SQL: Viste

Prof. Alfredo Pulvirenti
Prof. Salvatore Alaimo

(Atzeni-Ceri Capitolo 5)

Viste (View)

- Oltre alle tabelle di base che fanno parte dello schema si possono creare delle **tabelle ausiliarie** virtuali
- Sono “virtuali” in quanto sembrano tabelle a tutti gli effetti ma sono delle relazioni “create al volo”

- Utilizzate per vari scopi:
  - Semplificazione
  - Protezione dati
  - Scomposizione query complesse
  - Riorganizzazione dati secondo nuovi schemi
  - Etc.

Definizione VIEW

- Sintassi creazione VIEW:

```
CREATE VIEW NomeVista
       [“(” Attributo {,Attributo} “)”]
       AS Query-Select
```

Esempio definizione VIEW

```
CREATE VIEW MediaVoti (Matricola,Media) AS
      SELECT Matricola, AVG(Voto)
        FROM Esami
       GROUP BY Matricola
```

- Uso della view:

```
SELECT *
FROM MediaVoti
```

Uso delle VIEW per query complesse

- Semplificare query complesse
- Esempio: non possiamo scrivere

```
SELECT AVG(COUNT(*))
FROM AGENTI
GROUP BY ZONE
```

  - AVG deve agire sui valori di un attributo.

Uso delle VIEW per query complesse

```
CREATE VIEW AgPerZona (Zona,NumAg)
AS
SELECT Zona,COUNT(*)
FROM AGENTI
GROUP BY Zona


SELECT AVG(NumAg)
 FROM AgPerZona


SELECT AVG(NumAg)
 FROM (SELECT Zona,COUNT(*) as NumAg
FROM AGENTI
GROUP BY Zona)
```

Uso delle VIEW per Sicurezza

```
CREATE VIEW EsamiPubblici AS
SELECT Corso,Voto
FROM Esami
```

- Data la tabella
  ClientiBanca(Nome,Indirizzo,Saldo)

```
CREATE VIEW ClientiInd
AS SELECT Nome,Indirizzo
FROM ClientiBanca
```

Le VIEW possono essere usate come tabelle

```
SELECT Nome, Media
FROM Studenti, MediaVoti
WHERE  Studenti.Matricola = MediaVoti.Matricola
```

- Le VIEW possono essere distrutte alla pari di tabelle
  - DROP (TABLE | VIEW) Nome [RESTRICT|CASCADE]
  - Con RESTRICT non viene cancellata se e' utilizzata in altre viste
  - Con CASCADE verranno rimosse tutte le viste che usano la View o la Tabella rimossa
  - La distruzione di una VIEW non altera le tabelle su cui la VIEW si basa

Le VIEW possono essere usate come tabelle

- Una VIEW può essere definita sulla base di un'altra VIEW
- Nelle prime versioni di SQL non era possibile modificare una VIEW tramite *Insert*, *Delete*, *Update*
  - Non piu' vero nei nuovi DBMS (Vedremo dopo)
- Che succede se una tabella usata in una VIEW viene alterata o cancellata (senza specificare RESTRICT o CASCADE)?
  - Dipende dal DBMS:
    - la VIEW viene marcata 'inoperative', oppure
    - La modifica/cancellazione viene negata
    - Etc.

Mascherare l'organizzazione logica dei dati tramite VIEW

- Immaginiamo la seguente tabella:
  - Agenti(CodiceAgente, Nome, Zona, Commissione, Supervisore)

- Per riorganizzazione aziendale si decide di assegnare un Supervisore ad una zona intera invece del singolo agente

```
1)
CREATE TABLE Zone (Zona CHAR(8), Supervisore CHAR(3))
   AS SELECT DISTINCT Zona,Supervisore
        FROM Agenti
2)
CREATE TABLE NuoviAgenti
     AS SELECT CodiceAgente,Nome,Zona,Commissione
          FROM Agenti
3)
DROP Agenti
4)
CREATE VIEW Agenti
       AS SELECT *
            FROM NuoviAgenti NATURAL JOIN Zone
```

Aggiornamento delle VIEW

- Le operazioni INSERT/UPDATE/DELETE sulle VIEW non erano permesse nelle prime edizioni di SQL
- I nuovi DBMS permettono di farlo con *certe limitazioni* dovute alla definizione della VIEW stessa
- Che senso ha aggiornare una VIEW? Dopotutto si potrebbe aggiornare la tabella di base direttamente…

Aggiornamento delle VIEW, cont.

