SQL - Elementi più avanzati

Informatica@SEFA 2018/2019 - Lezione 24

Massimo Lauria <massimo.lauria@uniroma1.it>
http://massimolauria.net/courses/infosefa2018/

Venerdì, 14 Dicembre 2018

Funzioni di gruppo

Oltre le query classiche

Consideriamo il nostro DB del registro automobilistico e supponiamo di volere rispondere a domande come

- quanti veicoli sono registrati?
- quanti veicoli sono stati acquistati in media da ogni proprietario?
- qual è la massima velocità tra quella dei veicoli con cilindrata maggiore di 1800?

Nell’algebra relazionale il calcolo delle tabelle e la valutazione delle condizioni della query sono eseguite tupla per tupla.

Ora stiamo chiedendo di **effettuare calcoli** sul risultato di una query, calcoli che **aggregano** i dati di più righe.

Opzione 1

Usare un **linguaggio di programmazione** generico.

È possibile connettersi DB con un linguaggio di programmazione (e.g. Python)

- Eseguire query SQL
- Analizzare il risultato

L'approccio più flessibile ma complesso.

Opzione 2

Usare le funzioni di gruppo (o operatori aggregati) per calcolare funzioni specifiche sui dati della tabella.

- `MAX` calcola il valore massimo
- `MIN` calcola il valore minimo
- `SUM` somma dei valori
- `AVG` calcolo della media
- `count` conteggio dei valori

Altre funzioni sono aggiunte dall'implementazione.

Un semplice esempio

Contiamo il numero di veicoli

```
select count(*) from Veicoli;                                    1
```

count(*)
15

Calcoliamo la massima cilindrata

```
select max(Cilindrata) from Veicoli;                             1
```

max(Cilindrata)
1998

All'interno di query più sofisticate

Vogliamo calcolare la velocità media dei veicoli Fiat, e averne il numero.

```
select avg(Velocità) as 'Velocità media',count(*) as 'N.          1
    Macchine',
       Nome_Fabbrica                                              2
from Veicoli natural join Modelli natural join Fabbriche          3
where Fabbriche.Nome_Fabbrica='Fiat';                             4
```

Velocità media	N. Macchine	Nome_Fabbrica
150.5	6	Fiat

Il meccanismo degli operatori aggregati

Come funzionano questi operatori? Consideriamo la query

```
select Exp1,Exp2,... from <Tabelle> where <Condizione>
```

dove alcune delle espressioni colonne usano degli operatori

1. Prima viene effettuata la query

   ```
   select * from <Tabelle> where <Condizione>
   ```

2. il valore di `Exp1`, `Exp2`, … viene calcolato dalle colonne della query. E.g. `sum(Velocità)` è la somma dei valori nella colonna `Velocità`

Esempio

```
select * from Veicoli where Posti=5;                                    1
```

Targa	Cilindrata	Cavalli_Fiscali	Velocità	Posti	Immatricolazione	Cod_C
A123456X	1796	85	195	5	1998-12-30	
B256787Y	708	10	120	5	1989-09-21	
C76589AG	1106	54	130	5	1998-08-13	
CFT340VB	1390	75	170	5	1995-01-12	
DD4567XX	1581	17	NULL	5	1997-06-05	
DGH789JC	1590	114	170	5	1995-10-05	
DH79567H	1589	107	170	5	NULL	
ERG567NM	1598	107	175	5	1997-12-18	
F96154NH	1781	125	185	5	1992-03-08	
FGH673FV	899	39	140	5	1998-08-09	
XCH56GJK	1918	110	210	5	1998-09-04	
XF5789CY	1587	107	175	5	1996-05-05	

```
select max(Velocità)-min(Velocità),count(Velocità),                     1
       sum(Velocità),count(*) from Veicoli where Posti=5;               2
```

max(Velocità)-min(Velocità)	count(Velocità)	sum(Velocità)	count(*)
90	11	1840	12

Esempio (II)

La query

```
select * from Veicoli where Posti=5;
```

restituisce 12 righe di veicoli. Sui quali abbiamo calcolato quattro espressioni contenenti funzioni di gruppo.

- `count(*)` conta il numero di righe della query
- `count(Velocità)` conta i valori non nulli in `Velocità`
- `sum(Velocità)` somma i valori in `Velocità`
- `avg(Velocità)` media dei valori in `Velocità`

Espressioni colonna

Le espressioni nella lista colonne possono avere anche diversi funzioni di gruppo, che possono essere composte in espressioni algebriche, come abbiamo già visto per le espressioni colonna.

