SQL 2

Master GIS 2019

Claudio Rocchini
claudio@rockini.name

Istituto Geografico Militare

gennaio 2019

Espressioni e funzioni

Una parte importante dell'SQL è rappresentata dai valori:

- constanti letterali (numeri , parole);
- espressioni con operatori;
- chiamate di funzione;
- attributi di una tabella.

I valori sono restituti come risultato di una query o sono usati nei filtri.

Espressioni e funzioni

```
-- costanti letterali
SELECT 'Pippo', 3, TRUE;

-- cast (forzatura del tipo)
SELECT '42'::INTEGER, '2019-02-04'::DATE;

-- espressioni
SELECT 3*5 + 7,  TRUE OR FALSE, 3 <= 4;

-- chiamate di funzione
SELECT COS(3.1414), ROUND(2.349999, 2);

-- chiamate di funzione
SELECT TOLOWER('FIRENZE'), LENGTH('Roma');

-- chiamate di funzione
SELECT NOW(), DATE_PART('month', '2019-02-04'::DATE);
```

Interrogazione semplici

Il comando SELECT è utilizzato principalmente per:

- estrarre una parte degli attributi di una tabella;
- estrarre una parte delle righe;
- elaborare gli attributi con espressioni e funzioni.

Interrogazione semplici

```sql
SELECT   cognome, nome    -- alcune colonne
FROM     utenti;

SELECT   *                -- tutte le colonne
FROM     utenti;

         -- attributi calcolati
SELECT   cognome, nome, DATE_PART('year', data_nascita)
FROM     utenti;

         -- attributi calcolati (iniziali)
SELECT   LEFT(cognome, 1) || LEFT(nome, 1)
FROM     utenti;
```

Filtri

- sono espressioni booleane (vere o false);
- coinvolgono di solito attributi ed espressioni su di esse;
- contengono confronti (uguale, maggiore/minore, diverso);
- sono composti attraverso gli operatori AND, OR, NOT.

Filtri

```
SELECT   cognome, nome    -- filtro su attributo
FROM     utenti
WHERE    data_nascita > '1967-01-01';

SELECT   *          -- filtro su attributi
FROM     utenti
WHERE    cognome='rocchini'
AND      nome='claudio';

SELECT   *          -- filtro su attributi
FROM     docenti
WHERE    materia='GIS' OR materia='DATABASE';

SELECT   *          -- filtro su attributi (in)
FROM     docenti
WHERE    materia IN ('GIS', 'DATABASE', 'CARTO');
```

Filtri su stringhe

- La ricerca (filtraggio) sulle parole è un duro lavoro;
- per $=$ si intende esattamente uguale (compresi spazi, minuscole/maiuscole, accenti, ...);
- esistono espressioni per la ricerca avanzata (LIKE, espressioni regolari);
- esistono estensioni per ricerche più elastiche o ricerche su testi estesi (es. Full Text Search, stile Google).

Filtri su stringhe

```
        -- falso per 'Rocchini', ma ...
WHERE cognome='rocchini'
        -- vero per 'Rocchini' e 'ROCCHINI'
WHERE TOLOWER(cognome)='rocchini'
        -- % = qualsiasi stringa, inizia per roc
WHERE cognome LIKE 'roc%';
        -- finisce per ini
WHERE cognome LIKE '%ini';
        -- contiene cch
WHERE cognome LIKE '%cch%';
        -- '_' : qualsiasi singolo carattere:
        -- rocchini, rocchino, ...
WHERE cognome LIKE 'rocchin_'
        -- estensioni evolute: fuzzystrmatch, ...
```

Ordinamento

- L'ordine delle righe di un tabella non è definito;
- come l'ordine dei risultati di una query;
- attraverso ORDER BY è possibile definire un ordine nel risultato di una query.

Ordinamento

```
SELECT   cognome, nome    -- ordine per cognome
FROM     utenti
ORDER    BY cognome;

         -- prima per cognome, quindi per nome
SELECT   cognome, nome
FROM     utenti
ORDER    BY cognome, nome;

         -- per chiamata di funzione
SELECT   cognome, nome
FROM     utenti
ORDER    BY codice_fiscale(cognome, nome, data_n ...);

         -- Per data decrescente! Defaut= ASC
SELECT   cognome, nome
FROM     utenti
ORDER    BY data_nascita DESC;
```

Limitazione

L'opzione LIMIT permette di limitare il numero massimo di righe nel risultato di una query:

- per fare dei test;
- per imporre un limite di spazio (es. su una pagina web);
- combinato ad ORDER, permette di estrarre i primi (o gli ultimi) N elementi.