- … utile nel caso di accesso dati controllato
- Esempio:
  - Impiegato(Nome, Cognome, Dipart, Ufficio, Stipendio)

- Il personale della segreteria non puo' accedere ai dati sullo stipendio ma puo' modificare gli altri campi della tabella, aggiungere e/o cancellare tuple

- Si puo' controllare l'accesso tramite la definizione della VIEW:
  - CREATE VIEW Impiegato2 AS
    SELECT Nome, Cognome, Dipart, Ufficio
    FROM Impiegato

- INSERT INTO Impiegato2 VALUES (…)
  - Stipendio verra' inizializzato a Null
  - Se Null non e' permesso per Stipendio l'operazione fallisce

Aggiornamento VIEW 2

- Immaginiamo la seguente VIEW:

```
CREATE VIEW ImpiegatoRossi
  AS
  SELECT *
  FROM Impiegato
  WHERE Cognome='Rossi'
```

- La seguente operazione ha senso:
  - INSERT INTO ImpiegatoRossi (…'Rossi',…)

Aggiornamento VIEW 2, cont.

- Ma che succede nel caso di:
  - INSERT INTO ImpiegatoRossi (…'Bianchi',…)
  - In genere e' permesso, finisce nella tabella base ma non e' visibile dalla VIEW
  - Si puo' controllare tramite l'opzione "WITH CHECK OPTION":

```
CREATE VIEW ImpiegatoRossi AS
  SELECT *
  FROM Impiegato
  WHERE Cognome='Rossi'
  WITH CHECK OPTION
```

- Adesso l'insert con 'Bianchi' fallisce, quella con 'Rossi' viene invece eseguita.

Aggiornamento VIEW 3

- Consideriamo il seguente caso:

```
Impiegato( Nome, Cognome, Dipart, Ufficio, Stipendio)
Dipartimenti( Dipart, Indirizzo)

CREATE VIEW IMP_IND AS
SELECT Nome, Cognome, d.dipart, indirizzo
FROM Impiegato i join Dipartimenti d ON
i.Dipart=d.Dipart
```

- Un INSERT sulla VIEW IMP_IND dovrebbe inserire su entrambe le tabelle base

- In alcuni casi potrebbe inserire in una ma non nell'altra
- In genere quest'operazione non è consentita

- Alcuni DBMS consentirebbero l'INSERT se "Impiegati.Dipart" fosse una foreign key su "Dipartimenti.Dipart" e quest'ultima fosse chiave primaria

Aggiornamento VIEW, riepilogo

- In genere una VIEW definita su una singola tabella è modificabile se gli attributi della VIEW contengono la chiave primaria (e altre chiavi)
- In genere VIEW definite su piu' tabelle non sono aggiornabili
  - Alcuni DBMS, come discusso prima, lo permettono nel caso certe condizioni, molto restrittive, siano rispettate
- VIEW che usano funzioni di aggregazione non sono aggiornabili
- PRINCIPIO di base per l'aggiornamento delle VIEW:
  - Ogni riga ed ogni colonna della VIEW deve corrispondere ad una ed una sola riga ed una ed una sola colonna della tabella base

Esempio 1

Dato il seguente schema:

AEROPORTO(Città, Nazione,NumPiste)

VOLO(IdVolo,GiornoSett,CittaPart,OraPart,CittaArr,OraArr,TipoAereo)

AEREO(TipoAereo,NumPasseggeri,QtaMerci)

- scrivere, facendo uso di una vista, l'interrogazione SQL che permette di determinare il massimo numero di passeggeri che possono arrivare in un aeroporto italiano dalla Francia di giovedì (se vi sono più voli, si devono sommare i passeggeri).

```
CREATE VIEW Passeggeri(Numero)
AS SELECT SUM( NumPasseggeri )
FROM AEROPORTO AS A1 JOIN VOLO ON A1.Citta=CittaPart
JOIN AEROPORTO AS A2 ON A2.Citta=CittaArr
JOIN AEREO ON VOLO.TipoAereo=Aereo.TipoAereo
WHERE A1.Nazione='Francia' AND A2.Nazione='Italia' AND GiornoSett='Giovedì'
GROUP BY A2.Citta

SELECT MAX(Numero)
FROM Passeggeri
```

