Modulo 7 - SQL

L'Ereditarietà in SQL: Il Rapporto di Specializzazione

Introduzione al concetto di specializzazione

Il concetto di ereditarietà nel contesto dei database relazionali viene spesso indicato come rapporto di specializzazione. Questo meccanismo permette di rappresentare situazioni in cui un oggetto di un tipo A può appartenere anche a un tipo B, dove B rappresenta una sottocategoria più specifica di A.

Il problema della rappresentazione degli insegnanti

Immaginiamo di voler tracciare informazioni specifiche relative agli insegnanti, come la scuola di riferimento denominata "scuola polo", gli anni di anzianità e il titolo di studio, oltre alle materie per cui sono abilitati. Una prima soluzione potrebbe consistere nel modificare la tabella Person aggiungendo le colonne relative all'anzianità, al titolo di studio e alla scuola polo. Tuttavia, queste colonne resterebbero vuote per tutte le righe che non riguardano insegnanti, generando uno spreco di spazio e una struttura poco elegante.

Simulare l'ereditarietà con il rapporto uno a uno

Quello che desideriamo realizzare è invece una simulazione dell'ereditarietà, poiché i DBMS SQL standard non supportano nativamente questo costrutto. La strategia adottata prevede l'utilizzo di un rapporto uno a uno opzionale, creando una tabella "figlio" che conterrà i dati specifici del sottotipo, separati da quelli del supertipo, facente capo a una tabella "padre".

La categoria più generale, rappresentata dalla classe "padre", viene mappata sulla tabella Person che già conosciamo. A questa tabella principale aggiungiamo una tabella Teacher con la seguente struttura:

- Un campo ID che funge contemporaneamente da chiave primaria e da chiave esterna verso Person

- Un campo SCHOOL per indicare la scuola polo
- Un campo DEGREE per il titolo di studio
- Un campo SUBJECTS per le materie abilitate
- Un campo YEARS per gli anni di anzianità

Il vincolo di chiave esterna garantisce l'integrità referenziale tra le due tabelle.

La duplice funzione della colonna ID

La colonna ID in Teacher svolge una duplice funzione fondamentale. All'interno della tabella Teacher funge da chiave primaria, ma contemporaneamente è anche una chiave esterna che fa riferimento alla tabella Person. Questo significa che ogni riga di Teacher deve necessariamente avere una riga di Person corrispondente per poter essere completa.

Teacher non conosce autonomamente il proprio nome, la propria città o la propria data di nascita, poiché queste informazioni risiedono nella tabella Person. D'altra parte, Person non è obbligato a possedere chiavi esterne verso Teacher, poiché per questa tabella la relazione non è obbligatoria ma opzionale.

Il rapporto risulta quindi:

- **Obbligatorio per Teacher**: tutti gli insegnanti sono necessariamente persone
- **Opzionale per Person**: non tutte le persone sono necessariamente insegnanti

Esempio pratico di dati

Considerando un esempio pratico, supponiamo di avere nella tabella Person i seguenti record:

- Identificativo 1 → A
- Identificativo 2 → B
- Identificativo 3 → C

Nella tabella Teacher troveremo:

- Identificativo 1 → associato alla scuola S1
- Identificativo 3 → associato alla scuola S2

- Identificativo 2 → **assente**

L'assenza dell'identificativo 2 in Teacher non è casuale, ma riflette il fatto che l'ID di Teacher non ha un'esistenza autonoma e deve necessariamente corrispondere a un ID già presente in Person. L'ID di Teacher non può essere auto-incrementale proprio perché potremmo dover saltare dei numeri in base agli ID già presenti in Person.

In questo scenario, i signori A e C risultano essere insegnanti, poiché le loro righe in Person sono collegate a righe in Teacher, e lavorano rispettivamente presso S1 e S2. Il signor B si occupa di altro, e da questi dati non siamo in grado di determinare la sua occupazione specifica.

INNER JOIN: la forma esplicita

Quando abbiamo due tabelle in rapporto uno a uno, che è sempre opzionale per la tabella "padre" rappresentante la superclasse, il predicato di INNER JOIN si ottiene semplicemente uguagliando le chiavi primarie. Più precisamente, si stabilisce l'uguaglianza tra TabellaSuperClasse.PK e TabellaSottoClasse.PK.

Nel nostro caso specifico, la condizione di join diventa semplicemente Person.ID = Teacher.ID, poiché entrambe le tabelle condividono lo stesso identificativo che rappresenta la stessa entità.

Sintassi dell'INNER JOIN

SQL offre una forma esplicita per l'INNER JOIN attraverso un operatore dedicato, che permette di imporre il predicato di join separatamente dalle condizioni di ricerca dei dati. Questa separazione consente di distinguere adeguatamente i predicati di join, che rappresentano condizioni strutturali, dalle condizioni di dominio relative al filtraggio dei dati.

La forma esplicita:

```
SELECT *
FROM PERSON
INNER JOIN TEACHER ON PERSON.ID = TEACHER.ID
```

Forma generale:

```
SELECT *
FROM T1
INNER JOIN T2 ON (PREDICATO DI JOIN)
```

Il risultato ottenuto è identico a quello calcolato con la forma implicita, ma si tratta di una forma più elegante e leggibile.

Il significato di INNER

Il termine INNER è una parola chiave che indica che dobbiamo prendere solo le righe per cui quella condizione è vera, vale a dire soltanto le righe che partecipano effettivamente alla relazione. Possiamo ricordare questo concetto come **"prendi le righe collegate e solo quelle"**.

