



LINGUAGGIO SQL

1

Dati derivati

- **colonne derivate**
- **relazioni derivate**
- **viste**



COLONNE DERIVATE

- Vogliamo modificare la tabella Noleggio, aggiungendo la durata in giorni

i valori sono calcolati ed assegnati automaticamente in caso di inserimento e modifica (INSERT ed UPDATE possono assegnare alle colonne derivate solo DEFAULT)

il dominio, se specificato, deve essere compatibile con il tipo dell'espressione

```
ALTER TABLE Noleggio ADD COLUMN durata INTEGER  
GENERATED ALWAYS AS ((dataRest-dataNoI) DAY);
```

l'espressione deve valutarsi sempre nello stesso modo se le istanze del DB sono le stesse (es. NON deve contenere Current_Date)

tutte le colonne che compaiono nell'espressione devono appartenere alla relazione in cui viene definita la colonna e non possono essere calcolate



RELAZIONI DERIVATE

- Vogliamo creare una nuova relazione ClienteV con lo stesso schema di Cliente più una colonna bonus

copia in ClienteV la struttura completa (nomi delle colonne, domini e vincoli di obbligatorietà) di Cliente

i vincoli generali **non** vengono copiati

eventuali dati **non** vengono copiati

```
CREATE TABLE ClienteV  
(LIKE Cliente,  
bonus NUMERIC(4,2));
```

La nuova tabella è vuota e non ha connessioni permanenti a quelle su cui è basata

nessuna relazione fra le istanze

nessuna propagazione di modifiche



RELAZIONI DERIVATE DA QUERY

- Decidiamo di memorizzare in due relazioni separate i noleggi conclusi ed i noleggi in corso

Creo una tabella con schema uguale a una parte di schema di altre relazioni

```
CREATE TABLE NoleggioA AS  
SELECT colloc, dataNoI, codCli  
FROM Noleggio  
WHERE dataRest IS NULL  
WITH NO DATA;
```

Viene creata una relazione vuota

```
CREATE TABLE NoleggioC AS  
SELECT * FROM Noleggio  
WHERE dataRest IS NOT NULL  
WITH DATA;
```

la relazione è popolata con le tuple risultato della valutazione dell'interrogazione, che vengono memorizzate (*materializzate*) su disco

l'interrogazione viene utilizzata **solo** per la creazione ed eventualmente per la popolazione iniziale della relazione

modifiche alle tabelle usate nella query **NON** si propagano alla nuova tabella



VISTE (VIEW)

- In alcune situazioni creare una tabella con schema e contenuto iniziale derivato non basta
- Ad esempio, supponiamo che un nuovo socio della videoteca abbia necessità di gestire (vedere, manipolare) solo i noleggi dei clienti over 65

```
CREATE TABLE NoleggioOver65 AS  
SELECT *  
FROM Noleggio NATURAL JOIN Cliente  
WHERE (CURRENT_DATE – dataN) YEAR >= 65  
WITH DATA;
```

Se vengono aggiunti nuovi clienti over 65 in Noleggio, la relazione NoleggioOver65 non li contiene

Se si aggiunge una tupla a NoleggioOver65, la tupla non viene anche inserita in Noleggio

- Serve un meccanismo per «legare» l'istanza di NoleggioOver65 e la query che ne definisce l'istanza iniziale → **Viste**



VISTA (VIEW)

- Relazione *virtuale*
- Permette di accedere diversamente ai dati memorizzati nelle relazioni “reali” (di *base*)
- Si definisce tramite un'interrogazione su relazioni di base e viste
- Non corrisponde a dati memorizzati secondo il suo schema ma al risultato dell'interrogazione
- Può essere usata a quasi tutti gli effetti come una relazione di base
- Il meccanismo delle viste è utile per
 - semplificare l'accesso ai dati
 - fornire indipendenza logica
 - garantire la privatezza dei dati



ESEMPIO

```
CREATE VIEW Over65 AS
```

```
SELECT *
```

```
FROM Noleggio NATURAL JOIN Cliente
```

```
WHERE (CURRENT_DATE – dataN) YEAR >= 65
```

La vista può poi essere usata come una normale relazione

```
SELECT * FROM Over65
```



