Il Linguaggio di interrogazione SQL

1 COSTRUTTO DI BASE 2
1.1 Proiezione 2
1.2 Selezione 2
1.3 Eliminazione di duplicati 2
1.4 Prodotto Cartesiano 3
1.5 Ridenominazione 3
1.6 Join 4
2 OPERATORI INSIEMISTICI 5
3 ESTENSIONI LINGUISTICHE: SOTTOQUERY 6
3.1 L'operatore IN 6
3.2 L'operatore NOT IN 7
3.3 L'operatore EXISTS 8
3.4 L'operatore NOT EXISTS 9
3.5 Operatori di confronto 10
3.6 Operatori ALL e ANY 10
4 FUNZIONI DI AGGREGAZIONE 12
4.1 Funzione COUNT 12
4.2 Funzioni MAX e MIN 12
4.3 La funzione SUM 13
4.4 La Clausola GROUP BY 13
4.5 La clausola HAVING 16
5 APPENDICE 19

1 COSTRUTTO DI BASE

SELECT A_1, …, A_n

FROM R_1, .., R_k

WHERE condizione

Significato:

$$\pi_{A1,\dots,An}(\sigma_{cond}(R_1 \bowtie \cdots \bowtie R_k)$$

Attraverso questo costrutto è possibile rappresentare le query algebriche (positive) basate su proiezione, selezione, join (prodotto cartesiano).

Se due attributi presenti nelle relazioni **R_1, .., R_k** hanno lo stesso nome, essi vengono considerati diversi in virtù del fatto di appartenere a relazioni diverse. Per cui l'espressione $R_1 \bowtie \cdots \bowtie R_k$ equivale al prodotto cartesiano tra le relazioni che appaiono nella clausola FROM. Ciò implica che, per effettuare un join, è necessario esplicitare la condizione di join nella clausola WHERE.

NOTA: gli esempi di seguito riportati fanno riferimento alla base di dati descritta in Appendice.

1.1 Proiezione

```
SELECT   codf, nome
FROM     F
```

Equivalente a $\pi_{codf,nome}F$ – codice e nome di tutti i fornitori

1.2 Selezione

```
SELECT   *
FROM     F
WHERE età > 30
```

Equivalente a $\sigma_{città=roma}$F – codice, nome, età e città dei fornitori ultratrentenni

1.3 Eliminazione di duplicati

A differenza dell'AR, la risposta generata da una query SQL può contenere duplicati. Ad esempio,

```
SELECT città
FROM   F
```

restituisce [Roma, Roma, Milano, Cosenza] (vedi BD in Appendice). Per eliminare i duplicati, è necessario usare la parola chiave DISTINCT. La query

```
SELECT DISTINCT città
FROM   F
```

restituisce {Roma, Milano, Cosenza}.

1.4 Prodotto Cartesiano

Come sopra accennato, tra le relazioni che appaiono nella parte FROM viene fatto il prodotto cartesiano.

Interrogazione: Coppie <forn, prod>, in cui il fornitore è rappresentato in termini di codice (codf) e città, ed il prodotto attraverso il suo codice (codp)

```
SELECT   codf, città, codp
FROM     F, P
```

Interrogazione: Coppie <forn, prod>, in cui il fornitore è rappresentato in termini di codice (codf) e nome, ed il prodotto attraverso il suo codice (codp)

```
SELECT codf, F.nome, codp
FROM     F, P
```

Siccome il prodotto cartesiano tra F e FP contiene il nome dei fornitori ed il nome dei prodotti, la notazione F.nome consente di disambiguare il significato di "nome" come nome del Fornitore (e NON del Prodotto). In generale, per disambiguare eventuali attributi con lo stesso nome che appaiono nella SELECT, si usa la *dot notation.*

1.5 Ridenominazione

È possibile ridenominare le relazioni attraverso l'uso di variabili. La ridenominazione delle relazioni comporta la ridenominazione implicita dei relativi attributi. L'uso delle variabili è quindi necessario nel caso di prodotto cartesiano di una relazione con sè stessa.

