08Modulo - Le relazioni Uno a Molti in SQL

Le Relazioni Uno a Molti in SQL

I rapporti uno a molti rappresentano le relazioni che in Java abbiamo indicato con termini diversi come uso, associazione, aggregazione o composizione, e che abbiamo già avuto modo di applicare nella pratica durante la prima sezione dedicata al join. Si tratta di rapporti di tipo has_many, diversamente dai rapporti is_a che caratterizzavano l'ereditarietà.
In termini generali, a un'entità corrispondente a una riga di una tabella t1, che definiamo come lato uno, si associano n entità corrispondenti a n righe di una tabella t2, che definiamo invece come lato n. Queste vengono collegate tipicamente attraverso l'utilizzo di una chiave esterna, sebbene questo non rappresenti sempre l'unico caso possibile, e come abbiamo avuto modo di vedere in precedenza, due tabelle possono essere legate da più di una relazione.

Uso, Aggregazione e Composizione

Nel caso in cui desideriamo esprimere in SQL un rapporto di associazione, aggregazione o uso, la modalità naturale di operare prevede l'utilizzo dell'integrità referenziale accompagnata dalle clausole on update cascade e on delete restrict. Nel caso specifico della composizione, la clausola più appropriata potrebbe risultare on delete cascade, poiché gli oggetti dipendenti in un rapporto di composizione non dispongono di solito di un'esistenza indipendente rispetto al contenitore che li ospita.
Il rapporto che intercorre tra una squadra e i suoi membri rappresenta un esempio tipico di relazione uno a molti di associazione, uso o aggregazione in Java, sebbene le differenze tra queste tre tipologie siano piuttosto sottili. In questo contesto, cancellare una squadra non dovrebbe comportare la cancellazione dei giocatori che ne facevano parte. Il rapporto che intercorre invece tra un prodotto e le sue recensioni dovrebbe essere configurato come una relazione uno a molti di tipo composizione: eliminare il prodotto rende infatti insensate le righe relative alle recensioni, e di conseguenza potrebbe risultare opportuno cancellarle

contestualmente. Dal punto di vista delle interrogazioni, tuttavia, tutte queste tipologie di relazione vengono gestite allo stesso identico modo.
Struttura Generale per Due Tabelle Collegate da un Rapporto Uno a Molti
Posto di avere due tabelle A e B e di voler stabilire una relazione uno a n tra di esse, dove una riga di A risulta associata a n righe di B, si procede secondo le seguenti modalità: la tabella A viene creata per prima, mentre la tabella B viene creata successivamente. La tabella B contiene, oltre ai propri campi specifici tra cui l'identificativo, anche la chiave primaria di A che in B assume il ruolo di chiave esterna verso A. La tabella A contiene unicamente il proprio identificativo A.ID, mentre la tabella B contiene sia il proprio identificativo B.ID sia l'identificativo di una riga di A, indicato come B.AID, che funge da chiave esterna verso A.
Considerando un esempio formale di struttura uno a n, supponiamo di avere nella tabella A tre righe con identificativi uno, due e tre. Nella tabella B avremo tre righe: la prima e la seconda riga di B risultano collegate alla prima riga di A, mentre la terza riga di B è collegata alla terza riga di A. La seconda riga di A non risulta collegata a nessuna riga di B, ma questo non costituisce un problema, poiché una riga posta sul lato uno non è obbligata a possedere righe corrispondenti nella tabella lato n.

Predicato di Join per una Relazione Uno a Molti

Il predicato per l'INNER JOIN di una relazione uno a n si esprime attraverso la formula A.ID = B.aID, dove A rappresenta la tabella lato uno e B la tabella lato n. In termini di linguaggio naturale, eseguire l'inner join tra A e B significa uguagliare la chiave primaria di A con la chiave esterna di B verso A, stabilendo quindi la corrispondenza A.PK = B.FK.
Esempio Concreto di INNER JOIN Uno a Molti
Considerando le tabelle Product e Review come esempio pratico: un prodotto può ricevere molte recensioni, mentre una recensione può riguardare un solo prodotto. Il rapporto risulta quindi uno a n, con Product posizionata sul lato uno e Review sul lato n. Seguendo le regole illustrate in precedenza, la tabella Review contiene la colonna productID che funge da chiave esterna verso Product. L'inner join tra le due tabelle viene scritto nella seguente forma:

```
SELECT PRODUCT.*, REVIEW.*
FROM PRODUCT
```

```
INNER JOIN REVIEW ON PRODUCT.ID = REVIEW.PRODUCTID
```

