SQL

Operatori insiemistici,
interrogazioni nidificate e viste

Operatori insiemistici

- SQL mette a disposizione i seguenti operatori insiemistici: **`union`** (unione), **`intersect`** (intersezione), **`except`** (o **`minus`**, differenza).

- Mentre i risultati di intersezione e differenza possono essere espressi utilizzando altri costrutti del linguaggio (interrogazioni nidificate), l'operatore di unione arricchisce il potere espressivo di SQL permettendo di scrivere interrogazioni altrimenti non formulabili.

Operatori insiemistici

- Sintassi:

```
SelectSQL { < union [ all ] |
intersect | except > SelectSQL }
```

- Gli operatori insiemistici assumono come default l'eliminazione dei duplicati (se si vogliono mantenere i duplicati basta includere la clausola **`all`**).

Operatori insiemistici

- SQL non richiede che gli schemi su cui vengono effettuate le operazioni insiemistiche siano identici ma solo che gli attributi siano in ugual numero e che abbiano **domini compatibili**.

- La corrispondenza tra attributi si basa sulla loro posizione. In presenza di attributi con nome diverso il risultato ha *normalmente* il nome del primo operando.

Interrogazione 33:

- Estrarre in un unico insieme, nome e cognome degli impiegati:

```
select Nome from Impiegato
union
select Cognome from Impiegato
```

nome
Carla
Angela
Augusto
Luca
Laura
Piercarlo
Cancelli
Casini
Fochini
Moschini
Alfredi
Sottani
Rampollo
Magri

Dipartimento

Nome	Indirizzo	Città

Impiegato

Nome	Cognome	Dipart	Ufficio	Stipendio	Città

Interrogazione 33:

- Estrarre in un unico insieme, nome e cognome degli impiegati mantenendo i duplicati:

```
select Nome from Impiegato
union all
select Cognome from Impiegato
```

nome
Carla
Angela
Augusto
Luca
Laura
Laura
Alfredo
Giulio
Piercarlo
Cancelli
Casini
Casini
Fochini
Moschini

Dipartimento

Nome	Indirizzo	Città

Impiegato

Nome	Cognome	Dipart	Ufficio	Stipendio	Città

Interrogazione 34:

- Estrarre in un unico insieme, nome e cognome degli impiegati eccetto quelli del dipartimento Amministrazione, mantenendo i duplicati:

```
select Nome
from Impiegato
where Dipart <> 'Amministrazione'
union all
select Cognome
from Impiegato
where Dipart <> 'Amministrazione'
```

Dipartimento

Nome	Indirizzo	Città

Impiegato

Nome	Cognome	Dipart	Ufficio	Stipendio	Città

Interrogazione 35:

- Estrarre i cognomi degli impiegati che sono anche nomi:

```
select Nome from Impiegato
intersect
select Cognome from Impiegato
```

Dipartimento

Nome	Indirizzo	Città

Impiegato

Nome	Cognome	Dipart	Ufficio	Stipendio	Città

Interrogazione 36:

- Estrarre i nomi degli impiegati che non sono anche cognomi:

```
select Nome from Impiegato
except
select Cognome from Impiegato
```

Dipartimento

Nome	Indirizzo	Città

Impiegato

Nome	Cognome	Dipart	Ufficio	Stipendio	Città

Interrogazione 48:

- Con riferimento alla base dati definita dai seguenti schemi:
  Cantante(Nome, Canzone) e **Autore(Nome, Canzone)**
  …estrarre i nomi dei cantautori puri, vale a dire i cantanti che hanno cantato solo canzoni di cui erano anche autori.

Riassunto

```
select ListaAttributi
from ListaTabelle
where CondizioniSemplici
group by ListaAttributiDiRaggruppamento
having CondizioniAggregate
order by ListaAttributiDiOrdinamento
```

Operatori insiemistici: **`union`**, **`intersect`**, **`except`**

Interrogazioni nidificate

- Sino ad ora abbiamo visto interrogazioni in cui l'argomento della clausola where si basa su condizioni espresse da predicati semplici:
  - Confronto riga per riga tra il valore di un attributo ed una costante o il valore di un altro attributo

- Ci sono tuttavia interrogazioni che richiedono una modalità di specifica più complessa, che consenta di **effettuare la selezione in base al risultato di una seconda interrogazione**

- Esempio: estrarre nome e cognome dell'impiegato che guadagna lo stipendio massimo
  - Da un'analisi riga per riga non siamo in grado di sapere se lo stipendio di un impiegato è maggiore di quello di tutti gli altri