Interrogazione: Coppie di nomi di fornitori (X,Y) tali per cui X è più anziano di Y.

```
SELECT     X.nome, Y.nome
FROM       F as X, F as Y
WHERE      X.età>Y.età
```

X				Y			
codF	**nome**	**età**	**città**	**codF**	**nome**	**età**	**città**
f1	aldo	30	roma	f1	aldo	30	roma
f1	aldo	30	roma	f2	clara	40	milano
..	..						
f2	clara	40	milano	f1	aldo	30	roma

Senza l'uso delle variabili non sarebbe possibile differenziare i nomi degli attributi.

La suddetta query è quivalente alla seguente espressione algebrica, dove si usa l'operatore di ridenominazione:

$$\pi_{X.nome,Y.nome}(\sigma_{X.età>Y.età}(\rho_{X.codF\leftarrow codF,X.nome\leftarrow nome,X.età\leftarrow età,X.città\leftarrow città}F \bowtie \rho_{Y.codF\leftarrow codF,Y.nome\leftarrow nome,Y.età\leftarrow età,Y.città\leftarrow città}F))$$

Interrogazione: Codici e nomi dei fornitori più giovani del fornitore f1

```
SELECT   Y.codf as codiceGiovane, Y.nome as nomeGiovane
FROM     F as X, F as Y
WHERE    X.codf = f1 and X.età > Y.età
```

Interrogazione: Nome e cognome di fornitori che hanno lo stesso cognome, ma nome diverso, di fornitori di Roma

```
SELECT   Y.nome as Nome, Y.Cognome as Cognome
FROM   F as X, F as Y
WHERE X.città=Roma and X.cognome = Y.cognome and X.nome ≠ Y.nome
```

1.6 Join

Siccome le relazioni che appaiono nella parte FROM vengono messe in prodotto cartesiano, per eseguire il join tra due relazioni è necessario specificare la condizione di join nella clausola WHERE.

Interrogazione: Nomi dei fornitori, ed i codici e i prezzi dei prodotti da essi forniti.

```
SELECT   F.nome, FP.codP, FP.prezzo
FROM     F, FP
WHERE  F.codf= FP.codf   //condizione di join
```

Interrogazione: Codici e nome dei fornitori di Roma, con età inferiore ai 40 anni, che forniscono qualche prodotto

```
SELECT   F.codf, F.nome
FROM     F, FP
WHERE F.codf= FP.codf and città=Roma and età < 40
```

- NOTA: il join tra F e FP è necessario per restringere l'insieme dei fornitori solo a quelli che forniscono almeno un prodotto.
- NOTA: la clausola WHERE contiene sia la condizione di selezione < città=Roma and età < 40> sia quella di join <F.codf= FP.codf>

2 OPERATORI INSIEMISTICI

- UNIONE - Nomi dei fornitori che forniscono p1 o p2

```
SELECT Nome
FROM F, FP
WHERE F.codf = FP.codf  AND FP.codp=p1
        UNION
SELECT Nome
FROM Fornitori F, FP
WHERE F.codf = FP.codf and FP.codp=p2
```

Alternativamente

```
SELECT Nome
FROM  F, FP
WHERE F.codf = FP.codf AND (FP.codp=p1 OR FP.codp=p2)
```

- INTERSEZIONE - Nomi dei fornitori che forniscono sia p1 che p2

```
SELECT Nome
FROM F, FP
WHERE F.codf = FP.codf and FP.codp=p1
        INTERSECT
SELECT Nome
FROM F, FP
WHERE F.codf = FP.codf and FP.codp=p2
```

Alternativamente

```
SELECT X.codf
FROM  FP as X, FP as Y
WHERE X.codf = Y.codf and X.codp=p1 and Y.codp=p2
```

oppure usando l'operatore In come mostrato nell prossimo paragrafo.

