cefriel

Consorzio per la formazione e la ricerca in Ingegneria dell'Informazione

Structured Query Language *parte 2*

Come interrogare BENE una base di dati relazionale (e vivere felici)

17 Ottobre 2003

Gennaro Pepe
gennaro.pepe@cefriel.it

Informatica Applicata

Query di join(1)

- Talvolta i dati richiesti sono una combinazione di quelli presenti in più di una tabella
- Altre volte i dati richiesti sono collegati a quelli presenti su altre tabelle
- Esempio :

clienti

Cod_fisc	Nome	Cognome

ordini

Cod_fisc	Importo	Quantità	Cod_prod

  - Se si vuole conoscere nome e cognome dei clienti che hanno fatto ordini superiori ad un certo importo occorre collegare queste due tabelle

- L'operatore che consente di combinare i dati di due o più tabelle è la virgola nella clausola from

Structured Query Language - parte 2 - 2 - *Basi di Dati*

Query di join(2)

- Il join permette di eseguire l'operazione di 'prodotto cartesiano' tra tabelle. Il risultato è una tabella con tutte le possibili combinazioni dei record delle due tabelle

Tab1

D	E
D1	E1
D2	E2

JOIN

Tab2

A	B	C
A1	B1	C1
A2	B2	C2
A3	B3	C3

=

D	E	A	B	C
D1	E1	A1	B1	C1
D1	E1	A2	B2	C2
D1	E1	A3	B3	C3
D2	E2	A1	B1	C1
D2	E2	A2	B2	C2
D2	E2	A3	B3	C3

Attraverso il comando di ***select*** è possibile:
- effettuare una proiezione su una una parte dei campi
- indicare una condizione che deve essere verificata da tutte le tuple

Structured Query Language - parte 2 - 3 - *Basi di Dati*

Query di join – condizioni

- Il join permette di specificare direttamente nella clausola **from** quelle condizioni legate alla corrispondenza dei valori di campi delle tabelle che si vuole combinare: solo se tali valori si corrispondono i record corrispondenti entrano nella combinazione

Tab1

A	B	C
A1	B1	C1
A2	B2	C2
A3	B3	C3

JOIN

Tab2

D	E	C
D1	E1	C1
D2	E2	C3
D3	E3	C4
D4	E4	C6

ON Tab1.C=Tab2.C

A	B	C	D	E
A1	B1	C1	D1	E1
A3	B3	C3	D2	E2

Structured Query Language - parte 2 - 4 - *Basi di Dati*

Query di join – es.

clienti

Cod_fisc	Nome	Cognome
cod1	Marco	Pace
cod3	Sara	Gelo
cod2	Luca	Neri

ordini

Cod_fisc	Importo	Cod_Prod	Quantità
cod1	200	A1	2
cod2	10	B54	4
cod2	100	S32	5

```
select clienti.*,Importo
from clienti, ordini
where clienti.Cod_fisc=Ordini.Cod_fisc;
```

Individuare i clienti e l'importo dei relativi ordini

Cod_fisc	Nome	Cognome	Importo
cod1	Marco	Pace	200
cod2	Luca	Neri	10
cod2	Luca	Neri	100

Structured Query Language - parte 2 - 5 - *Basi di Dati*

Query di join – es.

Azienda

IDAzienda	Fatturato	Rag Soc
A1	100	SRL
A2	200	SPA
A3	150	SNC

Dipendente

IDDipend.	Nome	Azienda
D1	Pippo	A1
D2	Pluto	A3
D3	Paperino	A1
D4	Topolino	A1
D5	Minni	A2

Equivalentemente:

```
SELECT Nome, Azienda, Fatturato
FROM Azienda JOIN Dipendente ON
          Azienda.IDAzienda = Dipendente.Azienda
```

- In una query con JOIN sono coinvolte più tabelle
  - Può capitare che ci siano dei campi con nomi uguali
- Per non fare confusione si usa il . (dot notation)

per specificare a quale tabella appartiene ogni campo

Structured Query Language - parte 2 - 6 - *Basi di Dati*

Tipi di join

- Il tipo di join ora considerato è l'**INNER JOIN**: esso permette di includere nella tabella risultato solo i record che si corrispondono come specificato dalla condizione ON su un campo di ciascuno dei record
- Altri tipi di JOIN sono gli OUTER JOIN: essi permettono di includere nella combinazione anche dei record che non hanno record corrispondenti nell'altra tabella secondo la condizione ON
  - **LEFT OUTER JOIN**
    Permette di includere nella combinazione tutti i record della prima tabella e quelli della seconda che soddisfano la condizione
  - **RIGHT OUTER JOIN**
    Permette di includere nella combinazione tutti i record della seconda tabella e quelli della prima che corrispondono

