Basi di Dati: Corso di laboratorio

Lezione 4

Raffaella Gentilini

Sommario

Lo Statement JOIN

Join espliciti di tabelle nella clausola FROM

- In SQL e' possibile scrivere operazioni esplicite di join nella clausola FROM del comando SELECT, mediante lo statement JOIN
- si possono specificare esplicitamente diversi tipi di join:
  - CROSS JOIN
  - INNER JOIN
  - LEFT OUTER JOIN
  - RIGHT OUTER JOIN
  - FULL OUTER JOIN
  - NATURAL JOIN

Join Naturale

- In algebra relazionale, il join naturale e' un operatore che collega dati in relazioni diverse, sulla base di valori uguali in attributi con lo stesso nome
- Il join naturale $R_1 \bowtie R_2$ puo' essere introdotto esplicitamente nella clausola `FROM` del comando `SELECT` utilizzando
  1. il costrutto `NATURAL [INNER] JOIN`:

     `SELECT` $*$ `FROM` R_1 `NATURAL [INNER] JOIN` R_2
  2. il costrutto `[INNER] JOIN` + `USING`:

     `SELECT` $*$ `FROM` R_1 `[INNER] JOIN` R_2 `USING <attributi>`

     Lo statement `USING` permette di eliminare le colonne duplicate nel risultato

Esempio (I)

Si considerino le tabelle *insegnante* e *corso* illustrate di seguito:

id_insegnante	*cognome*	*nome*
3	Donato	Rocco
2	Bella	Donatella
7	Pupo	Pino
10	Moreno	Mario

id_corso	*id_insegnante*	*titolo*
1	3	'Circuiti'
2	2	'Programmazione'
3		'Algoritmi'
10		'Logica'

Esempio (II)

Example (join naturale di *insegnante* e *corso*)

Il join naturale delle tabelle *corso* e *insegnante* si puo' ottenere usando indifferentemente uno dei seguenti comandi:

- `SELECT * FROM insegnate NATURAL JOIN corso;`
- `SELECT * FROM insegnate NATURAL INNER JOIN corso;`
- `SELECT * FROM insegnate JOIN corso USING (id_insegnante);`
- `SELECT * FROM insegnate INNER JOIN corso USING (id_insegnante);`

e produce il risultato:

id_insegnante	*cognome*	*nome*	*id_corso*	*titolo*
3	Donato	Rocco	1	'Circuiti'
2	Bella	Donatella	2	'Programmazione'

Theta Join

L'operatore dell'algebra relazionale di theta join

$$R_1 \bowtie_{exp} R_2 \equiv \sigma_{exp}(R_1 \times R_2)$$

puo' essere introdotto esplicitamente nella clausola `FROM` del comando `SELECT` utilizzando:

- il costrutto `[INNER] JOIN` + `ON`:

  `SELECT` $*$ `FROM` R_1 `[INNER] JOIN` R_2 `ON` <*predicato*>

dove <*predicato*> e' un'espressione logica (come nella clausola `WHERE`).

Esempio

Si considerino le tabelle *persona* e *frequenta* illustrate di seguito:

Example

id_insegnante	*cognome*	*nome*
3	Donato	Rocco
2	Bella	Donatella
7	Pupo	Pino
10	Moreno	Mario

id_corso	*id_insegnante*	*titolo*
1	3	'Circuiti'
2	2	'Programmazione'
3		'Algoritmi'
10		'Logica'

Esempio (II)

Example

Selezionare i corsi per cui e' definito il corrispondente titolare del corso:

- utilizzando il costrutto `INNER JOIN` esplicitamente nella clausola `FROM`

```
SELECT id_corso, titolo
FROM insegnante INNER JOIN corso
        ON insegnante.id_insegnante = corso.id_insegnante;
```

- oppure, data l'equivalenza $R_1 \bowtie_{exp} R_2 \equiv \sigma_{exp}(R_1 \times R_2)$

```
SELECT id_corso, titolo
FROM insegnante, corso
WHERE insegnante.id_insegnante = corso.id_insegnante;
```

CROSS JOIN

Il costrutto CROSS JOIN nella clausola FROM

- L'operazione di prodotto cartesiano nell'algebra relazionale puo' essere vista come un'operazione di theta-join:

$$R_1 \times R_2 \equiv R_1 \bowtie_{true} R_2$$

- Le seguenti espressioni sono equivalenti in SQL:
  1. `SELECT * FROM` *tabella1*, *tabella2*;
  2. `SELECT * FROM` *tabella1* `CROSS JOIN` *tabella2*;
  3. `SELECT * FROM` *tabella1* `JOIN` *tabella2* `ON TRUE`;
  4. `SELECT * FROM` *tabella1* `INNER JOIN` *tabella2* `ON TRUE`;