COMANDO DI CREAZIONE

obbligatoria solo se l'interrogazione contiene colonne virtuali senza nome nella clausola di proiezione

nome della vista
che viene creata

lista di nomi da assegnare alle
colonne della vista (= in
numero a quelle restituite
dall'interrogazione)

CREATE VIEW <nome vista> [(<lista nomi colonne>**)]**

AS <interrogazione>

interrogazione di definizione
non ci sono restrizioni sulla forma

[WITH {LOCAL| CASCADED} CHECK OPTION];

discusse in seguito



ESEMPIO

Vista che restituisce codice cliente, data di inizio noleggio e collocazione dei video in noleggio da più di tre giorni

```
CREATE VIEW Nol3gg AS  
SELECT codCli, dataNol, colloc  
FROM Noleggio  
WHERE dataRest IS NULL AND  
(CURRENT_DATE - dataNol) DAY  
> INTERVAL '3' DAY;
```

I nomi delle colonne della vista sono
codCli, dataNol e colloc

```
CREATE VIEW Nol3gg  
(codiceCliente, dataNoleggio, Video)  
AS  
SELECT codCli, dataNol, colloc  
FROM Noleggio  
WHERE dataRest IS NULL AND  
(CURRENT_DATE - dataNol) DAY  
> INTERVAL '3' DAY;
```

I nomi delle colonne della vista sono
codiceCliente, dataNoleggio, Video



PERCHÉ LE VISTE SEMPLIFICANO L'ACCESSO AI DATI?

CASI COMUNI – JOIN

Per semplificare l'accesso agli utenti che lavorano sempre/spesso con particolari associazioni fra entità

- una sola *tabella* invece di un join esplicito
- si possono prefiltrare eventuali dati *superflui*

Esempio: Vista che restituisce il numero di telefono del cliente, la data di inizio noleggio, il titolo del film e l'importo dovuto per i video in noleggio da più di tre giorni (supponendo che la tariffa giornaliera di noleggio sia 5 Euro)

CREATE VIEW InfoNol3gg AS

```
SELECT telefono, dataNol, titolo, 5 * ((CURRENT_DATE - dataNol)
DAY) AS importo
FROM Cliente NATURAL JOIN Noleggio NATURAL JOIN Video
WHERE dataRest IS NULL AND
(CURRENT_DATE - dataNol) DAY > INTERVAL '3' DAY;
```

I nomi delle colonne della vista sono telefono, dataNol, titolo e importo



PERCHÉ LE VISTE SEMPLIFICANO L'ACCESSO AI DATI?

CASI COMUNI – GROUP BY

Per astrarre i dati e presentarli agli utenti in modo aggregato

- permette di garantire riservatezza
- es. si vogliono dare informazioni sul numero di noleggi dei singoli film alle ditte di distribuzione, senza divulgare chi ha noleggiato cosa per ragioni di privacy

Esempio Vista che per ogni cliente visualizza il nome, la residenza, l'anno di nascita, il numero di noleggi effettuati e la durata massima di tali noleggi

```
CREATE VIEW InfoCli (nome, residenza, annoN, numNol, durataM) AS
SELECT nome, residenza, YEAR(dataN), COUNT(*), MAX((dataRest -
dataNol) DAY)
FROM Cliente NATURAL JOIN Noleggio
GROUP BY codCli, nome, residenza, dataN;
```

I nomi delle colonne della vista sono nome, residenza, annoN, numNol, durataM



INTERROGAZIONI E AGGIORNAMENTI SU VISTE

- L'utente percepisce le viste come tabelle
 - vorrebbe manipolarle come le relazioni di base
- Su una vista si possono eseguire
 - interrogazioni
 - inserimenti
 - aggiornamenti
- Tutte le operazioni però subiscono forti restrizioni
 - facili da capire se ci si ricorda che una vista corrisponde a una query



INTERROGAZIONI SU VISTE

- Una vista può essere utilizzata nelle interrogazioni
- Su una vista **ad esempio** è possibile
 - effettuare proiezioni
 - specificare condizioni di ricerca
 - effettuare dei join con altre relazioni o viste
 - effettuare raggruppamenti e calcolare funzioni di gruppo
 - definire altre viste