Dipartimento

Nome	Indirizzo	Città

Impiegato

Nome	Cognome	Dipart	Ufficio	Stipendio	Città

Interrogazioni nidificate

- La soluzione consiste nell'estendere il tipo di confronti che possono essere specificati nella clausola where
  - Confronto riga per riga tra
    - il valore di un attributo ed una costante
    - il valore di un attributo ed un altro attributo
    - il valore di un attributo ed il **risultato di una select**

- Nell'ultimo caso, l'interrogazione usata per il confronto viene definita direttamente all'**interno del predicato della clausola where** e si parla di interrogazioni nidificate (**nested queries**) o **subqueries**

Interrogazioni nidificate

- Esempio: estrarre nome e cognome dell'impiegato che guadagna lo stipendio massimo

1. Calcolare lo stipendio massimo
   ```
   select max(stipendio) from Impiegato
   ```

2. Confrontare lo stipendio di ciascun impiegato con quello massimo
   …**where stipendio = select max(stipendio)**…

Dipartimento

Nome	Indirizzo	Città

Impiegato

Nome	Cognome	Dipart	Ufficio	Stipendio	Città

Interrogazioni nidificate

```
Select Nome, Cognome
From Impiegato
Where Stipendio = ( select max(stipendio)
                    from Impiegato )
```

Dipartimento

Nome	Indirizzo	Città

Impiegato

Nome	Cognome	Dipart	Ufficio	Stipendio	Città

Interrogazioni nidificate

- Nell'esempio visto, il risultato della select nidificata è costituito da un singolo valore (il massimo)

```
Select Nome, Cognome
From Impiegato
Where Stipendio = ( select max(stipendio)
                    from Impiegato )
```

- Tuttavia, non sempre è così. In generale, nelle interrogazioni nidificate può esserci una disomogeneità (un **conflitto d'impedenza**) tra i termini messi a confronto nella clausola where:
  - da una parte il valore dell'attributo per una particolare tupla (…where stipendio = …)
  - dall'altra un insieme di tuple (il risultato dell'interrogazione nidificata).

Interrogazioni nidificate

- Esempio: estrarre gli impiegati che lavorano in dipartimenti con sede a Milano

Dipartimento

Nome	Indirizzo	Città
Amministrazione	Via Tito Livio, 27	Milano
Produzione	P.Le Lavater, 3	Torino
Distribuzione	Via Segre, 9	Roma
Direzione	Via Tito Livio, 27	Milano
Ricerca	Via Morone, 6	Milano

Impiegato

Nome	Cognome	Dipart	Ufficio	Stipendio	Città
Mario	Rossi	Amministrazione	10	45	Milano
Carlo	Bianchi	Produzione	20	36	Torino
Giuseppe	Verdi	Amministrazione	20	40	Roma
Franco	Neri	Distribuzione	16	45	Napoli
Carlo	Rossi	Direzione	14	80	Milano
Lorenzo	Lanzi	Direzione	7	73	Genova
Paola	Borroni	Amministrazione	75	40	Venezia
Marco	Franco	Produzione	20	46	Roma

Interrogazioni nidificate

- Esempio: estrarre gli impiegati che lavorano in dipartimenti con sede a Milano

```
select cognome from impiegato
where dipart  =
        select nome from dipartimento
        where città = 'Milano'
```

Dipartimento

Nome	Indirizzo	Città
Amministrazione	Via Tito Livio, 27	Milano
Produzione	P.Le Lavater, 3	Torino
Distribuzione	Via Segre, 9	Roma
Direzione	Via Tito Livio, 27	Milano
Ricerca	Via Morone, 6	Milano

Impiegato

Nome	Cognome	Dipart	Ufficio	Stipendio	Città
Mario	Rossi	Amministrazione	10	45	Milano
Carlo	Bianchi	Produzione	20	36	Torino
Giuseppe	Verdi	Amministrazione	20	40	Roma
Franco	Neri	Distribuzione	16	45	Napoli
Carlo	Rossi	Direzione	14	80	Milano
Lorenzo	Lanzi	Direzione	7	73	Genova
Paola	Borroni	Amministrazione	75	40	Venezia
Marco	Franco	Produzione	20	46	Roma

Interrogazioni nidificate

- SQL gestisce questa disomogeneità con le parole chiave **`any`** e **`all`** che estendono gli operatori di confronto relazionale (=, <>, <, >, <=, >=).
  - Con **`any`** la condizione risulta vera se il confronto tra attributo e risultato è vero per almeno uno degli elementi restituiti dall'interrogazione.
  - Con **`all`** la condizione risulta vera se il confronto tra attributo e risultato è vero per tutti gli elementi restituiti dall'interrogazione.
- Deve esserci compatibilità di dominio tra l'attributo oggetto del confronto ed il risultato dell'interrogazione.

