



Basi di dati

Maurizio Lenzerini

***Dipartimento di Informatica e Sistemistica “Antonio Ruberti”
Università di Roma “La Sapienza”***

Anno Accademico 2023/2024

<http://www.dis.uniroma1.it/~lenzerini/?q=node/44>



SQL

- originariamente "**S**tructured **Q**uery **L**anguage", ora "nome proprio"
- è un linguaggio con varie funzionalità, che contiene:
 - il DDL (Data Definition Language)
 - il DML (Data Manipulation Language)
- ne esistono varie versioni
- analizziamo gli aspetti essenziali non i dettagli
- un po' di storia:
 - prima proposta **SEQUEL** (IBM Research, 1974);
 - prime implementazioni in SQL/DS (IBM) e Oracle (1981);
 - dal 1983 ca., "standard di fatto"
 - standard (1986, poi 1989, poi **1992**, 1999, e infine 2003):
recepito solo in parte



SQL-92

- è un linguaggio ricco e complesso
- ancora nessun sistema mette a disposizione tutte le funzionalità del linguaggio
- 3 livelli di aderenza allo standard:
 - **Entry SQL**: abbastanza simile a SQL-89
 - **Intermediate SQL**: caratteristiche più importanti per le esigenze del mercato; supportato dai DBMS commerciali
 - **Full SQL**: funzioni avanzate, in via di inclusione nei sistemi
- i sistemi offrono funzionalità non standard
 - incompatibilità tra sistemi
 - incompatibilità con i nuovi standard (es. trigger in SQL:1999)
- Nuovi standard conservano le caratteristiche di base di SQL-92:
 - **SQL:1999** aggiunge alcune funzionalità orientate agli oggetti
 - **SQL:2003** aggiunge supporto per dati XML



Utilizzo di un DBMS basato su SQL

- Un DBMS basato su SQL consente di gestire (diverse) basi di dati relazionali; dal punto di vista sistemistico è un **server**
- Quando ci si connette ad un DBMS basato su SQL, si deve indicare, implicitamente o esplicitamente, su quale basi di dati si vuole operare
- Se si vuole operare su una base di dati non ancora esistente, si utilizzerà un meccanismo messo a disposizione dal server per la sua creazione
- Coerentemente con la filosofia del modello relazionale, una base di dati in SQL è caratterizzata dallo **schema** (livello intensionale) e da una **istanza** (quella corrente -- livello estensionale)
- In più, una base di dati SQL è caratterizzata da un insieme di **meta-dati** (ossia “dati sui dati”, il catalogo – vedi dopo)



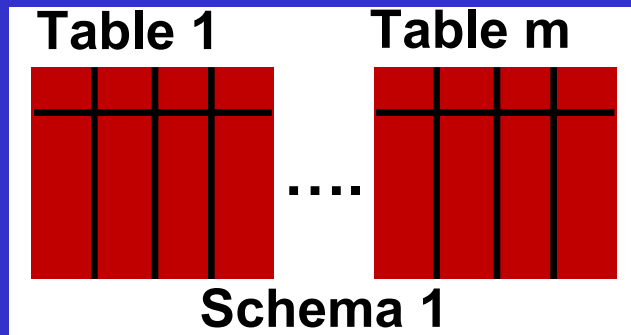
SQL e modello relazionale

- **Attenzione:** una tabella in SQL è definita come un multiinsieme di tuple
- In particolare, se una tabella non ha una primary key o un insieme di attributi definiti come unique (vedi dopo), allora potranno comparire due tuple uguali nella tabella; ne segue che una tabella SQL **non** è in generale una relazione
- Se invece una tabella ha una primary key o comunque un insieme di attributi definiti come superchiavi, allora non potranno mai comparire nella tabella due tuple uguali e quindi in questo caso la tabella è una relazione. Per questo, è consigliabile definire almeno una primary key per ogni tabella: per poi trattare quella tabella coerentemente con la definizione del modello relazionale
- Si noti comunque che, anche partendo da tabelle senza duplicati, eseguendo delle query potremo ottenere tabelle che i duplicati li hanno.

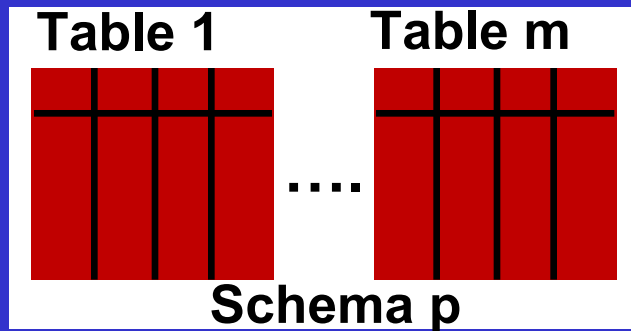
Basi di dati, schemi e tabelle in PostgreSQL

Installazione server PostgreSQL

Database 1

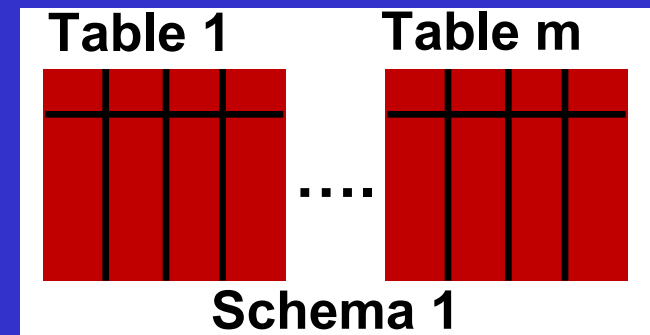


⋮

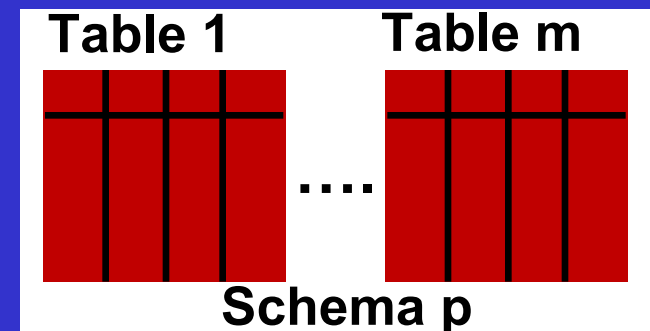


.....

Database n



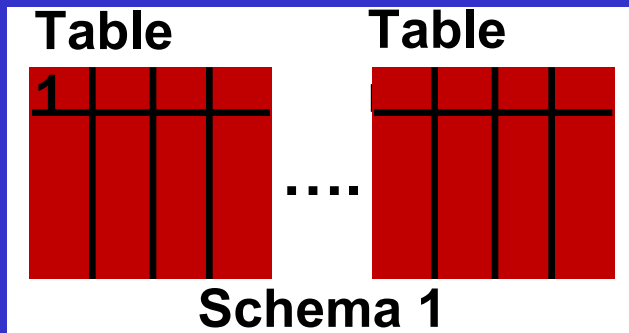
⋮



Basi di dati, schemi e tabelle in PostgreSQL

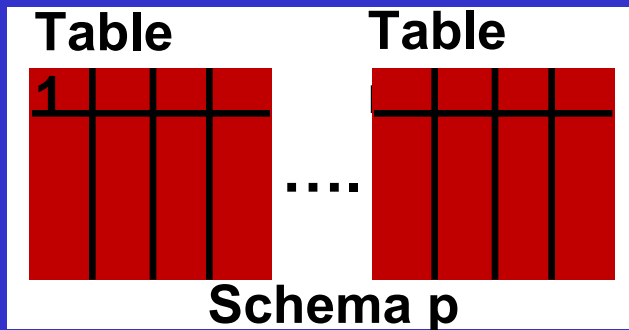
Installazione server PostgreSQL

Database 1



Schema 1

⋮

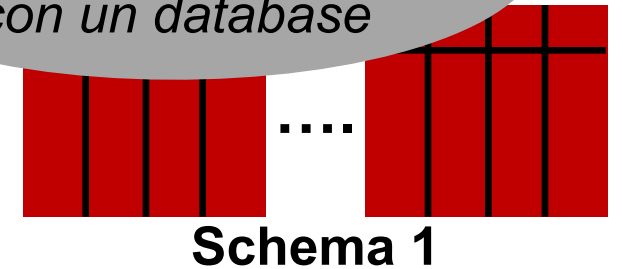


Schema p

.....