Raggruppamento e Join

Le funzioni di gruppo possono essere applicate anche al risultato delle operazioni di join, e questa possibilità risulta particolarmente utile nei rapporti uno a molti. Immaginiamo di dover calcolare la votazione media per ogni prodotto: il punteggio sarà contenuto nella tabella delle recensioni, mentre il nome del prodotto e il suo identificativo risiederanno nella tabella prodotti.
Il procedimento si articola in gradi successivi, assemblando prima i dati necessari attraverso una query che combina le due tabelle:

```
SELECT PRODUCT.ID, PRODUCT.NAME, REVIEW.SCORE
FROM PRODUCT
INNER JOIN REVIEW ON PRODUCT.ID = REVIEW.PRODUCTID
```

```
SELECT PRODUCT.ID, PRODUCT.NAME, AVG(REVIEW.SCORE) AS AVERAGE
FROM PRODUCT
INNER JOIN REVIEW ON PRODUCT.ID = REVIEW.PRODUCTID
GROUP BY PRODUCT.ID, PRODUCT.NAME
```

```
SELECT PRODUCT.ID, PRODUCT.NAME,
       AVG(REVIEW.SCORE) AS AVERAGE,
       MIN(SCORE) AS WORST,
       MAX(SCORE) AS BEST
FROM PRODUCT
INNER JOIN REVIEW ON PRODUCT.ID = REVIEW.PRODUCTID
GROUP BY PRODUCT.ID, PRODUCT.NAME
```

```
SELECT PRODUCT.ID, PRODUCT.NAME,
       AVG(REVIEW.SCORE) AS AVERAGE,
       MIN(SCORE) AS WORST,
       MAX(SCORE) AS BEST
```

```
FROM PRODUCT
INNER JOIN REVIEW ON PRODUCT.ID = REVIEW.PRODUCTID
WHERE YEAR(REVIEW.REVIEWDATE) >= 2020
GROUP BY PRODUCT.ID, PRODUCT.NAME
```

Il Problema delle Righe Non Collegate

Resta tuttavia un problema persistente quando utilizziamo INNER JOIN: non vedremo i prodotti che non possiedono alcuna recensione. Per ovviare a questa limitazione, si utilizza il LEFT JOIN al posto dell'INNER JOIN, mantenendo la stessa struttura di raggruppamento e aggregazione ma garantendo la visualizzazione di tutti i prodotti, anche di quelli sprovvisti di recensioni:

```
SELECT PRODUCT.ID, PRODUCT.NAME,
       AVG(REVIEW.SCORE) AS AVERAGE,
       MIN(SCORE) AS WORST,
       MAX(SCORE) AS BEST
FROM PRODUCT
LEFT JOIN REVIEW ON PRODUCT.ID = REVIEW.PRODUCTID
WHERE YEAR(REVIEW.REVIEWDATE) >= 2020
GROUP BY PRODUCT.ID, PRODUCT.NAME
```

Una Nota Finale sui JOIN

Per concludere questa trattazione, è importante sottolineare che non tutti i rapporti vengono dichiarati formalmente e non tutti i join richiedono la presenza di chiavi esterne. Consideriamo di possedere una tabella buyer contenente una colonna budget, che rappresenta quanto un acquirente intende spendere per un regalo, e una tabella present contenente i regali disponibili. Se desideriamo associare a ogni acquirente tutti i regali che rientrano nel suo budget, possiamo formulare la seguente query:

```
SELECT BUYER.NAME, PRESENT.NAME
FROM BUYER, PRESENT
```

```
WHERE BUYER.BUDGET >= PRESENT.COST
```