Esempio

```
select avg(Cilindrata)*count(Cilindrata), sum(Cilindrata)  1
from Veicoli;                                              2
```

avg(Cilindrata)*count(Cilindrata)	sum(Cilindrata)
19541.0	19541

Sintassi

Con l'eccezione di `count(*)` la sintassi delle funzioni di gruppo è una delle seguenti

- `func(NomeColonna)`
- `func(all NomeColonna)`
- `func(distinct NomeColonna)`

dove `NomeColonna` è una colonna del risultato della query, e `func` è una tra `avg`, `sum`, `max`, `min`, `count`.

I valori passati alla funzione di gruppo

Targa	Velocità
A123456X	195
B256787Y	120
C76589AG	130
C78905GT	212
C845905Z	NULL
CFT340VB	170
D239765W	NULL
DD4567XX	NULL
DGH789JC	170
DH79567H	170
ERG567NM	175
F96154NH	185
FGH673FV	140
XCH56GJK	210
XF5789CY	175

non nulli, con ripetizioni

count(Velocità)
12

non nulli, con ripetizioni

count(all Velocità)
12

non nulli, senza ripetizioni

count(distinct Velocità)
9

Colonne a valori tutti nulli

Table: Test

Valore
NULL
NULL
NULL

```
select count(*),count(Valore),max(Valore),                    1
       min(Valore),avg(Valore)                                2
from Test;                                                    3
```

count(*)	count(Valore)	max(Valore)	min(Valore)	avg(Valore)
3	0	NULL	NULL	NULL

Un altro esempio

E.g. Vediamo quante versioni di modelli sono stati prodotti da ogni fabbrica. Prima elenchiamo le fabbriche per avere un riferimento.

```
select * from Fabbriche;                                    1
```

```
Cod_Fabbrica  Nome_Fabbrica
------------  -------------
001           Fiat
003           Ford
004           Piaggio
005           Volvo
006           Renault
007           Toyota
008           Volkswagen
009           Honda
```

Numero di versioni nella fabbrica ’001’

- ogni modello esiste in diverse versioni
- ogni modello ha uno specifico `Cod_Fabbrica`

```sql
select sum(Numero_Versioni) as '# Vers. Fabbrica 001'
from Modelli
where Cod_Fabbrica = '001';
```

Vers. Fabbrica 001
13

Vediamo le fabbriche ’002’ e ’003’

```
select sum(Numero_Versioni) as '# Vers. Fabbrica 002'   1
from Modelli                                            2
where Cod_Fabbrica = '002';                             3
```

Vers. Fabbrica 002
NULL

```
select sum(Numero_Versioni) as '# Vers. Fabbrica 003'   1
from Modelli                                            2
where Cod_Fabbrica = '003';                             3
```

Vers. Fabbrica 003
3

Aggregare i dati per gruppi

Naturalmente non vogliamo fare queste query una alla volta per ogni fabbrica.

- In SQL è possibile segmentare la tabella in gruppi
- così gli operatori aggregati vengono applicati ai singoli gruppi.

Uso della clausola `GROUP BY`

```
select Cod_Fabbrica,                    1
    Numero_Versioni from
    Modelli
order by Cod_Fabbrica;                  2
```

Cod_Fabbrica	Numero_Versioni
1	3
1	2
1	5
1	1
1	2
3	3
4	4
5	2
6	5
6	2
6	2
7	4
8	4
9	3

```
select Cod_Fabbrica,                    1
       sum(Numero_Versioni)             2
    as 'Numero versioni'                3
from Modelli                            4
group by Cod_Fabbrica;                  5
```

Cod_Fabbrica	Numero versioni
1	13
3	3
4	4
5	2
6	9
7	4
8	4
9	3

Un esempio più articolato

Miglioriamo la query e contiamo le righe di ogni gruppo

```
select Nome_Fabbrica as Fabbrica,                                    1
       sum(Numero_Versioni) as Versioni,                             2
       count(*) as '# modelli'                                       3
from Modelli join Fabbriche                                          4
on Modelli.Cod_Fabbrica = Fabbriche.Cod_Fabbrica                     5
group by Nome_Fabbrica                                               6
order by Versioni desc;                                              7
```

Fabbrica	Versioni	# Modelli
Fiat	13	5
Renault	9	3
Piaggio	4	1
Toyota	4	1
Volkswagen	4	1
Ford	3	1
Honda	3	1
Volvo	2	1

Filtrare le righe con gli operatori aggregati

Fabbrica	Versioni	# modelli
Fiat	13	5
Renault	9	3
Piaggio	4	1
Toyota	4	1
Volkswagen	4	1
Ford	3	1
Honda	3	1
Volvo	2	1

E.g. Da questa query vogliamo solo le righe con `Versioni` dispari.

