52.5
Interrogazioni in Sql

52.5.1 Prodotto cartesiano

Sono date due tabelle `T1`, `T2`.

Il **prodotto cartesiano tra** `T1` **e** `T2` visualizza una tabella avente:

~ Per colonne: ogni colonna di `T1` e ogni colonna di `T2`.

~ Per righe: la concatenazione tra ogni riga di `T1` e ogni riga di `T2`.

Quindi, il prodotto cartesiano consente di comporre una nuova “tabella virtuale” contenente valori provenienti da due (o più) tabelle.

La sintassi per effettuare il prodotto cartesiano è:

```
SELECT ..
FROM T1, T2;
```

Tipicamente, il prodotto cartesiano non è utile, perché contiene alcune righe “non interessanti”; quindi, per renderlo interessante, lo dobbiamo completare eliminando queste righe; possiamo farlo mediante

~ un WHERE,

oppure

~ mediante l’operazione di join, che è la soluzione preferibile, e che vediamo adesso.

52.5.2 Join

Sono date:

~ Due tabelle `T1`, `T2`.

~ Un insieme di `n` attributi, detto `X1`, di `T1`.

~ Un insieme di `n` attributi, detto `X2`, di `T2`.

Per ottenere l'**equi join interno** tra gli attributi `X1` di `T1` e gli attributi `X2` di `T2`, dobbiamo eseguire due operazioni:

~ Facciamo il prodotto cartesiano tra `T1` e `T2`.

~ Dopo, applichiamo (a questo prodotto cartesiano) la selezione che mantiene soltanto le righe dove `X1` è uguale a `X2`.

Il **theta join interno** è un’estensione dell’equi join interno, dal quale si differenzia soltanto perché il confronto tra gli attributi `X1` e `X2` delle due tabelle avviene mediante un operatore qualsiasi di confronto (cioè, avviene mediante <, <=, <>, ecc.).

D’ora in poi in poi, tranne quando dirò diversamente, con “join” intenderò il “theta join interno.” (Quindi, intenderò anche l’equi join interno, perché è un caso particolare del theta join interno.)

NB: Esiste anche il join esterno, che è un’estensione del theta join interno, e che vedremo più avanti.

Tipicamente, il join (sia interno sia esterno) è utile quando:

~ C'è un vincolo di integrità referenziale da `X1` a `X2` o viceversa.

oppure

~ `X1` è in integrità referenziale con un insieme `Y` di attributi di una terza tabella, e anche `X2` è in integrità referenziale con lo stesso insieme `Y` precedente.

Ec: È dato il db formato dalle seguenti tabelle:

```
CITTA(NomeCitta, NAbitanti)

SCUOLE(IdeScuola, Citta)
    IR: Citta -> CITTA.NomeCitta

STUDENTI(MatriStu, CittaResi)
    IR: CittaResi -> CITTA.NomeCitta
```

Elencate gli studenti che abitano in una città differente da quella in cui vanno a scuola.

Soluzione parziale

Prima, devo concatenare ogni scuola con ogni studente.

Dopo, considero soltanto (tra queste nuove righe ottenute mediante la concatenazione suddetta) quelle dove i due nomi di città (`SCUOLE.Citta` e `STUDENTI.NomeCitta`) sono differenti. Ciò è coerente con quanto ho detto prima sull’uso del join, perché entrambi i due nomi di città sono in integrità referenziale con `CITTA.NomeCitta`.

52.5.3 Note importanti

~ La ridenominazione di una tabella è obbligatoria (sintatticamente):
  ~ Nei self join (che vedremo tra poco).
  e
  ~ In caso di omonimia tra attributi (che, ovviamente, appartengono a tabelle differenti).

~ Se il `FROM` contiene almeno due tabelle,
ci conviene prefissare il nome di ogni attributo inserendo il nome della sua tabella (o un suo alias). Quindi, in questo caso, ci conviene definire un alias “breve” per ogni tabella che compare nel FROM; così, davanti a ogni attributo inseriremo l’alias della tabella (anziché il suo nome, che è più lungo).

~ Nel JOIN ON, è opportuno inserire i due attributi da confrontare con lo stesso ordine con cui compaiono nel FROM.

~ È opportuno inserire la condizione di confronto nel JOIN ON (non nel WHERE).
Più in generale, propongo la seguente convenzione.
È preferibile mettere una condizione nel JOIN ON (anziché nel WHERE) quando la condizione confronta (mediante un qualsiasi operatore, come =, <=, <>, ecc.) due attributi appartenenti a tabelle differenti.

~ L’inserimento di tabelle inutili nel FROM aumenta il tempo computazionale, e quindi è un errore di inefficienza.

52.5.3.1 Equi join interno / Esercizi risolti

1. Considerate il db di riferimento.

Elencate ogni attributo degli impiegati che lavorano in un dipartimento che si trova nella stessa città in cui gli impiegati risiedono.

Soluzione

Applico il JOIN ON usando le due espressioni booleane seguenti e operandole mediante un AND:

```
1 DIPARTIMENTI.NomeDipa = IMPIEGATI. NomeDipa
2 DIPARTIMENTI.Citta = IMPIEGATI.Citta
```

Notate che non c'è un vincolo di integrità referenziale tra i due attributi (`DIPARTIMENTI.Citta` e `IMPIEGATI.Citta`) confrontati nella seconda espressione; però questi due attributi memorizzano informazioni“analoghe”; quindi, un JOIN ON sull’attributo `Citta` è plausibile.

```
SELECT I.*
FROM IMPIEGATI AS I JOIN DIPARTIMENTI AS D
  ON I.Citta = D.Citta AND I.Citta = D.Citta;
```

52.5.4 Self join

Il self join è una variante del join, dal quale si differenzia perché la seconda tabella su cui applichiamo il join coincide con la prima tabella `T1`.

Quindi, il self join deve confrontare alcuni attributi `X1` di `T1` con altri attributi `X2` di `T1`.