Operatori di confronto

Amministrazione

= any

Amministrazione
Produzione
Distribuzione
Direzione
Ricerca

Operatori di confronto

Amministrazione

= all

Amministrazione
Produzione
Distribuzione
Direzione
Ricerca

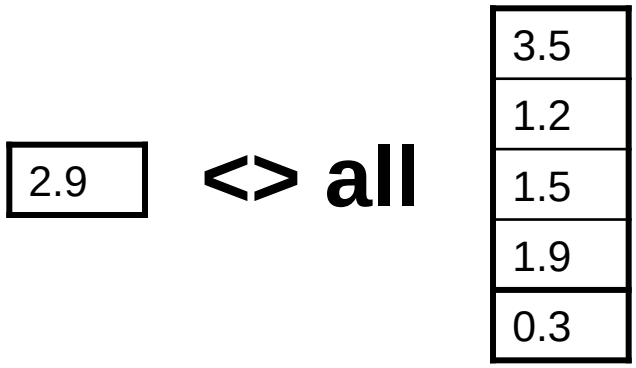
Operatori di confronto

Amministrazione

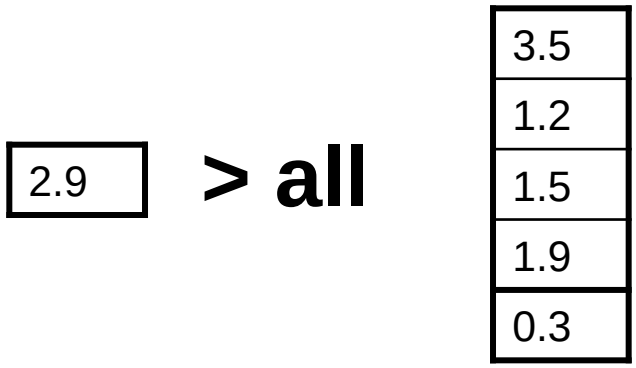
<> all

Amministrazione
Produzione
Distribuzione
Direzione
Ricerca

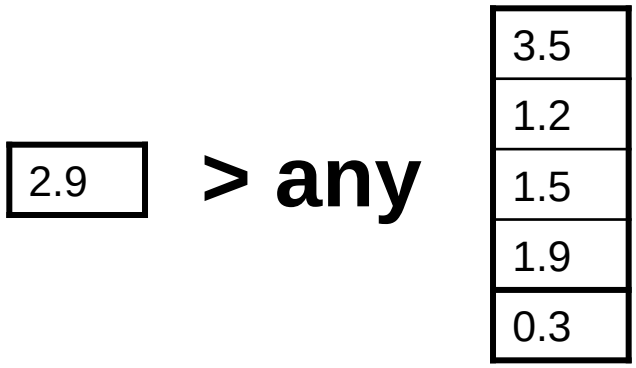
Operatori di confronto



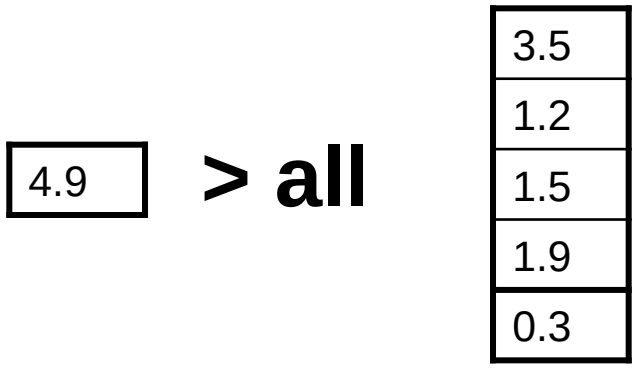
Operatori di confronto



Operatori di confronto



Operatori di confronto



Interrogazione 37:

- Estrarre il cognome degli impiegati che lavorano in dipartimenti con sede a Firenze:

```
select Cognome
from Impiegato
where Dipart = any (
                    select Nome
                    from Dipartimento
                    where Città='Firenze')
```

Questa interrogazione può anche essere espressa attraverso un join tra le tabelle Impiegato e Dipartimento.

Dipartimento

Nome	Indirizzo	Città

Impiegato

Nome	Cognome	Dipart	Ufficio	Stipendio	Città

Interrogazione 40

- Le query nidificate consentono di esprimere in modo equivalente interrogazioni che richiedono l'uso degli operatori intersezione e differenza.