Limitazione

```
-- 1000 laghi a caso (per test)
SELECT  *
FROM    laghi_italia
LIMIT   1000;

-- i primi 10 laghi per estensione
SELECT  *
FROM    laghi_italia
ORDER   BY area DESC
LIMIT   10;

-- IL lago maggiormente esteso
SELECT  *
FROM    laghi_italia
ORDER   BY area DESC
LIMIT   1;
```

Funzioni Aggreganti

- Esiste una classe di funzioni speciali, che aggregano i dati di una colonna;
- Il risultato, di solito, è composto di una sola riga;
- esempi sono MIN, MAX, AVG (media), COUNT, ...
- attraverso GROUP BY, è possibile ottenere sub-totali;
- *Esistono funzioni spaziali aggreganti: es. ST_Envelope.*

Funzioni Aggreganti

```
-- Un solo risultato: la superficie media
SELECT  AVG(area)
FROM    laghi_italia;

-- Una sola riga
SELECT  MIN(area), MAX(area), SUM(area)
FROM    laghi_italia;

-- count (non sua attributi)
SELECT  COUNT(*)
FROM    laghi_italia;

-- aggregazione parziale
SELECT  regione, MIN(area), MAX(area)
FROM    laghi_italia
GROUP   BY regione;
```

Funzioni Aggreganti 2

```
        -- ERRORE: non ha senso
SELECT  regione, MIN(area), MAX(area)
FROM    laghi_italia;

        -- filtro sul risultato aggregato
SELECT  regione, MIN(area), MAX(area)
FROM    laghi_italia
GROUP   BY regione
HAVING COUNT(*) > 1000;

        -- filtro sulle righe e sul risultato aggregato
SELECT  regione, MIN(area), MAX(area)
FROM    laghi_italia
WHERE   tipo_lago='perenne'      -- sulle righe
GROUP   BY regione
HAVING  COUNT(*) > 1000;         -- aggregato
```

Join e tabelle

- Due tabelle?
- É possibile costruire interrogazioni su più tabelle;
- il risultato standard è il prodotto cartesiano (tutte le combinazioni possibili);
- attraverso i filtri è possibile ottenere risulati che hanno un senso.
- In presenza di più tabelle, i nomi di colonna possono diventare ambigui: una colonna può essere indicata attraverso la dicitura: TABELLA.COLONNA;
- attraverso il comando AS è possibile assegnare un nickname ad una tabella (per comodità).

Join e tabelle

```
-- Prodotto cartesiano: ogni coppia a caso
SELECT  cognome, nom_corso
FROM    studenti, corsi;

-- Quale studente segue quale corso
SELECT  cognome, nom_corso
FROM    studenti, corsi
WHERE   corso_seguito=id_corso;

-- Join strade e descrizione classe
SELECT  nome, descrizione
FROM    strade, classi_stradali
WHERE   classifica=codice;
```

Join: nomi e cognomi

```
        -- Nomi e cognomi
SELECT  strade.nome,
        classi_stradali.descrizione
FROM    strade,
        classi_stradali
WHERE   strade.classifica=classi_stradali.codice;

        -- Nicknames: comando AS
SELECT  s.nome,
        c.descrizione
FROM    strade AS s,
        classi_stradali AS c
WHERE   s.classifica=c.codice;
```

Autojoin: una tabelle con se stessa

```
        -- ricerca duplicati
SELECT  s1.*, s2.*
FROM    studenti AS s1, -- nickname obbligatorio!
        studenti AS s2
WHERE   s1.cognome = s2.cognome
AND     s1.nome = s2.nome
AND     s1.id < s2.id   -- minore o diverso?

        -- ricerca errori topologici
SELECT  e1.id, e2.id
FROM    edifici AS e1,
        edifici AS e2
WHERE   ST_Intersects(e1.shape, e2.shape)
AND     e1.id < e2.id;
```

Join: filtri complessi

```
        -- Join multi-attributo
SELECT  s.*, d.*
FROM    studenti AS s,
        docenti AS d
WHERE   s.cognome = d.cognome
AND     s.nome = d.nome;

        -- combinazione per intervallo di date
SELECT  s.*, e.*
FROM    studenti AS s,
        eventi AS e
WHERE   s.data_nascita BETWEEN e.inizio AND e.fine;

        -- join spaziali
SELECT  s.*, r.*
FROM    strade AS s, regioni AS r
WHERE   ST_Contains(r.shape, s.shape);
```

Join Left Outer, Full Outer

Cosa succede quando un valore non si accoppia nella join?

- INNER (standard): solo i dati accoppiati;
- LEFT OUTER: tutta la prima tabella, se c'è anche la seconda;
- LEFT OUTER: tutta la seconda tabella, se c'è anche la prima;
- FULL OUTER: tutti, accoppiati e non.

Join Left Outer, Full Outer

```
        -- La join di default si chiama INNER

        -- Anche le strade con classifica errata
SELECT  s.nome,
        c.descrizione
FROM    strade AS s,
        classi_stradali AS c
        LEFT OUTER JOIN ON(s.classifica=c.codice);

        -- Differenza fra tabelle
SELECT  s1.id, s2.id
FROM    strade1 AS s1,
        strade2 AS s2
        FULL OUTER JOIN ON(s1.id = s2.id)
WHERE   s1.id IS NULL  OR  s2.id IS NULL;
```

Subquery

- Una query SQL racchiusa fra parentesi tonde diviene una subquery.
- Una subquery può essere sostituita in una query generale:
  - ad una tabella;
  - ad un valore se ha un solo risultato (una colonna ed una riga).