Structured Query Language - parte 2 - 7 - *Basi di Dati*

Outer join – es.(1)

clienti

Cod_fisc	Nome	Cognome
cod1	Marco	Pace
cod3	Sara	Gelo
cod2	Luca	Neri

ordini

Cod_fisc_cliente	Importo	Cod_Prod	Quantità
cod1	200	A1	2
cod2	10	B54	4
cod2	30	S32	5

```
select clienti.*, Cod_Prod
from clienti left outer join ordini on
clienti.Cod_fisc=ordini.Cod_fisc_cliente
```

Cod_fisc	Nome	Cognome	Cod_Prod
cod1	Marco	Pace	A1
cod2	Luca	Neri	S32
cod2	Luca	Neri	B54
cod3	Sara	Gelo	

Structured Query Language - parte 2 - 8 - *Basi di Dati*

Outer join – es.(2)

clienti

Cod_fisc	Nome	Cognome
cod1	Marco	Pace
cod3	Sara	Gelo
cod2	Luca	Neri

ordini

Cod_fisc_cliente	Importo	Cod_Prod	Quantità
cod1	200	A1	2
cod2	10	B54	4
cod4	30	S32	5

```
select clienti.Nome, clienti.Cognome,
ordini.Cod_fisc_cliente as Cod_fisc, ordini.Importo
from clienti right outer join ordini on
clienti.Cod_fisc=ordini.Cod_fisc_cliente
```

Nome	Cognome	Cod_fisc	Importo
Marco	Pace	cod1	200
Luca	Neri	cod2	10
		cod4	30

Self joins

- E se si volesse fare un JOIN di una tabella con se stessa ?
  - Ex: occorre effettuare delle operazioni di selezione su una tabella sulla base di valori ricavati da altre operazioni di selezione sulla stessa tabella
- Due soluzioni:
  - Utilizzo di query innestate
  - Utilizzo di alias per la tabella
    - La stessa tabella si chiama in 2 o più modi differenti nella stessa query

Self joins – es.(1)

Nome	Stipendio	Impiego
Angelo	3	Ricercatore
Marco	2,5	Ricercatore
Luca	4,2	Capo area
Martina	4	Capo area

Vogliamo conoscere gli stipendi di tutti gli impiegati che fanno lo stesso lavoro di Angelo

Vogliamo ottenere questo risultato:

p1.Nome	p1.Stipendio
Angelo	3
Marco	2,5

Dobbiamo confrontare i campi di un record con i campi di un altro record DELLA STESSA TABELLA

Possiamo Facciamo un Self Join…

Structured Query Language - parte 2 - 11 - *Basi di Dati*

Self joins – es.(2)

P1.Nome	*P1.Stipendio*	*P1.Impiego*	*P2.Nome*	*P2.Stipendio*	*P2.Impiego*
Angelo	3	Ricercatore	Angelo	3	Ricercatore
Angelo	3	Ricercatore	Marco	2,5	Ricercatore
Angelo	3	Ricercatore	Luca	4,2	Capo area
Angelo	3	Ricercatore	Martina	4	Capo area
Marco	2,5	Ricercatore	Angelo	3	Ricercatore
Marco	2,5	Ricercatore	Marco	2,5	Ricercatore
Marco	2,5	Ricercatore	Luca	4,2	Capo area
Marco	2,5	Ricercatore	Martina	4	Capo area
Luca	4,2	Capo area	Angelo	3	Ricercatore
Luca	4,2	Capo area	Marco	2,5	Ricercatore
Luca	4,2	Capo area	Luca	4,2	Capo area
Luca	4,2	Capo area	Martina	4	Capo area
Martina	4	Capo area	Angelo	3	Ricercatore
..	..	..	..	..	..

Structured Query Language - parte 2 - 12 - *Basi di Dati*

Self joins – es.(3)

Nome	Stipendio	Impiego
Angelo	3	Ricercatore
Marco	2,5	Ricercatore
Luca	4,2	Capo area
Martina	4	Capo area

Se selezionano solo i Record che hanno i valori delle colonne **Impiego** uguali

```
SELECT p2.Nome, p2.Stipendio
FROM Persona AS p1 INNER JOIN Persona AS p2
                   ON p1.Impiego = p2.Impiego
WHERE p1.Nome="Angelo"
```

p1.Nome	p1.Stipendio
Angelo	3
Marco	2,5

Esercizi

Aereoporto (Città, Paese, NumeroDiPiste)
Volo (FlightID, Giorno, CittàPart, OraPart, CittàArr, OraArr, TipoAereo)
Aereo (TipoAereo, NumeroPasseggeri)