- Il ricorso alle query nidificate è molto frequente su quei DBMS che non supportano gli operatori insiemistici

Interrogazione 40:

- Estrarre i dipartimenti in cui non lavorano impiegati di cognome Rossi (con except):

```
select distinct Nome
from Dipartimento
except
select distinct Dipart
from Impiegato
where Cognome = 'Rossi'
```

Dipartimento

Nome	Indirizzo	Città

Impiegato

Nome	Cognome	Dipart	Ufficio	Stipendio	Città

Interrogazione 40:

- Oppure, con le interrogazioni nidificate:

```
select Dipart
from Impiegato
where Dipart <> all (
                    select I.Dipart
                    from Impiegato I
                    where I.Cognome = 'Rossi')
```

Dipartimento

Nome	Indirizzo	Città

Impiegato

Nome	Cognome	Dipart	Ufficio	Stipendio	Città

Controllo di appartenenza

- Per rappresentare il controllo di appartenenza rispetto ad un insieme, SQL mette a disposizione gli operatori **`in`** e **`not in`** che sono del tutto equivalenti agli operatori **`=any`** e **`<>all`**.

Operatori aggregati

- Le interrogazioni nidificate sono spesso impiegate sia in combinazione che in sostituzione degli operatori max e min.

Interrogazione 42:

- Trovare il dipartimento dell'impiegato che guadagna lo stipendio massimo (usando max):

```
select Dipart from Impiegato
where Stipendio = (
      select max(I.Stipendio)
      from Impiegato I)
```

Dipartimento

Nome	Indirizzo	Città

Impiegato

Nome	Cognome	Dipart	Ufficio	Stipendio	Città

Interrogazione 42bis:

- Trovare il dipartimento dell'impiegato che guadagna lo stipendio massimo (non usando max):

```
select Dipart
from Impiegato
where Stipendio >= all (select I.Stipendio
                        from Impiegato I)
```

Dipartimento

Nome	Indirizzo	Città

Impiegato

Nome	Cognome	Dipart	Ufficio	Stipendio	Città

Int. 42bis: osservazione

- Abbiamo definito due modi per individuare l'impiegato con stipendio massimo:
  - … where stipendio >= all (select stipendio …
  - … where stipendio = (select max(stipendio) …

- Questi due modi sono sempre equivalenti?

Scope delle variabili

- Negli esempi visti sino ad ora abbiamo assunto che una query nidificata venga risolta attraverso i seguenti passi:

1. Eseguire la query nidificata (e.g. calcola max di stipendio)

2. Usare i risultati della query nidificata per valutare se i predicati nella clausola where siano veri (e.g. confronta un particolare stipendio con il valore massimo)

```
select Dipart
from Impiegato
where Stipendio = (select max(I.Stipendio)
                   from Impiegato I)
```

Scope delle variabili

```
select Dipart
from Impiegato
where Stipendio = (select max(I.Stipendio)
                   from Impiegato I)
```

- Ordine di esecuzione delle query nidificate:
  - Per ogni riga della query esterna si valuta la query nidificata e si calcola il valore del predicato a livello di riga della query esterna.

Questo processo può essere attuato un numero arbitrario di volte e consente alla query nidificata di fare riferimento al contesto (le variabili) dell'interrogazione che la racchiude.

Scope delle variabili

- Tipicamente questo accade tramite una variabile definita nell'ambito della query più esterna ed usata nell'ambito di una query in essa nidificata:
  - Si parla in questo caso di **correlated subqueries**

- Una variabile è usabile solo nell'ambito della query in cui è definita o nell'ambito di una query in essa nidificata (a qualsiasi livello).

Interrogazione:

- Estrarre matricola, nome, cognome e dipartimento degli impiegati che hanno lo stipendio massimo all'interno del dipartimento a cui sono affiliati:

```
select I1.matricola, I1.cognome, I1.nome, I1.dipart
from impiegato I1
where I1.stipendio = ( select max(I2.stipendio)
                       from impiegato I2
                       where I1.dipart = I2.dipart)
```

matricola	nome	cognome	dipartimento
AA999BB	Francesco	Casini	D002
AF445ED	Gino	Rampollo	D003
AF676GR	Giuliano	Magri	D001
RF132FR	Laura	Casini	D004
RF133FR	Laura	Casini	D005

Dipartimento

Nome	Indirizzo	Città

Impiegato

Matricola	Nome	Cognome	Dipart	Ufficio	Stipendio	Città

Interrogazione 45:

