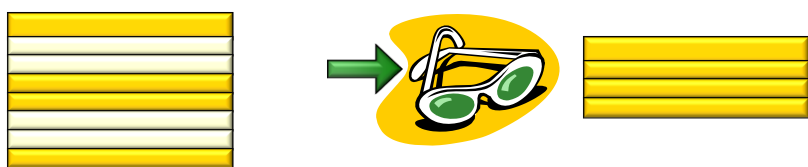




## Il linguaggio SQL: viste e tabelle derivate




**Dario Maio**  
<http://bias.csr.unibo.it/maio/>  
**Annalisa Franco**



Il linguaggio SQL: viste e tabelle derivate



1



## DB di riferimento per gli esempi

**Imp**


CodImp	Nome	Sede	Ruolo	Stipendio
E001	Rossi	S01	Analista	2000
E002	Verdi	S02	Sistemista	1500
E003	Bianchi	S01	Programmatore	1000
E004	Gialli	S03	Programmatore	1000
E005	Neri	S02	Analista	2500
E006	Grigi	S01	Sistemista	1100
E007	Violetti	S01	Programmatore	1000
E008	Aranci	S02	Programmatore	1200

**Sedi**


Sede	Responsabile	Citta
S01	Biondi	Milano
S02	Mori	Bologna
S03	Fulvi	Milano

**Prog**

CodProg	Citta
P01	Milano
P01	Bologna
P02	Bologna



Il linguaggio SQL: viste e tabelle derivate



2



## Definizione di viste

- Mediante l'istruzione **CREATE VIEW** si definisce una **vista**, ovvero una "tabella virtuale".
- Le **tuple della vista** sono il risultato di una query che viene **valutata dinamicamente** ogni volta che si fa riferimento alla vista.

```
CREATE VIEW ProgSedi (CodProg, CodSede)
AS      SELECT P.CodProg, S.Sede
        FROM    Prog P, Sedi S
        WHERE   P.Citta = S.Citta
```

ProgSedi

CodProg	CodSede
P01	S01
P01	S03
P01	S02
P02	S02

```
SELECT *
FROM    ProgSedi
WHERE   CodProg = 'P01'
```

CodProg	CodSede
P01	S01
P01	S03
P01	S02

Il linguaggio SQL: viste e tabelle derivate



3



## Uso delle viste

- Le viste possono essere create a vari scopi, tra i quali si ricordano i seguenti:
  - permettere agli utenti di avere una **visione personalizzata del DB** che in parte astragga dalla struttura logica del DB stesso;
  - far fronte a **modifiche dello schema logico** che comporterebbero una ricompilazione dei programmi applicativi;
  - **semplificare la scrittura di query complesse**.
- Inoltre le viste possono essere usate come **meccanismo per il controllo degli accessi**, fornendo a ogni classe di utenti gli opportuni privilegi.
- Si noti che nella definizione di una vista si possono referenziare anche altre viste.

Il linguaggio SQL: viste e tabelle derivate



4



## Indipendenza logica tramite VIEW

- A titolo esemplificativo si consideri un DB che contiene la tabella **EsamiBD** (Matr, Cognome, Nome, DataProva, Voto)
- Per evitare di ripetere i dati anagrafici, si decide di modificare lo schema del DB sostituendo alla tabella **EsamiBD** le due seguenti:

**StudentiBD** (Matr, Cognome, Nome)

**ProveBD** (Matr, DataProva, Voto)

- È possibile ripristinare la "visione originale" in questo modo:

```
CREATE VIEW EsamiBD (Matr, Cognome, Nome, DataProva, Voto)
AS
  SELECT S.*, P.DataProva, P.Voto
  FROM   StudentiBD S, ProveBD P
  WHERE  S.Matr = P.Matr
```

Il linguaggio SQL: viste e tabelle derivate



5



## Query complesse che usano VIEW

- Un "classico" esempio di uso delle viste si ha nella scrittura di query di raggruppamento in cui si vogliono confrontare i risultati della funzione aggregata:

*La sede che ha il massimo numero di impiegati*

- La soluzione senza viste è:

```
SELECT  I.Sede
FROM    Imp I
GROUP BY I.Sede
HAVING  COUNT(*) >= ALL (SELECT COUNT(*)
                        FROM    Imp I1
                        GROUP BY I1.Sede)
```