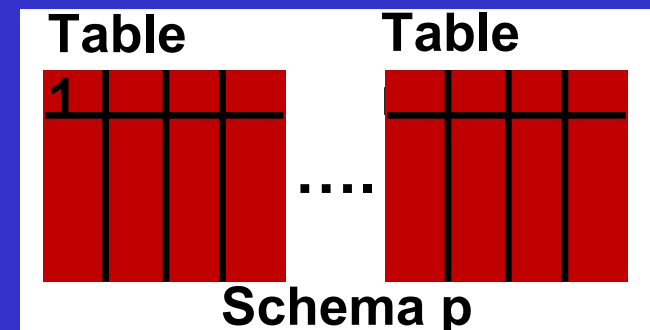
Database n

*ogni utente si
collega
con un database*



Schema 1

⋮

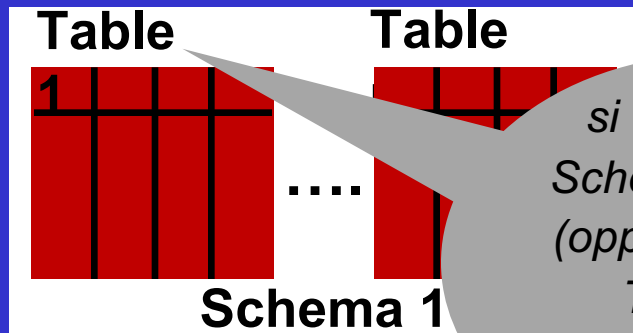


Schema p

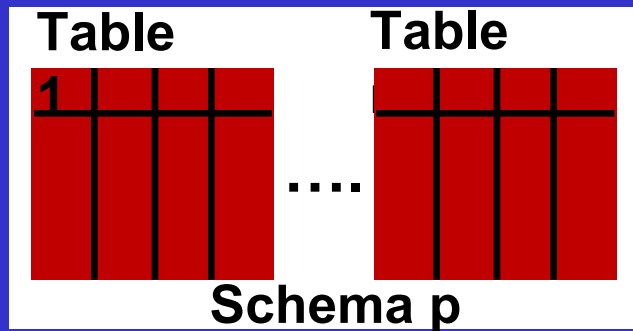
Basi di dati, schemi e tabelle in PostgreSQL

Installazione server PostgreSQL

Database 1

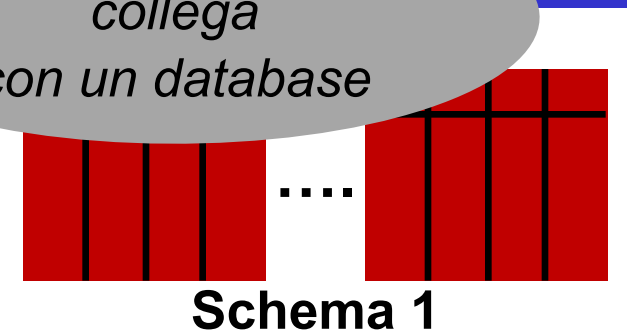


⋮



*si denota con
Schema1.Table1
(oppure con solo
Table1 se
Schema1
è implicito)*

Database n



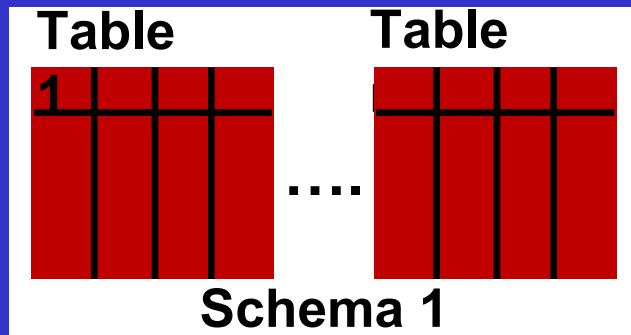
*un utente connesso
con Database1 "vede" tutti
gli schemi di Database1
e nessuno schema di un
altro Database (ad
esempio Database2)*

Schema p

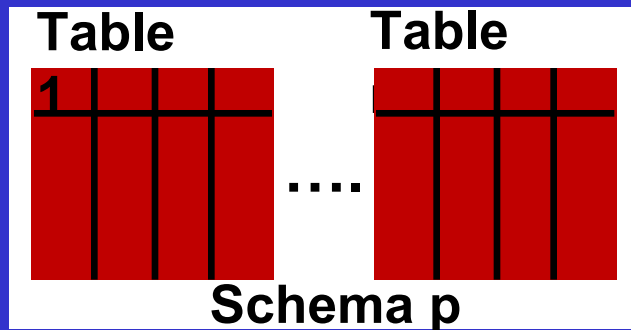
Basi di dati, schemi e tabelle in PostgreSQL

Installazione server PostgreSQL

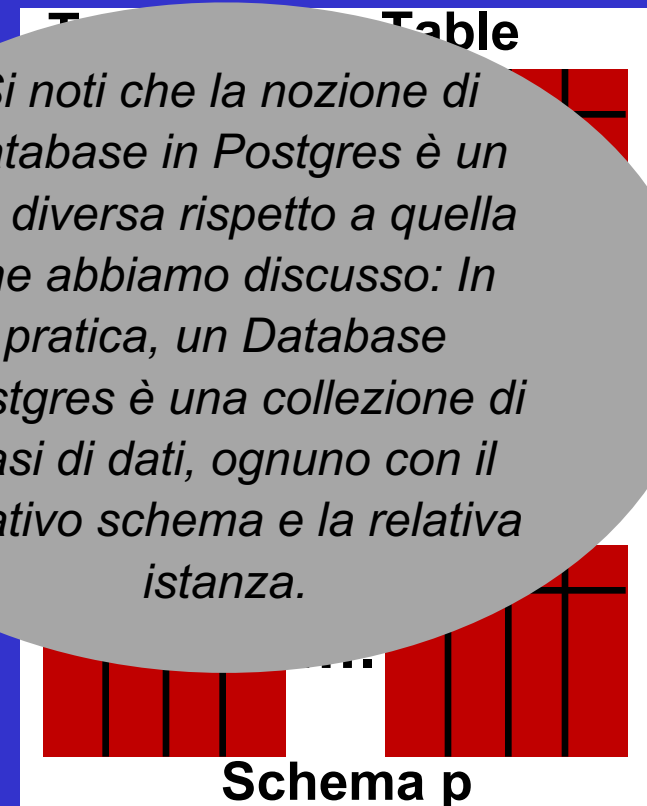
Database 1



⋮



Database n



Si noti che la nozione di Database in Postgres è un po' diversa rispetto a quella che abbiamo discusso: In pratica, un Database Postgres è una collezione di basi di dati, ognuno con il relativo schema e la relativa istanza.



Osservazione importante

Nel seguito di queste slides illustreremo **gli aspetti essenziali di SQL**, puntando a capire i concetti che lo caratterizzano e **NON tutti i dettagli**.