- Estrarre matricola, nome e cognome delle persone che hanno degli omonimi (stesso nome e cognome ma diversa matricola):

```
select I1.matricola, I1.cognome, I1.nome
  from impiegato I1
  where (I1.Nome, I1.Cognome) = any (
      select I2.nome, I2.cognome
      from impiegato I2
      where I1.matricola <> I2.matricola)
```

Dipartimento

Nome	Indirizzo	Città

Impiegato

Matricola	Nome	Cognome	Dipart	Ufficio	Stipendio	Città

Scope delle variabili

- Esempio di interrogazione con uso errato delle variabili:

```
select *
from Impiegato
where Dipart in (
    select Nome
    from Dipartimento D1
    where Nome = 'Produzione')
or Dipart in (
    select Nome
    from Dipartimento D2
    where D1.Città = D2.Città)
```

Riassunto

Dipartimento

Nome	Indirizzo	Città
Amministrazione	Via Tito Livio, 27	Milano
Produzione	P.Le Lavater, 3	Torino
Distribuzione	Via Segre, 9	Roma
Direzione	Via Tito Livio, 27	Milano
Ricerca	Via Morone, 6	Milano

Impiegato

Nome	Cognome	Dipart	Ufficio	Stipendio	Città
Mario	Rossi	Amministrazione	10	45	Milano
Carlo	Bianchi	Produzione	20	36	Torino
Giuseppe	Verdi	Amministrazione	20	40	Roma
Franco	Neri	Distribuzione	16	45	Napoli
Carlo	Rossi	Direzione	14	80	Milano
Lorenzo	Lanzi	Direzione	7	73	Genova
Paola	Borroni	Amministrazione	75	40	Venezia
Marco	Franco	Produzione	20	46	Roma

Riassunto

select lista attributi

from lista tabelle

where condizioni di selezione

group by attributi di raggruppamento

having condizioni di selezione

Riassunto

select lista attributi

from lista tabelle

where condizioni di selezione <>= **any/all** select

group by attributi di raggruppamento

having condizioni di selezione <>= **any/all** select

VISTE…

Interrogazioni nidificate

- L'operatore logico **`exists`** (ed il suo duale **`not exists`**) può essere usato nelle interrogazioni nidificate (correlate) e restituisce il valore vero solo se l'interrogazione nidificata fornisce come risultato un insieme non vuoto.

Interrogazione 45:

- Usare il costrutto **`exists`** per estrarre i dati di persone che hanno degli omonimi (stesso nome e cognome ma diversa matricola):

Dipartimento

Nome	Indirizzo	Città

Impiegato

Matricola	Nome	Cognome	Dipart	Ufficio	Stipendio	Città

Interrogazione 45:

```
select *
from impiegato I1
where exists(
    select *
    from impiegato I2
    where I1.matricola<>I2.matricola AND
          I1.nome=I2.nome AND
          I1.cognome=I2.cognome)
```

Dipartimento

Nome	Indirizzo	Città

Impiegato

Matricola	Nome	Cognome	Dipart	Ufficio	Stipendio	Città

Interrogazione 46:

- Estrarre i dati degli impiegati che NON hanno degli omonimi:

Dipartimento

Nome	Indirizzo	Città

Impiegato

Matricola	Nome	Cognome	Dipart	Ufficio	Stipendio	Città

Interrogazione 45:

```
select *
from impiegato I1
where not exists(
    select *
    from impiegato I2
    where I1.matricola<>I2.matricola AND
          I1.nome=I2.nome AND
          I1.cognome=I2.cognome)
```

Dipartimento

Nome	Indirizzo	Città

Impiegato

Matricola	Nome	Cognome	Dipart	Ufficio	Stipendio	Città

Subqueries nella target list

- Una select che restituisce una tabella con un solo attributo ed una sola riga prende il nome di **scalar query**
- Una scalar query può essere inclusa come subquery nella target list di una query
- Questo può essere utile per produrre in uscita una tabella che consente il confronto tra una serie di valori elencati dalla query principale con il valore restituito dalla scalar subquery

Dipartimento

Codice	Nome	Responsabile

Impiegato

Matricola	Nome	Cognome	Dipart	Stipendio

Subqueries nella target list

- Esempio: Per ciascun impiegato che non è direttore di dipartimento, elencare matricola, stipendio e lo stipendio minimo tra i direttori di dipartimento

```
select matricola, stipendio, (select min(stipendio)
                              from impiegato, dipartimento
                              where responsabile=matricola) as StipMinDir
from impiegato
where matricola not in
        (select responsabile
        from dipartimento)
```

matricola	stipendio	MinStipDir
AA998CC	88000.00	16500.00
AA999BB	104500.00	16500.00
AA999BK	10500.00	16500.00
AB999RT	22000.00	16500.00
AD645GG	66000.00	16500.00
AD999RT	71500.00	16500.00
AF145GG	71500.00	16500.00
AF444ED	79000.00	16500.00
AF445ED	89000.00	16500.00
AF676GR	93500.00	16500.00
AF978BF	16500.00	16500.00

