

SQL



DOCENTE
PROF. ALBERTO BELUSSI

Anno accademico 2010/11

Il concetto di vista

2

DEFINIZIONE

È una relazione derivata. Si specifica l'espressione che genera il suo contenuto. Esso quindi dipende dal contenuto delle relazioni che compaiono nell'espressione.

OSSERVAZIONE

Una vista può essere funzione di altre viste purché esista un ordinamento in grado di guidare il calcolo delle relazioni derivate.

Il concetto di vista

3

VISTA MATERIALIZZATA

Una vista si dice “materializzata” se viene calcolata e memorizzata esplicitamente nella base di dati.

Conviene materializzare quando:

- L’aggiornamento è raro
- L’interrogazione che genera la vista è complessa

Se la vista viene materializzate è necessario ricalcolarne il contenuto ogni volta che le relazioni di cui dipende vengono aggiornate.

Il concetto di vista

4

VISTA VIRTUALE

Una vista si dice “virtuale” se viene calcolata ogni volta che serve.

Conviene quando:

- L'aggiornamento è frequente
- L'interrogazione che genera la vista è semplice

Utilità delle viste:

- Consentono di personalizzare l'interfaccia utente
- Facilitano la gestione della privacy dei dati
- Permettono di memorizzare nel DBMS interrogazioni complesse condivise
- Sono utili per rendere l'interfaccia delle applicazioni indipendente dallo schema logico (INDIPENDENZA LOGICA)

SQL

5

SQL (Structured Query Language) è stato definito nel 1973 ed è oggi il linguaggio più diffuso per i DBMS relazionali.

La semantica dell'SQL fa riferimento al CALCOLO RELAZIONALE.

Poiché questo linguaggio non è parte del programma del corso, la semantica di SQL verrà presentata in modo informale e quando possibile verrà assegnata utilizzando espressioni dell'algebra relazionale.

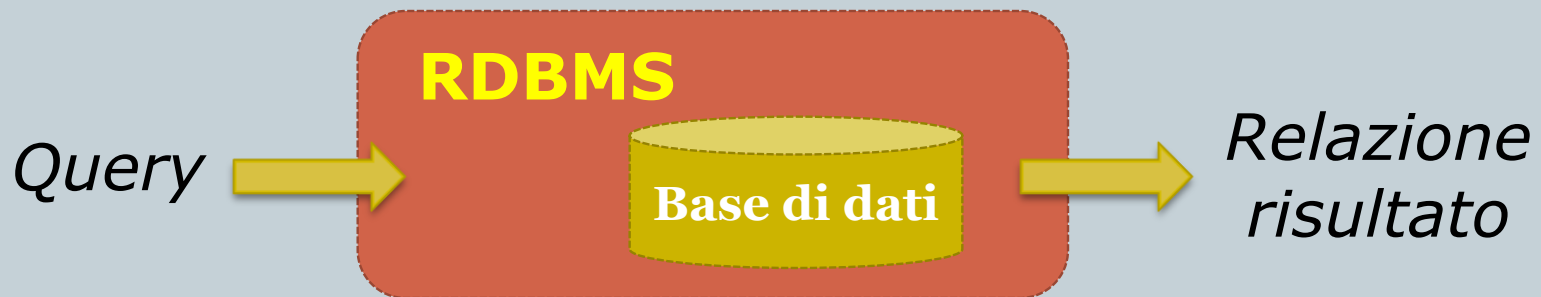
SQL

6

FORMA BASE di una interrogazione SQL (o QUERY)

```
SELECT <ListaAttributi>  
FROM <ListaTabelle>  
[WHERE <Condizione>]
```

[...] indica una parte
opzionale



SQL

7

Data la query SQL:

```
SELECT <ListaAttributi>  
FROM <ListaTabelle>  
[WHERE <Condizione>]
```

La sua esecuzione produce una relazione risultato con le seguenti caratteristiche:

SCHEMA: è costituito da tutti gli attributi indicati in <ListaAttributi>

CONTENUTO: è costituito da tutte le tuple t ottenute proiettando sugli attributi di <ListaAttributi> (clausola SELECT) le tuple t' appartenenti alle tabelle indicate in <ListaTabelle> (clausola FROM) che soddisfano l'eventuale condizione <Condizione> (clausola WHERE).