52.5.4.1 Self join / Esercizi risolti

1. Considerate il db di riferimento.

Elencate le coppie (Matricola di ogni impiegato che lavora nel dipartimento `'SW'`, Matricola di ogni impiegato con lo stesso cognome di un impiegato di `'SW'`).

Soluzione

Riporto soltanto la porzione del db che è funzionale a questa consegna:

IMPIEGATI

MatriImpie	Cognome	NomeDipa
1	A	SW
2	A	SW
3	B	HW
4	C	NULL
5	C	SW
6	D	NULL
7	E	SW

L’output corrispondente è:

I_S.MatriImpie	I_O.MatriImpie	I_O.Cognome
1	2	A
2	1	A
5	4	C

```
/* I_S: Impiegati di Software */
/* I_O: Impiegati di qualsiasi dipartimento e
        che sono omonimi (rispetto al cognome) di un I_S */
SELECT I_S.MatriImpie, I_O.MatriImpie, I_O.Cognome
FROM IMPIEGATI AS I_S JOIN IMPIEGATI AS I_O
  ON I_S.Cognome = I_O.Cognome AND I_S.MatriImpie <> I_O.MatriImpie
  /* L’ultimo confronto elimina le coppie composte dallo stesso impiegato. */
WHERE I_S.Diparti = 'SW';
```

2. Considerate il db di riferimento.

Elencate i capi (e ogni loro informazione) degli impiegati che guadagnano più di 25.

Soluzione

Riporto soltanto la porzione del db che è funzionale a questa consegna:

Devo congiungere due tabelle:

1) la selezione della tabella `IMPIEGATI` che ottengo selezionando gli impiegati che guadagnano più di 25:

MatriImpie	Stipendio	MatriImpieCapo
1	50	NULL
2	80	NULL
4	60	1
6	90	NULL
7	90	4

2) la tabella `IMPIEGATI` completa.

L’output corrispondente è:

MatriImpie	Cognome	Nome	Citta	Stipendio	NomeDipa	Ufficio	MatriImpieCapo
1	A	X	PD	50	SW	1	NULL
4	C	Y	MI	60	NULL	NULL	1

```
/* I_25: Impiegati che guadagnano più di 25 */
/* I_C: Capi di I_25 */
SELECT I_C.*
FROM IMPIEGATI AS I_25 JOIN IMPIEGATI AS I_C
  ON I_25.MatriImpieCapo = I_C.MatriImpie
WHERE I_25.Stipendio > 25;
```

3. Considerate il db di riferimento.

Elencate le matricole degli impiegati che guadagnano più del loro capo.

Soluzione

Riporto soltanto la porzione del db che è funzionale a questa consegna:

IMPIEGATI

MatriImpie	Stipendio	MatriImpieCapo
1	50	NULL
2	80	NULL
3	15	1
4	60	1
5	20	2
6	90	NULL
7	90	4

L’output corrispondente è

I_P.MatriImpie
4
7

```
/* I_P: Impiegati che guadagnano più del loro capo */
/* I_C: Capi di I_P */
SELECT I_P.MatriImpie
FROM IMPIEGATI AS I_P JOIN IMPIEGATI AS I_C
  ON I_P.MatriImpieCapo = I_C.MatriImpie
WHERE I_P.Stipendio > I_C.Stipendio;
```

52.5.5 Operatori aggregati

Consideriamo una tabella (che, eventualmente abbiamo ottenuto mediante un JOIN e dalla quale abbiamo eliminato alcune righe mediante un WHERE).

Ogni operatore di aggregazione consente di ottenere un’unica informazione proveniente da più righe di una tabella.

Per il momento, vediamo il caso particolare in cui vogliamo ottenere l’informazione da tutte le righe di una tabella (eventualmente ottenuta mediante i passaggi suddetti).

52.5.5.1 Note sull’operatore COUNT

```
COUNT(
    *
    | DISTINCT Lista di attributi
    | ALL Lista di attributi
)
```

`ALL` è il default.

52.5.5.2 Note sull’espressione di aggregazione

Gli operatori `MAX`, `MIN`, `SUM`, `AVG` possono essere applicati (oltre che su un attributo) anche su un’espressione `AttriEspre` calcolata mediante i valori di alcuni attributi. `AttriEspre` deve essere definita su ogni riga.

Ec: Calcolate la durata media dei prestiti usando la tabella seguente:

`PRESTITI(IdePrestito, DataInizio, DataFine*)`

Con i dati seguenti l’output è 6.

1	14/9/2022	22/9/2022
2	14/9/2022	18/9/2022

Soluzione

```
SELECT AVG(PRE.DataFine – PRE.DataInizio)
FROM PRESTITI AS PRE
WHERE DataFine IS NOT NULL;
```

52.5.6 Interrogazioni con raggruppamento

Il GROUP BY raggruppa le righe di una tabella in uno o più gruppi, per consentire (mediante altre clausole dell’interrogazione) di elaborare ogni singolo gruppo prodotto dal GROUP BY.

Nota 1

Nel SELECT possiamo inserire soltanto:

~ Operatori di aggregazione

~ Attributi che compaiono nel GROUP BY.

Nota 2

L’espressione sulla quale vogliamo applicare un operatore di aggregazione non può essere ottenuta (a sua volta) mediante un operatore di aggregazione.

Ec: Considerate il db di riferimento.

Ogni dipartimento spende una certa somma in stipendi.

Trovate il massimo tra tutte queste spese dei dipartimenti.

La soluzione seguente è sintatticamente errata.

```
SELECT MAX(SUM(I.Stipendio)) /* Errore di sintassi */
FROM IMPIEGATI AS I
GROUP BY I.NomeDipa
```

Non siamo capaci di risolvere questo problema usando le istruzioni viste finora.

Lo potremo risolvere mediante i select annidati (che vedremo più avanti).

52.5.7 Predicati sui gruppi

L’HAVING seleziona (mediante un opportuno criterio) i gruppi ottenuti mediante il GROUP BY; quindi, elimina i gruppi che non soddisfano il criterio suddetto.