Dipartimento

Codice	Nome	Responsabile

Impiegato

Matricola	Nome	Cognome	Dipart	Stipendio

Subqueries nella target list

- All'interno della scalar query si può fare riferimento ad una variabile definita nella query esterna
- Per esempio: per ciascun impiegato indicare matricola, stipendio e stipendio massimo tra quelli del dipartimento a cui l'impiegato è affiliato

```
SELECT matricola, stipendio, (select max(stipendio)
                              from Impiegato I2
                              where I1.Dipart=I2.Dipart)
FROM Impiegato I1;
```

matricola	stipendio	(select max(stip...
AA998CC	88000.00	93500.00
AA999BK	10500.00	104500.00
AA999BB	104500.00	104500.00
AB999RT	22000.00	93500.00
AD645GG	66000.00	93500.00
AD999RT	71500.00	104500.00
AF145GG	71500.00	104500.00
AF444ED	79000.00	93500.00
AF445ED	89000.00	89000.00
AF676GR	93500.00	93500.00
AF978BF	16500.00	NULL
AG642GZ	NULL	93500.00

Dipartimento

Codice	Nome	Responsabile

Impiegato

Matricola	Nome	C

Esercizio SQL - Azienda

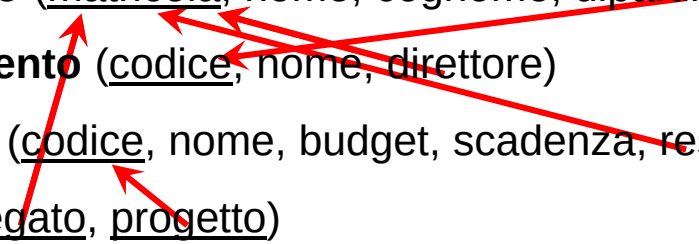
- Il modello relazionale sotto riportato organizza i dati di interesse su dipendenti e progetti di una azienda. In particolare,
  - Per ciascun dipendente sono memorizzate matricola, nome, cognome, dipartimento di affiliazione e stipendio
  - Per ciascun dipartimento sono memorizzati il codice, nome, e matricola del direttore (che è uno dei dipendenti dell'azienda)
  - Nell'azienda vengono portati avanti un certo numero di progetti, per ciascuno dei quali è noto un codice di identificazione, un nome, un budget complessivo, una data di scadenza e la matricola del responsabile di progetto (che è uno dei dipendenti dell'azienda).
  - Ogni impiegato può partecipare a più di un progetto.
  - La relazione PP tiene traccia dei progetti cui partecipano gli impiegati. Essere responsabile di un progetto non equivale a parteciparvi.

Impiegato (matricola, nome, cognome, dipartimento, stipendio)

Dipartimento (codice, nome, direttore)

Progetto (codice, nome, budget, scadenza, responsabile)

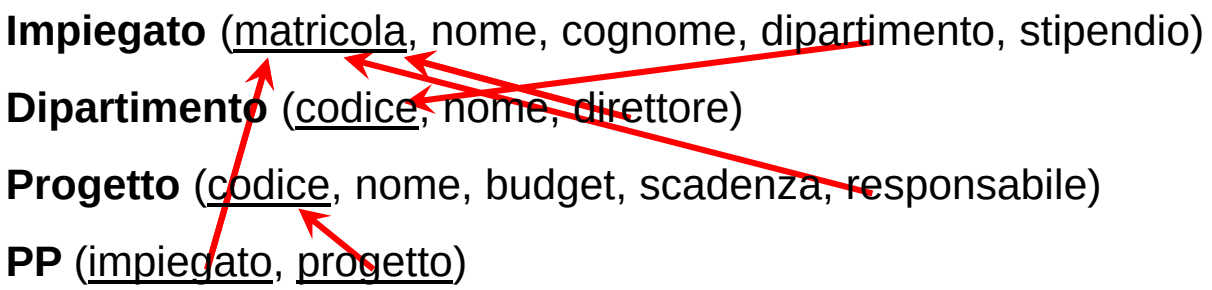
PP (impiegato, progetto)



Esercizio SQL

Interrogazioni:

