SQL: interrogazioni avanzate, viste, trigger

corso di basi di dati e laboratorio

Prof. Alfio Ferrara

Anno Accademico 2017/2018

Indice

1 Interrogazioni avanzate

1.1 La clausola WITH

Common Table Expressions (CTEs)

- Lo standard SQL 99 ha introdotto in SQL la nozione di Common Table Expression (CTE).
- Una CTE è una tabella temporanea che esiste e può essere utilizzata solo nell'ambito di un'interrogazione.
- Le CTE sono realizzate attraverso la clausola `WITH`, secondo la seguente sintassi:

```
WITH [ RECURSIVE ] with_query [, ...]
SELECT ...
```

In cui la `with_query` è:
`with_query_name [ ( column_name [, ...] ) ] AS ( SELECT ... )`

Esempio (passo 1)
Individuare per ogni sistema monetario il numero di nazioni il cui valore di prodotto interno lordo diviso per la sola popolazione urbana è superiore alla media.

Trovo il valore di prodotto interno lordo diviso per la sola popolazione urbana:

```
SELECT P.country, (G.value / P.value) AS urban_gdp
FROM measure AS P JOIN measure AS G
ON P.country = G.country
WHERE P.label = 'Urban Population' AND G.label = 'GDP'
```

Esempio (passo 2)
Individuare per ogni sistema monetario il numero di nazioni il cui valore di prodotto interno lordo diviso per la sola popolazione urbana è superiore alla media.

Creo una CTE `urban_data` per l'interrogazione precedente e la uso per trovare il valore medio:

```
WITH urban_data(country, urban_gdp) AS (
  SELECT P.country, (G.value / P.value) AS urban_gdp
  FROM measure AS P JOIN measure AS G
  ON P.country = G.country
  WHERE P.label = 'Urban Population' AND G.label = 'GDP')
SELECT AVG(urban_gdp) FROM urban_data;
```

Esempio (passo 3)

Individuare per ogni sistema monetario il numero di nazioni il cui valore di prodotto interno lordo diviso per la sola popolazione urbana è superiore alla media.

Creo una successiva CTE `top_countries` che restituisca la moneta e il codice della nazione per le sole nazioni con un valore superiore alla media:

```
WITH urban_data(country, urban_gdp) AS (
  SELECT P.country, (G.value / P.value) AS urban_gdp
  FROM measure AS P JOIN measure AS G
  ON P.country = G.country
  WHERE P.label = 'Urban Population' AND G.label = 'GDP'),
top_countries AS (
  SELECT C.iso3, C.currency_type
  FROM country AS C JOIN urban_data AS U
  ON C.iso3 = U.country
  WHERE U.urban_gdp > (SELECT AVG(urban_gdp) FROM urban_data) )
```

Esempio (soluzione finale)
Sfrutto la CTE `top_countries` per calcolare il numero di nazioni il cui valore di prodotto interno lordo diviso per la sola popolazione urbana è superiore alla media raggruppando per sistema monetario:

```
WITH urban_data(country, urban_gdp) AS (
  SELECT P.country, (G.value / P.value) AS urban_gdp
  FROM measure AS P JOIN measure AS G
  ON P.country = G.country
  WHERE P.label = 'Urban Population' AND G.label = 'GDP'),
top_countries AS (
  SELECT C.iso3, C.currency_type
  FROM country AS C JOIN urban_data AS U
  ON C.iso3 = U.country
  WHERE U.urban_gdp > (SELECT AVG(urban_gdp) FROM urban_data) )
SELECT currency_type, COUNT(iso3) FROM top_countries
GROUP BY currency_type;
```

1.2 Interrogazioni ricorsive

Uso della clausola RECURSIVE

- In generale la clausola `WITH` è prevalentemente una facilitazione sintattica per interrogazioni che sarebbero comunque possibili attraverso query nidificate, correlate e viste (si veda lucidi successivi).
- Aggiungendo la clausola `RECURSIVE` diventa però possibile usare `WITH` per interrogazioni ricorsive altrimenti impossibili in SQL.
- Esempio astratto (generazione ricorsiva di numeri interi):

```
WITH RECURSIVE t(n) AS (
SELECT 1
UNION ALL
SELECT n+1 FROM t WHERE n < 100
)
SELECT n FROM t;
```