Non possiamo usare la clausola `WHERE` perché questa si applica **prima** degli operatori aggregati.

Clausola `HAVING`

La clausola `WHERE`

- filtra le righe di una query
- viene applicata PRIMA degli operatori aggregati
- che non si possono usare nelle espressioni

Per filtrare i **gruppi** si usa la clausola `HAVING`

- viene applicata DOPO gli operatori aggregati
- che possono essere usati nelle espressioni.

Clausola `HAVING` su un esempio semplice (I)

Semplifichiamo l'esempio di prima, usando il codice di fabbrica invece del nome, così evitiamo il `JOIN`

```
select Cod_Fabbrica,                                    1
       sum(Numero_Versioni) as Versioni,                2
       count(*) as '# modelli'                          3
  from Modelli                                          4
group by Cod_Fabbrica;                                  5
```

Cod_Fabbrica	Versioni	# modelli
1	13	5
3	3	1
4	4	1
5	2	1
6	9	3
7	4	1
8	4	1
9	3	1

Clausola `HAVING` su un esempio semplice (II)

Ora solo le fabbriche con numero di versioni dispari.

```
select Cod_Fabbrica,                                    1
       sum(Numero_Versioni) as Versioni,                2
       count(*) as '# modelli'                          3
  from Modelli                                          4
group by Cod_Fabbrica                                   5
having Versioni % 2 = 1;                                6
```

Cod_Fabbrica	Versioni	# modelli
1	13	5
3	3	1
6	9	3
9	3	1

La clausola `HAVING` sull'esempio di prima

```
select Nome_Fabbrica as Fabbrica,                                    1
       sum(Numero_Versioni) as Versioni,                             2
       count(*) as '# modelli'                                       3
from Modelli join Fabbriche                                          4
on Modelli.Cod_Fabbrica = Fabbriche.Cod_Fabbrica                     5
group by Nome_Fabbrica                                               6
having  (Versioni % 2) = 1                                           7
order by Versioni desc;                                              8
```

Fabbrica	Versioni	Numero modelli
Fiat	13	5
Renault	9	3
Ford	3	1
Honda	3	1

Abbiamo re-inserito anche ordinamento e `JOIN`.

Manipolazione dei dati

Inserimento dati

Inserimenti espliciti

```
insert into Veicoli values                                              1
  ('C845905Z',NULL,NULL,NULL,3,'1995-04-11','04','03','006'),          2
  ('D239765W',NULL,NULL,NULL,2,'1997-08-12','03','01','002');           3
```

anche omettendo colonne

```
insert into Veicoli(                                                    1
       Targa,Posti,Immatricolazione                                     2
       Cod_Categoria, Cod_Combustibile, Cod_Modello) values             3
  ('C845905Z',3,'1995-04-11','04','03','006'),                         4
  ('D239765W',2,'1997-08-12','03','01','002');                          5
```

Inserimento dati (II)

Inserimenti attraverso una query

```
create table VeicoliLenti ( [...omissis...]);                    1
insert into VeicoliLenti                                         2
       select * from Veicoli where Velocità < 130;               3
```

Eliminazione dati

Si possono eliminare tutt e righe di una tabella che verificano la condizione di una clausola `WHERE`.

```
delete from Veicoli                                          1
  where Cod_Modello in ('001','002');                        2
```

Aggiornamento dati

Si possono aggiornare le righe di una tabella, se soddisfano la condizione di una clausola `where`

```
update Fabbriche                                    1
   set Nome_Fabbrica = 'FCA'                        2
where Nome_Fabbrica = 'Fiat'                        3
```

Notate che la condizione viene controllata **prima** dell'aggiornamento dati.

Fine della panoramica su SQL di base

Tutto quello che è stato visto finora è sul manuale SQL del corso.

`http://massimolauria.net/courses/infosefa2017/docs/manuale_sql.pdf`

Altri elementi SQL

che non fanno parte del programma di esame

Query annidate (I)

Le espressioni SQL usate nelle clausole `WHERE`, `HAVING`,... possono contenere sotto-espressioni della forma

```
<Expr> in ('V1','V2','V3',...)
```

```
select Nome_Modello from Modelli join Fabbriche                    1
    on Modelli.Cod_Fabbrica = Fabbriche.Cod_Fabbrica               2
where  Nome_Fabbrica in ('Fiat','Renault');                        3
```

Nome_Modello
Panda
Brava
Ducato
Clio
Coupè
Megane
Seicento
Laguna

Query annidate (II)

É possibile calcolare le sequenze su cui usare `IN` attraverso una sotto-query, invece di scriverle esplicitamente.