Join Esterno

- Il risultato del join tralascia le tuple di una relazione senza controparte nell'altra
- Il join esterno prevede che tutte le tuple diano un contributo al risultato, estendendo con valori nulli le tuple che non hanno controparte
  - Il join sinistro estende le tuple del primo operando
  - Il join destro estende le tuple del secondo operando
  - Il join completo le estende tutte

Join Esterno in SQL

In SQL l'operatore di outer join puo' essere introdotto esplicitamente nella clausola `FROM` del comando `SELECT` utilizzando i costrutti:

- `{LEFT|RIGHT|FULL}[OUTER]JOIN` + `ON` <*predicato*>
- `{LEFT|RIGHT|FULL}[OUTER]JOIN` + `USING` <*colonne*>
- `NATURAL{LEFT|RIGHT|FULL}[OUTER]JOIN`

Join Esterno Sinistro in SQL

Si considerino le tabelle *insegnante* e *corso* illustrate di seguito:

id_insegnante	*cognome*	*nome*
3	Donato	Rocco
2	Bella	Donatella
7	Pupo	Pino
10	Moreno	Mario

id_corso	*id_insegnante*	*titolo*
1	3	'Circuiti'
2	2	'Programmazione'
3		'Algoritmi'
10		'Logica'

Join Esterno Sinistro in SQL

Example (Join sinistro *insegnante* e *corso*)

Il join esterno sinistro delle tabelle *insegnante* e *corso*:

- permette di preservare nel risultato gli insegnanti che non tengono alcun corso

id_insegnante	*cognome*	*nome*	*id_corso*	*titolo*
3	Donato	Rocco	1	'Circuiti'
2	Bella	Donatella	2	'Programmazione'
7	Pupo	Pino		
10	Moreno	Mario		

Join Esterno Sinistro in SQL

Example (Join sinistro *insegnante* e *corso*)

- puo' essere realizzato in SQL mediante uno dei seguenti comandi:
  - `SELECT * FROM` *insegnante* `NATURAL LEFT JOIN` *corso*;
  - `SELECT * FROM` *insegnante* `NATURAL LEFT OUTER JOIN` *corso*;
  - `SELECT * FROM` *insegnante* `LEFT JOIN` *corso* `USING` (*id_insegnante*);
  - `SELECT * FROM` *insegnante* `LEFT OUTER JOIN` *corso* `USING` (*id_insegnante*);

Join Esterno Sinistro in SQL

Example

Se si usa uno dei due comandi:

- `SELECT` $*$ `FROM` *insegnante* `LEFT JOIN` *corso*
  `ON` $insegnante.id\_insegnante = corso.id\_corso$;
- `SELECT` $*$ `FROM` *insegnante* `LEFT OUTER JOIN` *corso*
  `ON` $insegnante.id\_insegnante = corso.id\_corso$;

La colonna che mantiene gli *id* degli insegnanti appare duplicata nel risultato.

Join Esterno Destro in SQL

Example (Join destro *insegnante* e *corso*)

Il join esterno sinistro delle tabelle *insegnante* e *corso*:

- permette di preservare nel risultato i corsi per cui non e' stato designato alcun titolare del corso

id_insegnante	*cognome*	*nome*	*id_corso*	*titolo*
3	Donato	Rocco	1	'Circuiti'
2	Bella	Donatella	2	'Programmazione'
			3	'Algoritmi'
			7	'Logica'

Join Esterno Destro in SQL

Example (Join destro *insegnante* e *corso*)

- puo' essere realizzato in SQL mediante uno dei seguenti comandi:
  - `SELECT * FROM` *insegnante* `NATURAL RIGHT JOIN` *corso*;
  - `SELECT * FROM` *insegnante* `NATURAL RIGHT OUTER JOIN` *corso*;
  - `SELECT * FROM` *insegnante* `RIGHT JOIN` *corso* `USING` (*id_insegnante*);
  - `SELECT * FROM` *insegnante* `RIGHT OUTER JOIN` *corso* `USING` (*id_insegnante*);

Join Esterno Completo in SQL

Example (Join completo *insegnante* e *corso*)