Applicando l'INNER JOIN ai nostri dati di esempio, abbiamo perso i dati relativi al signor B, poiché la seconda riga di Person, corrispondente all'identificativo 2 con nome B, non ha un corrispettivo nella tabella Teacher e quindi non partecipa alla relazione, venendo conseguentemente esclusa dall'INNER JOIN.

LEFT JOIN: preservare tutte le righe

Tuttavia, se desiderassimo visualizzare anche B nella nostra query, non potremmo utilizzare semplicemente l'INNER JOIN. Chiaramente non potremmo associargli una scuola, poiché non è un insegnante, ma potremmo comunque volerlo includere nella lista dei risultati.

Per ottenere questo risultato utilizziamo un tipo di join diverso, chiamato **LEFT JOIN**, che viene tipicamente impiegato per queste evenienze.

Sintassi del LEFT JOIN

Forma generale:

```
SELECT *
FROM T1
LEFT JOIN T2 ON (PREDICATO DI JOIN)
```

Come funziona il LEFT JOIN

Il LEFT JOIN si interpreta nel seguente modo: prendi le righe di T1 e collegale alle righe di T2 per cui vale la relazione espressa dal predicato di join. Se non trovi righe di T2 che soddisfano la condizione, presenta comunque la riga di T1 combinandola con una riga contenente valori NULL per tutti i campi di T2.

Possiamo ricordare questo comportamento come **"preserva tutte le righe di T1"** o **"preserva tutte le righe a sinistra"**, indipendentemente dal fatto che partecipino o meno alla relazione.

Esempio pratico di LEFT JOIN

Applicando questo join ai nostri dati di esempio:

```
SELECT *
FROM PERSON
LEFT JOIN TEACHER ON PERSON.ID = TEACHER.ID
```

Otterremo tre righe:

1. I dati del signor A e la sua scuola S1
2. I dati del signor B e valori NULL per i campi di Teacher
3. I dati del signor C e la sua scuola S2

La riga relativa al signor B, che non partecipa alla relazione con Teacher, è comunque presente nel risultato ed è stata associata a una riga composta interamente da valori nulli per i campi della tabella Teacher.

L'importanza dell'ordine

È importante sottolineare che il LEFT JOIN non escluderà mai le righe della tabella che si trova a sinistra dell'operatore, che possono essere quelle della tabella rappresentante la superclasse, ma possono anche essere quelle della tabella lato uno in un rapporto uno a molti, come vedremo successivamente.

L'ordine delle tabelle nel join ha una importanza cruciale: è la tabella che compare a sinistra dell'operatore LEFT JOIN che viene preservata, mentre non ci sono garanzie analoghe per quella posta sulla destra.

Combinare JOIN con condizioni di dominio

Nulla vieta che una riga venga esclusa da una condizione di dominio. Supponiamo di voler trovare tutte le persone che lavorano nella città con identificativo uguale a uno, visualizzando per gli insegnanti anche la scuola di appartenenza.

La soluzione:

```
SELECT PERSON.NAME, PERSON.SURNAME, PERSON.JOB, TEACHER.SCHOO
L
FROM PERSON
LEFT JOIN TEACHER ON PERSON.ID = TEACHER.ID
WHERE PERSON.CITYID = 1
```

Sebbene il LEFT JOIN mantenga tutte le righe della tabella Person, molte di queste righe verranno eliminate dalla condizione di dominio, che impone che CITYID sia uguale a uno.

Aggiungere una terza tabella

L'utilizzo di CITYID non è casuale: utilizzando esclusivamente le tabelle Person e Teacher non avremmo accesso al nome della città, poiché questa informazione risiede in una tabella separata. Se avessimo voluto ottenere lo stesso risultato per gli abitanti di Milano, ci sarebbe servita una terza tabella denominata City.

La query completa:

```
SELECT PERSON.NAME, PERSON.SURNAME, PERSON.JOB, TEACHER.SCHOO
L
FROM PERSON
INNER JOIN CITY ON CITY.ID = PERSON.CITYID
LEFT JOIN TEACHER ON PERSON.ID = TEACHER.ID
WHERE CITY.NAME = 'MILANO'
```

Questa query si traduce così:

1. Associa a ogni persona la propria città, pretendendo di avere la città o scartando la persona attraverso l'INNER JOIN
2. A questa combinazione aggiungi, se esiste, una riga di Teacher collegata a Person

3. Altrimenti aggiungi una riga di valori nulli per i campi di Teacher

La struttura del risultato

Analizzando l'esempio precedente e stabilendo delle regole generali, il risultato di questa query, prima della proiezione dei campi nome, cognome, lavoro e scuola, è una lista di righe del tipo **{P(i), C(k), T(i)}**, dove:

- **P(i)** rappresenta la riga di Person con identificativo i
- **C(k)** rappresenta la riga di City con identificativo k collegata a P(i)
- **T(i)** rappresenta l'eventuale Teacher collegato a P(i)

È importante notare che Person e Teacher condividono lo stesso identificativo i, poiché il join viene eseguito sulla chiave primaria. T(i) sarà nullo nel caso in cui la persona non sia un insegnante, presentando quindi campi vuoti per tutte le informazioni specifiche del docente.

Regola generale per i JOIN multipli

Come regola generale, quando abbiamo **n tabelle** nella clausula FROM, avremo **n-1 join**, che possono essere di tipo LEFT o INNER, opzionali o obbligatori.

Il risultato sarà composto da **super-righe divise in n parti**, prese una da ogni tabella. Su quella super-riga eseguiremo la proiezione per ricavare esattamente le informazioni che ci interessano.

Questa regola si applica a ogni tipologia di relazione, non soltanto a quelle di tipo uno a uno.