ESEMPIO 1

- Determinare per ciascun cliente di Genova nato tra il 1970 e il 1976 nome, numero e durata massima dei suoi noleggi
- Si può fare a partire dalla vista InfoCli che fornisce per ciascun cliente tutti i suoi dati ed i dati aggregati sui suoi noleggi (definita nella slide 12)

```
CREATE VIEW InfoCli (nome, residenza, annoN, numNol, durataM) AS  
  SELECT nome, residenza, YEAR(dataN), COUNT(*),  
         MAX((dataRest - dataNol) DAY)  
  FROM Cliente NATURAL JOIN Noleggio  
  GROUP BY codCli, nome, residenza, dataN;
```

- InfoCli viene usata come fosse una tabella di base

```
SELECT nome, NumNol, durataM  
FROM InfoCli  
WHERE annoN BETWEEN 1970 AND 1976 AND residenza LIKE  
'%genova';
```



ESEMPIO 4

Uso di vista per definirne un'altra: definizione alternativa della vista InfoNol3gg, basata sulla vista Nol3gg (definita nelle slide 10/11)

```
CREATE VIEW Nol3gg (codiceCliente,  
dataNoleggio,Video) AS  
    SELECT codCli, dataNol, colloc  
    FROM Noleggio  
    WHERE dataRest IS NULL AND (CURRENT_DATE  
- dataNol) DAY > INTERVAL '3' DAY;
```

```
CREATE VIEW InfoNol3ggB AS  
SELECT telefono, dataNol, titolo, 5 * ((CURRENT_DATE -  
dataNol) DAY) AS importo  
FROM Nol3gg NATURAL JOIN Cliente NATURAL JOIN  
Video;
```



ESEMPIO 4 – ESPANSIONE DI INFO NOL3GG B

CREATE VIEW InfoNol3ggB AS

SELECT telefono, dataNol, titolo, 5 * ((CURRENT_DATE - dataNol) DAY)
AS importo

FROM Nol3gg NATURAL JOIN Cliente NATURAL JOIN Video;

CREATE VIEW Nol3gg AS

SELECT codCli, dataNol, colloc

FROM Noleggio

WHERE dataRest IS NULL AND (CURRENT_DATE - dataNol) DAY >
INTERVAL '3' DAY;

Riscritto
in

CREATE VIEW InfoNol3ggB AS

SELECT telefono, dataNol, titolo, 5 * ((CURRENT_DATE - dataNol) DAY)
AS importo

FROM Noleggio NATURAL JOIN Cliente NATURAL JOIN Video

WHERE dataRest IS NULL AND (CURRENT_DATE - dataNol)
DAY > INTERVAL '3' DAY;



INTERROGAZIONI SU VISTE - PROBLEMI

- **Restrizione:** non è ammesso l'uso di funzioni di gruppo su colonne di viste che sono a loro volta definite tramite funzioni di gruppo
- **Motivazione**
 - la valutazione di un'interrogazione Q su una vista può essere effettuata **componendo** Q con l'interrogazione di definizione
 - in SQL non è possibile applicare funzioni di gruppo in cascata
- Questa restrizione richiede che l'utente finale
 - sia conscio di stare usando una vista
 - ne conosca la definizione

⇒ **si perde la trasparenza**

INTERROGAZIONI SU VISTE – ESEMPIO (OK)

```
SELECT nome, NumNol, durataM  
FROM InfoCli  
WHERE annoN BETWEEN 1970 AND 1976 AND residenza LIKE '%genova';
```



```
CREATE VIEW InfoCli (nome, residenza, annoN, numNol, durataM) AS  
SELECT nome, residenza, YEAR(dataN), COUNT(*), MAX((dataRest -  
dataNol) DAY)  
FROM Cliente NATURAL JOIN Noleggio  
GROUP BY codCli, nome, residenza, dataN;
```

Composizione



```
SELECT nome, COUNT(*), MAX((dataRest - dataNol) DAY  
FROM Cliente NATURAL JOIN Noleggio  
WHERE YEAR(dataN) BETWEEN 1970 AND 1976 AND residenza LIKE  
'%genova'  
GROUP BY codCli, nome, residenza, dataN;
```

INTERROGAZIONI SU VISTE – ESEMPIO (KO)

```
SELECT AVG(durataM)
FROM InfoCli
WHERE annoN BETWEEN 1970 AND 1976 AND residenza LIKE '%genova';
```

```
CREATE VIEW InfoCli (nome, residenza, annoN, numNol, durataM) AS
SELECT nome, residenza, YEAR(dataN), COUNT(*),
MAX((dataRest - dataNol) DAY)
FROM Cliente NATURAL JOIN Noleggio
GROUP BY codCli, nome, residenza, dataN;
```