Il linguaggio SQL: viste e tabelle derivate



6



## Query complesse che usano VIEW

- La soluzione con viste è:

```
CREATE VIEW NumImp (Sede, NImp)
AS SELECT Sede, COUNT (*)
FROM Imp
GROUP BY Sede
```

NumImp

Sede	NImp
S01	4
S02	3
S03	1

```
SELECT Sede
FROM NumImp
WHERE NImp = (SELECT MAX (NImp)
              FROM NumImp)
```

che permette di trovare "il MAX dei COUNT(\*)", cosa che, si ricorda, non si può fare direttamente scrivendo MAX(COUNT(\*)).

Il linguaggio SQL: viste e tabelle derivate



7



## Aggiornamento di viste

- Le viste possono essere utilizzate per le interrogazioni come se fossero tabelle del DB, ma per le operazioni di aggiornamento ci sono dei limiti.

```
CREATE VIEW NumImp (Sede, NImp)
AS SELECT Sede, COUNT (*)
FROM Imp
GROUP BY Sede
```

NumImp

Sede	NImp
S01	4
S02	3
S03	1

```
UPDATE NumImp
SET NImp = NImp + 1
WHERE Sede = 'S03'
```

- Che cosa significa? Non si può fare!

Il linguaggio SQL: viste e tabelle derivate



8



## Aggiornabilità di viste (1)

- Una vista è di fatto una funzione che calcola un risultato  $y$  a partire da un'istanza di database  $r$ ,  $y = V(r)$ .
- L'aggiornamento di una vista, che trasforma  $y$  in  $y'$ , può essere eseguito **se e solo se** è univocamente definita la nuova istanza  $r'$  tale che  $y' = V(r')$ , e ciò corrisponde a dire che **la vista è "invertibile", ossia  $r' = V^{-1}(y')$** .
- Data la complessità del problema, di fatto **ogni DBMS pone dei limiti sulle tipologie di viste che possono essere aggiornate**.
- Le **più comuni restrizioni** riguardano la non aggiornabilità di viste in cui **il blocco più esterno** della query di definizione contiene:
  - GROUP BY;
  - funzioni aggregate;
  - DISTINCT;
  - join (espliciti o impliciti).



## Aggiornabilità di viste (2)

- La precisazione che è **il blocco più esterno** della query di definizione che non deve contenere, ad es., dei join ha importanti conseguenze.
- Ad esempio, la vista:

```
CREATE VIEW ImpBO (CodImp, Nome, Sede, Ruolo, Stipendio)
AS
  SELECT I.*
  FROM   Imp I JOIN Sedi S ON (I.Sede = S.Sede)
  WHERE  S.Citta = 'Bologna'
```

non è aggiornabile mentre lo è la seguente, di fatto equivalente alla prima:

```
CREATE VIEW ImpBO (CodImp, Nome, Sede, Ruolo, Stipendio)
AS SELECT I.*
  FROM   Imp I
  WHERE  I.Sede IN (SELECT S.Sede FROM Sedi S
                   WHERE S.Citta = 'Bologna')
```



## Viste con CHECK OPTION

- Per le viste aggiornabili si presenta un nuovo problema. Si consideri il seguente inserimento nella vista **ImpBO**:

```
INSERT INTO ImpBO (CodImp, Nome, Sede, Ruolo, Stipendio)
VALUES ('E009', 'Azzurri', 'S03', 'Analista', 1800)
```

in cui il valore di Sede ('S03') non rispetta la specifica della vista. Ciò comporta che una successiva query su **ImpBO** non restituirebbe la tupla appena inserita (!?).

- Per evitare situazioni di questo tipo, all'atto della creazione di una vista si può specificare, facendola seguire alla query che definisce la vista, la clausola **WITH CHECK OPTION**, che garantisce che ogni tupla inserita nella vista sia anche restituita dalla vista stessa.



