SQL – prima parte

DOCENTE
PROF. ALBERTO BELUSSI

Anno accademico 2018/'19

Il concetto di vista

DEFINIZIONE

È una relazione derivata. Si specifica l'espressione che genera il suo contenuto. Esso quindi dipende dalle 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

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 privatezza dei dati
- Permettono di memorizzare nel DBMS interrogazioni complesse condivise
- Sono utili per rendere l’interfaccia delle applicazioni indipendente dallo schema logico (INDIPENDENZA LOGICA)

Il concetto di vista

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 materializzata è necessario ricalcolarne il contenuto ogni volta che le relazioni da cui dipende vengono aggiornate.

SQL

SQL (Structured Query Language) è stato definito negli anni ’70; è stato standardizzato negli anni ‘80 e ‘90 ed è oggi il linguaggio più diffuso per i DBMS relazionali.

L’SQL è un linguaggio di *interrogazione dichiarativo*. La sua semantica fa riferimento al CALCOLO RELAZIONALE.

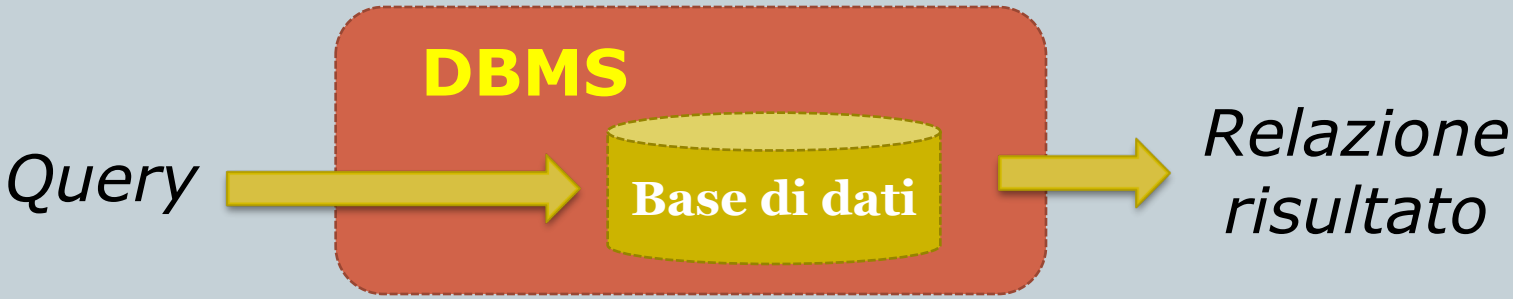
Poiché quest’ultimo linguaggio basato sulla logica non è parte del programma del corso, la semantica di SQL verrà presentata in modo informale e quando possibile verrà assegnata utilizzando espressioni dell’algebra relazionale.

SQL

FORMA BASE di una interrogazione SQL (o QUERY)

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

[...] indica una parte opzionale



SQL

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

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

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 nella relazione risultato delle tuple duplicate

Clausola SELECT: esempi

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

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

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

Clausola FROM

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

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

Clausola WHERE

WHERE <ListaCondizioni>

<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:

- **θ ∈ {=, <>, <, >, <=, >=}.**
- **<espr>: espressione che contiene riferimenti agli attributi delle tabelle che compaiono nella clausola FROM.**
- **<const>: valore dei domini di base compatibile con il tipo dell'espressione <espr>.**

Clausola WHERE: esempi

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

FERMATA(NumTreno, Stazione, Orario)

```
SELECT  NumTreno, Cat, Part
FROM  TRENO
WHERE (Cat ='FB' OR Cat='FR')
            AND Part > '12.00'
```

Interpretazione algebrica

Interpretazione algebrica di un'interrogazione SQL

Caso 1: una sola tabella nella clausola FROM

SELECT A1, …, An
FROM T
WHERE cond

equivale alla seguente espressione
in algebra relazionale

$\Pi_{\{A1, \ldots, An\}} (\sigma_{cond}(T))$

Interpretazione algebrica

Caso 2: più di una tabella nella clausola FROM

SELECT A1, …, An

FROM T1, … Tm

WHERE cond

equivale alla seguente espressione in algebra relazionale

$$\Pi_{\{A1, \ldots, An\}}(\sigma_{cond}(T1 \bowtie \ldots \bowtie Tm))$$

prodotto cartesiano (nessun attributo comune)

Variabili tupla

Uso delle variabili tupla (o 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

Esempio

Trovare il numero e l'orario di partenza dei treni 'FB' che hanno la stessa destinazione di almeno un treno 'Reg'.

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

```
SELECT T1.NumTreno, T1.Part
FROM TRENO T1, TRENO T2
WHERE T1.Cat='FB' AND T2.Cat='Reg'
        AND T1.Dest = T2.Dest
```

Clausola ORDER BY

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.

ASC e DESC indicano se l'ordinamento va fatto in modo crescente o decrescente. Il default è ASC.

Clausola ORDER BY: esempi

Esempio

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

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

```
SELECT NumTreno, Part
FROM TRENO
WHERE Cat='FR'
ORDER BY Part
```