Funzionamento delle query ricorsive

Una query ricorsiva è sempre composta da un termine di passo (non ricorsivo) seguito da `UNION (ALL)` e da un termine ricorsivo. Solo il termine ricorsivo può contenere un riferimento all'output della query.
Esecuzione:

1. Valutazione del termine non ricorsivo. Usando `UNION` (e non `UNION ALL`), si scartano i dublicati. Inclusione di tutte le tuple in una tabella temporanea T_0.
2. Fintanto che la tabella temporanea T_0 non è vuota si ripetono i seguenti passi:
   (a) Valutazione del termine ricorsivo, interrogando la tabella temporanea T_0. Con `UNION` (ma non con `UNION ALL`), si scarta ogni duplicato. Includere tutte le tuple risultanti nel risultato R della query e anche in una successiva tabella temporanea T_1.
   (b) Sostituire il contenuto di T_0 con il contenuto di T_1 e successivamente svuotare T_1.

Esempio

```
WITH RECURSIVE t(n) AS (
SELECT 1
```

```
UNION ALL
SELECT n+1 FROM t WHERE n < 100
)
SELECT SUM(n) FROM t;
```

Passaggi di esecuzione:

1. Esecuzione del temine non ricorsivo: $T_0 = \{\langle 1 \rangle\}$, $T_1 = \emptyset$, $R = \emptyset$
2. Esecuzione del temine ricorsivo: $T_0 = \{\langle 1 \rangle\}$, $T_1 = \{\langle 2 \rangle\}$, $R = \{\langle 1 \rangle \langle 2 \rangle\}$
3. Aggiornamento di T_0: $T_0 = \{\langle 2 \rangle\}$, $T_1 = \emptyset$, $R = \{\langle 1 \rangle \langle 2 \rangle\}$
4. ...
5. Condizione di chiusura: $T_0 = \{\langle 100 \rangle\}$, $T_1 = \emptyset$, $R = \{\langle 1 \rangle \ \ldots, \langle 100 \rangle\}$

Osservazioni

- Si noti che la condizione di selezione del termine ricorsivo determina anche la condizione di terminazione della ricorsione.
- L'algoritmo di esecuzione è di natura iterativa e non ricorsiva, anche se il risultato e la logica di impostazione dell'interrogazione sono ricorsive.
- Le query ricorsive sono spesso utilizzate per interrogare tabelle che rappresentano relazioni gerarchiche (visita dell'albero) o relazioni di connessione fra oggetti (visite del grafo).
- Nel secondo caso, occorre prestare attenzione ai cicli che impedirebbero la terminazione della ricorsione: in questi casi, l'uso della condizione `UNION` invece che `UNION ALL` può essere utile per eliminare i duplicati (ovvero per non ripercorrere lo stesso cammino nel grafo).

Esempio (1)

Individuare tutte le nazioni raggiungibili direttamente o indirettamente dall'Italia.

BORDERS

country	country1
ITA	FRA
ITA	AUT
ITA	SVN
ITA	CHE
FRA	ESP
FRA	DEU
FRA	BEL
DEU	BEL

```
WITH RECURSIVE
search_path(f_country, t_country, depth)
AS (
 SELECT g.country, g.country1, 1
 FROM borders g
  UNION
 SELECT sg.f_country, g.country1, sg.depth + 1
 FROM borders g, search_graph sg
 WHERE g.country = sg.t_country
 AND sg.f_country = 'ITA'
)
SELECT * FROM search_path WHERE f_country = 'ITA';
```

Esempio (2)

SEARCH_PATH

f_country	t_country	depth
ITA	FRA	1
ITA	AUT	1
ITA	SVN	1
ITA	CHE	1
ITA	ESP	2
ITA	DEU	2
ITA	BEL	2
ITA	BEL	3

L'interrogazione trova tutti i percorsi possibili a partire dall'Italia e la distanza.

Si noti l'uso della clausola `UNION` e il fatto che avendo proiettato anche la lunghezza del percorso, ciò che è univoco ai fini dell'unione è l'insieme della nazione di provenienza, di arrivo e anche la lunghezza del persorso.