Il join esterno completo delle tabelle *insegnante* e *corso*:

- permette di preservare nel risultato gli insegnanti che non tengono alcun corso ed i corsi non ancora assegnati ad alcun docente.

id_insegnante	*cognome*	*nome*	*id_corso*	*titolo*
3	Donato	Rocco	1	'Circuiti'
2	Bella	Donatella	2	'Programmazione'
			3	'Algoritmi'
			7	'Logica'
7	Pupo	Pino		
10	Moreno	Mario		

Join Esterno Completo in SQL

Example (Join completo *insegnante* e *corso*)

- puo' essere realizzato in SQL mediante uno dei seguenti comandi:
  - `SELECT * FROM` *insegnante* `NATURAL FULL JOIN` *corso*;
  - `SELECT * FROM` *insegnante* `NATURAL FULL OUTER JOIN` *corso*;
  - `SELECT * FROM` *insegnante* `FULL JOIN` *corso* `USING` (*id_insegnante*);
  - `SELECT * FROM` *insegnante* `FULL OUTER JOIN` *corso* `USING` (*id_insegnante*);

Operazioni insiemistiche su tabelle

- L'istruzione `SELECT` non permette di eseguire unione, intersezione e differenza di tabelle;
- Si puo' pero' combinare in modo opportuno i risultati di due istruzioni `SELECT` utilizzando la clausola:
  [{ `UNION` | `INTERSECT` | `EXCEPT` [`ALL`]}<richiesta>]
- gli elementi delle `SELECT` list devono avere tipi compatibili e gli stessi nomi se si vogliono colonne con un'intestazione definita;
- L'ordine degli elementi e' importante (notazione posizionale);
- Il risultato e' di default privo di duplicati. Per ottenerli occorre aggiungere l'opzione `ALL`:

Algebra Insiemistica su Tabelle

Example

Tabella S:

A	B
1	a
1	a
2	a
2	b
2	c
3	b

Tabella R:

C	B
1	a
1	b
2	a
2	c
3	c
4	d

```
SELECT A
FROM R
UNION
SELECT C
FROM S;
```

Il risultato e':

1
2
3
4

Algebra Insiemistica su Tabelle

Example

Tabella S:

A	B
1	a
1	a
2	a
2	b
2	c
3	b

Tabella R:

C	B
1	a
1	b
2	a
2	c
3	c
4	d

```
SELECT A
FROM R
UNION
SELECT C AS A
FROM S;
```

Il risultato e':

A
1
2
3
4

Algebra Insiemistica su Tabelle

Example

Tabella S:

A	B
1	a
1	a
2	a

Tabella R:

C	B
1	a
1	b
2	a

```
SELECT B
FROM R
UNION ALL
SELECT B
FROM S;
```

Il risultato e':

B
a
a
a
a
b
a

Raggruppamenti

- Quanto sinora visto permette di estrarre dal DB informazioni che si riferiscono a singole tuple (eventualmente ottenute mediante operazioni di join)
  - Esempio: Il titolare del corso di logica.
- Vedremo ora come ottenere dal DB informazioni di sintesi che caratterizzano gruppi di tuple mediante le funzioni d'aggregazione e le clausole `GROUP BY` e `HAVING` del comando `SELECT` di SQL
  - Esempio: Il numero di insegnanti che tengono almeno un corso.

Funzioni d'aggregazione

Lo standard SQL mette a disposizione una serie di funzioni d'aggregazione (o 'di colonna')

Funzioni d'aggregazione

- `MAX`: massimo
- `MIN`: minimo
- `SUM`: somma
- `AVG`: media aritmetica
- `COUNT`: contatore

Funzioni d'aggregazione

- L'argomento di una funzione d'aggregazione e' una qualunque espressione della SELECT list

  SELECT MAX(*data_nascita*) FROM *persona*;
  SELECT MIN(*voto* + 2) FROM *frequenza*;

- E' possibile utilizzare l'opzione DISTINCT, per considerare soltanto i valori distinti

  SELECT SUM (DISTINCT *voto*) FROM *frequenza*;

Funzioni D'aggregazione e Valori Nulli

- Tutte le funzioni, ad eccezione di COUNT, ignorano i valori nulli
- Per quanto riguarda la funzione d'aggregazione COUNT:
  - La forma COUNT(∗) conta le tuple del risultato, considerando anche i valori nulli.
  - Specificando invece il nome di una colonna (ad esempio, COUNT(*id_corso*)) , i valori nulli vengono ignorati.

