Lezione 11

SQL: DML avanzato

Gli operatori di aggregazione

- Per ***funzioni aggregate*** si intende
  - un insieme di funzioni che permettono di operare su *collezioni di valori della base di dati*

- SQL supporta cinque diverse funzioni di aggregazione
  - possono essere applicate ad un qualsiasi attributo di una relazione
    - COUNT()
      - restituisce il numero di tuple o di valori presenti nel risultato di una interrogazione
    - SUM(), MAX(), MIN(), AVG()
      - restituiscono la somma, il massimo, il minimo e la media aritmetica dei valori numerici selezionati

COUNT

count (<* | [distinct | all] listaattributi)

- Esempi

select count(*) from Impiegati

conta quante tuple sono presenti nella relazione impiegati

select count(all nome,cognome) from Impiegati

conta quante tuple sono presenti nella relazione impiegati con nome e cognome

select count(distinct nome,cognome) from Impiegati

conta quante tuple sono presenti nella relazione impiegati con nomi e cognomi differenti

Operatori Aggregati

<sum | avg | max | min> ([distinct | all] Attr)

- sum e avg richiedono che l'attributo sia numerico o di tipo intervallo temporale
- max e min possono riferirsi a qualsiasi attributo sul cui dominio sia definito un ordinamento

Esempio:

select max(Stipendio) from Impiegato where eta<35

Errore:

select matricola, max(Stipendio) from Impiegato where eta<35

Raggruppamenti

- In alcune applicazioni sorge l'esigenza di operare su gruppi di tuple con una caratterista in comune

- Le operazioni di raggruppamento si ottengono con:
  - GROUP BY *ElencoDiAttributi*:
    - per raggruppare le tuple della relazione in sottoinsiemi caratterizzati dallo stesso valore degli attributi che compaiono come argomento della clausola stessa SELECT
  - HAVING (*Condizioni*)
    - per specificare delle condizioni logiche (predicati) che devono essere verificate sul sottoinsieme di tuple precedentemente raggruppate con la clausola GROUP BY

GROUP BY

SELECT lista_attributi FROM nome_tab
GROUP BY lista_attributi_group

Si noti che lista_attributi deve coincidere o contenere lista_attributi_group (in più può avere operatori di aggregazione)

Esempi:

SELECT cognome FROM impiegati GROUP BY cognome

raggruppa gli impiegati che hanno lo stesso cognome

SELECT dipartimento, cognome, nome FROM impiegati
WHERE mansione='direttore'
GROUP BY dipartimento, cognome, nome

raggruppa i direttori per dipartimento

Funzionamento del GROUP BY

SELECT dipart FROM Impiegati GROUP BY dipart

- Step 1:
  si esegue

  SELECT dipart FROM Impiegati
- Step 2:
  Le righe della tabella ottenuta sono analizzate e raggruppate in sottoinsiemi in funzione dell'attributo di group by,
  - ossia per i valori di dipart
- Attenzione: il select può avere un where che interviene per produrre la tabella prima del raggruppamento

HAVING

```
SELECT lista_attributi FROM nome_tab
        WHERE condizione_select
        GROUP BY lista_attributi_group
        HAVING condizione_gruppo
```

- Mentre GROU BY organizza i gruppi HAVING seleziona quelli che soddisfano la condizione
- Differenze tra where e having
  - where seleziona le tuple prima del raggruppamento
  - having seleziona i gruppi

Condizioni di raggruppamento

- In una interrogazione SQL contenente la clausola GROUP BY,
  - un attributo può comparire come argomento della clausola SELECT solo se è presente nell'elenco degli attributi della clausola GROUP BY

- NON CONFONDERE WHERE con HAVING.
  - Una semplice regola da seguire è la seguente
    - se le condizioni sono da verificare a livello delle singole tuple, allora si usa la clausola WHERE
    - se le condizioni sono da verificare su un gruppo, si usa la clausola HAVING

Esempio

```
SELECT dipart, SUM(Stip) FROM Impiegati
        GROUP BY dipart
        HAVING SUM(Stip) >= 20
```

- GROUP BY organizza per dipartimenti
- SUM somma gli stipendi per ogni dipartimento
- HAVING mostra solo quei dipartimenti che soddisfano la condizione