52.5.8 Ordinamento

Adesso che abbiamo visto tutte le sei clausole, riassumo l’ordine con cui dobbiamo posizionarle e l’ordine con qui l’Sql le elabora.

Posizione sintattica	Ordine di elaborazione	Clausola
1	5	`SELECT`
2	1	`FROM`
3	2	`WHERE`
4	4	`GROUP BY`
5	5	`HAVING`
6	6	`ORDER BY`

L'`ORDER BY` si usa quando:

~ È richiesto esplicitamente un ordine nel risultato.

~ È opportuna un'esposizione non mescolata dei dati.

Non bisogna abusare dell'`ORDER BY`, perché questo può aumentare il tempo di esecuzione.

52.5.8.1 Ordinamento / Esercizi risolti

Uso di un intero positivo come argomento di ORDER BY

1. Considerate il db di riferimento.

Elencate la matricola e lo stipendio di ogni impiegato.

Ordinate l’elenco rispetto allo stipendio.

Soluzione:

```
SELECT MatriImpie, Stipendio
FROM IMPIEGATI
ORDER BY 2;
```

Uso di un alias come argomento di ORDER BY

2. Considerate il db di riferimento.

Elencate, per ogni dipartimento, la sua spesa in stipendi.

Ordinate l’elenco rispetto alla somma spesa per gli stipendi.

Soluzione:

```
SELECT NomeDipa, SUM(Stipendio) AS SommaStipendi
FROM IMPIEGATI
GROUP BY NomeDipa
ORDER BY SommaStipendi;
```

Uso di ORDER BY senza richiesta esplicita di ordinamento

3. Considerate il db di riferimento.

Elencate, per ogni dipartimento, i dipendenti che ci lavorano e il loro stipendio.

Soluzione

In questo caso è opportuno ordinare rispetto ai dipartimenti.

```
SELECT NomeDipa, MatriImpie, Stipendio
FROM IMPIEGATI
WHERE NomeDipa IS NOT NULL
ORDER BY NomeDipa;
```

Senza `ORDER BY` avremmo una visualizzazione disordinata, come la seguente:

NomeDipa	MatriImpie	Stipendio
SW	1	50
SW	2	80
HW	3	15
SW	5	20
SW	7	90

52.5.9 Interrogazioni di tipo insiemistico

Alcune versioni di Sql non possiedono gli operatori `INTERSECT` e `EXCEPT`.

Quindi, quando questi operatori non esistono, dovremo progettare le operazioni suddette mediante un select annidato (che vedremo tra poco).

Io e voi useremo gli operatori suddetti.

52.5.9.1 Operatori insiemistici / Esercizi risolti

NB Inserite un commento prima di ogni sottoproblema nel quale avete scomposto il problema iniziale.

1. Considerate il db di riferimento.

Elencate le matricole dei capi i cui impiegati guadagnano tutti più di 40, senza usare operatori di aggregazione.

Es: L’input seguente:

MatriImpie	Stipendio	MatriImpieCapo
10	50	1
20	30	1
30	60	2
44	70	2

genera l’output seguente:

MatriImpieCapo
2

Soluzione

La consegna equivale a:

```
Matricole dei capi
EXCEPT
Matricole dei capi di (almeno) un impiegato che guadagna al massimo 40
```

Codifico ognuno dei due sottoproblemi precedenti:

```
/* Matricole dei capi */
SELECT MatriImpieCapo
FROM IMPIEGATI

EXCEPT

/* Matricole dei capi di (almeno) un impiegato che guadagna al massimo 40 */
SELECT MatriImpieCapo
FROM IMPIEGATI
WHERE Stipendio <= 40;
```

52.5.10 Join completo e incompleto

Il **risultato** di un join è detto **completo** se ogni riga di T1 e ogni riga di T2 compaiono nel risultato del join; altrimenti, è detto **incompleto**.

Quando il join è completo, si ha:
Cardinalità del join >= max {cardinalità di T1, cardinalità di T2}.

52.5.11 Join esterno

Sono date:

~ Due tabelle `T1`, `T2`.

~ Una condizione `C` come nel join interno.

Il **join esterno [sinistro | destro | completo]** tra gli attributi `X1` di `T1` e gli attributi `X2` di `T2` è l'unione tra il join interno (eseguito mediante la condizione `C`) e:

~ la concatenazione tra ogni riga di `T1` che non compare nel join interno e i valori `NULL` negli attributi di `T2`, se il join esterno è **sinistro**.

~ la concatenazione tra ogni riga di `T2` che non compare nel join interno e i valori `NULL` negli attributi di `T1`, se il join esterno è **destro**.

~ ogni riga individuata dal join sinistro oppure dal join destro, se il join esterno è **completo**.

52.5.11.1 Join esterno / Esercizi risolti

1. Considerate il db di riferimento.

Elencate ogni dipartimento insieme alle matricole degli eventuali impiegati che lavorano nel dipartimento.

L’output richiesto è qualcosa del tipo:

NomeDipa	MatriImpie
CE	NULL
HW	3
SW	1
SW	2
SW	5
SW	7

Soluzione

```
SELECT D.NomeDipa, I.MatriImpie
FROM DIPARTIMENTI AS D LEFT JOIN IMPIEGATI AS I ON D.NomeDipa = I.NomeDipa
ORDER BY D.NomeDipa;
```

1B. Risolvete l’esercizio precedente senza usare il join esterno.

Soluzione

La consegna equivale a elencare:

1: Per ogni dipartimento, il suo nome e le matricole degli impiegati che lavorano nel dipartimento

UNION

2: Ogni dipartimento senza impiegati

La consegna 2, a sua volta, equivale a elencare:

2.1: Ogni dipartimento

EXCEPT

2.2: Ogni dipartimento in cui lavora qualche impiegato

Uniamo quanto sopra e lo scriviamo in Sql

```
/* 1: Dipartimenti e le matricole degli impiegati che lavorano nel dipartimento*/
SELECT D.NomeDipa, I.MatriImpie
FROM DIPARTIMENTI AS D LEFT JOIN IMPIEGATI AS I ON D.NomeDipa = I.NomeDipa
ORDER BY D.NomeDipa

UNION

( /* 2: Dipartimenti senza impiegati */

/* 2.1: Dipartimenti */
SELECT NomeDipa, NULL
FROM DIPARTIMENTI

EXCEPT

/* 2.2: Dipartimenti in cui lavora qualche impiegato */
SELECT NomeDipa, NULL
FROM IMPIEGATI
)
```

2. Considerate il db di riferimento.

Elencate, per ogni dipendente, la sua matricola e l’eventuale città in cui lavora.