Esempio

Example

id_corso	*id_insegnante*	*titolo*	*retribuzione*
1	3	'Circuiti'	100
2	3	'Programmazione'	
3	2	'Algoritmi'	300
10	1	'Logica'	300

```
SELECT COUNT(*) AS r3
FROM corso WHERE id_insegnante = 3;
```

r3
2

```
SELECT COUNT(retribuzione) AS r3
FROM corso WHERE id_insegnante = 3;
```

r3
1

Clausola Select e Funzioni Aggregate

La SELECT list deve essere omogenea!

Example

SELECT *crediti*, MAX(*crediti*)
FROM *corso*

Clausola Select e Funzioni Aggregate

La SELECT list deve essere omogenea!

Example

SELECT *crediti*, MAX(*crediti*)
FROM *corso*

wrong!

Clausola Select e Funzioni Aggregate

La SELECT list deve essere omogenea!

Example

```
SELECT  crediti, MAX(crediti)
FROM    corso
```

wrong!

```
SELECT  MIN(crediti), MAX(crediti)
FROM    corso
```

La Clausola GROUP BY

- I valori di sintesi calclati dalle funzioni aggregate si riferiscono a tutte le tuple che soddisfano le condizioni della clausola WHERE
- In molti casi e' utile fornire tali valori per gruppi omogenei di tuple (e.g. gruppo degli studenti di un certo corso)
- la clausola GROUP BY permette di definire tali gruppi, specificando una o piu' colonne di raggruppamento sulla base della/e quale/i le tuple sono raggruppate per valori uguali

GROUP BY

La clausola GROUP BY

id_studente	*id_corso*	*voto*
1	3	30
2	3	25
1	1	20
10	1	23
7	1	14

```
SELECT id_corso, COUNT(*) AS NumStudenti
FROM frequenza
WHERE voto >= 18
GROUP BY id_corso
```

id_corso	*NumStudenti*
3	2
1	2

La Clausola GROUP BY

GROUP BY step by step

SELECT *id_corso*, COUNT(*) AS *NumStudenti*
FROM *frequenza*
WHERE *voto* >= 18
GROUP BY *id_corso*

Le tuple che soddisfano la clausola WHERE

id_studente	*id_corso*	*voto*
1	3	30
2	3	25
1	1	20
10	1	23
7	1	14

La Clausola GROUP BY

GROUP BY step by step

```
SELECT id_corso, COUNT(*) AS NumStudenti
FROM frequenza
WHERE voto >= 18
GROUP BY id_corso
```

sono raggruppate per valori uguali della/e colonna/e nella clausola GROUP BY

id_studente	*id_corso*	*voto*
1	3	30
2	3	25
1	1	20
10	1	23

La Clausola GROUP BY

GROUP BY step by step

SELECT *id_corso*, COUNT(*) AS *NumStudenti*
FROM *frequenza*
WHERE *voto* >= 18
GROUP BY *id_corso*

e infine a ciascun gruppo si applica la funzione aggregata

id_corso	*NumStudenti*
3	2
1	2

Condizioni sui Gruppi

- Oltre a poter formare dei gruppi, e' anche possibile selezionare dei gruppi sulla base di loro proprieta' complessive
- La clausola HAVING ha per i gruppi una funzione simile a quella che la clausola WHERE ha per le tuple

Example

Selezionare per ogni corso superato da almeno 10 studenti, il numero di studenti promossi.

```
SELECT id_corso, COUNT(*)
FROM frequenza
WHERE voto >= 18
GROUP BY id_corso
HAVING COUNT(*)>= 10
```

Raggruppamenti: Esempi

- *persona*(*id_persona*, *codice_fiscale*, *nome*, *cognome*, *data_nascita*)
- corso(*id_corso*, *id_insegnante*, *sigla*, *crediti*, *descrizione*)
- frequenza(*id_studente*,*id_corso*,*voto*): dove *id_studente* ed *id_corso* sono chiavi esterne su persona e corso

Example

Definire il numero dei corsi impartiti da ogni docente

```
SELECT id_insegnante,cognome, nome, COUNT(*)
FROM corso JOIN persona ON id_insegnante = id_persona
GROUP BY id_insegnante, cognome, nome;
```