## Tipi di CHECK OPTION

- Se la vista **V1** è definita in termini di un'altra vista **V2**, e si specifica la clausola **WITH CHECK OPTION**, il DBMS verifica che la nuova tupla **t** inserita soddisfi sia la definizione di **V1** sia quella di **V2**, indipendentemente dal fatto che **V2** sia stata a sua volta definita **WITH CHECK OPTION**.
- Questo comportamento di default, che è equivalente a definire **V1**  
**WITH CASCADED CHECK OPTION**  
si può alterare definendo **V1**  
**WITH LOCAL CHECK OPTION**
- In modalità **LOCAL**, il DBMS verifica solo che **t** soddisfi la specifica di **V1** e quelle di tutte e sole le viste da cui **V1** dipende per cui è stata specificata la clausola **WITH CHECK OPTION**.



## Table expressions

- Tra le caratteristiche più interessanti di SQL vi è la possibilità di usare all'interno della clausola FROM una **subquery** che definisce "dinamicamente" una tabella derivata, e che qui viene anche detta "table expression".

*Per ogni sede, lo stipendio massimo e quanti impiegati lo percepiscono*

```
SELECT SM.Sede, SM.MaxStip, COUNT(*) AS NumImpWMaxStip
FROM    Imp I, (SELECT  Sede, MAX(Stipendio)
                FROM    Imp
                GROUP BY Sede) AS SM(Sede,MaxStip)
WHERE   I.Sede = SM.Sede
AND     I.Stipendio = SM.MaxStip
GROUP BY SM.Sede, SM.MaxStip
```

Sede	MaxStip
S01	2000
S02	2500
S03	1000

SM

Il linguaggio SQL: viste e tabelle derivate



13



## Table expressions correlate (1)

- Una table expression può essere correlata a un'altra tabella che la precede nella clausola FROM. Ad esempio:

*Per ogni sede, la somma degli stipendi pagati agli analisti*

```
SELECT S.Sede, Stip.TotStip
FROM    Sedi S,
        TABLE(SELECT SUM(Stipendio) FROM Imp I
                WHERE I.Sede = S.Sede
                AND I.Ruolo = 'Analista') AS Stip(TotStip)
```

- Si noti che sedi senza analisti compaiono in output con valore nullo per **TotStip**. Usando il GROUP BY lo stesso risultato si potrebbe ottenere con un LEFT OUTER JOIN, ma occorre fare attenzione...

Il linguaggio SQL: viste e tabelle derivate



14



## Table expressions correlate (2)

*Per ogni sede, il numero di analisti e la somma degli stipendi ad essi pagati*

```
SELECT S.Sede, Stip.NumAn, Stip.TotStip
FROM   Sedi S,
       TABLE(SELECT COUNT(*) ,SUM(Stipendio) FROM Imp I
              WHERE I.Sede = S.Sede
              AND I.Ruolo = 'Analista') AS Stip(NumAn,TotStip)
```

- Per sedi senza analisti **NumAn** vale 0 e **TotStip** è nullo. Viceversa

```
SELECT S.Sede, COUNT(*) AS NumAn, SUM(Stipendio) AS TotStip
FROM   Sedi S LEFT OUTER JOIN Imp I ON
       (I.Sede = S.Sede) AND (I.Ruolo = 'Analista')
GROUP BY S.Sede
```

ha per le sedi senza analisti **TotStip** nullo, ma **NumAn** pari a 1! (Infatti per ognuna di tali sedi c'è una tupla nel risultato dell'outer join). E quindi necessario usare, ad esempio, **COUNT (CodImp)**.

Il linguaggio SQL: viste e tabelle derivate



15



## Limiti delle table expressions

- Si consideri la query:

*la sede in cui la somma degli stipendi è massima*

- La soluzione con table expressions è:

```
SELECT Sede
FROM   (SELECT Sede, SUM(Stipendio) AS TotStip
       FROM   Imp
       GROUP BY Sede) AS SediStip
WHERE  TotStip = (SELECT MAX(TotStip)
                FROM (SELECT Sede, SUM(Stipendio) AS TotStip
                      FROM   Imp
                      GROUP BY Sede) AS SediStip2)
```

- Benché la query sia corretta, non è sfruttato il fatto che **le due table expressions sono identiche**, e ciò porta a una **valutazione inefficiente** e a una **formulazione poco leggibile**.