L’output richiesto è qualcosa del tipo:

MatriImpie	Citta
1	PD
2	PD
3	MI
4	NULL
5	PD
6	NULL
7	PD

Soluzione

```
SELECT I.MatriImpie, D.Citta
FROM DIPARTIMENTI AS D RIGHT JOIN IMPIEGATI AS I ON D.NomeDipa = I.NomeDipa
```

52.5.12 Select annidati

Interrogazione (o query):

~ 1 select;

~ 2 o più select:

  ~ Select allo stesso livello combinati da un operatore insiemistico (`UNION`, `INTERSECT`, `EXCEPT`);

  ~ Select annidati:

    ~ Select interno indipendente;

    ~ Select interno collegato.

Un select è **annidato** quando è composto da un select esterno e da (almeno) un select interno.

Risultato

Un select annidato confronta (mediante un operatore):

~ alcuni valori individuati dal select interno

con

~ ogni valore elencato del select esterno.

Il valore del select esterno proviene:

~ da ogni riga, quando il select interno è inserito nel `WHERE`

oppure

~ da ogni gruppo, quando il select interno è inserito nell'`HAVING`.

Dopo, il select annidato visualizza ogni riga (oppure ogni gruppo) del select esterno che soddisfa il confronto.

Es:

```
SELECT T1.X, COUNT(T1.K)
FROM T1
GROUP BY T1.X
HAVING COUNT(T1.K) < (
    SELECT COUNT(T1.K)
    FROM T1
    WHERE T1.X = 13
);
```

T1

K	X
A	11
C	12
D	13
E	13

Esecuzione della traccia

Il select interno individua:

COUNT(K)
2

Il select esterno "diventa":

```
SELECT T1.X, COUNT(T1.K)
FROM T1
GROUP BY T1.X
HAVING COUNT(T1.K) < 2;
```

Risultato finale:

X	COUNT(K)
11	1
12	1

Indicazioni sulla sintassi

Un select annidato confronta le righe individuate dal select interno con ogni riga o gruppo del select esterno; quindi:

- Il select interno deve avere un unico argomento.
- I due valori confrontati (uno proviene dal select esterno e l'altro proviene dal select interno) in ogni singolo passo devono:
  - appartenere a domini compatibili (es: un valore integer e un valore decimal);
  - possedere l'operatore di confronto usato nel confronto (es: l'operatore <).

Possiamo combinare due o più confronti mediante i consueti operatori logici:

```
..
WHERE AttrEst1 = (
    Select Interno 1
  )
  AND AttrEst2 = (
    Select Interno 2
);
```

Select interno indipendente

Un select interno è **indipendente** quando usa soltanto tabelle indicate nel proprio `FROM`.

Quindi:

~ Il risultato del select interno non dipende dal select esterno.

~ Sql esegue il select interno soltanto una volta, prima del select esterno.

Esempio importante.

Il select interno seguente è indipendente, e individua i valori di `K` che corrispondono al massimo di `X` (c.d. **punti di massimo**).

```
SELECT T1.K, T1.X
FROM T1
WHERE T1.X = (
    SELECT MAX(T1.X)
    FROM T1
);
```

Select interno collegato

Un select interno è **collegato** quando non è indipendente, cioè quando usa almeno una tabella che è indicata nel `FROM` del select esterno. Quindi:

~ Il risultato del select interno dipende dal select esterno.

~ Sql esegue il select interno per ogni riga del select esterno.

Es: Il select interno dell'esempio seguente è collegato perché (nella clausola `WHERE`) usa la tabella `T1` che non è indicata nel `FROM` del select interno.

Vedremo il suo risultato più avanti.

```
SELECT T1.K, T1.X
FROM T1
WHERE T1.X = (
    SELECT MAX(T2.X)
    FROM T2
    WHERE T1.Y = T2.Y
);
```

52.5.12.1 Sintesi di quanto visto finora

Tra poco approfondirò i select annidati.

Prima, evidenzio le cose più importanti che dovrete considerare per gestire i select annidati:

1 Un select interno serve a selezionare qualcosa del select esterno.
Quindi deve essere associato al `WHERE` o all’`HAVING` del select esterno.

2 La selezione suddetta avviene mediante un operatore di confronto applicato tra:
   ~ un’(unica) espressione che compare nel `WHERE` o nell’`HAVING` del select esterno
   e
   ~ un’(unica) espressione che compare nel `SELECT` del select interno;
   quindi il `SELECT` del select interno deve essere seguito da un unico argomento.

   Le due espressioni devono essere confrontate mediante un operatore di confronto.
   (Tra poco vedremo che gli operatori di confronto (>, >=, ecc.) possono essere “arricchiti” da alcuni argomenti che possono modificare il risultato.)

3 L’espressione del select esterno individua un valore, che può essere confrontato con:

   ~ un (unico) valore individuato dal select interno

   oppure

   ~ un insieme di valori individuati dal select interno;
   Soltanto alcuni operatori di confronto sono utilizzabili in questo secondo caso.

4 Dovete distinguere le due tipologie di select interno:
   ~ Indipendente

   oppure
   ~ Collegato

5 Il select interno indipendente (a differenza di quello collegato) è eseguito:
   ~ soltanto una volta;
   ~ prima del select esterno.

52.5.12.2 Sintassi e risultato di ogni modalità di confronto

Ci sono tre modalità per confrontare l’espressione del select interno con quella del select esterno.