Nel momento in cui un utente del linguaggio lo utilizza in progetti reali deve necessariamente acquisire maggiori dettagli e giungere ad una più approfondita conoscenza del linguaggio, in tutti i suoi aspetti. Ma questo sarà possibile solo se avrà acquisito e compreso i concetti essenziali che qui miriamo ad impartire.

Come si fa ad acquisire la conoscenza su tutti i dettagli? Si deve consultare il manuale del linguaggio, facendo in particolare riferimento alla formulazione del linguaggio nel DBMS che viene utilizzato. Tali manuali si trovano gratuitamente in rete.

Si invitano gli studenti ad abituarsi a consultare il manuale di SQL, facendo riferimento ad esempio alla sua realizzazione in PostgreSQL. Anche imparare a consultare i manuali fa parte della preparazione di un buon ingegnere informatico.



3. Il Linguaggio SQL

3.1 Definizione dei dati

1. **interrogazioni semplici**
2. definizione dei dati
3. manipolazione dei dati
4. interrogazioni complesse
5. ulteriori aspetti



Convenzioni sui nomi

- Ogni **tabella** si denota con *NomeSchema . NomeTabella*
- Quando l'ambiguità sullo schema non sussiste (per esempio quando istruiamo il sistema a fare riferimento una volta per tutte ad uno specifico schema, che diventa implicito), si può omettere *NomeSchema* . e scrivere semplicemente *NomeTabella*
- Ogni **attributo** di una tabella si denota con *NomeSchema . NomeTabella . Attributo*
- Quando l'ambiguità sullo schema non sussiste si può ancora una volta omettere *NomeSchema* . e scrivere *NomeTabella . Attributo*
- Quando anche l'ambiguità sulla tabella non sussiste (ad esempio all'interno di una query in cui si usa una sola tabella con quel nome), si può omettere anche *NomeTabella* . e scrivere semplicemente *Attributo*

Istruzione select (versione elementare)

- L'istruzione di interrogazione in SQL è

select

che definisce una interrogazione (query) e restituisce il risultato della valutazione di quella query sulla base di dati in forma di tabella. La sua forma elementare è:

```
[  
select  Attributo,...,Attributo  
from    Tabella  
where   Condizione
```

Istruzione select (versione elementare)

- L'istruzione di interrogazione in SQL è

select

che definisce una interrogazione (query) e **restituisce il risultato della valutazione di quella query sulla base di dati in forma di tabella**. La sua forma elementare è:

select *Attributo,...,Attributo*
from *Tabella*
where *Condizione*

- le tre parti vengono di solito chiamate

- **target list**
- **clausola from**
- **clausola where**

Istruzione select (versione elementare)

- L'istruzione di interrogazione in SQL è

select

che definisce una interrogazione (query) e **restituisce il risultato della valutazione di quella query sulla base di dati in forma di tabella**. La sua forma elementare è:

```
select  Attributo,...,Attributo  
from    Tabella  
where   Condizione
```



- le tre parti vengono di solito chiamate
 - **target list**
 - **clausola from**
 - **clausola where**

Istruzione `select` (versione elementare)

- L'istruzione di interrogazione in SQL è

`select`

che definisce una interrogazione (query) e **restituisce il risultato della valutazione di quella query sulla base di dati in forma di tabella**. La sua forma elementare è:

```
select  Attributo,...,Attributo  
from    Tabella  
where   Condizione
```



- le tre parti vengono di solito chiamate
 - **target list**
 - **clausola from**
 - **clausola where**



Istruzione `select` (versione elementare): **semantica**

La semantica di

<code>select</code>	<i>Attributo, ..., Attributo</i>
<code>from</code>	<i>Tabella</i>
<code>where</code>	<i>Condizione</i>

si può descrivere così: ogni tupla t della tabella il cui nome *Tabella* è indicato nella clausola `from` viene analizzata. Se t non soddisfa la condizione nella clausola `where`, allora viene ignorata. Altrimenti da t viene prodotta la target list secondo quanto specificato nella target list che appare dopo `select` e la tupla risultante da tale target list viene inserita nel risultato.

Il risultato della esecuzione della query (la tabella che contiene le tuple calcolate) viene restituito nel canale di output del sistema (e riportato all'utente per la visualizzazione). Vedremo successivamente cosa occorre fare per memorizzarlo nella base di dati (ad esempio in una nuova tabella)



Istruzione `select` (versione elementare): semantica

La semantica di

<code>select</code>	<i>Attributo ... Attributo</i>
<code>from</code>	<i>Tabella</i>
<code>where</code>	<i>Condizione</i>

che abbiamo descritto chiarisce che l'istruzione è **analoga** alla seguente espressione dell'algebra relazionale

$$\text{PROJ}_{\text{Attributo}, \dots, \text{Attributo}}(\text{SEL}_{\text{Condizione}}(\text{Tabella}))$$

Perché diciamo che è “analoga” e non “equivalente”? Perché in SQL la tabella ed il risultato possono contenere duplicati, mentre nel modello relazionale le relazioni sono insiemi (non multiinsiemi) e quindi il risultato di una espressione dell'algebra, essendo una relazione, è anch'essa **sempre** un insieme di tuple. Ne segue che l'unica cosa che possiamo asserire è che, per ogni base di dati, l'istruzione `select` di SQL fornisce lo stesso risultato dell'espressione dell'algebra relazionale a meno dei duplicati che può contenere. È in questo senso che usiamo il termine “analoga”.



maternita

madre	figlio
Luisa	Maria
Luisa	Luigi
Anna	Olga
Anna	Filippo
Maria	Andrea
Maria	Aldo

paternita

padre	figlio
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo

Lo schema di questa base di dati è S

persone

<u>nome</u>	eta	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

*nelle slides che seguono
assumiamo che persone
diverse abbiano nomi
diversi*



Selezione e proiezione

Vogliamo nome e reddito delle persone con meno di 30 anni.

$\text{PROJ}_{\text{nome, reddito}}(\text{SEL}_{\text{eta} < 30}(\text{persone}))$

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

Qui e nelle prossime slides, per le query SQL mostriamo talvolta anche le “analoghe” espressioni dell'algebra relazionale

nome	reddito
Andrea	21
Aldo	15
Filippo	30



Selezione e proiezione

Vogliamo nome e reddito delle persone con meno di 30 anni.

Da ora in poi assumiamo di essere nell'ambito dello schema S (che quindi è implicito) e quindi possiamo omettere il nome della schema. In questo caso possiamo quindi scrivere:

nome	reddito
Andrea	21
Aldo	15
Filippo	30

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



Ricordiamo le convenzioni sui nomi

Nella query che vediamo qui sotto non c'è ambiguità su quali sono gli attributi: essi sono certamente quelli della tabella “persone”. Quindi la query

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

si può scrivere anche come:

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



SELECT: “as” per ridenominazione

“as” nella lista degli attributi serve a ridenominare gli attributi, specificando esplicitamente un nome per un attributo del risultato. Quando per un attributo manca tale ridenominazione, il nome dell’attributo nel risultato sarà uguale a quello che compare nella tabella menzionata nella clausola **from**.