- Selezionare codice e nome dei dipartimenti tra i cui affiliati non ci sono responsabili di progetto
- Selezionare i codici dei progetti che hanno budget massimo
- Selezionare matricola nome e cognome degli impiegati che non sono direttori di dipartimento o responsabili di progetto
- Selezionare codice, nome e budget dei progetti che hanno come responsabile l'impiegato che tra tutti i responsabili ha lo stipendio massimo
- Selezionare il codice dei progetti a cui partecipano solo impiegati dello stesso dipartimento
- Selezionare il codice di ogni dipartimento insieme alla matricola, nome, cognome e stipendio dell'impiegato che in quel dipartimento ha lo stipendio massimo
- Selezionare matricola, nome e cognome dell'impiegato responsabile del maggior numero di progetti insieme al numero di progetti di cui è responsabile

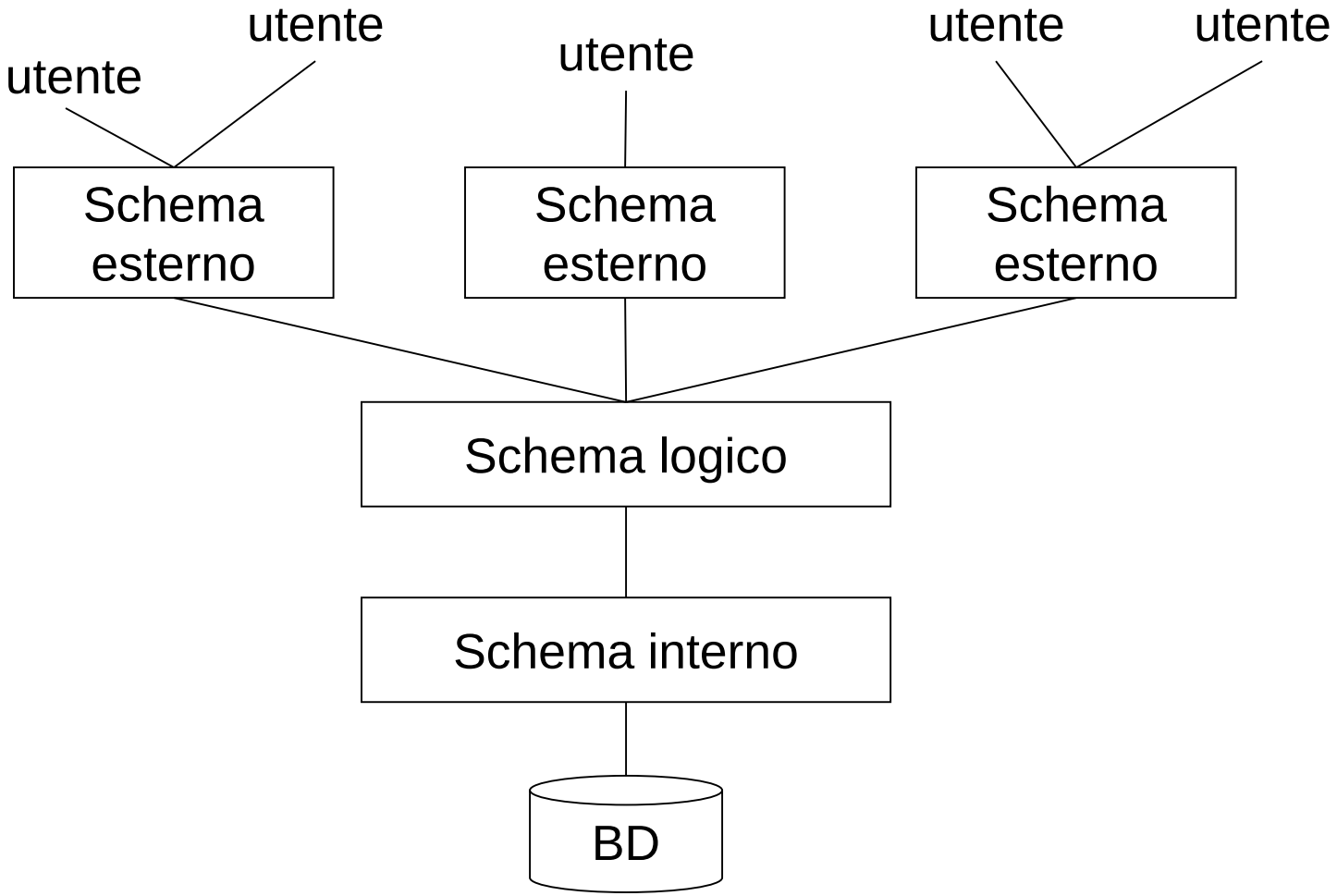


Viste

- In alcuni casi può risultare utile mettere a disposizione degli utenti (e delle applicazioni) rappresentazioni diverse per gli stessi dati