Modalità 1

Questa modalità può essere usata soltanto se il select interno individua sempre (e non a seconda dei dati) un’unica riga; altrimenti, Sql darà un errore in esecuzione.

Quindi, per essere certi che l’istruzione funzioni sempre (cioè, con qualsiasi input), l’argomento del select interno deve essere una funzione di aggregazione.

Sintassi:

```
..
[WHERE | HAVING] EspreEste OperatoreDiConfronto (
    SELECT EspreInte
    ..
);
```

Risultato

Il risultato mostra ogni riga del select esterno che soddisfa il confronto seguente:

```
    EspreEste OperatoreDiConfronto EspreInte
```

Es:

```
..
HAVING COUNT(T1.X) >= (
    SELECT COUNT(T2.Y)
    ..
```

Es:

```
..
WHERE T1.X >= (
    SELECT T2.X
    FROM T2
    WHERE T2.Y = 5
);
```

Modalità 2

Questa modalità può essere usata qualunque sia il numero (anche zero) di righe individuate dal select interno.

Sintassi

Il select interno deve essere preceduto da una di queste quattro forme alternative, che ho inserito tra parentesi quadre:

```
..
[WHERE | HAVING] EspreEste
  [ OperatoreDiConfronto ANY | OperatoreDiConfronto ALL | IN | NOT IN ]  (
    SELECT EspreInte
    ..
)
```

Risultato

Una riga del select esterno compare nel risultato a seconda della parola chiave indicata:

~ con `ANY`:
la riga compare se `EspreEste` soddisfa il confronto con almeno un valore del select interno.

~ con `ALL`:
la riga compare se `EspreEste` soddisfa il confronto con ogni valore del select interno.

~ con `IN`:
la riga compare se `EspreEste` è tra i valori del select interno. (Quindi, `IN` equivale a `= ANY`).

~ con `NOT IN`:
la riga compare se `EspreEste` non è tra i valori del select interno. (Quindi, `NOT IN` equivale a `<> ALL`).

Es:

```
..
WHERE T1.X >= ANY (
    SELECT T2.X
    FROM T2
    WHERE T2.Y = 5
);
```

Modalità 3

Questa modalità,a differenza delle due precedenti, non deve confrontare un confronto tra il select esterno e quello interno; deve soltanto individuare se il select interno genera un elenco vuoto.

Sintassi:

```
..
WHERE [NOT] EXISTS (
    SELECT *
    FROM T2
);
```

Risultato:

~ `NOT EXISTS`: la riga del select esterno compare se non c'è nessuna riga nel select interno, cioè se il select interno genera un elenco vuoto.

~ `EXISTS`: la riga del select esterno compare se c'è almeno una riga nel select interno.

NB Nella clausola `SELECT` del select interno è meglio mettere sempre `*` (invece di un'espressione `EspreInte`), perché la presenza (o l'assenza) di righe in un select interno non dipende dai particolari attributi della clausola `SELECT` del select interno.

Nota sull'utilità di EXISTS

Se il select interno è indipendente, l’`EXISTS` non ha nessuna utilità perché il suo esito è identico su tutte le righe del select esterno (cioè, è sempre vero oppure sempre falso) e quindi non seleziona le righe del select esterno.

Es: Il seguente select interno è indipendente (perché usa una tabella definita nel proprio `FROM`); provo a confrontarlo (impropriamente) con quello esterno mediante un `EXISTS`.

```
SELECT T1.K, T1.X
FROM T1
WHERE EXISTS (
    SELECT *
    FROM T1
    WHERE T1.X = 3
);
```

L’istruzione precedente ha il seguente effetto.

Se il select interno genera una tabella vuota (cioè, se `T1.X` di ogni riga è sempre `!= 3`),
allora nessuna riga del select esterno compare nel risultato.

Se, invece, il select interno genera almeno una riga,
allora ogni riga del select esterno compare nel risultato.

Quindi, l'`EXISTS` in questo caso è inutile, perché il select interno (indipendente) non seleziona soltan-

to alcune righe del select esterno, ma le visualizza tutte o nessuna.

L'`EXISTS` è utile soltanto nei select collegati, come vedremo tra poco.

52.5.12.3 Select collegati

Risultato

Per ogni riga `r` del select esterno

Si individua l’insieme `I(r)` di righe del select interno. (Quindi, `I(r)` dipende dalla riga `r`).

Se il confronto (indicato nel `WHERE` o nell'`HAVING`) tra `r` e `I(r)` è soddisfatto

La riga `r` va nel risultato

Esempio:

Traccio l'istruzione:

```
SELECT *
FROM T1
WHERE T1.A = (
    SELECT COUNT(*)
    FROM T2
    WHERE T1.B = T2.B
);
```

sui seguenti dati:

T1

A	B
3	12
2	14

T2

B	C
12	8
14	7
13	5
14	6

Esecuzione della traccia:

Select esterno		Select interno		
T1.A	T1.B	WHERE T1.B = T2.B	T2.COUNT(*)	WHERE T1.A = (SELECT COUNT(*) ..
3	12	12	1	3 = 1 ? F
2	14	14	2	2 = 2 ? T

Risultato:

A	B
2	14

Uso: Quando l'interrogazione deve soltanto generare le righe che soddisfano un confronto tra:

~ un valore del select esterno, e

~ una tabella che varia al variare della riga del select esterno.

52.5.12.4 Select annidati / Esercizi risolti

1. Considerate il db di riferimento.

Elencate ogni matricola che ha lo stesso cognome di un’altra matricola. Usate un select annidato.

Soluzione

Abbiamo già risolto un problema simile mediante un self join.

Adesso lo risolviamo mediante un algoritmo del tipo:

Per ogni impiegato `impie`

Si individua l’insieme `Cogno(impie)` composto da quegli impiegati del select interno che hanno lo stesso cognome di `impie` e che hanno inoltre un identificativo differente.

Se l’insieme `Cogno(impie)` contiene almeno un impiegato,

La matricola `impie` va nel risultato.