Esempio:

```
select nome as name, reddito as salary
from persone
where eta < 30
```

restituisce come risultato una tabella con due attributi, il primo di nome **name** ed il secondo di nome **salary**

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

restituisce come risultato una tabella con due attributi, il primo di nome **nome** ed il secondo di nome **reddito**

SELECT: “as” per alias

“as” serve anche ad assegnare un nuovo nome (alias) alle tabelle nell’ambito di una query. Ad esempio:

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

ridenominazione

si può scrivere anche:

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

ridenominazione

o anche:

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


SELECT: “as” per alias

“as” serve anche ad assegnare un nuovo nome (alias) alle tabelle nell’ambito di una query. Ad esempio:

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

ridenominazione

si può scrivere anche:

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

alias

o anche:

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

Nota: “as” si
può anche omettere



Proiezione in SQL

Cognome e filiale di tutti gli impiegati

impiegati

matricola	cognome	filiale	stipendio
7309	Neri	Napoli	55
5998	Neri	Milano	64
9553	Rossi	Roma	44
5698	Rossi	Roma	64

PROJ *cognome, filiale* **(impiegati)**

Proiezione: attenzione ai duplicati

```
select cognome,  
       filiale  
from impiegati
```

cognome	filiale
Neri	Napoli
Neri	Milano
Rossi	Roma
Rossi	Roma

senza "distinct":
con duplicati

```
select distinct cognome,  
       filiale  
from impiegati
```

cognome	filiale
Neri	Napoli
Neri	Milano
Rossi	Roma

con "distinct":
senza duplicati

Selezione senza proiezione

Nome, età e reddito delle persone con meno di 30 anni

SEL_{eta<30}(persone)

```
select *  
from persone  
where eta < 30
```

dammi tutti gli
attributi

è un'abbreviazione per:

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

tutti gli
attributi



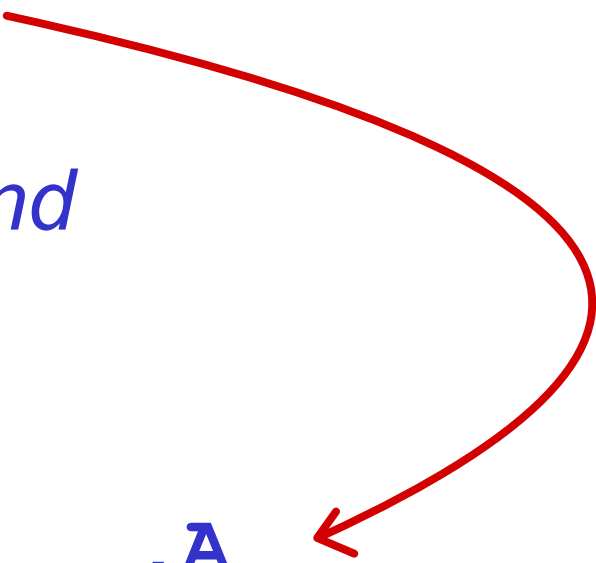
SELECT con asterisco

Data una tabella R sugli attributi A_1, \dots, A_n

```
select  *  
from    R  
where   cond
```

equivale a

```
select   $A_1, \dots, A_n$   
from    R  
where   cond
```





Proiezione senza selezione

Nome e reddito di tutte le persone

PROJ_{nome, reddito}(persone)

```
select nome, reddito  
from persone
```

è un'abbreviazione per:

```
select p.nome, p.reddito  
from persone p  
where true
```

Condizione complessa nella clausola “where”

Fino ad ora abbiamo usato espressioni semplici (ossia atomiche, formate da una sola condizione elementare).

Ovviamente, però, nella clausola where possono comparire espressioni booleane qualunque, semplici o complesse, ossia formate con i classici operatori booleani ed eventualmente parentesi. Ad esempio:

```
select *  
from   persone  
where  reddito > 25 and (eta < 30 or eta > 60)
```

Condizioni con operatore “LIKE”

Nelle condizioni che compaiono nella clausola `where` si possono usare **molte operatori che SQL mette a disposizione**. Rimandiamo al manuale del linguaggio per avere un quadro completo di tali operatori.

Menzioniamo qui l'operatore `like` che consente di verificare che una stringa appartenga al linguaggio definito da una **espressione regolare**. Ad esempio, se vogliamo conoscere quali sono le persone che hanno un nome che inizia per 'A', ha 'd' come terza lettera e può continuare con altri caratteri, scriviamo la query:

```
select *  
from   persone  
where  nome like 'A_d%'
```

espressione regolare
 $(\text{'A'} + \Sigma + \text{'d'})^*$
[Σ denota l'alfabeto]



Gestione dei valori nulli – “is null” e “is not null”

Nelle condizioni che compaiono nella clausola `where` si possono usare anche i predicati “is null” e “is not null” per gestire i valori nulli (già visti in algebra relazionale)

Vogliamo le persone la cui età è o potrebbe essere maggiore di 40

SEL `eta > 40 OR eta IS NULL` (impiegati)

```
select *  
from   persone  
where  età > 40 or età is null
```

Espressioni nella target list

Fino ad ora abbiamo usato solo nomi di attributi nella target list (quella che appare dopo `select`). In realtà ogni elemento della target list può essere una espressione che fa uso dei valori memorizzati negli attributi delle tuple del risultato. Ad esempio:

*espressione aritmetica che
fa uso del valore dell'attributo
reddito e lo divide per 2*

```
select età, reddito/2
from persone
where nome = 'Luigi'
```

*in assenza di
ridenominazione, il nome
dell'attributo nella tabella
risultato è uguale
all'espressione*

Risultato:

età	reddito/2
50	20



Espressioni nella target list

Nella target list può comparire un numero qualunque di espressioni e all'interno di tali espressioni possono ovviamente comparire anche costanti (nell'esempio precedente abbiamo usato la costante 2). Ulteriore esempio:

*espressione costituita da
una costante di tipo stringa*

```
select 'Luigi' as nomePersona, età,  
       reddito/2 as redditoSemestrale  
from   persone  
where  nome = 'Luigi'
```

Risultato:

nomePersona	età	redditoSemestrale
Luigi	50	20



Esercizio 1

Calcolare la tabella ottenuta dalla tabella **persone** ignorando l'attributo età, selezionando solo le persone con reddito tra 20 e 30, aggiungendo un attributo che ha, in ogni tupla, un valore booleano che indica se la persona corrispondente a quella tupla sta sotto i 50 anni o no ed aggiungendone un altro che indica il reddito mensile. Mostrare poi il risultato dell'interrogazione.

persone

<u>nome</u>	eta	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

```
SELECT  NOME, REDDITO, ETÀ < 50 AS GIOVANE,  
        REDDITO / 12  
FROM    PERSONE  
WHERE   REDDITO ≥ 20    AND    REDDITO ≤ 30
```

Soluzione esercizio 1

*espressione
booleana*

```
select nome, reddito, età < 50 as sotto50,  
       reddito/12 as redditoMensile  
from   persone  
where  reddito >= 20 and reddito <= 30
```

Risultato:

nome	reddito	sotto50	redditoMensile
Andrea	21	true	1.75
Filippo	30	true	2.50
Franco	20	false	1,67



Esercizio 2

Calcolare la tabella ottenuta dalla tabella **persone** selezionando solo quelli con età minore di 30 o maggiore di 60, proiettando i dati sugli attributi **nome** e **reddito**, ed aggiungendo un attributo che ha, in ogni tupla, il valore dell'anno in corso.

