

Interrogazioni

L'istruzione base per le interrogazioni è **select**

select <i>ListaAttributi</i>	(<i>target list</i>)
from <i>ListaTabelle</i>	(<i>clausola from</i>)
[where <i>Condizione</i>]	(<i>clausola where</i>)

Più in dettaglio:

```
select AttrEspr [ [as] Alias] { , AttrEspr [ [as] Alias] }  
from    Tabella [ [as] Alias] { , Tabella [ [as] Alias] }  
[ where Condizione ]
```

seleziona le righe che soddisfano la condizione **where** fra quelle appartenenti al prodotto cartesiano delle tabelle in *ListaTabelle*, mostrandone gli attributi specificati.

Ogni colonna (tabella) può essere ridenominata con un alias.

Interrogazioni

Es. Data una base di dati che contiene le tabelle:

IMPIEGATO (Nome, Cognome, Dipart, Ufficio, Stipendio, Città)

DIPARTIMENTO (Nome, Indirizzo, Città)

```
select Stipendio/12 as SalarioMensile  
from Impiegato  
where Cognome = `Rossi`
```

genera una tabella con una colonna ridenominata *SalarioMensile* e tante righe quanti sono gli impiegati che si chiamano Rossi.

Se si indica **select *** si selezionano tutti gli attributi

Interpretazione algebrica delle interrogazioni

select $T_1.\textit{Attributo}_{11}, \dots T_h.\textit{Attributo}_{hm}$
from $\textit{Tabella}_1$ **as** $T_1, \dots \textit{Tabella}_h$ **as** T_h
where $\textit{Condizione}$

equivale a

$$\Pi_{T_1.\textit{Attributo}_{11} \dots T_h.\textit{Attributo}_{hm}} \left(\sigma_{\textit{Condizione}} (\textit{Tabella}_1 \times \textit{Tabella}_2 \times \dots \times \textit{Tabella}_h) \right)$$

X = prodotto cartesiano

Persone

Nome	Età	Reddito
Andrea	27	21
Aldo	25	15
Maria	55	42
Anna	50	35
Filippo	26	30
Luigi	50	40
Franco	60	20
Olga	30	41
Sergio	85	35
Luisa	75	87

Interrogazioni

Nome e reddito delle persone con meno di trenta anni

PROJ_{Nome, Reddito}(SEL_{Eta<30}(Persone))

```
select   nome, reddito
from     persone
where    eta < 30
```

```
select   p.nome as nome, p.reddito as reddito
from     persone as p
where    p.eta < 30
```

Interrogazioni su più tabelle

Se si vogliono estrarre informazioni da più tabelle, queste devono apparire come argomento della clausola **from** o in forma di lista o esplicitamente come (theta-)join fra le tabelle stesse.

Infatti, la clausola **from** denota la singola tabella a cui applicare le operazioni di proiezione (specificate dalla clausola **select**) e selezione (specificate dalla clausola **where**) per ottenere il risultato voluto.

Se viene applicata a una lista di tabelle denota il prodotto cartesiano fra tali tabelle.

In pratica tale situazione indica che si possono combinare tuple provenienti da qualsiasi tabella, senza che debbano soddisfare alcun vincolo interrelazionale.

Interrogazioni su più tabelle

Se si deve formulare un theta-join fra le tabelle, è possibile farlo in modo esplicito: **from** reggerà allora un **join** fra le tabelle (**join** esplicito).

```
select Studenti.Nome, Studenti.Cognome, Esami.voto  
from Studenti join Esami  
        on Esami.studente=Studenti.matricola
```

Poiché un theta-join equivale a un prodotto cartesiano seguito da una selezione (sulla condizione di join), si può indicare come *join implicito*, specificando solo la lista delle tabelle nella clausola **from** e inserendo la condizione di join nella clausola **where**.

```
select Studenti.Nome, Studenti.Cognome, Esami.voto  
from Studenti, Esami  
where Esami.studente=Studenti.matricola
```