Possiamo notare che l’insieme `Cogno(impie)` dipende dall’impiegato `impie`. Ciò implica che:

~ Il select interno deve essere collegato.

~ Il select interno confronta il cognome della sua tabella `IMPIEGATI` con il cognome di `IMPIEGATI` del select esterno. Quindi, usa la tabella `IMPIEGATI` con due finalità differenti; per questo, dobbiamo rinominare `IMPIEGATI` all’interno del select interno.

```
SELECT I.MatriImpie
FROM IMPIEGATI AS I
WHERE EXISTS ( /* Elenca gli impiegati omonimi della matricola I.MatriImpie */
    SELECT *
    FROM IMPIEGATI AS I_O
    WHERE I.Cognome = I_O.Cognome
      AND I.MatriImpie <> I_O.MatriImpie
)
```

Esecuzione della traccia:

Uso questi valori:

MatriImpie	Cognome
1	A
2	A
3	B
4	A

SELECT ESTERNO		SELECT INTERNO	
I. MatriImpie	I. Cognome	WHERE I.Cognome = I_O.Cognome AND I.MatriImpie <> I_O.MatriImpie	WHERE EXISTS
1	A	'A' = 'A' AND 1 <> 1 ? F	EXISTS({2, 4}) ? T
		'A' = 'A' AND 1 <> 2 ? T	
		'A' = 'B' AND 1 <> 3 ? F	
		'A' = 'A' AND 1 <> 4 ? T	
2	A	È analogo a I.MatriImpie = 1	EXISTS({1, 4}) ? T
3	B	'B' = 'A' AND 3 <> 1 ? F	EXISTS({ }) ? F
		'B' = 'A' AND 3 <> 2 ? F	
		'B' = 'B' AND 31 <> 3 ? F	
		'B' = 'A' AND 4 <> 4 ? F	
4	A	È analogo a I.MatriImpie = 1	EXISTS({1, 2}) ? T

L'interrogazione visualizzerà questa tabella:

I.MatriImpie
1
2
4

2. Considerate il db di riferimento.

Elencate, per ogni dipartimento:

~ la matricola con lo stipendio massimo nel dipartimento;

~ il valore di questo stipendio.

Ordinate l’elenco rispetto ai dipartimenti.

Soluzione

Per ogni terna (`d, m, s`),
(dove: `d` è un dipartimento; `m` è una matricola che lavora nel dipartimento `d`; `s` è lo stipendio di `m`)

Si individua l’insieme `Stipe(d)` composto dagli stipendi di quelle matricole che lavorano nel dipartimento `d` del select interno.

Se lo stipendio `s` è uguale al massimo di `Stipe(d)`,

La matricola `m` va nel risultato.

Possiamo notare che l’insieme `Stipe(d)` dipende dal dipartimento `d`. Ciò implica che:

~ Il select interno deve essere collegato.

~ Il select interno usa la tabella `IMPIEGATI` con due finalità differenti, in modo analogo all’esercizio precedente. Quindi, anche in questo esercizio dobbiamo rinominare `IMPIEGATI` all’interno del select interno.

```
SELECT I.NomeDipa, I.MatriImpie, I.Stipendio
FROM IMPIEGATI AS I
WHERE I.Stipendio = ( /* Individua lo stipendio massimo tra tutti gli impiegati
       che lavorano nel dipartimento I.NomeDipa */
    SELECT MAX(I_DIPA.Stipendio)
    FROM IMPIEGATI AS I_DIPA
    WHERE I.NomeDipa = I_DIPA.NomeDipa
)
ORDER BY I.NomeDipa;
```

Questo esercizio è un esempio molto importante di esercizi con questa tipologia:

Per ogni sottoinsieme individuato al variare di un attributo `A`, trovate le righe che hanno il valore massimo nell’attributo `B`.

(In questo caso, `A` è il dipartimento, le righe cercate sono gli impiegati, e `B` è lo stipendio.)

Esecuzione della traccia:

Uso questi valori:

MatriImpie	Stipendio	NomeDipa
1	50	SW
3	15	HW
5	15	SW
6	90	SW
7	90	SW

SELECT ESTERNO — SELECT INTERNO

I.MatriImpie, I.Stipendio, I.NomeDipa	ON I.NomeDipa = I_DIPA.NomeDipa	I_DIPA.Stipendio	WHERE I.Stipendio = MAX(I_DIPA.Stipendio)
1, 50, SW	SW	50	50 = 90 ? F
		15	
		90	
		90	
3, 15, HW	HW	15	15 = 15 ? T
5, 15, SW	SW	50	15 = 90 ? F
		15	
		90	
		90	
6, 90, SW	SW	50	90 = 90 ? T
		15	
		90	
		90	
7, 90, SW	SW	50	90 = 90 ? T
		15	
		90	
		90	

L'interrogazione visualizzerà questa tabella:

I.NomeDipa	I.MatriImpie	I.Stipendio
SW	6	90
SW	7	90
HW	2	15

52.6 Viste

52.6.1 Viste

Le tabelle che abbiamo visto finora sono soltanto un caso particolare (e il più importante) tra le varie categorie di “tabelle nel senso più ampio del termine” che possiamo trovare in un DB.

Possiamo classificare le “tabelle nel senso più ampio del termine” mediante questo albero:

~ Tabelle di base
Sono le tabelle che abbiamo visto finora.

~ Tabelle derivate
Sono tabelle che contengono soltanto alcuni dei dati contenuti nelle “tabelle di base” o ricavabili da essi. Quindi, contengono soltanto dati ridondanti.
Queste tabelle ridondanti si distinguono nelle seguenti categorie, a seconda della modalità con cui il DBMS le memorizza.
  ~ Viste materializzate
  Il DBMS memorizza una “vista materializzata”, in modo da averla pronta per le interrogazioni successive.
  ~ Viste (o Viste virtuali)
  Il DBMS calcola una “vista virtuale” ogni volta che un’interrogazione ha bisogno della vista; il calcolo avviene partendo dai valori contenuti nelle tabelle di base. Eventualmente, se c'è spazio in memoria, il DBMS può decidere di memorizzare la vista temporaneamente, anziché ricalcolarla ogni volta

52.6.1.1 Vantaggi delle viste

Ci conviene usare una vista in questi casi:

1 Vogliamo definire soltanto una volta un'espressione che dovremo riusare in più interrogazioni.

  Allora, definiamo una vista che calcola questa espressione; poi la riuseremo nelle varie interrogazioni.