Mostrare il risultato dell'interrogazione

persone

<u>nome</u>	eta	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



Soluzione esercizio 2

```
select nome, reddito, 2022 as annoInCorso
from persone
where età < 30 or età > 60
```

*espressione
costante*

nome	reddito	annoInCorso
Andrea	21	2022
Aldo	21	2022
Filippo	30	2022
Sergio	35	2022
Luisa	87	2022



Selezione, proiezione e join

- Le istruzioni **select** che abbiamo visto finora hanno una sola tabella nella clausola **from** e quindi permettono di realizzare:
 - selezioni
 - proiezioni
 - ridenominazioni
- Il **join** (e i prodotti cartesiani) si possono realizzare indicando due o più tabelle nella clausola **from**, separate da virgola

La forma base della select: sintassi e semantica

La forma base delle select in SQL è:

```
[select <target list>
from   R1,R2,...,Rn
where  <condizione>
```

lista di attributi

lista di relazioni

La sua semantica si può descrivere semplicemente dicendo che essa è **analoga** all'espressione dell'algebra relazionale:

IL RISULTATO È LO
STESSO CHE SI
OTTERREBBE CALCOLANDO
IL PROD. CART.

$$\text{PROJ}_{\text{<target list>}}^3 (\text{SEL}_{\text{<condizione>}}^2 (R1 \times R2 \times \dots \times Rn)^1)$$

Ogni tupla t del prodotto cartesiano viene analizzata. Se t non verifica la condizione della clausola WHERE, essa viene ignorata. Se invece t verifica la condizione della clausola WHERE, allora da t viene prodotta la target list secondo la proiezione specificata nella clausola SELECT e la tupla risultante da tale target list viene inserita nel risultato

La forma base della select: sintassi e semantica

Abbiamo appena detto che la semantica di

```
select <target list>  
from   R1,R2,...,Rn  
where  <condizione>
```

si può descrivere come:

PROJ_{<target list>} (**SEL**_{<condizione>} (R1 × R2 × ... × Rn))

Attenzione: questo non significa che il DBMS calcola davvero il prodotto cartesiano di R1,R2,...,Rn!

Significa che il **risultato ottenuto è lo stesso di quello che** si ottiene calcolando prima il prodotto cartesiano delle tabelle nella clausola from, poi eseguendo la selezione sulla base della clausola where e poi eseguendo la proiezione sulla base della clausola select (con **distinct** il tutto avviene eliminando eventuali duplicati). **Il vero modo con cui il DBMS giunge al risultato dipende dall'algoritmo interno che usa, algoritmo che farà di tutto per evitare di calcolare il costoso prodotto cartesiano.**

SQL e algebra relazionale (1)

Date le relazioni: $R1(A1,A2)$ e $R2(A3,A4)$

```
select R1.A1, R2.A4
from   R1, R2
where  R1.A2 = R2.A3
```

è analoga quindi a:

$\text{PROJ}_{A1,A4} (\text{SEL}_{A2=A3} (R1 \times R2))$

a sua volta equivalente a

$\text{PROJ}_{A1,A4} (\text{SEL}_{A2=A3} (R1 \text{ JOIN } R2))$

a sua volta equivalente al Theta-join:

$\text{PROJ}_{A1,A4} (R1 \text{ JOIN}_{A2=A3} R2)$

Siccome R1 e R2 non hanno attribute in comune, il join naturale corrisponde al prodotto cartesiano



SQL e algebra relazionale (2)

Possono essere necessarie ridenominazioni

- nella target list (per avere nomi di attributi significativi negli attributi del risultato)
- nel prodotto cartesiano (in particolare, introdurre alias consente di riferirsi due o più volte alla stessa tabella)

Esempio:

```
select X.A1 as B1, ...  
from    R1 as X, R2 as Y, R1 as Z  
where   X.A2 = Y.A3 and Y.A4 = Z.A1
```

che, come al solito, si scrive anche senza “as”

```
select X.A1 B1, ...  
from    R1 X, R2 Y, R1 Z  
where   X.A2 = Y.A3 and Y.A4 = Z.A1
```

SQL e algebra relazionale: esempio

Date le tabelle: $R1(A1,A2)$ e $R2(A3,A4)$
la query in SQL

```
select distinct X.A1 as B1, Y.A4 as B2
from    R1 as X, R2 as Y, R1 as Z
where   X.A2 = Y.A3 and Y.A4 = Z.A1
```

è equivalente alla query in algebra relazionale:

```
RENB1,B2←A1,A4 (
  PROJA1,A4 (SELA2 = A3 and A4 = C1 (
    R1 JOIN R2 JOIN RENC1,C2 ← A1,A2 (R1))))
```

Il self-join in SQL

Come sappiamo già, il self-join è un join in cui la stessa relazione compare sia come operando sinistro sia come operando destro ed è cruciale quando dobbiamo combinare due tuple della stessa relazione. Supponiamo ad esempio di volere le coppie di persone con lo stesso reddito.

persone


<u>nome</u>	eta	reddito
Andrea	27	21
Aldo	25	15
Maria	55	42
Anna	50	35
Filippo	26	21

Il self-join in SQL

Come sappiamo già, il self-join è un join in cui la stessa relazione compare sia come operando sinistro sia come operando destro ed è cruciale quando dobbiamo combinare due tuple della stessa relazione. Supponiamo ad esempio di volere le coppie di persone con lo stesso reddito. È immediato verificare che le due tuple collegate dalla linea rossa formano una coppia che soddisfa la condizione. Ma come facciamo a combinarle?

persone

<u>nome</u>	eta	reddito
Andrea	27	21
Aldo	25	15
Maria	55	42
Anna	50	35
Filippo	26	21



Il self-join in SQL

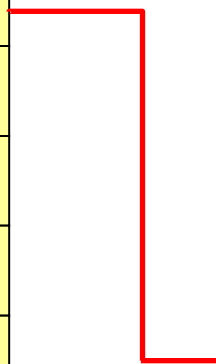
Come sappiamo già, il self-join è un join in cui la stessa relazione compare sia come operando sinistro sia come operando destro ed è cruciale quando dobbiamo combinare due tuple della stessa relazione. Supponiamo ad esempio di volere le coppie di persone con lo stesso reddito. Consideriamo una «copia virtuale» della relazione, ovviamente usando opportuni alias, e usiamo il join per combinare le due tuple collegate dalla linea rossa sulla condizione di uguale reddito.

persone as p1

<u>nome</u>	eta	reddito
Andrea	27	21
Aldo	25	15
Maria	55	42
Anna	50	35
Filippo	26	21

persone as p2

<u>nome</u>	eta	reddito
Andrea	27	21
Aldo	25	15
Maria	55	42
Anna	50	35
Filippo	26	21



Il self-join in SQL

```
select p1.nome, p1.eta, p1.reddito, p2.nome, p2.eta  
from persone p1, persone as p2  
where p1.reddito=p2.reddito
```

persone as p1

<u>nome</u>	eta	reddito
Andrea	27	21
Aldo	25	15
Maria	55	42
Anna	50	35
Filippo	26	21

persone as p2

<u>nome</u>	eta	reddito
Andrea	27	21
Aldo	25	15
Maria	55	42
Anna	50	35
Filippo	26	21





Il self-join in SQL

```
select p1.nome, p1.eta, p1.reddito, p2.nome, p2.eta  
from persone p1, persone as p2  
where p1.reddito=p2.reddito
```

Eseguendo questa query otteniamo:

p1.nome	p1.eta	p1.reddito	p2.nome	p2.eta
Andrea	27	21	Filippo	26
Andrea	27	21	Andrea	27
Filippo	26	21	Andrea	27
Filippo	26	21	Filippo	26
Aldo	25	15	Aldo	25
Maria	55	42	Maria	55
Anna	26	21	Anna	26

Il self-join in SQL

```
select p1.nome, p1.eta, p1.reddito, p2.nome, p2.eta  
from persone p1, persone as p2  
where p1.reddito=p2.reddito
```

Eseguendo questa query otteniamo:

p1.nome	p1.eta	p1.reddito	p2.nome	p2.eta
Andrea	27	21	Filippo	26
Andrea	27	21	Andrea	27
Filippo	26	21	Andrea	27
Filippo	26	21	Filippo	26
Aldo	25	15	Aldo	25
Maria	55	42	Maria	55
Anna	26	21	Anna	26

non
significative,
perché
chiaramente
ridondanti

Il self-join in SQL

```
select p1.nome, p1.eta, p1.reddito, p2.nome, p2.eta  
from persone p1, persone as p2  
where p1.reddito=p2.reddito and p1.nome<p2.nome
```

Eseguendo questa query otteniamo:

p1.nome	p1.eta	p1.reddito	p2.nome	p2.eta
Andrea	27	21	Filippo	26

lasciando solo
le tuple in cui
p1.nome viene
prima in ordine
alfabetico di
p2.nome
eliminiamo le
tuple non
significative



Altro esempio di self-join in SQL

Data la relazione **Volo(partenza,arrivo)**, ogni tupla della quale rappresenta un volo aereo da una certa città ad un'altra, vogliamo sapere quali sono le città raggiungibili da Roma con due voli.

```
SELECT DISTINCT V2.ARRIVO  
FROM      VOLO AS V1 , VOLO AS V2  
WHERE     V1.PARTENZA = "ROMA" AND  
          V1.ARRIVO = V2.PARTENZA
```



Altro esempio di self-join in SQL

È facile verificare che la query prevede di trovare due tuple t_1 e t_2 nella relazione Volo tale che $t_1.\text{partenza} = \text{'Roma'}$ e $t_1.\text{arrivo} = t_2.\text{partenza}$. Come abbiamo visto prima, questo si realizza con un self-join.