Subquery tabella

```
SELECT   reg2.id,
         reg2.nome,        -- campo calcolato nella sub
         reg2.area
FROM     ( SELECT id,      -- Inizio sub
                  TOUPPER(nome) AS nome,
                  ST_Area(shape) AS area
           FROM   laghi
           WHERE   regione='Toscana'
         ) AS reg2         -- fine sub: nome obbligatorio
WHERE reg2.area > 10000; -- filtro su campo calcolato
```

Costrutto Exists

```
        -- elenco le regioni che hanno sciovie
SELECT  r.*
FROM    regioni AS r
WHERE EXISTS
( SELECT        *       -- cosa non importa
  FROM  sciovie AS s
  WHERE s.codice_regionale=r.codice -- attributo esterno
);
```

Subquery valore risultato

```
        -- Elenco regioni e lago magg. esteso
SELECT  r.nome,
        ( SELECT l.nome
          FROM laghi AS l,
          WHERE  l.cod_reg=r.codice
          ORDER BY area DESC
          LIMIT 1          -- importante: un solo valore
        ) as lago
FROM    regioni AS r;
```

Subquery valore filtro

```
        -- i laghi che superano l'estensione media
        -- dei laghi della regione di cui fanno parte
SELECT  l.*
FROM    laghi AS l
WHERE   l.area >
( SELECT avg(l2.area)
  FROM   laghi AS l2
  WHERE l2.cod_reg=l1.cod_reg
);      -- non serve il nickaname del valore
```

Manipolazione dei risultati

Il risultato di una SELECT:

- di default è stampato su schermo;
- può essere usato per
  - creare una vista;
  - creare una nuova tabella;
  - inserire nuove righe in una tabella esistente;
  - aggiornare dati esistenti.

Creazione di una vista

```
CREATE VIEW laghi_elaborati AS
SELECT ...
```

Anteponendo CREATE VIEW nome AS ad una query si salva una query importante come vista.

```
SELECT ... FROM laghi_elaborati ...
```

Una volta creata la query si usa come una tabella. Il suo contenuto è dinamico e dipende dai cambiamenti delle tabelle di origine.

Creazione di una tabella da query

```
CREATE TABLE laghi_elaborati2 AS
SELECT ...
```

Anteponendo CREATE TABLE nome AS ad una query si *salvano* i dati del risultato di una query in una nuova tabella.

```
SELECT ... FROM laghi_elaborati2 ...
```

In questo caso il risultato è *congelato* al momento della query; eventuali cambiamenti nella tabelle di origine non vengono riportati.
Perchè usare CREATE TABLE al posto di CREATE VIEW? Ad esempio nel caso di elaborazioni lente.

Creazione di nuovi dati

```
INSERT INTO laghi_elaborati2
SELECT ...
```

Antepondendo INSERT INTO tabella ad una query i dati del risultato vengono *accodati* in una tabella esistente.
Nota: numero, tipo ed ordine delle colonne della tabella esistente devono coicidere con quelli del risultato della query.

Sintassi di UPDATE

```
        -- Modifica di un attributo costante, attenzione!
UPDATE  laghi
SET     nome = 'UNK';

        -- Modifica vari attributi costanti
UPDATE  laghi
SET     nome = 'UNK',
        tipo = '999';

        -- Modifica con filtro
UPDATE  laghi
SET     nome = 'UNK'
WHERE   nome IS NULL;
```

Update Dinamici

```
        -- Con espressioni
UPDATE  laghi
SET     area = ROUND(area, 1);

        -- Con chiamate di funzione
UPDATE  laghi
SET     area = ST_Area(shape);

        -- Con chiamate di funzione e filtri
UPDATE  laghi
SET     shape = ST_MakeValid(shape)
WHERE   NOT ST_IsValid(shape);
```

Update da subquery

```
        -- inserisco il codice regionale
        -- nota la tabella di update non ha nickname
UPDATE  laghi
SET     cod_reg =
( SELECT r.codice
  FROM  regione AS r
  WHERE ST_Contains(r.shape, laghi.shape)
);
```

Estensioni Utili

```
        -- accodamento di due query
        -- nota: gli attributi devono coincidere
SELECT * FROM laghi_naturali
UNION ALL
SELECT * FROM bacini_artificiali;

        -- costrutto CASE WHEN
SELECT id,
        CASE
                WHEN area>70000000 THEN 1
                WHEN area>45000000 THEN 2
                WHEN area> 4000000 THEN 3
                ELSE 5
        END AS classifica
FROM dbsn.sp_acq;
```