- Dato il precedente schema scrivere le seguenti query in SQL:
  1. Il paese di arrivo del volo AZ274
  2. Il tipo di aereo e il numero di passeggeri dei voli in partenza da Boston; se non ci sono le informazioni dell'aereo, visualizzare solo il tipo

Soluzione

1.1
```
select Aereoporto.Paese
from Aereoporto, Volo
where (FlightID="AZ274") AND
      (CittàArr=Città)
```

1.2
```
select Aereoporto.Paese
from Aereoporto
     inner join Volo on
CittàArr=Città
where (FlightID="AZ274")
```

2.
```
select Volo.TipoAereo,
    Aereo.NumeroPasseggeri
from Volo left join Aereo on

Volo.TipoAereo=Aereo.TipoAereo
where Volo.CittàPart="Boston"
```


Esercizio

CD (CDNumber, Title, Year, Price)
Track (CDNumber, PerformanceCode, trackNo)
Recording (Performance, SongTitle, Year)
Composer (CompName, SongTitle)
Singer (SingerName, PerformanceCode)

- Dato il precedente schema scrivere le seguenti query in SQL:
  1. Le persone che hanno scritto e suonato canzoni il cui nome inizia per “D”
  2. I titoli dei CD che contengono canzoni il cui anno di registrazione non è conosciuto
  3. Le tracce dei CD con numero di serie 78574. Fornirle in ordine di numero indicando il cantante per le tracce per cui è definito.

Soluzione (1)

1.1

```
select SingerName
from Singer, Recording, Composer
where (Singer.PerformanceCode=Recording.Performance) AND
        (Recording.SongTitle=Composer.SongTitle) AND
        (SingerName=CompName) AND (SongTitle like "D*")
```

1.2

```
select SingerName
from (Singer join Recording on
        Singer.PerformanceCode=Recording.Performance),
         join Composer on Recording.SongTitle=Composer.SongTitle
where (SingerName=CompName) AND (SongTitle like "D*")
```

Structured Query Language - parte 2 - 17 - *Basi di Dati*

Soluzione (2)

2.

```
select CD.title
from CD join Track on CD.CDNumber=Track.CDNumber
     join Recording on Tack.PerformanceCode=Recording.Performance
where Recording.Year is null
```

3.

```
select trackNo,Singer.SingerName
from Track left join Singer on
        Track.PerformanceCode=Singer.PerformanceCode
where Track.CDNumber=78754
order by trackNo
```

Structured Query Language - parte 2 - 18 - *Basi di Dati*

Funzioni aggregate(1)

- Spesso si ha bisogno di informazioni che concernono il complesso dei dati e non singole tuple
- In tali casi calcolare tali dati complessivi a partire dai singoli sarebbe un dispendio di risorse
- SQL mette a disposizione 5 funzioni per fare ciò in modo semplice:

AVG(campox)	valore medio dei valori di campox
MAX(campox)	massimo valore assunto da campox
MIN(campox)	minimo valore assunto da campox
SUM(campox)	somma dei valori della colonna 'campox'
COUNT(campox)	numero di righe nella colonna 'campox'

Funzioni aggregate(2)

- A parte "count", le altre funzioni ammettono come argomento una espressione
  - **sum** e **avg**
    - Attributo o campo calcolato (numeri o intervalli di tempo)
  - **min** e **max**
    - Attributo o espressione il cui risultato sia ordinabile (anche per le stringhe di cartatteri)
- La presenza di una condizione WHERE nella query naturalmente limita il numero di righe su cui sono calcolate le funzioni
- Per fare calcoli sui valori distinti si può utilizzare l'operatore **distinct**
  - es.: `AVG(DISTINCT campox)`
- Esistono altre operazioni possibili che dipendono dal tipo di DBMS che si sta utilizzando

Funzioni aggregate - es.(1)

persone

Nome	Compenso_ mensile	Impiego	Mesi_ di_lavoro
Angelo	3	ricercatore	3
Marco	2,5	ricercatore	5
Martina	4	capo area	6

Estrarre il numero degli impiegati

Estrarre il totale delle spese per i ricercatori

```
select count(*) as numimpiegati
from persone;
```

→ Numimpiegati 3

```
select sum(Compenso_mensile*Mesi_di_lavoro) as spese_ric
from persone
where Impiego= 'ricercatore';
```