AVG(MAX(..)) non è ammesso

```
SELECT AVG(MAX((dataRest - dataNol) DAY))
FROM Cliente NATURAL JOIN Noleggio
WHERE YEAR(dataN) BETWEEN 1970 AND 1976 AND residenza LIKE '%genova'
GROUP BY codCli, nome, residenza, dataN;
```

Composizione

V

A	B	C
1	2	3
5	4	9
2	8	10

AGGIORNAMENTI SU VISTE

R

A	B	C
1	2	3
4	1	6
5	4	9
2	8	10

- **SE** possibile, l'esecuzione di un'operazione di aggiornamento su una vista viene propagata alla relazione su cui la vista è definita
- Creo una vista V su R
 Create view V as Select * from R where B>=2
- Voglio eseguire **Insert into V Values (2,8,10)**
- V è definita in termini di R \Rightarrow devo inserire (2,8,10) in R
- In questo caso semplice tutto va bene
 - analogamente per update e delete
- In casi complicati le operazioni potrebbero non andare a buon fine, quindi sono proibite



PROBLEMA 0 (ESEMPIO MODIFICA)

- Una modifica a una colonna di una vista viene scaricata sulla modifica alla colonna corrispondente nella relazione di base
- Se la colonna della vista è virtuale (definita da un'espressione) non è sempre possibile stabilire su quale colonna della relazione di base agire e/o quale valore assegnarle

- Esempio

```
CREATE VIEW InfoNoCosti AS
```

```
SELECT codCli, dataNo, titolo, 5 * ((dataRest - dataNo) DAY) AS  
importo
```

```
FROM Cliente NATURAL JOIN Noleggio NATURAL JOIN Video  
WHERE dataRest IS NOT NULL;
```

- UPDATE InfoNoCosti importo = importo + 2
 - modifica dataNo o dataRest?
 - ammesso di scegliere arbitrariamente di modificare dataRest..che valore scelgo?
- In questi casi la modifica verrà rifiutata



PROBLEMA 1 (ESEMPIO INSERIMENTO)

- Un inserimento in una vista viene scaricato su un inserimento nella relazione di base
- Se la vista **non** contiene una colonna della relazione di base su cui
 - è specificato un vincolo NOT NULL
 - non è specificato nello schema un valore di default
 1. il comando di inserimento sulla vista non specifica un valore per la colonna
 2. la colonna è obbligatoria e senza valore di default
 3. non si può effettuare l'inserimento nella relazione di base
- Esempio

```
CREATE VIEW ClientiConNoleggiAperti AS
SELECT codCli, dataNol
FROM Noleggio
WHERE dataRest IS NULL;
```

 - **INSERT INTO ClientiConNoleggiAperti**
VALUES (6635, CURRENT_DATE)
 - che colloc usa per inserire la tupla in Noleggio?
 - l'inserimento verrà rifiutato



PROBLEMA 2 (ESEMPIO CANCELLAZIONE)

- Una cancellazione da una vista viene scaricata su una cancellazione dalle relazioni di base
- Possono esserci diverse possibilità che portano a modificare in modi diversi l'istanza della base di dati

S

A	D	E
1	2	7
1	4	8

PROBLEMA 2 (ESEMPIO ESTESO)

VV

A	B	C	D	E
1	2	3	2	7
1	2	3	4	8

R

A	B	C
4	2	3
4	1	6
5	4	9

- Creo una vista VV su R ed S
Create view V as Select * from R NATURAL JOIN S
- Voglio eseguire **DELETE FROM VV WHERE A=1**
 - Posso
 - cancellare in R
 - cancellare in S
 - cancellare in entrambe
 - modificare la tupla (1,2,3) in R....
- Il non determinismo è un problema perché è indice di incertezza sulle intenzioni dell'utente e potrebbe portare la base di dati in uno stato incomprensibile all'utente
 - soprattutto ad uno che veda le relazioni base e non la vista

DELETE SU VISTE: RESTRIZIONI

- È possibile eseguire il comando DELETE su V se la query di definizione Q
 - è su una singola relazione R
 - non contiene
 - GROUP BY
 - DISTINCT
 - operatori insiemistici
 - funzioni di gruppo
 - eventuali sotto-interrogazioni in Q non fanno riferimento ad R