Il linguaggio SQL: viste e tabelle derivate



16





## Common table expressions

- L'idea alla base delle "common table expressions" è definire una "**vista temporanea**" che può essere usata in una query come se fosse a tutti gli effetti una VIEW:

```
WITH SediStip(Sede,TotStip)
AS  (SELECT  Sede,SUM(Stipendio)
     FROM    Imp
     GROUP BY Sede)
SELECT Sede
FROM  SediStip
WHERE TotStip = (SELECT MAX(TotStip)
                 FROM  SediStip)
```

Il linguaggio SQL: viste e tabelle derivate



17



## WITH e interrogazioni ricorsive (1)

- Si consideri la tabella **Genitori** (**Figlio**, **Genitore**) e la query
 

trova tutti gli antenati (genitori, nonni, bisnonni,...) di Anna
- La query è **ricorsiva** e pertanto **non è esprimibile in algebra relazionale**, in quanto richiede un numero di (self-)join non noto a priori.
- La formulazione mediante common table expressions definisce la vista temporanea (ricorsiva) **Antenati** (**Persona**, **Avo**) facendo l'unione di:
  - una "subquery base" non ricorsiva (che inizializza **Antenati** con le tuple di **Genitori**);
  - una "subquery ricorsiva" che a ogni iterazione aggiunge ad **Antenati** le tuple che risultano dal join tra **Genitori** e **Antenati**.

**Genitori**

Figlio	Genitore
Anna	Luca
Luca	Maria
Luca	Giorgio
Giorgio	Lucia

**Antenati**

Persona	Avo
Anna	Luca
Luca	Maria
Luca	Giorgio
Giorgio	Lucia

+

**Antenati**

Persona	Avo
Anna	Maria
Anna	Giorgio
Luca	Lucia

+

**Antenati**

Persona	Avo
Anna	Lucia

Il linguaggio SQL: viste e tabelle derivate



18

## WITH e interrogazioni ricorsive (2)

```

WITH Antenati(Persona,Avo)
AS  ((SELECT Figlio, Genitore      -- subquery base
      FROM    Genitori)
  UNION ALL
    (SELECT G.Figlio, A.Avo        -- subquery ricorsiva
      FROM    Genitori G, Antenati A
      WHERE   G.Genitore = A.Persona))

SELECT Avo
FROM    Antenati
WHERE   Persona = 'Anna'

```

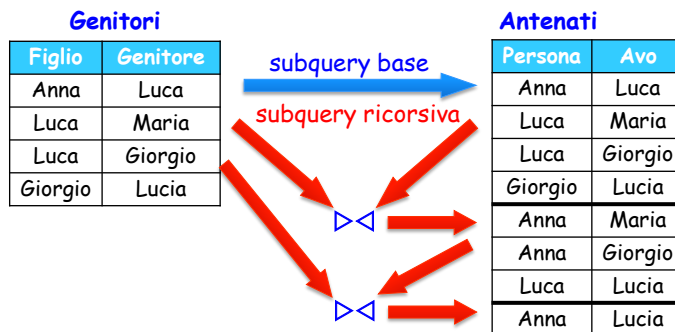
Il linguaggio SQL: viste e tabelle derivate



19

## WITH e interrogazioni ricorsive (3)

- Per comprendere meglio come funziona la valutazione di una query ricorsiva, e come "ci si ferma", si tenga presente che a ogni iterazione il DBMS aggiunge ad **Antenati** le tuple che risultano dal join tra **Genitori** e le sole tuple aggiunte ad **Antenati** al passo precedente:



Il linguaggio SQL: viste e tabelle derivate



20



## Uso delle common table expressions

- Una common table expression è soggetta ad alcune limitazioni d'uso, in particolare si può usare solo nel **blocco più esterno di un SELECT** (anche se usato per creare VIEW o eseguire INSERT).
- Per le subquery ricorsive vi sono alcune restrizioni, tra cui va ricordato che **non si possono usare funzioni aggregate, GROUP BY e SELECT DISTINCT**.



## Sommario:

- Le **viste** sono **tabelle virtuali**, interrogabili come le altre, ma **soggette a limiti** per ciò che riguarda gli **aggiornamenti**.
- Una **table expression** è una **subquery** che definisce una **tabella derivata utilizzabile nella clausola FROM**.
- Una **common table expression** è una "**vista temporanea**" che può essere usata in una query come se fosse a tutti gli effetti una VIEW.
- Mediante common table expression è anche possibile formulare **interrogazioni ricorsive**, definendo "**viste temporanee ricorsive**" come unione del risultato di una "subquery base" e una "subquery ricorsiva".