

Inizio SQL

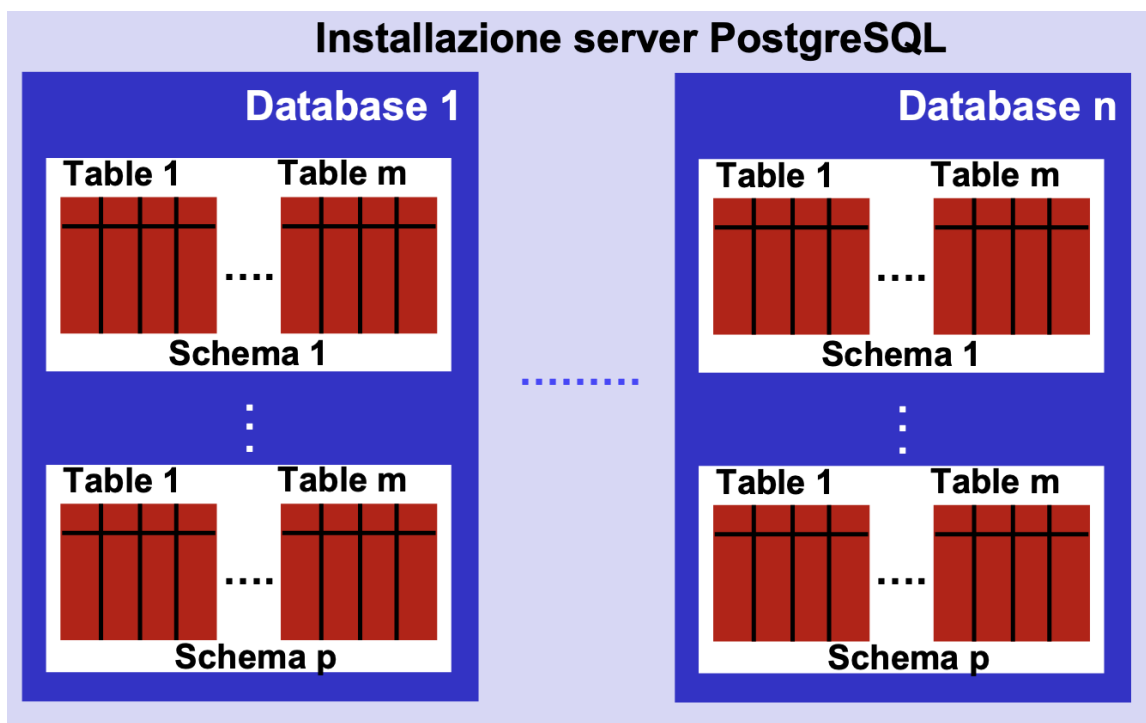
▼ Introduzione

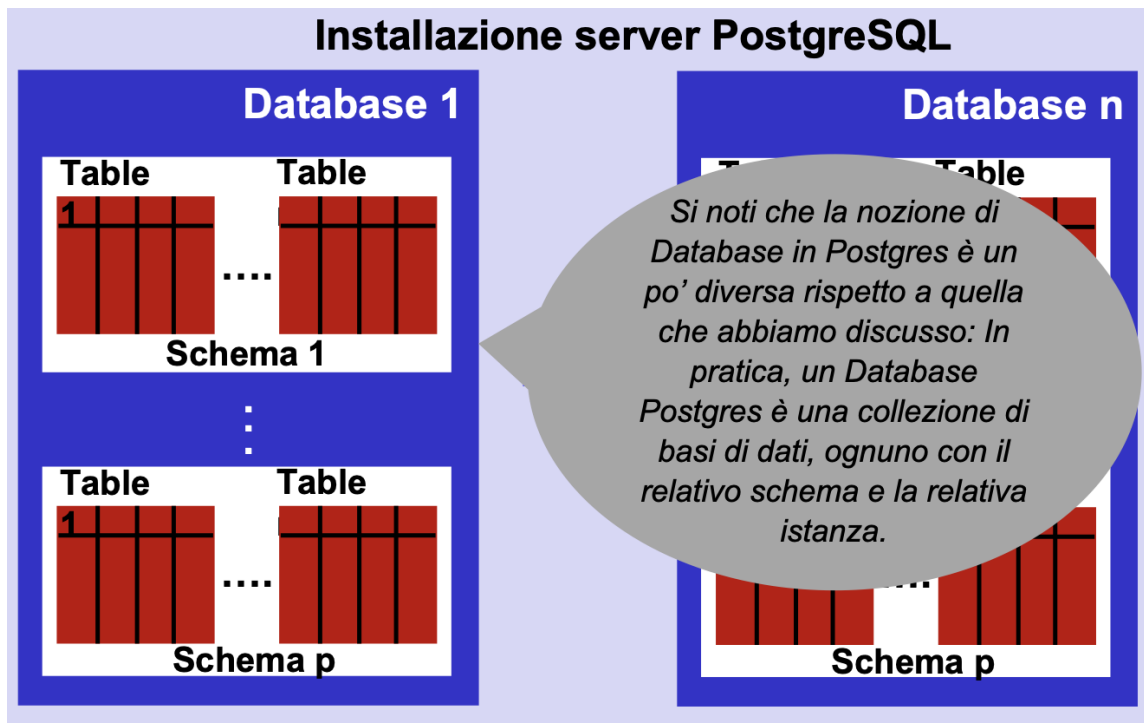
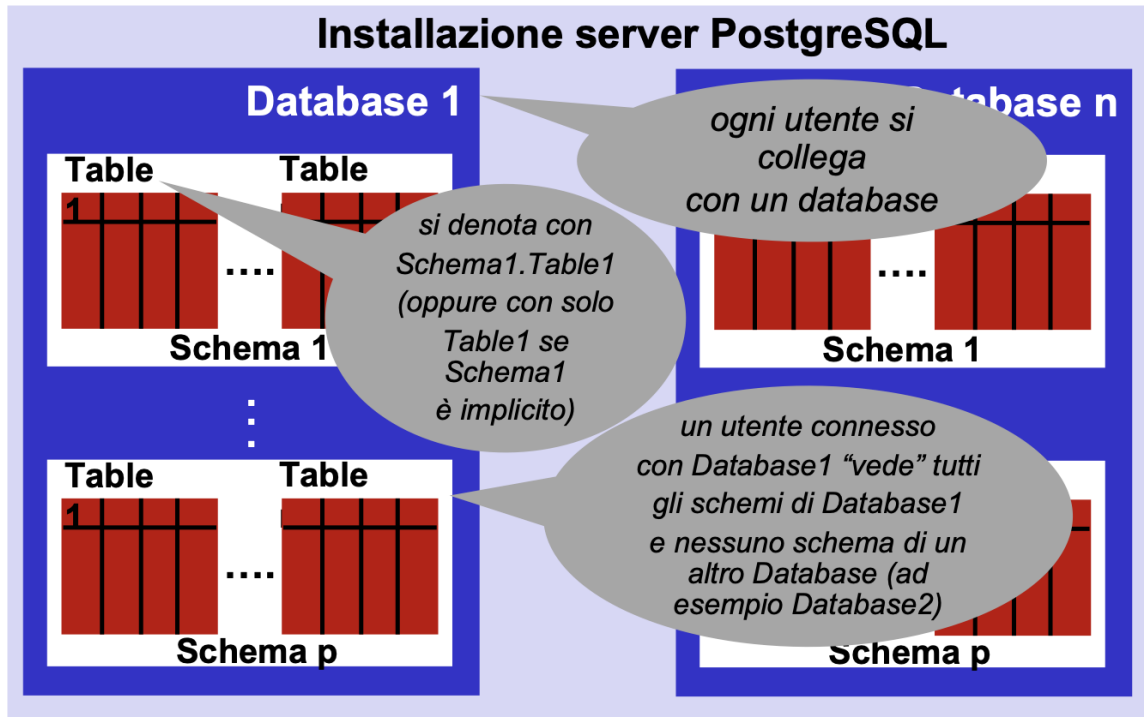
Sebbene SQL sia basato sul modello relazionale, esso non è la stessa cosa:

Differenze:

- Una tabella SQL può contenere duplicati —> È un multinsieme di tuple
 - Una tabella può contenere duplicati, una relazione no
- All'interno di una base di dati in PostgreSQL ci sono diversi schemi

