Azzolini Riccardo 2019-10-24

Sotto-interrogazioni

1 Sotto-interrogazioni

Una **sotto-interrogazione** (**subquery**) è un'istruzione `SELECT` contenuta all'interno di un'altra interrogazione. Essa può essere introdotta:

- in un predicato della clausola `WHERE`;
- in un predicato della clausola `HAVING`;
- nella clausola `FROM`.

Con le sotto-interrogazioni, è possibile suddividere un problema complesso in sotto-problemi più semplici.

Una sotto-interrogazione può contenere qualsiasi costrutto ammesso nelle interrogazioni "normali", compresi join, raggruppamenti, e altre sotto-interrogazioni.

2 Tipi di sotto-interrogazioni

Le sotto-interrogazioni si suddividono in tre tipi, a seconda dei risultati che restituiscono:

Subquery scalare: restituisce un singolo valore. Ad esempio:

```
SELECT MAX(valutazione) FROM Film;
```

Subquery di colonna: restituisce una colonna, cioè un insieme di valori. Ad esempio:

```
SELECT valutazione FROM Film;
```

Subquery di tabella: restituisce una tabella con più di un attributo. Ad esempio:

```
SELECT titolo, regista FROM Film;
```

3 Subquery in `WHERE` e `HAVING`

Nelle clausole `WHERE` e `HAVING`, le sotto-interrogazioni possono essere usate per:

- determinare uno o più valori da usare come valori di confronto in un predicato dell'interrogazione esterna;
- esprimere quantificazioni universali o esistenziali.

3.1 Subquery scalari

L'uso di una subquery scalare in `WHERE` o `HAVING` permette di eseguire un confronto con un valore che non è noto a priori, ma è calcolabile. Infatti, il valore restituito dalla subquery può essere usato con gli operatori `=`, `<`, ecc.

Quando una subquery è usata in questo modo:

- Se nessuna tupla verifica la sotto-interrogazione, viene restituito il valore `NULL`.
- Se più di una tupla verifica la sotto-interrogazione (cioè la subquery non è effettivamente scalare), si ha un errore a runtime. Per questo, se non si è sicuri che una determinata subquery restituisca sempre un solo valore, è meglio non usarla con operatori scalari.

3.1.1 Esempi

Determinare il titolo di tutti i film che hanno la stessa valutazione del film "Le iene":

```
SELECT titolo
FROM Film
WHERE valutaz = (
  SELECT valutaz
  FROM Film
  WHERE titolo = 'le iene'
);
```

In questo caso, la formulazione mediante subquery ha permesso di separare il problema in due sotto-problemi:

1. valutazione del film "Le iene";
2. titolo dei film con la stessa valutazione.

Si può anche ottenere una formulazione equivalente con il join:

- la clausola `FROM` deve contenere le tabelle referenziate nelle `FROM` di tutte le `SELECT`;

- devono essere specificate le opportune condizioni di join (nella clausola `WHERE`, o direttamente nel `FROM`);
- eventuali predicati di selezione devono essere aggiunti nella clausola `WHERE`.

```
SELECT F2.titolo
FROM Film F1, Film F2
WHERE F1.titolo = 'le iene'
  AND F1.valutaz = F2.valutaz;
```

Non tutte le interrogazioni che possono essere espresse mediante subquery hanno una formulazione equivalente con il join. Ad esempio, le subquery permettono il confronto dei singoli valori di una colonna con un valore aggregato, che non si può fare con una singola interrogazione. In seguito sono riportati alcuni esempi di query che possono essere espresse solo mediante sotto-interrogazioni.

Determinare i film la cui valutazione è superiore alla media:

```
SELECT *
FROM Film
WHERE valutaz > (
  SELECT AVG(valutaz)
  FROM Film
);
```

Determinare il titolo e il regista del film drammatico di valutazione minima:

```
SELECT titolo, regista
FROM Film
WHERE genere = 'drammatico'
  AND valutaz = (
    SELECT MIN(valutaz)
    FROM Film
    WHERE genere = 'drammatico'
  );
```

3.2 Subquery di colonna

Per utilizzare una subquery di colonna in `WHERE` o `HAVING`, è necessario specificare come i valori multipli restituiti devono essere usati, aggiungendo dopo l'operatore di confronto la parola chiave `ANY` oppure `ALL`:

- Con `ANY`, il predicato è vero se la condizione è soddisfatta da almeno uno dei valori restituiti dalla sotto-interrogazione. Se non è stato restituito alcun valore, non ce ne può essere uno che soddisfa la condizione, quindi il predicato è falso.

- Con `ALL`, il predicato è vero se tutti i valori restituiti dalla sotto-interrogazione soddisfano la condizione. Se la subquery non restituisce nessun valore, allora è vero che "tutti" i valori dell'insieme vuoto risultante soddisfano la condizione, quindi il predicato è vero.

Come casi particolari:

- `>= ALL` è equivalente a `MAX`;
- `<= ALL` è equivalente a `MIN`;
- `= ANY` verifica l'appartenenza di un elemento all'insieme di valori restituiti dalla subquery, e si può scrivere anche come `IN`;
- `<> ALL` verifica se un elemento non appartiene all'insieme, e si può scrivere anche come `NOT IN`.