Raggruppamenti: Esempi

- *persona*(*id_persona*, *codice_fiscale*, *nome*, *cognome*, *data_nascita*)
- corso(*id_corso*, *id_insegnante*, *sigla*, *crediti*, *descrizione*)
- frequenza(*id_studente*,*id_corso*,*voto*): dove *id_studente* ed *id_corso* sono chiavi esterne su persona e corso

Example

Definire il numero dei corsi di informatica impartiti dai docenti che insegnano in almeno due corsi di informatica:

```
SELECT id_insegnante,cognome, nome, COUNT(*)
FROM corso JOIN persona ON id_insegnante = id_persona
WHERE sigla LIKE 'INF%'
GROUP BY id_insegnante, cognome, nome
HAVING COUNT(*)>= 2;
```

Raggruppamenti: Esempi

- *persona*(*id_persona*, *codice_fiscale*, *nome*, *cognome*, *data_nascita*)
- corso(*id_corso*, *id_insegnante*, *sigla*, *crediti*, *descrizione*)
- frequenza(*id_studente*,*id_corso*,*voto*): dove *id_studente* ed *id_corso* sono chiavi esterne su persona e corso

Example

Definire il numero dei corsi di informatica impartiti dai docenti che insegnano in almeno due corsi di informatica, dei quali uno garantisca piu' di 4 crediti:

```
SELECT id_insegnante,cognome, nome, COUNT(*)
FROM corso JOIN persona ON id_insegnante = id_persona
WHERE sigla LIKE 'INF%'
GROUP BY id_insegnante, cognome, nome
HAVING COUNT(*)>= 2 AND MAX(crediti) > 4 ;
```

Sommario (I)

Sinassi del costrutto di Join

Ricapitolando:

- e' possibile definire esplicitamente una joined (intermediate) table nella clausola `FROM` del comando `SELECT` utilizzando una delle tre sinassi:
  1. < tabella1> [{{`RIGHT`|`LEFT`|`FULL`}[`OUTER`]}] `JOIN` <tabella2> `ON` <predicato> [...]
  2. < tabella1> [{{`RIGHT`|`LEFT`|`FULL`}[`OUTER`]}] `JOIN` <tabella2> `USING` (<colonna> [, ...]) [...]
  3. < tabella1> `NATURAL` [{{`RIGHT`|`LEFT`|`FULL`}[`OUTER`]}] `JOIN` <tabella2> [...]

Sommario (II)

Traduzione degli operatori insiemistici

- L'operatore insiemistico di unione $relazione_1 \cup relazione_2$ si puo' tradurre in SQL con:

  `SELECT * FROM` $relazione_1$ `UNION SELECT * FROM` $relazione_2$;

- L'operatore insiemistico di intersezione $relazione_1 \cap relazione_2$ si puo' tradurre in SQL con:

  `SELECT * FROM` $relazione_1$ `INTERSECT SELECT * FROM` $relazione_2$;

- L'operatore insiemistico di differenza $relazione_1 \setminus relazione_2$ si puo' tradurre in SQL con:

  `SELECT * FROM` $relazione_1$ `EXCEPT SELECT * FROM` $relazione_2$;

Sommario (III)

Gruppi: Sintassi

La sintassi di una query che utilizza i costrutti di raggrupamento (funzioni di aggregazione, clausole `GROUP BY` e `HAVING`) e data da:

`SELECT` <*espressione*$_1$> [,...], <*espressione*$_n$> [, <*funzione_aggreg*> [,...]]
`FROM` <*tabella*> [[`AS` <*alias*>]] [,...]
[`WHERE` <*predicato*>]
[`GROUP BY` <*espressione*$_1$> [,...], <*espressione*$_n$>]
[`HAVING` <*predicato*>]

Sommario (IV)

Funzioni d'aggregazione: Sintassi

- `AVG` ([`DISTINCT`|`ALL`] <*espressione*>)
- `COUNT` ({$*$|[`DISTINCT`|`ALL`] <*espressione*>}})
- `MAX` ([`DISTINCT`|`ALL`] <*espressione*>)
- `MIN` ([`DISTINCT`|`ALL`] <*espressione*>)
- `SUM` ([`DISTINCT`|`ALL`] <*espressione*>)

Bibliografia

Bibliografia ed Approfondimenti

- R.A.Elmasri, S.B. Navathe. Sistemi di Basi di Dati – Fondamenti: Capitolo 8
- Capitolo 6 (Data Manipulation) del manuale di PostgreSQL (http://www.postgresql.org/docs/manuals/)