Esempi

8

TRENO(Num, Cat, Part, Arrivo, Dest)

FERMATA(Treno, Stazione, Orario)

```
SELECT Num, Cat, Dest  
FROM TRENO  
WHERE PART > '14.00'
```

```
SELECT Stazione, Orario  
FROM FERMATA  
WHERE Treno = 123
```


Clausola SELECT

9

Clausola SELECT

SELECT <ListaAttributi>

<ListaAttributi> è una lista di espressioni con la seguente sintassi:

< [DISTINCT] <espr> [[AS] <alias>] {, <espr>[[AS]<alias>]} | * >

dove:

- **<espr>** è un'espressione che coinvolge gli attributi della tabella (può anche essere semplicemente il nome di un attributo)
- **"*"** è una stenografia per indicare tutti gli attributi delle tabelle
- **<alias>** è il nome assegnato all'attributo che conterrà il risultato della valutazione dell'espressione **<espr>** nella relazione risultato
- **DISTINCT**: se presente richiede l'eliminazione delle tuple duplicate

Clausola SELECT: esempi

10

```
SELECT *  
FROM TRENO  
WHERE Cat = 'IC'
```

```
SELECT DISTINCT Dest  
FROM TRENO  
WHERE Cat='Reg' AND Part > '9.00'
```

Clausola FROM

11

Clausola FROM

FROM<ListaTabelle>

<ListaTabelle> è una lista di espressioni con la seguente sintassi:

<tabella> [[AS] <alias>] {, <tabella> [[AS] <alias>]}

dove:

- Se sono presenti due o più tabelle la semantica prevede che si esegua il prodotto cartesiano tra tutte le tabelle e successivamente si applichi una selezione basata sulla condizione specificata nella clausola WHERE.
- Non è richiesto che le tabelle abbiano schemi disgiunti.
- Se ci sono attributi con lo stesso nome in tabelle diverse, per evitare ambiguità, è possibile premettere al nome dell'attributo il nome della tabella (o l'alias) come segue:

<NomeTabella>.<NomeAttributo>

Clausola FROM: esempi

12

TRENO(NumTreno, Cat, Part, Arrivo, Dest)

FERMATA(NumTreno, Stazione, Orario)

```
SELECT T.NumTreno, F.Stazione
```

```
FROM TRENO AS T, FERMATA AS F
```

```
WHERE T.NumTreno = F.NumTreno
```

Clausola WHERE

13

Clausola WHERE

WHERE <ListaCondizioni>

dove

<ListaCondizioni> è un'espressione booleana ottenuta combinando condizioni semplici **<Cond>** con i connettivi logici **AND, OR e NOT:**

<Cond>:

<espr> θ <espr>

<espr> θ <const>

dove:

- $\theta \in \{=, <>, <, >, <=, >=\}$.
- **<espr>**: espressione che contiene riferimenti agli attributi delle tabelle della clausola **FROM**.
- **<const>**: valore dei domini elementari.

Clausola WHERE: esempi

14

TRENO(NumTreno, Cat, Part, Arrivo, Dest)

FERMATA(NumTreno, Stazione, Orario)

```
SELECT NumTreno, Cat, Part
```

```
FROM TRENO
```

```
WHERE (Cat = 'IC' OR Cat = 'ES')
```

```
AND Part > '12.00'
```

Interpretazione algebrica

15

Interpretazione algebrica di un'interrogazione SQL

Caso 1: una sola tabella nella clausola FROM

SELECT A₁, ..., A_n

FROM T

WHERE cond

$$\Pi_{\{A_1, \dots, A_n\}}(\sigma_{\text{cond}}(T))$$

Interpretazione algebrica

16

Caso 2: più di una tabella nella clausola FROM

SELECT A₁, ..., A_n

FROM T₁, ... T_m

WHERE cond

**prodotto
cartesiano**

$\Pi_{\{A_1, \dots, A_n\}}(\sigma_{\text{cond}}(T_1 \bowtie \dots \bowtie T_m))$

Variabili tupla

17

Uso delle variabili tupla (Alias di tabella)

Vengono usate con due obiettivi:

- **per risolvere l'ambiguità sui nomi degli attributi**
- **per gestire il riferimento a più esemplari della stessa tabella (in questo caso aggiungono potere espressivo al linguaggio)**

Variabili tupla: esempi

18

Esempio

Trovare il numero e l'orario di partenza dei treni ES che hanno la stessa destinazione di un treno IC.

TRENO(NumTreno, Cat, Part, Arrivo, Dest)

FERMATA(NumTreno, Stazione, Orario)

SELECT T1.NumTreno, T1.Part

FROM TRENO T1, Treno T2

WHERE T1.Cat='ES' AND T2.Cat='IC'

AND T1.Dest = T2.Dest

Clausola ORDER BY

19

Ordinamento (clausola ORDER BY)

```
ORDER BY <Attributo> [<ASC|DESC>]  
        {,<Attributo> [<ASC|DESC>]}
```

permette di specificare un'insieme di attributi da utilizzare per ordinare le tuple della relazione risultato.

Clausola ORDER BY: esempi

20

Esempio

Trovare il numero e l'orario di partenza dei treni ES ordinati per orario di partenza

TRENO(NumTreno, Cat, Part, Arrivo, Dest)

FERMATA(NumTreno, Stazione, Orario)

SELECT NumTreno, Part

FROM TRENO

WHERE Cat='ES'

ORDER BY Part