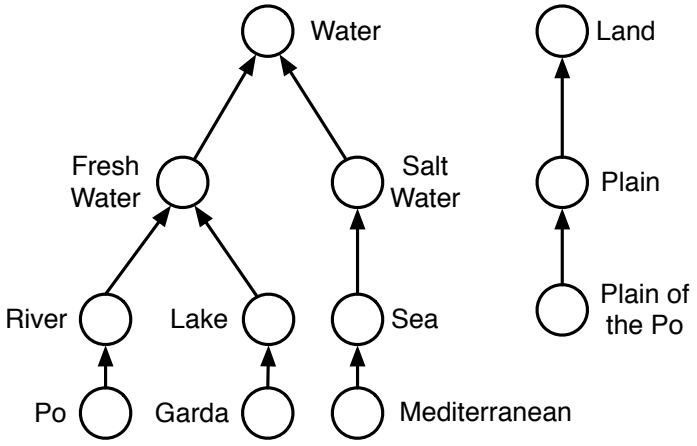
Per testare la possibile presenza di cicli senza rischiare di generarli è sempre possibile usare la clausola `LIMIT` nella query esterna, che forza l'interruzione del processo ricorsivo.

Esempio su relazioni gerarchiche (1)

```
CREATE TABLE areas ( area VARCHAR(20) PRIMARY KEY, super
VARCHAR(20) REFERENCES areas(area) )
```

AREAS

area	super
Water	NULL
Land	NULL
Plain of the Po	Plain
Plain	Land
Fresh Water	Water
River	Fresh Water
Po	River
Garda	Lake
Lake	Fresh Water
Mediterranean	Sea
Salt Water	Water
Sea	Salt Water



Esempio su relazioni gerarchiche (2)

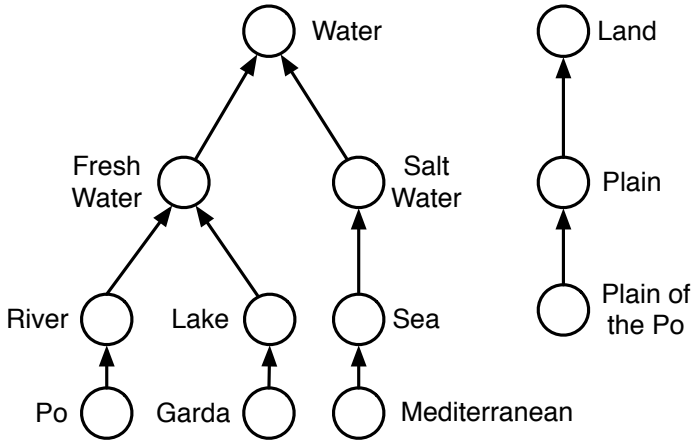
Per ogni area, trovare il numero di sotto-aree (considerando la semantica della relazione di inclusione).

```
WITH RECURSIVE sub_areas(super_area, sub_area) AS (
  SELECT super AS super_area, area AS sub_area
  FROM areas WHERE super IS NOT NULL
 UNION ALL
  SELECT S.super_area AS super_area, A.area AS sub_area
  FROM sub_areas AS S, areas AS A
  WHERE A.super = S.sub_area
)
SELECT super_area, COUNT(DISTINCT sub_area) AS sub_areas
FROM sub_areas GROUP BY super_area ORDER BY sub_areas DESC;
```

Esempio su relazioni gerarchiche (3)

Per ogni area, trovare il numero di sotto-aree (considerando la semantica della relazione di inclusione).

super_area	sub_areas
Water	8
Fresh Water	4
Land	2
Salt Water	2
Plain	1
River	1
Sea	1
Lake	1



2 Viste

2.1 Uso di viste in interrogazioni SQL

Viste

- Una **vista** in SQL è una tabella derivata a partire da altre tabelle (tabelle di base o altre viste).
- Una vista è una *tabella virtuale*: non ci sono tuple istanza della vista memorizzate nella BD (a differenza delle tabelle definite mediante `CREATE TABLE` che hanno tuple memorizzate nella BD).
- Anche se virtuale, una vista può essere utilizzata nella formulazione di interrogazioni come se fosse memorizzata.
- Possono esistere anche viste materializzate, ovvero tabelle derivate effettivamente memorizzate nella BD (va effettuata la gestione dell'allineamento).