Esempio 2

- Definire una vista che mostra per ogni dipartimento il valore medio degli stipendi superiori alla media del dipartimento

```
CREATE VIEW SalariSopraMedia (Dipartimento,Stipendio)
AS SELECT Dipartimento, AVG(Stipendio)
FROM Impiegato AS I
WHERE Stipendio > (
    SELECT AVG(I2.Stipendio)
    FROM Impiegato AS I2
    WHERE I.Dipartimento=I2.Dipartimento)
GROUP BY I.Dipartimento
```

Esempio 3 (vincoli)

- Definire sulla tabella Impiegato il vincolo che il dipartimento Amministrazione abbia meno di 100 dipendenti, con uno stipendio medio superiore ai 40 mila €.

```
CHECK (100 >= ( SELECT COUNT(*)
                FROM Impiegato
                WHERE Dipartimento='Amministrazione' )
      AND 40000 <= (SELECT AVG(Stipendio)
                    FROM Impiegato
                    WHERE
  Dipartimento='Amministrazione'))
```

Esempio 4 (vincoli)

- Definire a livello di schema il vincolo che il massimo degli stipendi degli impiegati di dipartimenti con sede a Firenze sia minore dello stipendio di tutti gli impiegati del dipartimento Direzione.

```
CREATE ASSERTION ControlloSalari
CHECK ( NOT EXISTS ( SELECT *
FROM Impiegato I JOIN Dipartimento D ON I.Dipartimento=D.Nome
WHERE D.Città='Firenze'
   AND Stipendio > ( SELECT MIN(Stipendio)
                     FROM Impiegato
                     WHERE Dipartimento='Direzione') ) )
```

Algebra relazionale: limiti

- L'insieme di interrogazioni esprimibili con l'algebra relazionale è significativo; il concetto è robusto

- Ci sono però interrogazioni interessanti non esprimibili:
  - **calcolo di valori derivati**: possiamo solo estrarre valori, non calcolarne di nuovi; calcoli di interesse:
    - a livello di ennupla o di singolo valore (conversioni somme, differenze, etc.)
    - su insiemi di ennuple (somme, medie, etc.)
  - le estensioni sono ragionevoli, sono state viste in SQL
  - interrogazioni inerentemente ricorsive, come la chiusura transitiva

Chiusura transitiva

Supervisione(Impiegato, Capo)

Per ogni impiegato, trovare tutti i superiori (cioè il capo, il capo del capo, e cosi' via)

Impiegato	Capo
Rossi	Lupi
Neri	Bruni
Lupi	Falchi

Impiegato	Superiore
Rossi	Lupi
Neri	Bruni
Lupi	Falchi
Rossi	Falchi

Chiusura transitiva, come si fa?

Nell'esempio, basterebbe il join della relazione con se stessa, previa opportuna ridenominazione

Ma:

Impiegato	Capo
Rossi	Lupi
Neri	Bruni
Lupi	Falchi
Falchi	Leoni

Impiegato	Superiore
Rossi	Lupi
Neri	Bruni
Lupi	Falchi
Falchi	Leoni
Rossi	Falchi
Lupi	Leoni
Rossi	Leoni

Chiusura transitiva, impossibile!

Non esiste in algebra e calcolo relazionale la possibilità di esprimere l'interrogazione che, per ogni relazione binaria, ne calcoli la chiusura transitiva

Per ciascuna relazione, è possibile calcolare la chiusura transitiva, ma con un'espressione ogni volta diversa:

quanti join servono?

non c'è limite!

Datalog

Un linguaggio di programmazione logica per basi di dati derivato dal Prolog

Utilizza predicati di due tipi:

- estensionali: relazioni della base di dati
- intensionali: corrispondono alle viste

Il linguaggio è basato su regole utilizzate per "definire" i predicati estensionali

Datalog, sintassi

Regole:

testa ← *corpo*

testa è un predicato atomico (intensionale)

corpo è una lista (congiunzione) di predicati atomici

Le interrogazioni sono specificate per mezzo di predicati atomici (convenzionalmente preceduti da "?")