E.g. “Le fabbriche che producono veicoli con velocità almeno 180 km/h.”

```
select distinct Nome_Fabbrica from Fabbriche join Modelli      1
    on Modelli.Cod_Fabbrica = Fabbriche.Cod_Fabbrica           2
where  Cod_Modello in                                          3
       (select Cod_Modello from Veicoli                        4
         where Velocità >= 180);                               5
```

Sotto-query correlate (I)

Se la sotto-query dipende da attributi della query esterna, deve essere **rieseguita** per ogni riga.

```
select Targa,Nome_Modello                                          1
       from Veicoli as V natural join Modelli                      2
where  Velocità = ( select max(Velocità) from Veicoli              3
                    where Cod_Modello=V.Cod_Modello);              4
```

Targa	Nome_Modello
A123456X	Mondeo
C76589AG	Panda
C78905GT	Coupè
CFT340VB	Clio
DGH789JC	Civic
DH79567H	Megane
ERG567NM	Laguna
F96154NH	Golf
FGH673FV	Seicento
XCH56GJK	V-10
XF5789CY	Corolla

Sotto-query correlate (II)

E.g. ”Tutti i dischi le cui tracce occupano meno di 2MB”

```
select AlbumId, Title from Albums                                    1
where  2*1024*1024 >  (select sum(Bytes)                             2
                       from Tracks                                   3
                       where Tracks.AlbumId = Albums.AlbumId)        4
order by title;                                                      5
```

AlbumId	Title
277	Bach: Goldberg Variations
328	Charpentier: Divertissements, Airs & Concerts
345	Monteverdi: L’Orfeo

Altri usi delle query annidate

Le sotto-query possono apparire:

- dentro altre sotto-query
- nella clausola `from`
- nella clausola `select`
- ...

Vincoli di integrità

É possibile inserire dei vincoli di integrità della base di dati, che vengono verificati ad ogni cambiamento dei dati.

```
create table Esame(                                                    1
    Matricola         nchar(20),                                       2
    Cod_Insegnamento  nchar(10),                                       3
    Voto      integer check (Voto >= 18 and Voto <= 30),               4
    Lode      boolean,                                                 5
    check( Voto=30 or not Lode)                                        6
);                                                                     7
```

- verifica che il voto sia tra 18 e 30
- non permette la lode se il voto non è 30

Viste

Una **vista** è il risultato di una `select` su altre tabelle, che viene mantenuta come una tabella virtuale

- Viene mantenuta aggiornata
- Si può usare come una tabella qualunque
- Se modificata, le modifiche vengono propagate alle tabelle di origine.

```
create view InfoVeicoli(Veicolo,Modello,Fabbrica) as          1
select Targa, Nome_Modello, Nome_Fabbrica                     2
from Veicoli natural join Modelli natural join Fabbriche;     3
```

Veicolo	Modello	Fabbrica
A123456X	Mondeo	Ford
B256787Y	Panda	Fiat
C76589AG	Panda	Fiat
C78905GT	Coupè	Fiat
C845905Z	Ducato	Fiat
CFT340VB	Clio	Renault
D239765W	Vespa	Piaggio
DD4567XX	Brava	Fiat
DGH789JC	Civic	Honda
DH79567H	Megane	Renault
ERG567NM	Laguna	Renault
F96154NH	Golf	Volkswagen
FGH673FV	Seicento	Fiat
XCH56GJK	V-10	Volvo
XF5789CY	Corolla	Toyota

Trigger

Un trigger (trad. innesco, detonatore) è sequenza di operazioni che viene innescata se si verifica una condizione precisa nella base di dati. Ha due componenti:

- la condizione di innesco
- le istruzioni da eseguire

Può essere usata per mantenere dei vincoli di integrità dei dati troppo complessi per essere formalizzati con `check`.

Controllo dell'accesso

SQL ha delle istruzioni per fornire/revocare privilegi agli utenti della base di dati.

- e.g. l'utente Valentina può effettuare la `select` sulla tabella `Dipartimenti`.
- e.g. all'utente Marco è stato impedito di modificare la tabella `Stipendi`.

SQLite non è un sistema multi-utente, ma molti server SQL lo sono.

Transazioni

Una **transazione** è una serie di operazioni che deve essere eseguita in maniera **atomica**, perché altrimenti lascerebbe il DB in uno stato incoerente.

Modifica la base di dati e termina con

- **commit** le modifiche vengono finalizzate

oppure, in caso di errori o interruzioni,

- **rollback** le modifiche vengono azzerate

Letture

Sezioni 4.5, 4.6, 4.7 e Capitolo 7 del Manuale SQL.