Azzolini Riccardo 2019-11-04

Sotto-interrogazioni correlate

1 Sotto-interrogazioni correlate

Una subquery "semplice" viene eseguita una sola volta, prima dell'interrogazione esterna, e il valore (o l'insieme di valori) restituito viene poi usato nel predicato dell'interrogazione esterna. L'esecuzione delle due query è quindi scorrelata.

A volte, invece, ha senso eseguire la subquery in modo **correlato**, con valori presi dalle *tuple candidate* della query esterna. In questo caso, la sotto-interrogazione viene eseguita ripetutamente, una volta per ogni tupla candidata considerata nella valutazione dell'interrogazione esterna (come se le due interrogazioni fossero cicli `for` annidati).

Per usare i valori della query esterna nella subquery, è necessario assegnare alla relazione della query esterna un *alias*. L'alias può essere definito nella query esterna e riferito nelle sotto-interrogazioni, ma non viceversa (come appunto nei cicli `for` annidati: quello interno utilizzare l'indice di quello esterno, ma non è consentito il contrario).

1.1 Esempi

Determinare titolo, regista e anno dei film la cui valutazione è superiore alla media delle valutazioni dei film *dello stesso regista*:

```
SELECT titolo, regista, anno
FROM Film X
WHERE valutaz > (
  SELECT AVG(valutaz)
  FROM Film
  WHERE regista = X.regista
);
```

Dato lo schema

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

trovare il codice dei viaggi che hanno una durata inferiore alla durata media dei viaggi sullo stesso percorso (caratterizzato dallo stesso luogo di partenza e di arrivo):

```sql
SELECT CodV
FROM Viaggio V
WHERE OraA - OraP < (
  SELECT AVG(OraA - OraP)
  FROM Viaggio
  WHERE Partenza = V.Partenza
    AND Arrivo = V.Arrivo
);
```

2 EXISTS e NOT EXISTS

L'operatore `EXISTS` verifica se esiste almeno una tupla che soddisfa una sotto-interrogazione: se la subquery restituisce almeno una tupla, l'operatore restituisce il valore booleano `TRUE`, altrimenti restituisce `FALSE`. Viceversa, `NOT EXISTS` dà risultato `TRUE` se la subquery *non* restituisce tuple, e `FALSE` se invece ne restituisce almeno una.

Questi operatori hanno senso con le sotto-interrogazioni correlate: una subquery semplice viene valutata una sola volta, quindi usandola con (`NOT`) `EXISTS` si otterrebbe un predicato che, per tutte le tuple della query esterna, sarebbe sempre vero oppure sempre falso.

2.1 Esempi

Determinare il nome e l'età delle persone che hanno almeno un figlio:

```sql
SELECT Nome, Eta
FROM Persone P
WHERE
  EXISTS (
    SELECT *
    FROM Paternita
    WHERE Padre = P.Nome
  )
  OR EXISTS (
    SELECT *
    FROM Maternita
    WHERE Madre = P.Nome
  );
-- oppure, con una subquery semplice:
SELECT Nome, Eta
FROM Persone
WHERE
  Nome IN (
```

```
    SELECT Padre
    FROM Paternita
  )
  OR Nome IN (
    SELECT Madre
    FROM Maternita
  );
```

Trovare il codice dei clienti che *non* hanno noleggiato film di Tim Burton:

```
SELECT codCli
FROM Cliente C
WHERE NOT EXISTS (
  SELECT *
  FROM Noleggio NATURAL JOIN Video
  WHERE regista = 'tim burton'
    AND codCli = C.codCli
);
```

Dato lo schema

$$\text{Studenti}(\underline{\text{Matricola}}, \text{Nome}, \text{Cognome})$$

trovare tutti gli studenti che *non* hanno un omonimo:

```
SELECT *
FROM Studenti S
WHERE NOT EXISTS (
  SELECT *
  FROM Studenti
  WHERE Nome = S.Nome
    AND Cognome = S.Cognome
    AND Matricola <> S.Matricola
);
```

2.2 Intersezione e differenza

Le operazioni di intersezione e differenza possono essere eseguite anche tramite `EXISTS` e `NOT EXISTS`.

Ad esempio, la query seguente, che determina gli anni in cui sono usciti sia film di Tim Burton sia film di Quentin Tarantino,

```
SELECT DISTINCT anno
FROM Film
WHERE regista = 'tim burton'
INTERSECT
```

```sql
SELECT DISTINCT anno
FROM Film
WHERE regista = 'quentin tarantino';
```

si può scrivere anche come:

```sql
SELECT DISTINCT anno
FROM Film F
WHERE regista = 'tim burton'
  AND EXISTS (
    SELECT *
    FROM Film
    WHERE regista = 'quentin tarantino'
      AND anno = F.anno
  );
```

3 Quantificatore universale

Usando le sotto-interrogazioni correlate insieme all'operatore `NOT EXISTS`, è possibile esprimere il quantificatore universale. A tale scopo, è necessario ragionare in base al concetto di *controesempio*: per verificare che una proprietà valga per tutti gli elementi di un insieme, bisogna controllare che non esista un controesempio, cioè un elemento per il quale essa non vale.

Ad esempio, se si vogliono determinare i codici dei clienti che hanno noleggiato tutti i film di Tim Burton, è necessario riformulare l'interrogazione come: "determinare i codici dei clienti per cui non esiste un film di Tim Burton che il cliente non ha mai noleggiato".

```sql
SELECT DISTINCT codCli
FROM Noleggio X
WHERE NOT EXISTS (
  -- film di Tim Burton che il cliente non ha mai noleggiato
  SELECT *
  FROM Film F
  WHERE regista = 'tim burton'
    AND NOT EXISTS (
      -- noleggio del film considerato da parte del cliente in esame
      SELECT *
      FROM Noleggio NATURAL JOIN Video
      WHERE codCli = X.codCli
        AND titolo = F.titolo
        AND regista = F.regista -- oppure: regista = 'tim burton'
    )
);
```

In questo caso, la query può essere espressa anche con le funzioni di gruppo, determinando quali clienti hanno noleggiato un numero di film di Tim Burton diversi uguale al numero di tutti i film di Tim Burton:

```
SELECT codCli
FROM Noleggio NATURAL JOIN Video
WHERE regista = 'tim burton'
GROUP BY codCli
HAVING COUNT(DISTINCT titolo) = (
  SELECT COUNT(*)
  FROM Film
  WHERE regista = 'tim burton'
);
```