→ spese_ric 21,5

Funzioni aggregate - es.(2)

persone

Nome	Compenso_ mensile	Impiego	Mesi_di_ lavoro
Angelo	3	ricercatore	3
Marco	2,5	ricercatore	5
Martina	4	capo area	6

Estrarre lo stipendio medio delle persone

Estrarre lo stipendio massimo

```
select avg(Compenso_mensile) as stipendio_medio
from persone;
```

→ stipendio_medio 31,6

```
select max(Compenso_mensile) as tetto_stipendi
from persone;
```

→ tetto_stipendi 4

Funzioni aggregate - es.(3)

- select Cognome, Nome, max (stipendio)
  from Impiegato
  where Dipartimento = NomeDip
  and Città = “Milano”
- Cosa fa questa operazione?
  - Problema 1: non è detto che Max dia un solo valore come risultato
  - Problema 2: non funziona per gli altri operatori
- Per questo motivo l’SQL non consente di mischiare fra i campi della select attibuti e funzioni aggregate a meno di non usare la clausola GROUP BY

Raggruppamento di dati

- Il raggruppamento di record consente di dividere le tabelle (originarie o risultati di operazioni di selezione) in gruppi logici distinti
- L’operatore usato è **GROUP BY**
- Sui gruppi così creati è possibile poi effettuare operazioni di calcolo applicando le funzioni aggregate

```
select Campo1, func(Campo2)
from Tabella
where Campo1 [cond]
group by Campo1
```

Nella select posso specificare solo campi presenti nel raggruppamento

- il raggruppamento può avvenire su uno o più campi
- se avviene su più campi il risultato sarà una tabella con un numero di righe pari al prodotto dei campi su cui è effettuato il raggruppamento

Raggruppamento di dati – es.

persone

Nome	Compenso_ mensile	Impiego	Mesi_di_ lavoro
Angelo	3	ricercatore	3
Marco	2,5	ricercatore	5
Luca	4,2	capo area	10
Martina	4	capo area	6

Estrarre lo stipendio medio di ogni tipologia di impiego

```
select avg(Compenso_mensile) as Paga_media, Impiego
from persone
group by Impiego;
```

Paga_media	Impiego
2,75	ricercatore
4,1	capo area

Filtri su raggruppamenti

- Per imporre delle condizioni sui valori di certe grandezze caratteristiche di un raggruppamento si può usare il comando **HAVING**

Estrarre lo stipendio medio di ogni Tipologia di impiego se superiore a 3 milioni

```
select avg(Compenso_mensile) as Paga_media, Impiego
from persone
group by Impiego
having avg(Compenso mensile)>3;
```

Paga_media	Impiego
4,1	capo area

Esercizio

Aereoporto (Città, Paese, NumeroDiPiste)
Volo (FlightID, Giorno, CittàPart, OraPart, CittàArr, OraArr, TipoAereo)
Aereo (TipoAereo, NumeroPasseggeri)

- Dato il precedente schema scrivere le seguenti query in SQL:
  1. Il numero di voli che lasciano Parigi il Giovedì
  2. Il numero di voli internazionali che lasciano città canadesi
  3. Le città francesi dalle quali partono più di 20 voli diretti verso la germania

Soluzioni (1)

1.
```
select count (*)
from Volo
where Giorno="Giovedì" and CittàPart="Parigi"
```

2.
```
select count(*)
from Aereporto as A1 join Volo on A1.Città = Volo.CittàPart
    join Aeroporto as A2 on Volo.CittàArr = A2.Città
where A1.Paese = "Canada" and A2.Paese <> "Canada"
```

3.
```
select A1.città
from Aereporto as A1 join Volo on A1.Città = Volo.CittàPart
    join Aeroporto as A2 on Volo.CittàArr = A2.Città
where A1.Paese = "Francia" and A2.Paese = "Germania"
group by A1.Città
having count(*)>20
```


Query innestate

- SQL permette di realizzare query in cui i valori utilizzati per specificare le condizioni sono ricavati direttamente da altre query
  - L'interrogazione viene specificata direttamente nel predicato all'interno della clausola **where**
- Caso tipico: confronto di un attributo con il risultato di una query. Come procedo?
  - Devo estendere i normali operatori di confronto (=,<>,<,>,<=.>=) con due nuovi operatori
  - **ANY**: la riga soddisfa la condizione se risulta vero il confronto tra l'attributo della riga stessa e **almeno uno** degli elementi dell'interrogazione nidificata
  - **ALL**: come sopra ma per **tutti** gli elementi

