

09Modulo- I self join e gli alias di tabella

Rapporti fra Righe della Stessa Tabella: Self-Join e Alias di Tabella

In alcuni casi può emergere l'esigenza di accoppiare i dati di una tabella con righe provenienti dalla stessa tabella. In questa situazione si parla di self-relationship per la tabella in questione. Nella pratica, si prende la tabella due volte, assegnandole due ruoli diversi. Consideriamo un esempio pratico: supponiamo di avere una tabella employee con un rapporto uno a n fra impiegati, quindi fra righe della stessa tabella. Un responsabile coordina n junior, il che significa che una riga della tabella impiegati avrà il ruolo di responsabile e sarà accoppiata a n righe con il ruolo di junior.

Struttura della Tabella Employee

Il tracciato della tabella potrebbe essere il seguente: la tabella Employee contiene un campo ID che funge da chiave primaria, un campo NAME per il nome, un campo SURNAME per il cognome e un campo COORDINATORID che rappresenta l'identificativo del coordinatore. Il campo coordinatorid è una chiave esterna della tabella employee verso se stessa, ed esprime il rapporto fra la riga e il suo coordinatore. Non viene espressa integrità referenziale poiché si prevede che ci siano impiegati senza responsabili, pertanto coordinatorid dovrà poter essere NULL.

Uso di una Self-Relationship con Alias di Tabella

Supponiamo di dover riportare il cognome della risorsa assieme al cognome del suo coordinatore. Cominciamo col prodotto cartesiano: `SELECT * FROM EMPLOYEE, EMPLOYEE`. Questa query produrrà un errore poiché MySQL ha bisogno di distinguere le due tabelle per poter capire da quale delle due stia prendendo i campi, o meglio, da quale riga. La soluzione consiste nel fornire un alias di tabella per ogni istanza della tabella: `SELECT * FROM EMPLOYEE COORDINATOR, EMPLOYEE JUNIOR`.

INNER JOIN per una Self-Relationship

La tabella employee viene presa una volta come coordinator e una volta come junior. Tutte le righe vengono combinate con tutte le altre, come abbiamo visto in precedenza. Questo comportamento viene evitato attraverso l'uso del predicato di join: `SELECT * FROM EMPLOYEE COORDINATOR, EMPLOYEE JUNIOR WHERE COORDINATOR.ID = JUNIOR.COORDINATORID`. In questo modo MySQL "crede" di avere due tabelle, coordinator e junior, che in realtà esistono solo in questa query, ma le vede come tabelle separate e possono essere usate esattamente come se lo fossero, applicando il consueto join uno a n.

Ricavare i Dati che Ci Interessano

A questo punto possiamo proiettare i campi di nostro interesse: `SELECT JUNIOR.SURNAME AS JUNIORSURNAME, COORDINATOR.SURNAME AS COORDINATORSURNAME FROM EMPLOYEE COORDINATOR, EMPLOYEE JUNIOR WHERE COORDINATOR.ID = JUNIOR.COORDINATORID`. La sintassi tabella.campo risulta obbligatoria in questo caso: le due tabelle sono identiche e hanno gli stessi campi, quindi MySQL non saprebbe distinguere altrimenti. È importante notare che abbiamo proiettato un campo dalla riga di Employee che funge da junior, ovvero junior.surname, e uno dalla riga di Employee che funge da coordinatore, ovvero coordinator.surname, ma usando sempre la stessa tabella Employee.

Self-Relationship e Join Atipici

Posto queste differenze, che consistono negli alias di tabella obbligatori e nella sintassi tabella.campo, si possono applicare tutti gli altri tipi di join. Ad esempio, possiamo scrivere una query pensata per identificare i possibili genitori e figli a partire dalle età nella tabella Person: `SELECT PARENT.ID AS PARENTID, CHILD.ID AS CHILID FROM PERSON PARENT, PERSON CHILD WHERE PARENT.SURNAME = CHILD.SURNAME AND YEAR(PARENT.DATEOFBIRTH) - YEAR(CHILD.DATEOFBIRTH) BETWEEN 20 AND 50`. Questa query trova tutte le persone con lo stesso cognome e per cui ci sia una differenza compresa fra i 20 e i 50 anni fra la possibile riga genitore e la possibile riga figlio. In questo caso il predicato di join è costituito da tutte le condizioni poste dopo il where, e si tratta di un inner join隐式的.