La query in algebra relazionale sarebbe:

$\text{PROJ}_a(\text{SEL}_{\text{partenza}=\text{'Roma'}}(\text{Volo}) \text{ JOIN}_{\text{arrivo}=p} \text{REN}_{p \leftarrow \text{partenza}, a \leftarrow \text{arrivo}}(\text{Volo}))$

In SQL la query è:

```
select V2.arrivo
from   Volo as V1, Volo as V2
where  V1.partenza='Roma' and
       V1.arrivo = V2.partenza
```



SQL: esecuzione delle interrogazioni

- Le espressioni SQL sono dichiarative e noi ne stiamo illustrando la semantica
- In pratica, i DBMS tentano di eseguire le operazioni in modo efficiente, ad esempio:
 - eseguono le selezioni al più presto
 - se possibile, eseguono join e **non** prodotti cartesiani
 - usano strutture ausiliarie, come gli indici
- La capacità dei DBMS di "**ottimizzare**" le interrogazioni rende (di solito) non necessario preoccuparsi dell'efficienza quando si specifica un'interrogazione
- È perciò più importante preoccuparsi della chiarezza (anche perché così è più difficile sbagliare ...)



maternita

madre	<u>figlio</u>
Luisa	Maria
Luisa	Luigi
Anna	Olga
Anna	Filippo
Maria	Andrea
Maria	Aldo

paternita

padre	<u>figlio</u>
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo

persone

<u>nome</u>	eta	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



Assunzioni

Nelle slide che seguono, facciamo queste assunzioni:

- Come abbiamo già detto, persone diverse hanno nomi diversi e non nulli: nome è chiave (primaria)
- Ogni figlio ha un solo padre (figlio è chiave primaria in paternità)
- Ogni figlio ha una sola madre (figlio è chiave primaria in maternità)
- I valori che troviamo nell'attributo nome delle tabelle paternita e maternita si trovano anche nell'attributo nome nella tabella persone (integrità referenziale)
- Se non esplicitamente detto, non ci preoccupiamo di eliminare i duplicati nel risultato delle query e quindi le espressioni dell'algebra relazionale che mostreremo sono analoghe (non necessariamente equivalenti) alle query SQL.



Esercizio 3: selezione, proiezione e join

I padri di persone che guadagnano più di venti milioni
(senza ripetizioni nel risultato)

Esprimere la query sia in algebra relazionale sia in SQL



Esercizio 3: soluzione

I padri di persone che guadagnano più di venti milioni
(senza ripetizioni nel risultato)

PROJ_{padre}(paternita JOIN_{figlio=nome} SEL_{reddito>20} (persone))

```
select distinct paternita.padre
from   persone, paternita
where  paternita.figlio = persone.nome
       and persone.reddito > 20
```



Esercizio 4: join

Padre e madre di ogni persona della quale entrambi i genitori sono noti.

Esprimere la query sia in algebra relazionale sia in SQL.



Esercizio 4: soluzione

Padre e madre di ogni persona della quale entrambi i genitori sono noti.

In algebra relazionale si calcola mediante il **join naturale**.

paternita JOIN maternita

In SQL:

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



Esercizio 4: soluzione

Se avessimo inteso la domanda come «padre e madre di ogni persona che appare nella tabella “persona” e della quale entrambi i genitori sono noti», allora avremmo dovuto usare un join in più:

In algebra:

$\text{PROJ}_{\text{figlio, padre, madre}} ((\text{paternita} \text{ JOIN } \text{maternita}) \text{ JOIN}_{\text{figlio=nome}} \text{persone})$

In SQL:

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



Esercizio 5: join e altre operazioni

Le persone che guadagnano più dei rispettivi padri, mostrando per ognuna il suo nome, il suo reddito e anche il reddito del padre

Esprimere la query sia in algebra relazionale sia in SQL



```

      PROJnome, reddito, RP
    (SELreddito>RP (RENNP,EP,RP ← nome,eta,reddito (persone)
      JOINNP=padre
      (paternita JOINfiglio =nome persone))
    )
  )
)

```

```
select figlio, f.reddito as reddito,
       p.reddito as redditoPadre
from   persone p, paternita t, persone f
where  p.nome = t.padre and
       t.figlio = f.nome and
       f.reddito > p.reddito
```

SELECT con join esplicito, sintassi

In SQL esiste un operatore che si può usare nella clausola **from** e che corrisponde al Theta-join.





Join esplicito

Padre e madre di ogni persona della quale entrambi sono noti.

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

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

**join
esplicito**



Esercizio 6: join esplicito

Le persone che guadagnano più dei rispettivi padri, mostrando per ognuna il suo nome, il suo reddito e anche il reddito del padre

Esprimere la query in SQL usando il join esplicito



SELECT con join esplicito, esempio

Le persone che guadagnano più dei rispettivi padri, mostrando per ognuna il suo nome, il suo reddito e anche il reddito del padre

```
select f.nome, f.reddito, p.reddito
from   persone p, paternita t, persone f
where  p.nome = t.padre and
       t.figlio = f.nome and
       f.reddito > p.reddito
```

*due applicazioni
dell'operatore
di join esplicito*

```
select f.nome, f.reddito, p.reddito
from   persone p join paternita t on p.nome = t.padre
      join persone f on t.figlio = f.nome
where  f.reddito > p.reddito
```



SELECT con join esplicito: ridenominazione

Ricordiamo che il risultato del join è una tabella che ha come attributi l'unione degli attributi dei due operandi.

```
select f.nome, f.reddito, p.reddito
from  persone p join paternita t on p.nome = t.padre
      join persone f on t.figlio = f.nome
where  f.reddito > p.reddito
```

Per esempio, nella query mostrata sopra, il join esplicito che compare nella clausola **from** dà come risultato una tabella con gli attributi: p.nome, p.eta, p.reddito, t.padre, t.figlio, f.nome, f.eta, ed f.reddito.