▼ Installazione server PostgreSQL





- Usiamo PostgreSQL perché è open-source

▼ Convenzione sui nomi

- Ogni tabella si denota con `NomeSchema.NomeTabella`

- Quando l'ambiguità sullo schema non sussiste, 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`
 - Esempio: Quando istruiamo il sistema a fare riferimento una volta per tutte ad uno specifico schema, che diventa implicito
- Quando anche l'ambiguità sulla tabella non sussiste, si può omettere anche `NomeTabella.` e scrivere semplicemente `Attributo`
 - Esempio: all'interno di una query in cui si usa una sola tabella con quel nome

▼ 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

▼ Forma elementare

`select` *Attributo,...,Attributo* → **target list**

`from` *Tabella* → **clausola from**

`where` *Condizione* → **clausola where**

▼ Semantica

La semantica di

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

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.



La semantica della **select** è **analogica** alla espressione dell'algebra relazionale

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

▼ 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)

▼ Esempio

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

| nome | 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

vogliamo nome e reddito delle persone con meno di 30 anni:



Algebra relazionale:

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

Semantica:

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

si può scrivere anche come:

```
select nome, reddito
from   persone
where  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 |

▼ Ridenominazione

Per ridenominare utilizziamo `as` nella lista degli attributi, specificando esplicitamente un nome per un attributo del risultato. Quando un attributo manca tale ridenominazione, il nome dell'attributo nel risultato sarà uguale a quello che compare nella tabella menzionata nella clausola `from`. Esso si può anche omettere:



Esempio:

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

È uguale a :

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

▼ Alias

Utilizziamo **as** anche per assegnare un nuovo nome (**alias**) alle tabelle nell'ambito di una query

```
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

$SEL_{eta < 30}(\text{persone})$

```
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



Se ometto `where` è come scrivere `where true`

▼ Condizione con operatore “LIKE”

L'operatore `like` è un operatore che lavora con le stringhe e consente di verificare che una stringa appartenga al linguaggio definito da una espressione regolare

▼ Esempio

Se vogliamo conoscere quali sono le persone che hanno un nome che inizia per `'A'`, ha `'d'` come terza lettera e può contenere altri caratteri,

scriviamo la query:

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

espressione regolare
 $(A + \Sigma + d)^*$
[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)

▼ Espressioni nella target list

- Ogni elemento della target list può essere una espressione che fa uso dei valori memorizzanti negli attributi delle tuple del risultato

▼ Esempio

esempio:

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

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

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

Risultato:

| età | reddito/2 |
|-----|-----------|
| 50 | 20 |

- Nella target list può comparire un numero qualunque di espressioni e all'interno di tali espressioni possono comparire anche costanti

▼ Esempio

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

espressione costituita da una costante di tipo stringa

Risultato:

| nomePersona | età | redditoSemestrale |
|-------------|-----|-------------------|
| Luigi | 50 | 20 |

▼ Selezione, proiezione e join

- LE istruzioni `select` che abbiamo visto fin'ora hanno una sola tabella nella clausola `from` e quindi permettono di realizzare:
 - selezioni
 - proiezioni
 - ridenominazioni
- I `join` (e i *prodotti cartesiani*) si possono realizzare indicando due o più tabelle nella clausola `from` separate da virgola



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

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



$\text{PROJ}_{\langle \text{target list} \rangle} (\text{SEL}_{\langle \text{condizione} \rangle} (R1 \times R2 \times \dots \times Rn))$

Questo **non** significa che il DBMS calcola davvero il prodotto cartesiano di R1, R2,..., Rn

Ciò 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`

▼ Esempio

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 :

$PROJ_{A1,A4} (SEL_{A2=A3} (R1 \times R2))$

a sua volta equivalente a

$PROJ_{A1,A4} (SEL_{A2=A3} (R1 JOIN R2))$

a sua volta equivalente al Theta-join:

$PROJ_{A1,A4} (R1 JOIN_{A2=A3} R2)$

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

Il self-join in SQL

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.

▼ Esempio

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 |

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 |



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.

```
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 |

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 |

| 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

```
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

Select con join esplicito, sintassi

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



```
select ...
from Tabella { join Tabella on CondizioneJoin}, ...
[where AltraCondizione]
```