- DIFFERENZA - codici dei fornitori che NON forniscono prodotti

```
SELECT codf
FROM Fornitori
        EXCEPT (o MINUS)
SELECT codf
FROM FP
```

NOTA: grazie agli operatori insiemistici (in particolare, all'operatore MINUS) si ottiene l'equivalenza con l'AR

3 ESTENSIONI LINGUISTICHE: SOTTOQUERY

Non aggiungono potere espressivo, ma rendono l'uso del linguaggio più semplice e naturale

3.1 L'operatore IN

Operatore insiemistico di appartenenza di un elemento: ∈

- **Interrogazione**: Nome dei fornitori di Milano, o Torino o Roma

```
SELECT nome
FROM  F
WHERE città IN ( "milano", "torino", "roma")
```

Equivalente a

```
SELECT nome
FROM  F
WHERE città = "milano" or città = "torino" or città = "roma"
```

L'insieme di appartenenza può essere calcolato attraverso una sotto-query.

- **Interrogazione**: Nomi dei fornitori che forniscono almeno un prodotto

```
SELECT nome
FROM  F
WHERE codf IN
        (SELECT codf
        FROM   FP )
```

la sottoquery restituisce l'insieme S dei codici dei fornitori che forniscono almeno un prodotto. La query esterna a sua volta restituisce i nomi dei fornitori i cui codici appartengono ad S.

La suddetta query si presta quindi alla seguente lettura: calcola i nomi dei fornitori i cui codici appartengono all'insieme dei codici dei fornitori che forniscono qualche prodotto.

Essa è equivalente a

```
SELECT nome
FROM  F, FP
WHERE F.codf  = FP.codf
```

- **Interrogazione**: Nomi dei fornitori di prodotti rossi

```
SELECT F.nome
FROM  F
WHERE codf   IN  (SELECT      codf
                  FROM      FP
                  WHERE     codp  IN  (SELECT  codp
                                       FROM    P
                                       WHERE  colore = rosso)
```

Con riferimento alla base di dati in appendice, la sottoquery di secondo livello restituisce l'insieme S=[p1, p2] – i codici dei prodotti di colore rosso. La sottoquery di primo livello restituisce quindi l'insieme T dei codici dei fornitori in FP che forniscono prodotti i cui codici sono in S, cioè, T = {f1,

f2, f3}. Infine, la query esterna restituisce i nomi dei fornitori i cui codici sono in T, quindi: [chiara, aldo, piero].

- **Interrogazione**: Nomi dei fornitori che forniscono sia p1 che p2

```
SELECT  codf
FROM    FP
WHERE codp=p1 and codf IN  IN
        (SELECT codf
         FROM    FP
         WHERE  codp=p2)
```

- **Interrogazione**: Fornitori che hanno lo stesso cognome di fornitori di Roma

```
SELECT  codf
FROM    F
WHERE cognome  IN
        (SELECT cognome
         FROM    F
         WHERE  città=Roma)
```

- **Interrogazione**: Fornitori che hanno lo stesso cognome, ma nome diverso, di fornitori di Roma

```
SELECT codf
FROM F as F1
WHERE cognome IN (SELECT cognome
         FROM  F as F2
         WHERE città=Roma and F1.nome <> F2.nome)
```

Si tratta di una **query nidificata correlata**: infatti, l'interrogazione nidificata fa riferimento a quella esterna attraverso la variabile F1. Da un punto di vista operazionale, essa può essere interpretata come segue: per ogni fornitore F1 (query esterna) vengono selezionati tutti i fornitori F2 (query interna) tali che la città di F2 sia Roma ed il suo nome sia diverso da quello di F1. Se il cognome di F1 è nell'insieme calcolato dalla query nidificata, allora la query è soddisfatta e, per il fornitore F1, vengono forniti nome e cognome. Pertanto, la query nidificata viene valutata una volta per ogni fornitore (tupla) di Fornitori.

Si noti che la variabile F1, definita nella query esterna, è visibile nella sottoquery. In quest'ultima non è necessario (ma non è vietato) l'uso di altre variabili, in quanto gli attributi "città" e "nome" che appaiono nella clausola WHERE fanno riferimento alla istanza di F che appare nella clausola FROM della sottoquery.

In generale, una variabile definita in una (sotto)query è visibile in tutte le sottoquery più interne – vale la regola di visibilità delle variabili nei linguaggi di programmazione.

