

Interpretazione delle query nidificate

Per analizzare il risultato di una interrogazione nidificata si può supporre di valutare prima il risultato dell'interrogazione nidificata (query interna) per poi 'sostituirlo' nell'interrogazione che la contiene (query esterna) e valutare quest'ultima. Questo migliora anche l'efficienza, in quanto l'interrogazione nidificata viene eseguita una sola volta.

Tuttavia, è possibile farlo solo se le due query sono indipendenti fra loro, cioè se la query interna non varia al variare della specifica tupla della query esterna che si deve confrontare col risultato della query interna.

Quando esiste questa dipendenza, espressa tramite l'uso di un alias comune alle due query, si parla di *passaggio di binding* fra le due interrogazioni.

Interpretazione delle query nidificate

Se esiste un riferimento (passaggio di binding) fra le due query, bisogna usare la definizione 'procedurale' di query, che calcola il prodotto cartesiano fra le tabelle e poi verifica la condizione **where** separatamente per ogni riga.

Quindi per ogni riga della query esterna (da cui dipende anche la query interna) si valuta prima la corrispondente query nidificata, per poi calcolare il predicato logico a livello di riga sulla query esterna.

Nota Di fatto, si realizzano n theta-join separati fra ciascuna delle n righe risultanti dalla query esterna e la tabella risultante dall'esecuzione della query interna, istanziando in essa gli alias coi valori corrispondenti della query esterna.

Interrogazioni nidificate

Le variabili SQL (alias condivisi fra più interrogazioni) sono utilizzabili solo nella query in cui sono definite o nell'ambito di una query nidificata al suo interno. Se due query sono allo stesso livello non possono condividere variabili.

exists

è un operatore logico applicabile a query nidificate.

Restituisce vero se la query dà un risultato non nullo, falso se è nullo.

E' utilizzabile in modo significativo solo se esiste un passaggio di binding fra interrogazione esterna e interrogazione nidificata.

Le persone che hanno almeno un figlio

```
select *  
from Persone P  
where exists ( select *  
               from Paternita  
               where Padre = P.Nome)  
  
or          exists ( select *  
                    from Maternita  
                    where Madre = P.Nome)
```

exists

```
select *  
from Persona P  
where exists ( select *  
               from Persona P1  
               where P1.Nome = P.Nome  
               and P1.Cognome = P.Cognome  
               and P1.CFiscale <> P.CFiscale)
```

In questo caso non è possibile eseguire prima la query nidificata, in quanto è indeterminata se non si risolve il riferimento (passaggio di binding). Quindi per ogni riga dell'interrogazione esterna dovrà essere valutata una diversa interrogazione nidificata.

Costruttore di tuple

Quando si deve valutare una condizione che coinvolge un insieme di attributi, la lista degli attributi coinvolti viene inserita all'interno di una parentesi tonda.

Es.

```
select *  
from   Persona P  
where  (Nome, Cognome) in  
      ( select Nome, Cognome  
        from   Persona Q  
        where  P.CFiscale <> Q.CFiscale)
```

Inserimento

```
insert into Dipartimento (NomeDip, Città)
values ('Produzione', 'Torino')
```

Si utilizza in genere mediante una maschera (*form*) per consentire agli utenti di inserire dati. L'ordinamento degli attributi (se presente) e dei valori è significativo e le due liste devono avere lo stesso numero di elementi

```
insert into ProdottiMilanesi
      (select codice, descrizione
       from   Prodotto
       where  LuogoProd = 'Milano')
```

Se in un inserimento non vengono inseriti tutti i dati di una riga si usa o il default (se esiste) o il valore nullo. La corrispondenza fra valori inseriti e attributi è per posizione.

```
INSERT INTO Persone VALUES  
('Mario',25,52)
```

```
INSERT INTO Persone(Nome, Eta, Reddito)  
VALUES('Pino',25,52)
```

```
INSERT INTO Persone(Nome, Reddito)  
VALUES('Lino',55), ('Gino',43)
```

```
INSERT INTO Persone ( Nome )  
SELECT Padre  
FROM Paternita  
WHERE Padre NOT IN (SELECT Nome  
FROM Persone)
```

Cancellazione

```
delete from NomeTabella [ where Condizione ]
```

se **where** non è specificato tutte le righe della tabella vengono eliminate. Se esiste un vincolo di integrità referenziale con politica di **cascade** ... ATTENZIONE!!!!