Generazione di viste

```
CREATE VIEW NomeVista [(ListaAttributi)] AS SelectSQL
[WITH [LOCAL | CASCADED] CHECK OPTION]
```

- In generale le viste sono utili per:
- specificare una nuova tabella che frequentemente viene utilizzata nelle interrogazioni, anche se non esiste nella BD;

- specificare una tabella contenente un sottoinsieme dei dati di una o più tabelle nella BD, al fine di restringere l'accesso al solo sottoinsieme contenuto nella vista (sicurezza).

Viste in SQL

Supponendo di avere il seguente schema:

COUNTRY(iso3, government, funding_year, ...)

NAME(code, script, gazeteer, country)

dovendo lavorare spesso su alcuni dati relativi ai nomi delle nazioni, può essere utile definire la seguente vista:

```
CREATE VIEW country_names (country, government, year, language, name) AS
SELECT C.iso3 AS country, C.government, C.founding_year AS year,
  N.script AS language, N.gazeteer AS name
FROM country AS C LEFT JOIN name N
ON N.country = C.iso3
WHERE N.gazeteer IS NOT NULL;
```

Altre interrogazioni possono essere direttamente eseguite su `country_names` senza specificare ulteriormente il join. (es., `SELECT * FROM country_names WHERE languge = 'en-us'`)

Esempio: uso con operatori aggregati

Determinare la nazione di cui sono noti più nomi della media:

```
CREATE VIEW count_names(country, number_of_names) AS
SELECT country, COUNT(gazeteer)
FROM name
GROUP BY country

SELECT country
FROM count_names
WHERE number_of_names > (
        SELECT AVG(number_of_names) FROM count_names
)
```

Aggiornamento di viste

- Le operazioni di aggiornamento sulle viste sono tradotte in opportuni comandi di modifica sulle tabelle di base da cui la vista è derivata.

- Non sempre è possibile determinare in modo univoco come riportare la modifica sulla vista in termini di modifiche sulle tabelle di base; i problemi sorgono con viste ottenute tramite join di più tabelle.
- I sistemi considerano aggiornabili viste definite su una sola tabella (in alcuni casi si richiede che la vista contenga la chiave primaria).

Aggiornamento di viste

- `CHECK OPTION`: si applica a viste aggiornabili.
- Sono ammessi aggiornamenti sulle righe della vista e dopo l'aggiornamento le righe devono continuare ad appartenere alla vista.
- `LOCAL` vs. `CASCADED`: profondità con cui applicare i controlli a seguito di aggiornamento.

3 Trigger e basi di dati attive

3.1 Asserzioni

Uso di asserzioni

- In SQL un'`ASSERTION` consente di specificare un ulteriore vincolo sulla base di dati che non è altrimenti rappresentabile dallo schema.
- Sintassi:

```
CREATE ASSERTION <Constraint name>
CHECK (search condition)
[ <constraint attributes> ]
```

Constraint attributes: DEFERRABLE

- Un'asserzione, come gli altri vincoli, può essere `DEFERRABLE` o `NOT DEFERRABLE`.
- `NOT DEFERRABLE` significa che il DBMS controlla il vincolo immediatamente dopo l'esecuzione di ogni SQL statement in una transazione.
- `DEFERRABLE` significa che il DBMS può posticipare il controllo del vincolo fino al termine della transazione.

Gestione del tempo di controllo

- Un'asserzione, e in generale un vincolo, può essere `INITIALLY DEFERRED` o `INITIALLY IMMEDIATE`.
- `INITIALLY DEFERRED` significa che il vincolo è necessariamente `DEFERRABLE` e controllato all'inizio di ogni transazione.
- `INITIALLY IMMEDIATE` significa che il vincolo può essere sia `DEFERRABLE` sia `NOT DEFERRABLE` e il tempo di controllo è immediato all'inizio di ogni transazione.

Esempio
Vogliamo vincolare le nazioni a possedere almeno un nome:

```
CREATE ASSERTION check_names   CHECK (
  NOT EXISTS (
   SELECT * FROM country C
   WHERE NOT EXISTS (
    SELECT * FROM name N
    WHERE C.iso3 = N.country )));
```

Molti DBMS non implementano le asserzioni (es. PostgreSQL), poiché gli stessi risultati sono ottenibili mediante trigger.