NOTA: una query correlata può essere trasformata in una query non correlata tramite il prodotto cartesiano

3.2 L'operatore NOT IN

Operatore insiemistico di non appartenenza di un elemento: $\notin$

- **Interrogazione**: Nomi dei fornitori che **non** forniscono alcun prodotto

```
SELECT nome
FROM  F
WHERE codf  NOT IN  (SELECT codf
```

```
                                    FROM   FP )
```

Equivalente all'uso dell'operatore MINUS

- **Interrogazione**: Codici dei fornitori che non hanno omonimi:

```
SELECT codf
FROM  F as F1
WHERE  nome NOT IN
        (SELECT nome
        FROM F as F2
        WHERE F1.codf <> F2.codf)
```

La sottoquery restituisce, per ogni fornitore F1, i nomi di tutti gli *altri* fornitori. NOTA: l'uso di F2 nella sottoquery non è necessario

La sottoquery correlata può essere sostituita con un prodotto cartesiano in una query non correlata

```
SELECT  codf
FROM   F
WHERE  codf  NOT IN
        (SELECT F1.codf
        FROM F as F1, F as F2
        WHERE F1.codf <> F2.codf and F1.nome = F2.nome)
```

La sottoquery restituisce l'insieme dei codici dei fornitori che hanno qualche omonimo.

3.3 L'operatore EXISTS

L'operatore EXISTS viene usato per controllare se il risultato di una **sottoquery correlata** (nei casi significativi) è vuoto oppure no. Corrisponde all'operatore ∃ del calcolo dei predicati.

- **Interrogazione**: Fornitori che hanno omonimi

```
SELECT nome
FROM  F as F1
WHERE  EXISTS
        (SELECT *
        FROM F as F2
        WHERE F1.codf <> F2.codf and F1.nome=F2.nome  )
```

Il meccanismo di valutazione è sempre quello delle query correlate: per ogni fornitore F1 viene calcolato l'insieme S delle tuple di F che rappresentano fornitori diversi da F1 e che hanno lo stesso nome. Se S è l'insieme non vuoto, allora F1 è un fornitore che ha omonimi e, quindi, fa parte della risposta.

- **Interrogazione**: codf fornitori tranne i più giovani

```
SELECT codf
FROM F as F1
WHERE EXISTS (SELECT *
```

```
        FROM  F as F2
        WHERE  F2.eta < F1.eta)
```

Le suddette query si possono riformulare, in maniera meno naturale, attraverso il prodotto cartesiano.

3.4 L'operatore NOT EXISTS

L'operatore NOT EXISTS (∄) viene usato per controllare se il risultato di una **sottoquery correlata** è vuoto oppure no.

- **Interrogazione**: Seleziona fornitori che non hanno omonimi:

```
SELECT codf
FROM   F as F1
WHERE NOT EXISTS
        (SELECT *
        FROM F as  F2
        WHERE F1.codf <> F2.codf and F1.nome=F2.nome)
```

Per ogni fornitore F1 viene calcolato l'insieme S delle tuple di F che rappresentano fornitori diversi da F1 e che hanno lo stesso nome. Se S è l'insieme vuoto, allora F1 è un fornitore che non ha omonimi e, quindi, fa parte della risposta.

- **Interrogazione**: Seleziona i codf dei fornitori con la minima età

```
SELECT codf
FROM    F as F1
WHERE NOT EXISTS (SELECT *
                  FROM F as F2
                    WHERE F2.eta < F1.eta)
```

Attraverso il NOT EXISTS (o NOT IN) è possibile calcolare il MIN e il MAX di un insieme – il min (risp. max) di un insieme è un elemento tale che NON ESISTE un altro elemento minore (risp. maggiore)

- **Interrogazione**: Seleziona i fornitori che forniscono **solo** il prodotto con codice p1

La query, nonostante le apparenze, è negativa; infatti, essa può essere riformulata come segue: seleziona tutti i fornitori che forniscono qualche prodotto e per cui **non esiste** una loro fornitura di un prodotto diverso da p1.

```
SELECT codf
FROM    FP as F1
WHERE codp=p1 and NOT EXISTS (SELECT *
                FROM FP
                WHERE F1.codf=codf and codp <> p1)
```

3.5 Operatori di confronto

Si tratta degli usuali operatori di confronto {=, >, <, …} che vengono utilizzati per collegare una query ad una sotto-query.