3.2.1 Esempi

Determinare titolo, regista e anno dei film più vecchi di tutti i film di Quentin Tarantino:

```sql
SELECT titolo, regista, anno
FROM Film
WHERE anno < ALL (
  SELECT anno
  FROM Film
  WHERE regista = 'quentin tarantino'
);
-- oppure, con MIN:
SELECT titolo, regista, anno
FROM Film
WHERE anno < (
  SELECT MIN(anno)
  FROM Film
  WHERE regista = 'quentin tarantino'
);
```

Determinare il titolo e il regista del film drammatico di valutazione minima (formulando la query con `<= ALL` invece di `MIN`):

```sql
SELECT titolo, regista
FROM Film
WHERE genere = 'drammatico'
  AND valutaz <= ALL (
    SELECT valutaz
```

```
    FROM Film
    WHERE genere = 'drammatico'
  );
```

Determinare nome e reddito dei padri di persone che guadagnano più di 20:

```
SELECT Padre, Reddito
FROM Paternita
  JOIN Persone ON Nome = Padre
WHERE Figlio = ANY ( -- si può scrivere come: Figlio IN (...)
  SELECT Nome
  FROM Persone
  WHERE Reddito > 20
);
-- oppure, con il join:
SELECT Padre, p.Reddito
FROM Paternita
  JOIN Persone p ON p.Nome = Padre
  JOIN Persone f ON f.Nome = Figlio
WHERE f.Reddito > 20;
```

Dato lo schema

$$\text{Impiegato}(\text{Nome}, \underline{\text{CF}}, \text{Indirizzo}, \text{Stipendio}, \text{Resp}^{\text{Impiegato}}, \text{Dip}^{\text{Dipartimento}})$$
$$\text{Dipartimento}(\text{Nome}, \underline{\text{IdD}}, \text{Manager}^{\text{Impiegato}}, \text{ManagerData\_In})$$

determinare il dipartimento che spende il massimo in stipendi:

```
SELECT Dip
FROM Impiegato
GROUP BY Dip
HAVING SUM(Stipendio) >= ALL (
  SELECT SUM(Stipendio)
  FROM Impiegato
  GROUP BY Dip
);
```

3.2.2 Concetto di esclusione

Alcune query possono essere espresse "a esclusione": si parte dall'insieme di tutte le tuple, e si escludono gli elementi che non interessano. Questo tipo di interrogazioni si possono realizzare sia con sotto-interrogazioni e `NOT IN` che con l'operatore insiemistico `EXCEPT`.

Ad esempio, per determinare il codice dei clienti che *non* hanno noleggiato film di Tim Burton, si ottiene l'insieme dei clienti che invece hanno noleggiato tali film e lo si sottrae all'insieme di tutti i clienti:

```
SELECT codCli
FROM Cliente
WHERE codCli NOT IN (
  SELECT codCli
  FROM Noleggio NATURAL JOIN Video
  WHERE regista = 'tim burton'
);
-- oppure, con EXCEPT:
SELECT codCli
FROM Cliente
EXCEPT
SELECT codCli
FROM Noleggio NATURAL JOIN Video
WHERE regista = 'tim burton';
```

Invece, la query

```
SELECT codCli
FROM Noleggio NATURAL JOIN Video
WHERE regista <> 'tim burton';
```

è sbagliata, perché seleziona invece i clienti che hanno noleggiato film di registi diversi da Tim Burton, ma gli stessi clienti potrebbero aver noleggiato anche film di Tim Burton.

Un altro esempio è trovare il codice dei clienti che hanno noleggiato *solo* film di Tim Burton. In questo caso, bisogna considerare l'insieme dei clienti che hanno noleggiato almeno un film ed escludere l'insieme dei clienti che hanno noleggiato film non di Tim Burton:

```
SELECT codCli
FROM Noleggio
WHERE codCli NOT IN (
  SELECT codCli
  FROM Noleggio NATURAL JOIN Video
  WHERE regista <> 'tim burton'
);
-- oppure, con EXCEPT:
SELECT codCli
FROM Noleggio
EXCEPT
SELECT codCli
```

```
FROM Noleggio NATURAL JOIN Video
WHERE regista <> 'tim burton'
```

Il fatto che i clienti selezionati abbiano noleggiato almeno un film è garantito estraendo i loro codici dalla tabella Noleggio. Se, invece, essi venissero presi dalla tabella Cliente,

```
SELECT codCli
FROM Cliente
WHERE codCli NOT IN (
  SELECT codCli
  FROM Noleggio NATURAL JOIN Video
  WHERE regista <> 'tim burton'
);
```

sarebbero inclusi nel risultato anche i clienti che non hanno noleggiato nessun film (e quindi neanche un film di Tim Burton), perciò la query non sarebbe corretta.

3.3 Subquery di tabella

Quando una sotto-interrogazione restituisce una tabella con più di un attributo, il confronto si effettua tra tuple, quindi nel predicato di confronto è necessario usare il **costruttore di tupla**: esso permette di definire la struttura temporanea di una tupla, elencando tra parentesi i nomi degli attributi che ne fanno parte.

L'ordine in cui si elencano gli attributi (nel costruttore di tupla e nella `SELECT` della sotto-interrogazione) è importante, perché il predicato confronta gli attributi nelle posizioni corrispondenti delle due tuple alle quali si applica.

Le subquery di tabella estendono il potere espressivo di `IN` e `NOT IN`, permettendo di verificare l'appartenenza di una tupla (invece che solo di un singolo valore) a un insieme.

3.3.1 Esempio

Dato lo schema

$$\text{Viaggio}(\underline{\text{CodV}}, \text{Partenza}, \text{Arrivo}, \text{OraP}, \text{OraA})$$

trovare le coppie di luogo di partenza e luogo d'arrivo per cui nessun viaggio dura più di due ore:

```
SELECT Partenza, Arrivo
FROM Viaggio
WHERE (Partenza, Arrivo) NOT IN (
  SELECT Partenza, Arrivo
  FROM Viaggio
```

```
  WHERE OraA - OraP > 2
);
-- oppure, con EXCEPT:
SELECT Partenza, Arrivo
FROM Viaggio
EXCEPT
SELECT Partenza, Arrivo
FROM Viaggio
WHERE OraA - OraP > 2;
```