Interrogazioni su più tabelle

Estrarre i nomi degli impiegati e le città dove lavorano.

```
select  Impiegato.Nome, Impiegato.Cognome,  
         Dipartimento.Città  
  
from    Impiegato, Dipartimento  
  
where    Impiegato.Dipart=Dipartimento.Nome  
oppure (con join esplicito)
```

```
select  Impiegato.Nome, Impiegato.Cognome,  
         Dipartimento.Città  
  
from    Impiegato join Dipartimento on  
         Impiegato.Dipart=Dipartimento.Nome
```

Interrogazioni su più tabelle

E' possibile omettere il nome della tabella per quegli attributi che non presentano ambiguità.

```
select  Impiegato.Nome, Cognome,  
         Dipartimento.Città  
from    Impiegato, Dipartimento  
where   Dipart = Dipartimento.Nome
```

e abbreviare ulteriormente il codice utilizzando gli alias

```
select  I.Nome, Cognome, D.Città  
from    Impiegato as I, Dipartimento as D  
where   Dipart = D.Nome
```

Clausola **where**

Ha come argomento una condizione logica, che ammette come operatori i predicati semplici (confronto attributo-costante o attributo-espressione):

=, <>, <, >, <=, >=

I predicati semplici possono essere modificati tramite gli operatori logici **and, or, not**.

not ha precedenza su **and** e **or**.

La precedenza fra **and** e **or** non è definita: quando si coordinano più predicati con **and** e **or** è bene esplicitare sempre le precedenze con le parentesi.

```
select    Nome  
from      Impiegato  
where     Cognome = 'Rossi'  
and       (Dipart='Amministratz' or Dipart='Produtz')
```

Operatore `like`

Per i confronti fra stringhe è definito anche l'operatore `like`.

Il confronto è effettuato con una stringa che può contenere i caratteri speciali `%` e `_`.

`_` rappresenta un carattere arbitrario

`%` rappresenta in numero arbitrario di caratteri (anche zero).

```
select *
```

```
from Impiegato
```

```
where Cognome like '_o%i'
```

La condizione è soddisfatta da Rossi Borroni Poli Pollastri ecc.

Gestione dei valori nulli

La gestione dei valori nulli, a seconda dell'implementazione, avviene attraverso una logica a due valori come in SQL-89, o a tre valori (vero, falso, unknown) come in SQL-2.

In MySQL logica a due valori con gestione del valore nullo.

Le condizioni sui valori nulli possono essere definite attraverso i predicati `is null` o `is not null`.

Duplicati

Al contrario di quanto avviene per l'algebra relazionale SQL **ammette** duplicati.

Quindi nella lista di città generata da

```
select      Città  
from        Persona  
where       Cognome = 'Rossi'
```

il nome di una città può comparire più volte.

Per evitare i duplicati SQL prevede la clausola **distinct** da inserire subito dopo **select**.

```
select      distinct Città  
from        Persona  
where       Cognome = 'Rossi'
```

Join

In SQL-2 è stata introdotta la seguente sintassi per esprimere il join ed estenderlo ai join esterni :

```
select AttrEspr [ [as] Alias ] { , AttrEspr [ [as] Alias ] }  
from    Tabella [ [as] Alias ]  
{ [ TipoJoin ] join Tabella [ [as] Alias ] on CondizioneJoin }  
[ where AltraCondizione ]
```

TipoJoin può assumere i valori

inner , **right** [**outer**] , **left** [**outer**] , **full** [**outer**]

inner è il default.

C'è anche l'estensione **natural** che implica la condizione di uguaglianza sugli attributi con lo stesso nome.

Paternità

Padre	Figlio
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo

Maternità

Madre	Figlio
Luisa	Maria
Luisa	Luigi
Anna	Olga
Anna	Filippo
Maria	Andrea
Maria	Aldo

Join implicito ed esplicito

Padre e madre di ogni persona

Con join implicito

```
select    paternita.figlio, padre, madre  
from      maternita, paternita  
where     paternita.figlio = maternita.figlio
```

Con join esplicito

```
select    paternita.figlio, padre, madre  
from      maternita join paternita on  
           paternita.figlio = maternita.figlio
```