- **Interrogazione**: Nomi dei fornitori più giovani del fornitore f1

```
SELECT  nome
FROM  F
WHERE età < (SELECT età  FROM   F
      WHERE  codf = f1)
```

NOTA: la sotto-query deve restituire un unico valore

- **Interrogazione**: Codici fornitori che forniscono il prodotto p2 ad un prezzo inferiore a quello a cui lo fornisce il fornitore f1

```
SELECT  codf
FROM  FP
WHERE codp = p2 and prezzo <
        (SELECT prezzo
        FROM      FP
        WHERE  codf = f1 and codp=p2)
```

3.6 Operatori ALL e ANY

È possibile usare operatori aritmetici con sotto-query che restituiscono più valori usando ALL e ANY

- **Interrogazione**: Nomi dei fornitori più anziani – max età

```
SELECT  nome
FROM   F
WHERE età >= ALL (SELECT età
                  FROM    F)
```

La sotto-query restituisce l'insieme delle età dei fornitori S={30, 40, 50}. La query esterna quindi restituisce il nome dei fornitori la cui età è maggiore o uguale ad ogni età in S.

Attraverso le condizioni >= ALL e <= ALL è possibile calcolare il MAX e il MIN di un insieme

- **Interrogazione**: Codici dei fornitori che forniscono il prodotto p1 al minimo prezzo

```
SELECT  codf
FROM  FP
WHERE codp=p1 and prezzo <= ALL
```

```
(SELECT prezzo
FROM      FP
WHERE    codp=p1)
```

Le suddette interrogazioni possono essere facilmente espresse con il NOT EXIST.

- **Interrogazione**: Nomi fornitori tranne i più giovani

```
SELECT  nome
FROM  F
WHERE età > ANY (SELECT età
                 FROM  F)
```

La suddetta query può essere espressa con l'EXISTS

```
SELECT  nome
FROM  F as X
WHERE EXISTS (SELECT *
              FROM F
              WHERE età <X.età)
```

4 FUNZIONI DI AGGREGAZIONE

Sono funzioni non rappresentabili in Algebra Relazionale – quindi rendono SQL più espressivo dell'AR. Sono: count, sum, min, max, avg.

4.1 Funzione COUNT

- **Interrogazione**: Numero di fornitori di Roma

```
SELECT COUNT(*) as #forn
FROM F
WHERE città = roma
```

- **Interrogazione**: Numero di città

```
SELECT COUNT(all città)          /* num. righe di F con valori di città non NULL
FROM F

SELECT COUNT(distinct città)
FROM F
```

4.2 Funzioni MAX e MIN

```
SELECT MAX(prezzo)
FROM  FP
```

Alternativamente si può usare, ad esempio, il NOT EXISTS

```
SELECT prezzo
FROM FP as F1
WHERE NOT EXISTS (SELECT *
                  FROM FP
                  WHERE F.prezzo > FP1.prezzo)
```

- **Interrogazione**: Codici dei prodotti con prezzo massimo

```
SELECT codp
FROM FP
WHERE prezzo = (SELECT max(prezzo)
          FROM FP)
```

La sotto-query restituisce il prezzo massimo Q che appare nella relazione FP. La query esterna a sua volta restituisce i codici dei prodotti il cui prezzo è pari a Q.

Se la clausola *select* contiene funzioni aggregate, tutti gli attributi nella clausola *select* devono essere contenuti nelle funzioni aggregate

Formulazione ERRATA:

```
SELECT codp, max(prezzo)
FROM FP
```

- **Interrogazione**: *Quanti* sono i prodotti a costo minimo?

```
SELECT COUNT(distinct codp)
FROM    FP
WHERE  prezzo = (SELECT MIN(prezzo)
                 FROM FP )
```

Con riferimento alla base di dati in appendice, la sotto-query restituisce il prezzo minimo, che è 20. La query esterna quindi (*a*) seleziona le tuple di FP in cui il prezzo è pari a 20 (le ultime 2), e (*b*) conta il numero di tali tuple (2, per l'appunto).