Azione combinata tra aggregatori e raggruppamenti

Operatori insiemistici

- In SQL si possono combinare i risultati di due query con gli operatori insiemistici
  - UNION
  - INTERSECT
  - EXCEPT
- si usano per determinare rispettivamente
  - l'unione, l'intersezione e la differenza sulle tabelle prodotte per effetto delle due query

Caratteristiche

- Sono operazioni binarie: si applicano a due relazioni
- Le due relazioni su cui è eseguita ognuna delle operazioni deve avere lo stesso tipo di tuple:
  - questa condizione è detta compatibilità all'unione
    - Si dice che due relazioni $R(A_1, A_2, , A_n)$ e $S(B_1, B_2, , B_n)$ sono compatibili all'unione se
      - hanno lo stesso grado n
      - e se $dom(A_i) = dom(B_i)$ per $1 \leq i \leq n$
- Ciò significa che le due relazioni hanno lo stesso numero di attributi e che ogni coppia di attributi corrispondenti è dello stesso dominio

Note

- INTERSECT, UNION e EXCEPT
  - eliminano i duplicati nel risultato
- Le versioni UNION ALL, INTERSECT ALL e EXCEPT ALL
  - consentono invece di mantenere i duplicati nel risultato

Esempio

```
SELECT cognome FROM impiegati
        UNION
SELECT cognome FROM dirigenti
```

- Genera un unico elenco di tutti i cognomi diversi prendendoli dalle due tabelle

```
SELECT cognome FROM impiegati
        INTERSET
SELECT cognome FROM dirigenti
```

- Genera un unico elenco di tutti i cognomi presenti contemporaneamente in entrambe le tabelle

```
SELECT cognome FROM impiegati
        EXCEPT
SELECT cognome FROM dirigenti
```

- Genera un unico elenco di tutti i cognomi della prima tabella che non sono presenti anche nella seconda

Interrogazioni nidificate

- Un'interrogazione può contenere al suo interno un'altra interrogazione
  - Tipicamente la interrogazione nidificata è contenuta nella clausola WHERE
    - ma sottointerrogazioni possono comparire anche nelle clausole FROM e HAVING
- Gli operatori che consentono la nidificazione sono:
  - IN e NOT IN
  - EXISTS e NOT EXISTS
  - UNIQUE e NOT UNIQUE
  - ANY
  - ALL

Nota

- Gli operatori consentono di confrontare un valore con il risultato di una select

```
SELECT * FROM impiegato
        WHERE dipart =   ANY(SELECT nome from dipartimento
                              WHERE citta = 'Napoli')
SELECT * FROM impiegato
        WHERE dipart        IN(SELECT nome from dipartimento
                              WHERE citta = 'Napoli')
SELECT nome FROM impiegato
        WHERE          EXISTS(SELECT nome from dipartimento
                              WHERE citta='Napoli')
```

Interpretazioni

- IN
  - verifica se il valore è presente nell'elenco generato dalla query nidificata
- EXISTS
  - ritorna TRUE se l'elenco generato non è vuoto
- UNIQUE
  - restituisce TRUE se non ci sono tuple duplicate nell'elenco generato
- ANY
  - restituisce TRUE se il confronto è vero per una qualunque delle tuple dell'elenco generato
- ALL
  - restituisce TRUE se il confronto è vero per tutte le tuple dell'elenco generato

Si noti che

- **IN** equivale a **=ANY** e **NOT IN** a **<>ALL**
- per EXISTS e UNIQUE la query nidificata deve riportare proprietà legate ai valori prodotti dalla query esterna

Nota

- In generale la query nidificata deve essere calcolata per ogni tupla della tabella coinvolta nella select principale
  - Se la sotto-interrogazione è indipendente dalla select principale, allora viene determinata una sola volta
  - Se invece dipende, allora deve essere ricalcolata per ogni tupla

L’operazione di DIVISIONE

- L’operazione di DIVISIONE si applica a due relazioni R(Z) e S(X) in cui X è contenuto in Z
- Sia allora Y l’insieme degli attributi di R che non sono attributi di S, cioè Y = Z − X
- Il risultato della divisione è una relazione T(Y) che comprende una tupla t se in R sono presenti tuple con $t_R[Y] = t$ e con $t_R[X] = t_s$ per ogni tupla t_s di S
- Perché una tupla compaia nel risultato, in R devono comparire i valori di t in combinazione con ogni tupla di S