3.2 Trigger

Trigger e basi di dati attive

- **Trigger:** regole attive, parte dello standard SQL-99.
- BD con componente per la gestione di regole **Evento-Condizione-Azione** (regole di produzione):
  - **Evento:** normalmente modifiche della base di dati
  - **Condizione:** determina se la regola debba essere eseguita
  - **Azione:** e.g., sequenza istruzioni SQL, programma esterno

  Le BD attive hanno comportamento **reattivo** (in contrasto con passivo); eseguono non solo le transazioni utente ma anche le regole.
- I trigger "mettono a fattor comune" parte dell'applicazione che è così "condivisa".

Trigger

Definiti con istruzioni DDL (`CREATE TRIGGER`).

- Basati sul paradigma evento-condizione-azione (ECA):
  - evento: modifica dei dati, specificata con `insert`, `delete`, `update`;
  - condizione (opzionale) predicato SQL;
  - azione: sequenza di istruzioni SQL (o estensioni, ad esempio PL/SQL in Oracle).
- Intuitivamente:
  - quando si verifica l'evento (**attivazione**);
  - se la condizione è soddisfatta (**considerazione**);
  - allora esegui l'azione (**esecuzione**).
- Ogni trigger fa riferimento ad una tabella (target): risponde ad eventi relativi a tale tabella.

Trigger: granularità e modalità

- Granularità:
  - di ennupla (**row-level**): il trigger viene attivato per ogni ennupla coinvolta nell'operazione;
  - di operazione (**statement-level**): una sola attivazione per ogni istruzione SQL, con riferimento a tutte le ennuple coinvolte ("set-oriented").

Trigger: granularità e modalità

- Modalità di valutazione della regola:
  - **immediata:** il valore della condizione è valutato come parte della stessa transazione che ha scatenato l'evento (subito prima (**before**), o dopo (**after**) l'evento);
  - **differita:** il valore della condizione è valutato alla fine della transazione che comprende l'evento (a seguito del comando di commit);
  - **separata (detached):** il valore della condizione è valutato come transazione a parte.

Trigger: granularità e modalità

- Relazione tra valutazione della condizione e esecuzione della regola:
  - **Immediata (immediate)**
  - **Posticipata (deferred)**
  - **Separata (detached)**
- La maggior parte dei sistemi usa l'opzione immediata, ovvero non appena il valore della condizione, se quest'ultima restituisce true, immediatamente viene eseguita l'azione.

Esempio
Supponiamo di aggiungere alla tabella delle nazioni il seguente attributo:
`ALTER TABLE country ADD COLUMN number_official_web_sites INTEGER`

Scrivere un trigger che aggiorni il nuovo campo `number_official_web_sites` (che mantiene il conto dei siti ufficiali registrati per una nazione) con il nuovo valore in caso di modifica della tabella `web(url, country, official)` che assegni un sito a una diversa nazione.

- **Evento:** aggiornamento di record nella tabella `web` relativamente al campo `country`.
- **Condizione:** i record modificati devono essere riferiti siti ufficiali.
- **Azione:** incremento del numero di siti della nuova country e decremento del numero di siti della precedente nazione.

Esempio

```
CREATE TRIGGER update_official_websites   AFTER UPDATE OF country ON
web
  FOR EACH ROW
  WHEN (NEW.official = TRUE)
  BEGIN
    UPDATE country SET
    number_official_web_sites = number_official_web_sites + 1
    WHERE iso3 = NEW.country;
    UPDATE country SET
    number_official_web_sites = number_official_web_sites - 1
    WHERE iso3 = OLD.country;
  END;
```

- **NEW:** per fare riferimento a una tupla appena inserita/aggiornata.
- **OLD:** si usa per fare riferimento a una tupla eliminata o a una tupla prima che sia aggiornata.

Principali applicazioni

- Specificare vincoli di integrità complessi (es., regole gestionali).
- Gestione dei dati derivati.
- Mantenimento della consistenza delle viste materializzate rispetto alle modifiche nelle relazioni di base.