2a Vogliamo scomporre un’interrogazione complicata.

2b Vogliamo scomporre “un’interrogazione non risolubile direttamente” in due o più sottointerrogazioni più semplici.

  Allora, in questi due sottocasi (2a e 2b), definiamo una vista che risolve un sottoproblema; poi la useremo, insieme ad altre istruzioni, per risolvere il problema iniziale.

3 Vogliamo nascondere a un utente alcuni dati che non gli sono necessari.

  Allora, definiamo una vista che contiene soltanto i dati che vogliamo rendere visibili; poi, consen-

52.8
Esercizi

52.8.1 Interrogazioni

~ Risolvete, mediante un'unica istruzione Sql, ogni seguente interrogazione in modo da visualizzare la consegna.

~ Potete usare soltanto costrutti visti a lezione.

~ Potete usare le viste soltanto quando l'interrogazione:
  ~ necessita di un'espressione che si deve usare almeno due volte, oppure
  ~ non è risolubile direttamente.

~ Una soluzione meno efficiente di quella migliore avrà una valutazione inferiore.

~ Le tracce sono facoltative, e quindi non vanno consegnate; però, sono fortemente consigliate.

~ Negli esercizi etichettati con (AR), non potete usare: select annidati, operatori aggregati, group by, having.

Commentate ogni:

~ Select interno.

~ Select che sarà combinato mediante un operatore insiemistico.

~ Tabella che state rinominando perché compare in un self join.

~ Parametro di un select.

~ Vista.

1. È dato il db AEROPORTI contenente le seguenti tabelle:

```
CITTA(NomeCitta, Stato)

TIPI_AEREI(TipoAereo, Capienza)
    Capienza è il numero massimo di passeggeri trasportabili da TipoAereo
VOLI(IdeVolo, GiornoSetti, NomeCittaParte, NomeCittaArri, TipoAereo*)
    TipoAereo == NULL => Il volo può essere schedulato mediante un tipo qualsiasi
    di aereo.
```

Il db è definito nel file `Aeroporti_CreateModify.sql`.

NB: Se avete bisogno di un self join con la tabella CITTA che compare due volte, usate le ridenominazioni seguenti:

~ C_A := Città d'arrivo

~ C_P := Città di partenza

A. Definite una vista contenente ogni volo (con ogni suo attributo) e aggiungendogli lo stato di partenza e quello di arrivo.

B. Elencate i voli gestiti da un tipo sconosciuto di aereo.

C. Elencate i tipi di aereo usati nei voli che partono da Milano.

D. Elencate gli stati dai quali partono voli che possono trasportare almeno 100 passeggeri.

E. Per il volo con identificativo 3, elencate il giorno della settimana, lo stato di partenza e quello di arrivo.

Risolvete in due modi:

I) Senza usare viste.

II) Mediante la vista individuata nel punto A.

F. Elencate le coppie di città collegate da voli internazionali.

Risolvete in due modi:

I) Senza usare viste.

II) Mediante la vista individuata nel punto A.

G. Indicate il numero di voli internazionali che partono la domenica da ‘Milano’.

La domenica è identificata mediante la stringa 'DO'.

L’Italia è identificata mediante la stringa 'ITA'.

H. Per ogni città italiana, indicate il numero di voli internazionali in partenza.'

L'elenco deve contenere soltanto le città in cui questo numero e` > 0.'

I. Elencate le città francesi da cui partono più di venti voli alla settimana diretti in ITA.

J. Elencate le città da cui partono voli diretti a Milano, ordinate alfabeticamente.

K. Elencate ogni città che è collegata a Roma (mediante un volo che parte da Roma o ci arriva).

L'elenco deve avere un'unica colonna.

L. Elencate le città collegate soltanto mediante voli effettuati la domenica. (Quindi, dovete escludere le città che non sono collegate e le città che sono collegate in almeno un giorno differente dalla domenica.)

L’elenco deve avere un'unica colonna.

Non usate join.

M. Per ogni volo che parte da Milano, elencate la città di arrivo e, se sono disponibili, il tipo e la capienza dell’aereo.

N. Elencate i tipi di aereo aventi la massima capienza.

O. Elencate le città che sono servite dai tipi di aereo aventi la massima capienza.

L'elenco deve avere un'unica colonna.

P. Elencate le città italiane che hanno la massima "ricezione". La ricezione di una città si ottiene sommando la capienza dei voli che arrivano in quella città.

L'elenco deve avere un'unica colonna.

Testate la soluzione usando questi dati:

TIPI_AEREI

TipoAereo	Capienza
A	100
B	70
C	70
D	200

VOLI

NomeCittaParte	NomeCittaArri	TipoAereo
Milano	Paris	D
Milano	Roma	B
Milano	Roma	C
Milano	Venezia	B
Milano	Venezia	B
Roma	Milano	A

I dati suddetti individuano le ricezioni seguenti:

```
Ricezione(Roma)          = 70 + 70   = 140
Ricezione(Milano)                    = 100
Ricezione(Paris)                     = 200
Ricezione(Venezia)       = 70 + 70   = 140
```

Quindi, l’output corrispondente è:

Roma
Venezia

Q. Elencate le città italiane dalle quali non partono voli per l'estero.

L'elenco deve avere un'unica colonna.

Testate la soluzione usando questi dati:

CITTA

NomeCitta	Nazione
Catania	ITA
Lyon	FRA
Milano	ITA
Paris	FRA
Rimini	ITA
Roma	ITA
Venezia	ITA

VOLI

NomeCittaParte	NomeCittaArri
Catania	Venezia
Milano	Lyon
Paris	Catania
Paris	Lyon
Paris	Venezia
Roma	Paris
Venezia	Paris

L’output corrispondente è:

Catania
Rimini

a) Risolvete questa interrogazione usando operatori insiemistici.

b) Risolvete questa interrogazione usando un select annidato indipendente e l'operatore `NOT IN`.

c) Risolvete questa interrogazione usando un select annidato collegato e l'operatore `NOT EXISTS`.