4.3 La funzione SUM

- **Interrogazione**: Quantità totale del prodotto p2 fornito dai fornitori di milano

```
SELECT SUM(quantità)
FROM     FP
WHERE codp=p2 and codf IN (SELECT codf
                           FROM  F
                           WHERE città = milano)
```

4.4 La Clausola GROUP BY

È utilizzata in query con funzioni di aggregazione con lo scopo di limitare la loro applicazione a sottoinsiemi di tuple.

- **Interrogazione**: Numero di fornitori per città

```
SELECT  città, count(*) as #forn
FROM  F
GROUP BY città
```

Fornitori

codF	nome	età	città
f1	chiara	30	roma
f2	aldo	40	roma
f3	piero	50	milano
f4	giovanna	35	cosenza

Risposta

città	#forn
roma	2
milano	1
cosenza	1

NOTA: viene creato un raggruppamento di tuple per ogni valore di città (attributo presente nel GroupBy). Il conteggio dei fornitori avviene all'interno di ogni singolo raggruppamento.

In generale, ogni raggruppamento è formato da tutte le tuple della relazione che hanno gli stessi valori degli attributi di raggruppamento – quelli che appaiono nel GROUPBY

- **Interrogazione**: Numero di forniture e quantità totale fornita da *ogni fornitore*, del quale vogliamo conoscere il codice

```
SELECT   codf, count(*) as #prod, sum(quantità) as quanTot
FROM     FP
GROUP BY         codf
```

FP

codF	codP	prezzo	quan
f1	p2	100	50
f1	p1	50	100
f2	p2	60	30
f3	p2	40	100
f3	p3	20	200

Risultato

codF	#prod	quanTot
f1	2	150
f2	1	30
f3	2	300

Gli attributi della *SELECT* devono apparire nel *GROUP BY*

- **Interrogazione**: Numero di forniture e quantità totale fornita da ogni fornitore, del quale vogliamo conoscere codice e nome

Siccome il nome è richiesto nella clausola SELECT, deve anche apparire tra gli attributi del GROUPBY – si noti che la presenza del nome non modifica i raggruppamenti in quanto ad un codice fornitore corrisponde esattamente un nome

```
SELECT      codf, nome, count(*) as #prod, sum(quantità) as quanTot
FROM        F, FP
WHERE       F.codf = FP.codf
GROUP BY    codf, nome
```

F join FP

codF	nome	età	città	codP	prezzo	quan
f1	chiara	30	roma	p2	100	50
f1	chiara	30	roma	p1	50	100
f2	piero	40	roma	p2	60	30
f3	piero	50	milano	p2	40	100
f3	piero	50	milano	p3	20	200

Risultato

codF	nome	#prod	quanTot
f1	chiara	2	150
f2	piero	1	30
f3	piero	2	300

- **Interrogazione**: Numero di forniture e quantità totale fornita da *ogni fornitore*, del quale vogliamo conoscere solo il nome

  Siccome il *nome* non è chiave primaria di Fornitore – ad esempio, sia f2 che f3 si chiamano entrambi Piero, se vogliamo mantenere la distinzione tra fornitori diversi, nel GROUPBY va comunque inserito il *codice*, anche se non viene usato nella clausola SELECT

```
SELECT      nome, count(*) as #prod, sum(quantità) as quanTot
FROM        F, FP
WHERE       F.codf = FP.codf
GROUP BY    codf, nome
```

Risultato

nome	#prod	quanTot
chiara	2	150
piero	1	30
piero	2	300

Usando solo il *nome* nel GROUPBY, si ottiene un risultato diverso, che risponde alla interrogazione: Numero di forniture e quantità totale fornita da *gruppi di fornitori con lo stesso nome*

```
SELECT      nome, count(*) as #prod, sum(quantità) as quanTot
FROM        F, FP
WHERE  F.codf = FP.codf
GROUP BY    nome
```

