di_riferimento
       [WHERE definizione_delle_condizioni]
       [GROUP BY {unsigned_integer | nome_col | formula} [ASC | DESC], ...]
       [HAVING definizione_delle_condizioni]
       [ORDER BY {unsigned_integer | nome_col | formula} [ASC | DESC] ,...]
       [LIMIT [offset,] righe | righe OFFSET offset]
       [PROCEDURE nome_procedura (lista_parametri)]
       [FOR UPDATE | LOCK IN SHARE MODE]]
```

Proprietà opzionali della SELECT

Molti di questi comandi sono già familiari. Brevemente qui di seguito sono elencati quelli non ancora usati.

- STRAIGHT_JOIN all'inizio della forma sintattica può essere usato per forzare l'ottimizzatore a collegare le tabelle secondo l'ordine in cui sono state elencate. Questa forma ha lo stesso effetto dello STRAIGHT_JOIN nella condizione WHERE. (comando estensione ANSI SQL).
- I comandi SQL_SMALL, SQL_BIG_RESULT e SQL_BUFFER_RESULT sono stati creati per aiutare il processo di ottimizzazione. Si possono usare SQL_SMALL_RESULT e SQL_BIG_RESULT per indicare a MySQL che ci si aspetta un risultato composto da poche o molte righe. SQL_BUFFER_RESULT indica a MySQL di inserire il risultato in un tabella temporanea. Si può usare questo comando quando l'invio del risultato al client occuperebbe molto tempo bloccando quindi l'uso delle tabelle oggetto dell'interrogazione. (comandi estensione ANSI SQL).
- SQL_CACHE e SQL_NOCACHE dicono a MySQL se memorizzare nella memoria cache il risultato (comando estensione ANSI SQL).

Proprietà opzionali della SELECT

- SQL_CALC_FOUND_ROWS viene usata con la condizione LIMIT; indica a MySQL di calcolare da quante righe sarebbe stato composto il risultato se non ci fosse stata la condizione LIMIT. Potete ottenere il numero totale con l'istruzione select found_rows( ) (comando estensione ANSI SQL). Lo scopo del comando è ridurre la duplicazione del lavoro. Nelle versioni di MySQL prive del comando, si può ottenere lo stesso usando un'interrogazione con count(*) e di seguito un SELECT con un LIMIT.
- HIGH PRIORITY dice a MySQL che l'esecuzione di quest'interrogazione deve precedere qualsiasi comando di UPDATE che sia in attesa di modificare le tabelle in oggetto.
- Si è già parlato di DISTINCT, DISTINCTROW è semplicemente un sinonimo. ALL funziona in modo opposto (restituisce tutti i duplicati) ed è l'opzione standard.

Proprietà opzionali del comando SELECT

- Il comando PROCEDURE permette di indicare una procedura da eseguire sull'insieme dei risultati prima di inviarlo al client. Può essere scritta in C++ e quindi esula dagli scopi di questo corso. Qualora ci fosse bisogno di maggiori dettagli si può consultare direttamente il manuale di MySQL.
- I comandi FOR UPDATE e LOCK IN SHARE MODE interessano soltanto se il sistema di memorizzazione utilizzato supporta il lock di riga o di pagina, in pratica si sta parlando di InnoDB o BDB. Se viene indicato FOR UPDATE si ottiene un lock completo altrimenti con LOCK IN SHARE MODE se ne ha uno condiviso.

ESERCIZI

1. Scrivete un’interrogazione usando il LEFT JOIN che elenchi i clienti che non hanno mai avuto degli impiegati che lavorassero sui loro progetti (assegnamenti).
2. Riscrivete l’interrogazione dell’esercizio 1 usando EXISTS.

Le Viste Logiche

Le viste sono tabelle virtuali, interrogabili come le altre, ma soggette a limiti per ciò che riguarda gli aggiornamenti.

Le viste possono essere create a vari scopi, tra i quali si ricordano i seguenti:

- Permettere agli utenti di avere una visione personalizzata del DB, e che in parte astragga dalla struttura logica del DB stesso
- Far fronte a modifiche dello schema logico che comporterebbero una ricompilazione dei programmi applicativi
- Semplificare la scrittura di query complesse

Inoltre le viste possono essere usate come meccanismo per il controllo degli accessi, fornendo ad ogni classe di utenti gli opportuni privilegi

Si noti che nella definizione di una vista si possono referenziare anche altre viste

Le Viste Logiche esempi

Mediante l'istruzione CREATE VIEW si definisce una vista.

Le tuple della vista sono il risultato di una query che viene valutata dinamicamente ogni volta che si fa riferimento alla vista

CREATE VIEW **ProgSedi**(CodProg,CodSede) AS SELECT P.CodProg,S.Sede FROM Prog AS P, Sedi AS S WHERE P.Citta = S.Citta

SELECT * FROM **ProgSedi** WHERE CodProg = 'P01'

Una vista può essere eliminata con il comando:

DROP VIEW **ProgSedi;**