R. Per ogni città, elencate i tipi degli aerei aventi la capienza minima tra tutti i voli che partono da quella città.

Testate la soluzione usando questi dati:

NomeCittaParte	TipoAereo	Capienza
Milano	A	100
Milano	B	70
Milano	C	70
Roma	C	70
Roma	D	200

L’output corrispondente è:

NomeCittaParte	TipoAereo
Milano	B
Milano	C
Roma	C

2. Considerate il db di riferimento.

A. Indicate, per ogni matricola, il suo stipendio e l'eventuale città del suo dipartimento.

B. Indicate, per ogni dipartimento, la città e l’eventuale stipendio del suo cassiere.

C. (AR) Elencate le matricole di ogni impiegato che ha il massimo stipendio.

Es: Con i valori dell’esempio iniziale si ha questo output:

MatriImpie
6
7

3. È dato il seguente db:

```
CITTA(NomeCitta, NomeStato, NAbita)
```

Elencate, per ogni stato, la città più abitata e la sua popolazione. Ordinate l’elenco rispetto allo stato.

Testate la soluzione usando questi dati:

NomeCitta	NomeStato	NAbita
A	X	300
B	X	200
C	Y	200
D	Y	100
E	Y	200

L’output corrispondente è:

NomeStato	NomeCitta	NAbita
X	A	300
Y	C	200
Y	E	200

4. È dato il db MUSICHE contenente le seguenti tabelle:

```
PERSONE(NomePersona, AnnoNascita)

ALBUM(IdeAlbum, TitoloAlbum, AnnoAlbum)

CANZONI(IdeCanzone, TitoloCanzone, AnnoCanzone*)
    AnnoCanzone = NULL => Canzone popolare (non è noto l'anno di composizione)

ALBUM_CANZO(IdeAlbum, IdeTraccia, IdeCanzone)
    CC: {IdeAlbum, IdeCanzone}
    AI: Nello stesso album, gli IdeTraccia devono partire dal valore 1 e
        devono essere contigui.

AUTORI_CANZO(NomePersona, IdeCanzone)

CANTANTI_CANZO(NomePersona, IdeCanzone)
```

A. Elencate i titoli degli album che contengono almeno una canzone popolare.

B. Elencate i titoli e i cantanti delle canzoni contenute nell'album avente l’identificatore 3.
Ordinate le righe rispetto all’identificatore della traccia.

C. Elencate i nomi delle persone che iniziano per `D` e che hanno cantato una canzone scritta da loro.

D. Elencate i nomi degli autori che non hanno mai cantato una canzone e i nomi dei cantanti che non hanno mai scritto una canzone.

L'elenco deve avere un'unica colonna.

E. Elencate gli identificatori degli l'album che contengono il massimo numero di canzoni.

F. Risolvete l'esercizio precedente senza usare l'attributo `IdeTraccia`.

G. Elencate gli identificatori degli album che contengono canzoni che sono tutte di un solo cantante e che contengono almeno tre canzoni di anni precedenti all’anno dell’album.

H. Elencate i nomi dei cantanti che non hanno mai cantato in un intero album da soli.

I. Elencate i nomi dei cantanti che non hanno mai cantato una canzone come solisti.

J. Elencate i nomi dei cantanti che hanno cantato soltanto come solisti.

Testate inserendo nell’input:

~ Un cantante solista in una canzone e non solista in un'altra (che, quindi, non deve comparire nel risultato).

~ Un cantante solista in due canzoni (che, quindi, deve comparire nel risultato).

~ Un cantante non solista in due canzoni (che, quindi, non deve comparire nel risultato).

K. Per ogni identificatore di canzone, elencate gli identificatori degli album nel quale la canzone è stata pubblicata per prima e mostrate l’anno di questa pubblicazione.

Testate la soluzione usando questi dati:

IdeAlbum	AnnoAlbum	IdeCanzone
1	2000	A
1	2000	B
2	2002	A
2	2002	B
2	2002	C
3	2004	C
4	2000	B

L’output corrispondente è:

IdeCanzone	IdeAlbum	AnnoAlbum
A	1	2000
B	1	2000
B	4	2000
C	2	2002

52.8.2 Viste

1. È dato il DB che gestisce gli impiegati di un'azienda composto da:

`DIPARTIMENTI(NomeDipa, Citta)`

`IMPIEGATI(MatriImpie, Stipendio, Ufficio, NomeDipa)`

Trovate la matricola dell'impiegato con stipendio minimo tra quelli che lavorano nell'ufficio `10` a `'Roma'`.

NB Una vista non è necessaria, ma semplifica la soluzione.

2. È dato il DB che gestisce la partecipazioni delle squadre ai campionati della massima serie di un dato sport (ad es., pallavolo).

Il DB è composto da:

`SQUADRE(NomeSquadra, AnnoFonda)`

`PARTECIPAZIONI(NomeSquadra, AnnoCampio, NPunti)`

Nessun campionato ha avuto due squadre arrivate prime con lo stesso numero di punti. (Cioè, nessun campionato si è concluso con uno spareggio.)

A1. Trovate la/e squadra/e che ha vinto più campionati, mediante una vista (che dovete definire opportunamente).

A2. Trovate la/e squadra/e che ha vinto più campionati, senza usare viste.

Es: Con i seguenti valori, dovete individuare la squadra A, perché ha vinto il campionato due volte (nel 2000 e nel 2010); invece, la squadra B lo ha vinto soltanto una volta (nel 2020).

`PARTECIPAZIONI`

NomeSquadra	AnnoCampio	NPunti
A	2000	100
B	2000	80
A	2010	200
B	2020	100
A	2020	60
B	2020	80