Esempio 1

Trovare matricola, nome, età e stipendio degli impiegati che hanno 30 anni

? Impiegati(Matricola: m, Nome: n, Età: 30, Stipendio: s)

Esempio 2

Trovare matricola, nome, età e stipendio degli impiegati che guadagnano più di 40

Serve un predicato intensionale

ImpRicchi(Matricola: m, Nome: n, Età: e, Stipendio: s) ←
Impiegati(Matricola: m, Nome: n, Età: e, Stipendio: s) , s >40

? ImpRicchi(Matricola: m, Nome: n, Età: e, Stipendio: s)

Esempio 3

Trovare matricola, nome ed età di tutti gli impiegati

InfoPubbliche(Matricola: m, Nome: n, Età: e)
    ← Impiegati(Matricola: m, Nome: n, Età: e, Stipendio: s)

? InfoPubbliche(Matricola: m, Nome: n, Età: e)

Esempio 4

Trovare le matricole dei capi degli impiegati che guadagnano più di 40

CapiDeiRicchi (Capo:c) ←
    ImpRicchi(Matricola: m, Nome: n, Età: e, Stipendio: s),
    Supervisione (Capo:c,Impiegato:m)

CapiDeiRicchi (Capo:c) ←
    Impiegati(Matricola: m, Nome: n, Età: e, Stipendio: s),
    Supervisione (Capo:c,Impiegato:m), s>40

? CapiDeiRicchi (Capo:c)

Esempio 5

Trovare matricola e nome dei capi i cui impiegati guadagnano tutti più di 40

serve la negazione

CapiDiNonRicchi (Capo:c) ←

Supervisione (Capo:c,Impiegato:m),
Impiegati (Matricola: m, Nome: n, Età: e, Stipendio: s) ,
$s \leq 40$

CapiSoloDiRicchi (Matricola: c, Nome: n) ←

Impiegati (Matricola: c, Nome: n, Età: e, Stipendio: s) ,
Supervisione (Capo:c,Impiegato:m),
not CapiDiNonRicchi (Capo:c)

? CapiSoloDiRicchi (Matricola: c, Nome: n)

Esempio 6

Per ogni impiegato, trovare tutti i superiori.

Serve la ricorsione

Superiore (Impiegato: i, SuperCapo: c) ←
    Supervisione (Impiegato: i, Capo: c)

Superiore (Impiegato: i, SuperCapo: c) ←
    Supervisione (Impiegato: i, Capo: c'),
    Superiore (Impiegato: c', SuperCapo: c)

Datalog, semantica

La definizione della semantica delle regole ricorsive è delicata (in particolare con la negazione)

Potere espressivo:

Datalog non ricorsivo con negazione è equivalente all'algebra

Datalog ricorsivo con negazione è più espressivo dell’algebra

Viste ricorsive in SQL:1999

```
WITH RECURSIVE factorial (n, fact) AS
(
  SELECT 0, 1 -- Initial Subquery
    UNION ALL
  SELECT n+1, (n+1)*fact
  FROM factorial -- Recursive Subquery
  WHERE n < 9
)
SELECT * FROM factorial;
```

Viste ricorsive

Per ogni persona, trovare tutti gli antenati, avendo

Paternita (Padre, Figlio)

Serve la ricorsione; in Datalog:

Discendenza (Antenato: a, Discendente: d) ←
    Paternita (Padre: a, Figlio: d)

Discendenza (Antenato: a, Discendente: d) ←
    Paternita (Padre: a, Figlio: f) ,
    Discendenza (Antenato: f, Discendente: d)

Viste ricorsive in SQL:1999

```
WITH RECURSIVE Discendenza(Antenato,Discendente) AS
(
    SELECT Padre, Figlio
    FROM Paternita
      UNION ALL
    SELECT D.Antenato, Figlio
    FROM Discendenza D, Paternita
    WHERE D.Discendente = Padre
)
SELECT * FROM Discendenza;
```

Funzioni scalari

Funzioni a livello di ennupla che restituiscono singoli valori

Temporali

current_date, extract(year from data)

Manipolazione stringhe

char_length, lower

Conversione

CAST(X AS TIPO)

Condizionali

…

Funzioni condizionali

CASE, COALESCE, NULLIF

```
SELECT Nome, Cognome, COALESCE(Dipart,'Ignoto')
FROM Impiegato
```

```
SELECT Targa,
  CASE Tipo
     WHEN 'Auto' THEN 2.58 * KWatt
     WHEN 'Moto' THEN (22.00 + 1.00 * KWatt)
     ELSE NULL
  END AS Tassa
FROM Veicolo
WHERE Anno > 1975
```