Query innestate – es.(1)

clienti

Cod_fisc	Nome	Cognome
cod1	Marco	Pace
cod3	Sara	Gelo
cod2	Luca	Neri

ordini

Cod_fisc_cliente	Importo	Cod_Prod	Quantità
cod1	200	A1	2
cod2	10	B54	4
cod4	30	S32	5

Estrarre i clienti che hanno fatto ordini con quantità maggiore di 3

```
select clienti.*
from clienti
where Cod_fisc = ANY
(select Cod_fisc_cliente from ordini where Quantità>3);
```

Cod_fisc	Nome	Cognome
cod2	Luca	Neri

Query innestate – es.(2)

Dipartimento

Nome	Num_Dip

Persone

Codice	Nome	Cognome	Dipartimento

Trovare i dipartimenti in cui non lavorano persone con Nome “Angelo”

select D.Nome
from Dipartimento as D **where** D.Nome <> **ALL**
(select Dipartimento from Persone as P
where P.Nome = “Angelo”)

- Per esprimere appartenenza ed esclusione ci sono anche altri due operatori (già visti in precedenza...)
  - **IN** → **= ANY**
  - **NOT IN** → **<> ALL**

Structured Query Language - parte 2 - 31 - *Basi di Dati*

Query innestate – es.(3)

Persone

Codice	Nome	Cognome	Dipartimento	Stipendio

Trovare il dipartimento dell’impiegato che guadagna di più

select Dipartimento
from Persone **where** Stipendio >= **ANY**
(select Stipendio from Persone)

select Dipartimento
from Persone **where** Stipendio =
(select max(Stipendio) from Persone)

Structured Query Language - parte 2 - 32 - *Basi di Dati*

Query innestate complesse (1)

- **EXISTS** e **NOT EXISTS**
  - Operatore che ammette come parametro una interrogazione nidificata e dà valore vero solo se l'interrogazione fornisce un risultato non vuoto

Persone

Codice_fiscale	Nome	Cognome

Trovare le persone che hanno omonimi

```
select *
from Persone as P1 where exists
        (select * from Persone as P2
        where P1.Nome = P2.Nome and
                P1.Cognome = P2.Cognome and
                P1.Codice_Fiscale <> P2.Codice_Fiscale)
```


Query innestate complesse (2)

- L'esempio precedente mostra che
  - Non si può sempre pensare che l'esecuzione delle interrogazioni nidificate avvenga eseguendo prima la più interna e poi la più esterna
  - Quando l'interrogazione nidificata usa variabili di quella esterna ciò non è più vero
    - Si chiama PASSAGGIO DI BINDING da un contesto ad un altro
  - In questi casi l'esecuzione è la seguente
    - Per ogni riga della query esterna calcolo la query nidificata e il predicato che ne deriva
- L'operatore EXISTS è significativo solo in casi di passaggio di binding (gli altri anche negli altri tipi di query nidificate)

Esercizio

Aereoporto (Città, Paese, NumeroDiPiste)
Volo (FlightID, Giorno, CittàPart, OraPart, CittàArr, OraArr, TipoAereo)
Aereo (TipoAereo, NumeroPasseggeri)

- Trovare gli aereoporti del Belgio che hanno solo voli domestici (in due modi: usando **not in** e usando **not exists**)

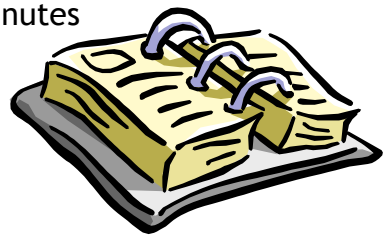
```
select CittàPart from Aeroporto join Volo on CittàPart=Città
where Paese = "Belgio" and CittàPart not in
    (select CittàPart
    from Aereoporto as A1 join Volo on A1.Città=CittàPart
                join Aereoporto as A2 on CittàArr=A2.Città
    where A1.Paese='Belgio' and A2.Paese<> 'Belgio')

select CittàPart from Aeroporto join Volo as A1 on CittàPart=Città
where Paese = "Belgio" and not exist
    (select * from Volo join Aereoporto as A2 on A2.Città=CittàArr
    where A1.Città=CittàPart and A2.Paese<>'Belgio' )
```

Structured Query Language - parte 2 - 35 - *Basi di Dati*

Bibliografia

- Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, Riccardo Torlone
  ***Basi di Dati** - Seconda edizione*
  settembre 1999 - McGraw Hill
  - **Capitolo 4.2**
- Teach Yourself SQL in 10 minutes second edition
  Ben Forta
  2001, SAMS



Structured Query Language - parte 2 - 36 - *Basi di Dati*