Si noti che il risultato di un join si può anche usare come una delle tabelle nella lista della clausola **from**, ma in questo caso occorre racchiudere il join esplicito tra parentesi tonde (e gli si può anche assegnare un alias, con la solita notazione, come vedremo anche più avanti):

```
select f.nome, f.reddito, p.reddito
from  (persone p join paternita t on p.nome = t.padre),
      persone f
where  f.reddito > p.reddito and t.figlio = f.nome
```



Ulteriore estensione: join naturale (meno diffuso)

PROJ_{figlio,padre,madre}(**paternita** JOIN **figlio**□**nome** **REN**_{nome}□**figlio(**maternita**))**

In algebra: paternita JOIN maternita

In SQL (con join esplicito):

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

In SQL (con natural join):

```
select paternita.figlio, padre, madre  
from maternita natural join paternita
```

come al solito: equi-join sugli attributi in comune, ovvero attributi che hanno lo stesso nome semplice (il nome semplice è quello ottenuto dal nome esteso ignorando nome di schema e nome di relazione)



Join esterno: "outer join"

Padre di ogni persona e, se nota, anche la madre

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

NOTA: "outer" si può anche omettere

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


maternita

madre	figlio
Luisa	Maria
Luisa	Luigi
Anna	Olga
Anna	Filippo
Maria	Andrea
Maria	Aldo

Padre di ogni persona e, se nota, anche la madre:

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

questa tupla dell'operando sinistro non si combina nel join e quindi compare nell'outer join con il valore NULL negli attributi dell'operando destro

paternita

padre	figlio
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo

Risultato:

figlio	padre	madre
Franco	Sergio	NULL
Olga	Luigi	Anna
Filippo	Luigi	Anna
Andrea	Franco	Maria
Aldo	Franco	Maria



Outer join, esempi

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

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

```
select nome, padre, madre  
from paternita full outer join maternita on  
       paternita.figlio = maternita.figlio  
       full outer join persone on  
       persone.nome = paternita.figlio or  
       persone.figlio = maternita.figlio
```

Outer join, esempi

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

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

```
select nome, padre, madre
from   paternita full outer join maternita
       on paternita.figlio = maternita.figlio
       full outer join persone on
       persone.nome = paternita.figlio or
       persone.figlio = maternita.figlio
```

*con questo full join non perdo le
persone che non compaiono in
alcuna delle due relazioni*

Ordinamento del risultato: order by

Nome e reddito delle persone con meno di 30 anni in ordine alfabetico

```
select nome, reddito  
from persone  
where eta < 30  
order by nome ASC
```



ordine
ascendente

```
select nome, reddito  
from persone  
where eta < 30  
order by nome desc
```



ordine
discendente



Ordinamento del risultato: order by

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

nome	reddito
Andrea	21
Aldo	15
Filippo	20

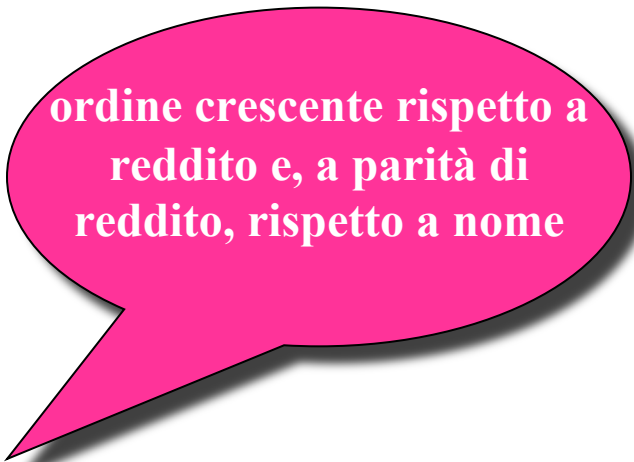
```
select nome, reddito  
from persone  
where eta < 30  
order by nome
```

nome	reddito
Aldo	15
Andrea	21
Filippo	20

Ordinamento del risultato: order by

Nome e reddito delle persone con meno di trenta anni in ordine crescente rispetto a reddito e, a parità di reddito, rispetto a nome

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



ordine crescente rispetto a
reddito e, a parità di
reddito, rispetto a nome



Limite alla dimensione del risultato: limit

Si può indicare un limite alla dimensione del risultato (con la clausola **limit** in SQL, con clausole diverse in altri sistemi), al fine di avere come risultato al massimo un prefissato numero di tuple

```
select nome, reddito
from   persone
where  eta < 30
order by nome
limit 2
```



Limite alla dimensione del risultato: limit

```
select nome, reddito  
from   persone  
where  eta < 30  
order by nome desc  
limit 2
```

nome	reddito
Andrea	21
Aldo	15



Operatori aggregati

Nelle espressioni della target list possiamo avere anche espressioni che calcolano valori a partire da insiemi di tuple:

- conteggio, minimo, massimo, media, totale

Sintassi base (semplificata):

Funzione ([distinct] EspressioneSuAttributi)



Operatori aggregati: **count**

Sintassi:

- conta il numero di tuple:

`count (*)`

- conta i valori di un attributo (considerando i duplicati):

`count (Attributo)`

- conta i valori distinti di un attributo:

`count (distinct Attributo)`

Operatore aggregato count: esempio e semantica

Esempio: Quanti figli ha Franco?

```
select count(*) as NumFigliDiFranco  
from   paternita  
where  padre = 'Franco'
```

Semantica: l'operatore aggregato (**count**), che conta le tuple, viene applicato al risultato della seguente interrogazione:

```
select *  
from   paternita  
where  padre = 'Franco'
```



Risultato di count: esempio

paternita

padre	figlio
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo

NumFigliDiFranco
2



count e valori nulli

```
select count(*)  
from persone
```

Risultato = numero di tuple
= 4

```
select count(reddito)  
from persone
```

Risultato = numero di valori
diversi da NULL
= 3

```
select count(distinct reddito)  
from persone
```

Risultato = numero di valori
distinti (escluso
NULL)
= 2

persone

nome	eta	reddito
Andrea	27	21
Aldo	25	NULL
Maria	55	21
Anna	50	35



Altri operatori aggregati

sum, avg, max, min

- ammettono come argomento un attributo o un'espressione (ma non “*”)
- **sum** e **avg**: argomenti numerici o tempo
- **max** e **min**: argomenti su cui è definito un ordinamento

Esempio: media dei redditi dei figli di Franco.

```
select avg(reddito)
from   persone join paternita on
       nome = figlio
where  padre = 'Franco'
```



Operatori aggregati e valori nulli

```
select avg(reddito) as redditoMedio  
from persone
```

persone

nome	eta	reddito
Andrea	27	30
Aldo	25	NULL
Maria	55	36
Anna	50	36

viene ignorato

redditoMedio
34

$(30+36+36)/3$



Operatori aggregati e target list

Un'interrogazione irragionevole (di chi sarebbe il nome?):

```
select nome, max(reddito)
from persone
```

L'interrogazione di sopra è irragionevole perché gli elementi della target list sono disomogenei: infatti abbiamo un valore di nome per ogni tupla, mentre abbiamo un valore di max(reddito) per tutta la tabella.

Affinché l'interrogazione sia ragionevole, la **target list** deve essere **omogenea**, ad esempio:

```
select min(eta), avg(reddito)
from persone
```




Operatori aggregati e raggruppamenti

- Nei casi visti in precedenza, gli operatori aggregati sono applicati all'insieme di tutte le tuple che formano il risultato di una query
- In molti casi, vorremmo che le funzioni di aggregazione venissero applicate a **gruppi di tuple** delle relazioni
- Per specificare i gruppi di tuple su cui applicare le funzioni, si utilizza la clausola **group by**:

group by *listaAttributi*



Semantica di interrogazioni con operatori aggregati e raggruppamenti

```
select <target list>  
from R  
group by Ai
```

```
SELECT CORSO, AVG(VOTO)  
FROM ESAME  
GROUP BY CORSO
```

1. Si esegue l'interrogazione **ignorando la group by** e la target list:

```
select *  
from R
```
2. Sulle tuple che risultano si formano i gruppi, dove ogni gruppo si ottiene raggruppando le **tuple che hanno lo stesso valore negli attributi che compaiono nella group by**. Si produce nel risultato una tupla per ogni gruppo e per ognuna di tali tuple si applica la target list, usando ovviamente gli operatori aggregati in essa presenti