UPDATE SU VISTE : RESTRIZIONI

- È possibile eseguire il comando UPDATE su una colonna C di V se
- È possibile eseguire il comando se:
 - la query di definizione Q soddisfa tutte le restrizioni richieste per il DELETE
 - C non è definita tramite un'espressione o funzione



INSERT IN VISTE: RESTRIZIONI

- È possibile eseguire il comando INSERT in V se
 - la query di definizione Q soddisfa tutte le restrizioni richieste per il DELETE
 - tutte le colonne di V soddisfano le restrizioni richieste per l'UPDATE
 - tutte le colonne di R su cui vale il vincolo di NOT NULL e per cui non è specificato valore di default sono incluse in V

AGGIORNAMENTI SU VISTE

- Le restrizioni indicate sono sufficienti affinché le operazioni di aggiornamento della vista possano essere scaricate in maniera univoca su aggiornamenti della relazione di base
- non sono però necessarie
- lo standard ammette anche aggiornamenti su viste la cui interrogazione di definizione contiene più di una relazione
 - purché sia possibile stabilire una corrispondenza uno a uno tra le tuple della vista e le tuple delle relazioni di base
- i DBMS commerciali ammettono al più aggiornamenti su viste contenenti un'unica relazione nell'interrogazione di definizione

ULTERIORE PROBLEMA

```
CREATE VIEW Nol3gg AS
SELECT codCli, dataNol, colloc
FROM Noleggio
WHERE dataRest IS NULL AND
      (CURRENT_DATE - dataNol) DAY >
      INTERVAL '3' DAY;
```

- Supponiamo di voler inserire nella vista Nol3gg la seguente tupla (6635, CURRENT_DATE, 1128)
 - la data di noleggio specificata è oggi
 - la condizione nell'interrogazione (noleggio iniziato almeno tre giorni fa) **non** è verificata dalla nuova tupla
 - le restrizioni per l'inserimento sono soddisfatte
 - **la tupla viene inserita in Noleggio ma non apparterrà alla vista Nol3gg**
- per assicurare che le tuple siano inserite/modificate tramite una vista solo se verificano la condizione nella sua interrogazione di definizione, si usa la clausola **CHECK OPTION** del comando CREATE VIEW

CHECK OPTION

- Nell'esempio precedente, se Nol3gg è definita come

```
CREATE VIEW Nol3gg AS  
SELECT codCli, dataNol, colloc  
FROM Noleggio  
WHERE dataRest IS NULL AND (CURRENT_DATE - dataNol)  
DAY > INTERVAL '3' DAY  
WITH CHECK OPTION;
```

l'inserimento di tuple che non soddisfano l'interrogazione di definizione della vista, come (6635, CURRENT_DATE, 1128), non è permesso

CHECK OPTION

- La situazione si complica ulteriormente nel caso di viste definite in termini di altre viste
 - ognuna di tali viste potrebbe o meno essere definita con CHECK OPTION
- la CHECK OPTION può essere specificata con due possibili alternative
 - LOCAL
 - CASCADED (default)
- la differenza tra LOCAL e CASCADED è rilevante nei casi in cui una vista è definita in termini di un'altra vista

CHECK OPTION

```
CREATE VIEW V1 AS  
SELECT *  
FROM V2  
WHERE B > 2
```

WITH LOCAL CHECK OPTION

WITH CHECK OPTION

gli inserimenti eseguiti su V1 devono verificare
l'interrogazione di definizione di V1
l'interrogazione di definizione di V2

```
CREATE VIEW V2 AS  
SELECT *  
FROM R  
WHERE D < 6;
```

gli inserimenti eseguiti su V1 devono verificare
l'interrogazione di definizione di V1

CHECK OPTION

```
CREATE VIEW V1 AS  
SELECT *  
FROM V2  
WHERE B > 2
```

**WITH CASCADED CHECK
OPTION**

WITH CHECK OPTION

gli inserimenti eseguiti su V1 devono verificare
l'interrogazione di definizione di V1
l'interrogazione di definizione di V2

```
CREATE VIEW V2 AS  
SELECT *  
FROM R  
WHERE D < 6;
```

gli inserimenti eseguiti su V1 devono verificare
l'interrogazione di definizione di V1
l'interrogazione di definizione di V2