Siccome **delete** ha la stessa sintassi di **select**, è possibile utilizzare interrogazioni nidificate al suo interno.

```
delete from Dipartimento  
      where Nome not in (select Dipart  
                        from Impiegato)
```

NB:

delete from Dipartimento distrugge *il contenuto della tabella*
drop table Dipartimento cascade altera lo schema

Modifica

update *NomeTabella*

set *Attributo* = <*Espressione*|*SelectSQL*|**null**|**default**>

{, *Attributo* = <*Espressione*|*SelectSQL*|**null**|**default**>}

[**where** *Condizione*]

aggiorna uno o più attributi delle righe di *NomeTabella* che soddisfano l'eventuale *Condizione*. Se non c'è condizione la modifica avviene per tutte le righe.

Il nuovo valore può essere:

- il risultato di un'espressione valutata sugli attributi della tabella
- il risultato di una generica interrogazione SQL
- il valore nullo
- il valore di default

Modifica

```
update Dipendente
```

```
set Stipendio = StipendioBase + 5
```

```
where Matricola = 'XYZ'
```

aggiorna una sola riga (Matricola è chiave)

```
update Dipendente
```

```
set Stipendio = Stipendio * 1.1
```

```
where Dipart = 'Amministrazione'
```

aggiorna un insieme di righe

Nella valutazione delle espressioni bisogna ricordarsi che SQL ha una natura orientata agli insiemi e non alle tuple. Quindi non è possibile pensare all'esecuzione di un comando come esecuzione riga per riga, ma come un'operazione effettuata contemporaneamente su tutto un insieme.

Modifica

Vogliamo aumentare del 10% gli stipendi degli impiegati con stipendio inferiore ai 30 milioni e del 15% quello degli impiegati con stipendio superiore ai 30 milioni.

Se scriviamo

```
update Impiegato set Stipendio = Stipendio*1.1  
where Stipendio <= 30
```

```
update Impiegato set Stipendio = Stipendio*1.15  
where Stipendio > 30
```

ad alcuni impiegati lo stipendio viene aumentato 2 volte!!!

Se invertiamo l'ordine la procedura risulta corretta. Ma non sempre è così facile.....

Vincoli di integrità generici

Con i costrutti visti sinora, non è sempre possibile definire tutti i possibili vincoli di integrità.

Per questo esiste l'istruzione

check (*Condizione*)

La condizione specificata deve essere verificata da tutte le tuple in ogni momento.

E' possibile con **check** definire tutti i vincoli predefiniti. Però è meno leggibile e non si possono applicare le politiche di reazione alle violazioni. Tuttavia è molto potente

Assertzioni

Le asserzioni sono vincoli che fanno parte dello schema; non sono associati ad alcun attributo o tabella. Permettono di definire tutti i vincoli utilizzabili nella definizione di una tabella, ma anche vincoli su più tabelle o vincoli che richiedono che una tabella abbia certe caratteristiche (es. una certa cardinalità).

```
create assertion NomeAsserzione  
check(Condizione)
```

Es.

```
create assertion AlmenoUnImpiegato  
check (1 <= (select count(*)  
                from Impiegato))
```

Assertzioni

Ogni vincolo di integrità è associato ad una politica di controllo che specifica se è *immediato* o *differito*.

Se è *immediato* è verificato immediatamente dopo una modifica.

Se è *differito* solo al termine dell'esecuzione di una serie di operazioni che si considerano parte di un'unica azione atomica (*transazione*).

Se un vincolo immediato viene violato, l'operazione di modifica può essere annullata immediatamente (*rollback parziale*). Tutti i vincoli predefiniti sono immediati.

Se invece è differito, al momento in cui si verifica la violazione non è più possibile identificare l'operazione che l'ha causata e quindi va annullata tutta la transazione (*rollback totale*).

Questo garantisce la consistenza della base di dati.

Per cambiare il tipo di controllo:

```
set constraints [NomeVincolo] immediate
```

```
set constraints [NomeVincolo] deferred
```