Operatori aggregati e raggruppamenti: esempio

Il numero di figli di ciascun padre

```
select padre, count(*) as NumFigli  
from paternita  
group by padre
```

paternita

padre	figlio
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo

Operatori aggregati e raggruppamenti

Il numero di figli di ciascun padre

```
select padre, count(*) as NumFigli
from paternita
group by padre
```

paternita

padre	figlio
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo

gruppo di
padre='Sergio'

padre	NumFigli
-------	----------

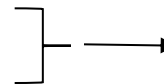
Operatori aggregati e raggruppamenti

Il numero di figli di ciascun padre

```
select padre, count(*) as NumFigli
from paternita
group by padre
```

paternita

padre	figlio
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo



padre	NumFigli
Sergio	1

Operatori aggregati e raggruppamenti

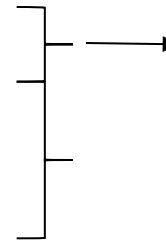
Il numero di figli di ciascun padre

```
select padre, count(*) as NumFigli
from paternita
group by padre
```

gruppo di
padre='Luigi'

paternita

padre	figlio
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo



padre	NumFigli
Sergio	1

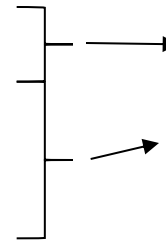
Operatori aggregati e raggruppamenti

Il numero di figli di ciascun padre

```
select padre, count(*) as NumFigli
from paternita
group by padre
```

paternita

padre	figlio
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo



padre	NumFigli
Sergio	1
Luigi	2

Operatori aggregati e raggruppamenti

Il numero di figli di ciascun padre

```
select padre, count(*) as NumFigli
from paternita
group by padre
```

gruppo di
padre='Franco'

paternita

padre	figlio
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo

padre	NumFigli
Sergio	1
Luigi	2

Operatori aggregati e raggruppamenti

Il numero di figli di ciascun padre

```
select padre, count(*) as NumFigli
from paternita
group by padre
```

paternita

padre	figlio		padre	NumFigli
Sergio	Franco	→	Sergio	1
Luigi	Olga	→	Luigi	2
Luigi	Filippo	→		
Franco	Andrea	→	Franco	2
Franco	Aldo	→		



Esercizio 7: group by

Massimo dei redditi per ogni gruppo di persone che sono maggiorenni ed hanno la stessa età (indicando anche l'età)

Esprimere la query in SQL

persone

nome	eta	reddito
-------------	------------	----------------



Esercizio 7: soluzione

Massimo dei redditi per ogni gruppo di persone che sono maggiorenni ed hanno la stessa età (indicando anche l'età)

```
select eta, max(reddito)
from   persone
where  eta > 17
group by eta
```



Raggruppamenti e target list

In una interrogazione che fa uso di `group by`, dovrebbero comparire solo target list “omogenee”, ovvero target list che comprendono, oltre a funzioni di aggregazione, **solamente** attributi che compaiono nella `group by`.

Esempio:

- Redditi delle persone, raggruppati per età (**non ragionevole**, perché la target list è disomogenea: potrebbero esistere più valori di reddito per le persone appartenenti allo stesso gruppo):

```
select eta, reddito
from persone
group by eta
```

- Media dei redditi delle persone, raggruppati per età (**ragionevole**, perché per ogni gruppo c'è una sola media dei redditi):

```
select eta, avg(reddito)
from persone
group by eta
```

Raggruppamenti e target list

La restrizione di target list omogenea sugli attributi nella `select` vale anche per interrogazioni che semanticamente sarebbero corrette (ovvero, per cui sappiamo che nella base di dati esiste un solo valore dell'attributo per ogni gruppo).

Esempio: i padri col loro reddito, e con reddito medio dei figli.

Target list disomogenea:

```
select padre, avg(f.reddito), p.reddito
from   persone f join paternita on figlio = nome
       join persone p on padre = p.nome
group by padre
```

sembra corretta, perché ogni padre ha un solo reddito,
ma SQL non lo sa e considera disomogenea la target list

Corretta:

```
select padre, avg(f.reddito), p.reddito
from   persone f join paternita on figlio = nome
       join persone p on padre = p.nome
group by padre, p.reddito
```



Target list disomogenea

Abbiamo visto che in una interrogazione che fa uso di **group by**, la target list dovrebbe essere omogenea.

Cosa succede se non lo è? PostgreSQL dà errore, ma alcuni sistemi non segnalano errore e restituiscono uno dei valori che sono associati al valore corrente degli attributi che formano il gruppo.

Esempio:

Redditi delle persone, raggruppati per età (**target list disomogenea**, perché potrebbero esistere più valori di reddito per lo stesso gruppo):

```
select eta, reddito
from persone
group by eta
```

ma MySQL, ad esempio, non dà errore, e sceglie per ciascun gruppo uno dei valori di reddito che compare nel gruppo e lo riporta nell'attributo reddito della target list.



Condizioni sui gruppi

Si possono anche imporre le condizioni di **selezione sui gruppi**. La selezione sui gruppi è **ovviamente diversa** dalla condizione che seleziona le tuple che devono formare i gruppi (clausola **where**). Per effettuare la selezione sui gruppi si usa la clausola **having**, che deve apparire dopo la “**group by**” e che di fatto opera un taglio sulle tuple del risultato della group by.

Esempio: i padri i cui figli hanno un reddito medio maggiore di 25.

```
select padre, avg(f.reddito)
from   persone f join paternita
       on figlio = f.nome
group by padre
having avg(f.reddito) > 25
```

Esercizio 8: where o having?

I padri i cui figli sotto i 30 anni hanno un reddito medio maggiore di 20

```
SELECT  PADRE
FROM    PERSONE P JOIN PATERNITA AS f
        ON f.FIGLIO = P.NOME
WHERE   P.ETÀ < 30
GROUP BY PADRE
HAVING  AVG (P.REDDITO) > 20
```




Esercizio 8: soluzione

I padri i cui figli sotto i 30 anni hanno un reddito medio maggiore di 20

```
select t.padre, avg(f.reddito)
from   persone f join paternita t
      on f.nome = t.figlio
where  f.eta < 30
group by t.padre
having avg(f.reddito) > 20
```



Sintassi, riassumiamo

SelectSQL ::=

select *ListaAttributiOEspressioni*
from *ListaTabelle*
[**where** *CondizioniSemplici*]
[**group by** *ListaAttributiDiRaggruppamento*]
[**having** *CondizioniAggregate*]
[**order by** *ListaAttributiDiOrdinamento*]
[**limit** *numero*]



Unione, intersezione e differenza

La **select** da sola non permette di eseguire l'unione

Serve un costrutto esplicito:

```
select ...  
union [all]  
select ...
```

Con **union**, i duplicati vengono eliminati (anche in presenza di proiezioni)

Con **union all** vengono mantenuti i duplicati



Notazione posizionale

```
select padre, figlio  
from paternita  
union  
select madre, figlio  
from maternita
```

Quali nomi per gli attributi del risultato? Dipende dal sistema:

- nuovi nomi decisi dal sistema, oppure
- quelli del primo operando, oppure
- ...



Risultato dell'unione

padre	figlio
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo
Luisa	Maria
Luisa	Luigi
Anna	Olga
Anna	Filippo
Maria	Andrea
Maria	Aldo



Differenza

```
select nome
from   impiegato
except / MINUS
select cognome as nome
from   impiegato
```

Nota: **except** elimina i duplicati

Nota: **except all** non elimina i duplicati

Vedremo che la differenza si può esprimere anche con **select** annidate.



Intersezione

```
select nome
from   impiegato
intersect
select cognome as nome
from   impiegato
```

equivale a

```
select distinct i.nome
from   impiegato i, impiegato j
where  i.nome = j.cognome
```

Nota: **intersect** elimina i duplicati

Nota: **intersect all** non elimina i duplicati