In sintesi

- La divisione ($R_1(Z) \div R_2(Y)$) produce una relazione R(X)
  - che contiene tutte le tuple t[X] di $R_1(Z)$
  - che in R_1 si presentano in combinazione con ogni tupla di $R_2(Y)$ dove $Z=X \cup Y$

- La divisione relazionale è un'operazione che date due relazioni r_1 ed r_2 , con r_1 su $R_1(X_1X_2)$ e r_2 su $R_2(X_2)$ la divisione è (il più grande) insieme di tuple aventi schema X_1 tale che, facendo il prodotto Cartesiano con r_2, ciò che si ottiene è una relazione contenuta in r_1

A cosa serve

- Consente di esprimere query del tipo
  - *Sedi in cui sono presenti tutti i ruoli (che* equivale a *Sedi in cui non esiste un ruolo non presente)*
  - *Date in cui ci sono voli per tutte le linee aeree*
  - *Clienti che hanno prenotato tutte le stanze di un albergo*

Esempio 1

- Relazioni:
  - **Impiegato (Matricola, Nome, Sede, Ruolo, Stipendio)**
  - **Sedi (Sede, Responsabile, Città)**
  - **Progetti (Codice, Città)**
- Query: Trovare le sedi in cui sono presenti tutti i ruoli

```
SELECT Sede FROM Sedi S WHERE
        NOT EXISTS (SELECT * FROM Impiegato I1
                        WHERE NOT EXISTS (SELECT * FROM Impiegato I2
                                                WHERE S.Sede = I2.Sede AND I1.Ruolo=I2.Ruolo))
```

  - Il blocco più interno viene valutato per ogni combinazione di S e I1
  - Il blocco intermedio funge da “divisore” (per il coinvolgimento di I1.Ruolo)
  - Data una sede S, se in S manca un ruolo:
    - la subquery più interna non restituisce nulla
    - quindi la subquery intermedia restituisce almeno una tupla
    - quindi la clausola WHERE non è soddisfatta per S

Esempio 1

Impiegato (Matricola, Nome, Sede, Ruolo, Stipendio)
Sedi (Sede, Responsabile, Città)
Progetti (Codice, Città)

- Trovare le sedi in cui sono presenti tutti i ruoli

```
SELECT Sede FROM Sedi S
    WHERE NOT EXISTS
        (SELECT * FROM Impiegato I1
            WHERE NOT EXISTS
                (SELECT * FROM Impiegato I2
                    WHERE S.Sede = I2.Sede
                    AND  I1.Ruolo=I2.Ruolo))
```

- Il blocco più interno viene valutato per ogni combinazione di S e I1
- Il blocco intermedio funge da “divisore” (per il coinvolgimento di I1.Ruolo)
- Data una sede S, se in S manca un ruolo:
  - la subquery più interna non restituisce nulla
  - quindi la subquery intermedia restituisce almeno una tupla
  - quindi la clausola WHERE non è soddisfatta per S

Esempio 2

Clienti (CF, Nome, Cognome, Email)
Camere (Id, Tipo, Piano)
Prenotazioni (ID, CF, IDCamera, Data, Durata)

- Trovare i nomi dei clienti che hanno prenotato tutti i tipi di camera

```
SELECT CL.nome FROM Clienti C
    WHERE NOT EXISTS
            (SELECT CA.Id  FROM Camere CA
                    WHERE NOT EXISTS
                            (SELECT P.ID FROM Prenotazioni P
                                    WHERE P.IDCamera = CA.Id
                                            AND P.CF = C.CF))
```

Esempio 2 (USANDO EXCEPT)

Clienti (CF, Nome, Cognome, Email)
Camere (Id, Tipo, Piano)
Prenotazioni (ID, CF, IDCamera, Data, Durata)

- Trovare i nomi dei clienti che hanno prenotato tutti i tipi di camera

```
SELECT CL.nome FROM Clienti C
        WHERE NOT EXISTS
                (SELECT CA.Id  FROM Camere CA
                 EXCEPT
                 SELECT P.IDCamera FROM Prenotazioni P
                        WHERE P.CF = C.CF)
```