F join FP

codF	nome	età	città	codP	prezzo	quan
f1	chiara	30	roma	p2	100	50
f1	chiara	30	roma	p1	50	100
f2	piero	40	roma	p2	60	30
f3	piero	50	milano	p2	40	100
f3	piero	50	milano	p3	20	200

Risultato

nome	#prod	quanTot
chiara	2	150
piero	3	330

4.5 La clausola HAVING

Consente di selezionare gruppi creati con il GroupBy – a differenza della clausola WHERE che consente di selezionare singole tuple

- **Interrogazione**: Codici dei fornitori che forniscono più di un prodotto

```
SELECT   codf
FROM     FP
GROUP BY  codf
HAVING count(*) > 1
```

Grazie al GroopBy, il *count* viene calcolato all'interno di ogni singolo raggruppamento.

FP

codF	codP	prezzo	quan
f1	p2	100	50
f1	p1	50	100
f2	p2	60	30
f3	p2	40	100
f3	p3	20	200

La clausola *having* restituisce quindi i raggruppamenti che soddisfano la condizione *count>1* (in rosso).

codF	count(*)
f1	2
f2	1
f3	2

Risultato: {f1, f3}

NOTA: tutti i nomi di attributi in *HAVING* devono essere contenuti in una funzione aggregata o apparire nell'elenco del *GROUP BY*

La suddetta query è equivalente a

```
SELECT codf
FROM FP as F1
WHERE 1 < (SELECT count(*)
        FROM FP
        WHERE F1.codf=FP.codf)
```

- **Interrogazione**: Fornitori che forniscono la massima quantità totale

```
SELECT   codf
FROM     FP
GROUP BY codf
HAVING SUM(quantità) > =  ALL
        ( SELECT SUM(quantità)
        FROM FP
        GROUP BY codf)
```

Dalla relazione FP

codF	codP	prezzo	quan
f1	p2	100	50
f1	p1	50	100
f2	p2	60	30
f3	p2	40	100
f3	p3	20	200

la sotto-query calcola il seguente insieme S= {150, 30, 300}, i cui elementi sono le somme delle quantità fornite da ogni fornitore. La query esterna quindi calcola, per ogni fornitore, la quantità totale SUM(quantità) fornita e restituisce il codice del fornitore presente nel raggruppamento che soddisfa la condizione SUM(quantità) >= ALL, cioè, in cui la quantità totale è maggiore o uguale ad ogni elemento di S.

La suddetta query è equivalente alla seguente: codici fornitori X tali che non esiste un fornitore Y tale che la somma delle quantità fornite da Y sia maggiore di quella fornita da X

```
SELECT     codf
FROM       FP as X
WHERE NOT EXISTS
           (SELECT *
            FROM FP as Y
            WHERE (SELECT Sum(quant)
                   FROM FP
                   WEHERE codf=Y.codf )
                         >
                   SELECT Sum(quant)
                   FROM FP
                   WEHERE codf=X.codf)
                  )
```

5 APPENDICE

BASE DI DATI:

Fornitori(codf, nome, età, città) -- F
Prodotti(codp, nome, colore) -- P
Forniture(codf*, codp*, prezzo, quantità) – FP

Nella relazione FP, gli attributi codf e codp sono chiavi secondarie. FP rappresenta l'associazione molti-a-molti tra Fornitori e Prodotti.

Base di dati di esempio definita sul suddetto schema:

F - Fornitori

codF	nome	età	città
f1	chiara	30	roma
f2	aldo	40	roma
f3	piero	50	milano
f4	giovanna	35	cosenza

FP - Forniture

codF	codP	prezzo	quan
f1	p2	100	50
f1	p1	50	100
f2	p2	60	30
f3	p2	20	100
f3	p3	20	200

P - Prodotti

codp	nome	colore
p1	sedia	rosso
p2	divano	rosso
p3	letto	verde