- Questo è reso possibile attraverso gli schemi esterni

Architettura standard a tre livelli



Relazioni di base e derivate

- Il modello relazionale permette di definire
  - Relazioni di base il cui contenuto è autonomo
  - Relazioni derivate, il cui contenuto è funzione di altre relazioni (sia di base che derivate)

Relazioni di base e derivate

- Un esempio di relazione derivata:

CorsiSedi

Corso	Aula	Edificio	Piano
Sistemi	N3	OMI	Terra
Reti	N3	OMI	Terra
Controlli	G	PIN	Primo

Corsi

Corso	Docente	Aula
Basi di dati	Rossi	DS3
Sistemi	Neri	N3
Reti	Bruni	N3
Controlli	Bruni	G

Aule

Nome	Edificio	Piano
DS1	OMI	Terra
N3	OMI	Terra
G	PIN	Primo

Viste virtuali e materializzate

- Le relazioni derivate possono essere di due tipi:
  - viste materializzate
  - relazioni virtuali (o viste)

CorsiSedi

Corso	Aula	Edificio	Piano
Sistemi	N3	OMI	Terra
Reti	N3	OMI	Terra
Controlli	G	PIN	Primo

Corsi

Corso	Docente	Aula
Basi di dati	Rossi	DS3
Sistemi	Neri	N3
Reti	Bruni	N3
Controlli	Bruni	G

Aule

Nome	Edificio	Piano
DS1	OMI	Terra
N3	OMI	Terra
G	PIN	Primo

Viste materializzate

- relazioni derivate associate ad istanze effettivamente memorizzate nella base di dati
  - vantaggi:
    - immediatamente disponibili per le interrogazioni
  - svantaggi:
    - ridondanti
    - appesantiscono gli aggiornamenti (problemi di allineamento)

CorsiSedi

Corso	Aula	Edificio	Piano
Sistemi	N3	OMI	Terra
Reti	N3	OMI	Terra
Controlli	G	PIN	Primo

Corsi

Corso	Docente	Aula
Basi di dati	Rossi	DS3
Sistemi	Neri	N3
Reti	Bruni	N3
Controlli	Bruni	G

Aule

Nome	Edificio	Piano
DS1	OMI	Terra
N3	OMI	Terra
G	PIN	Primo

Relazioni virtuali (viste)

- relazioni derivate definite per mezzo di funzioni di interrogazione e non effettivamente memorizzate
  - Vantaggi:
    - Non presentano problemi di allineamento
    - Non introducono ridondanza
  - Svantaggi:
    - Ogni volta che vi si accede devono essere ricalcolate

CorsiSedi

Corso	Aula	Edificio	Piano
Sistemi	N3	OMI	Terra
Reti	N3	OMI	Terra
Controlli	G	PIN	Primo

Corsi

Corso	Docente	Aula
Basi di dati	Rossi	DS3
Sistemi	Neri	N3
Reti	Bruni	N3
Controlli	Bruni	G

Aule

Nome	Edificio	Piano
DS1	OMI	Terra
N3	OMI	Terra
G	PIN	Primo

Viste virtuali e materializzate

- La maggior parte dei DBMS supporta le viste virtuali e non quelle materializzate
  - Quando presenti (PostgreSQL), le viste materializzate possono essere usate per memorizzare il risultato di interrogazioni molto onerose di cui non serva un dato perfettamente aggiornato
- Le viste virtuali vengono definite per mezzo di espressioni del linguaggio di interrogazione

Viste, esempio

Afferenza

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

Direzione

Reparto	Capo
A	Mori
B	Bruni

Supervisione

Impiegato	Capo
Rossi	Mori
Neri	Bruni
Bianchi	Bruni

```
Create view Supervisione as
    Select Impiegato, Capo
    from Afferenza A, Direzione D
    where A.Reparto=D.Reparto
```

Viste

- Sintassi:

  create view NomeVista [(ListaAttributi)] as
  SelectSQL [with [local | cascaded] check option]

- L'interrogazione SQL deve restituire lo stesso numero di attributi e nello stesso ordine di quelli previsti (sempre che lo siano) nello schema della vista
- Per le viste materializzate:

  create materialized view NomeVista …

  refresh materialized view NomeVista

  drop materialized view NomeVista

Viste

- Le interrogazioni sulle viste sono eseguite sostituendo alla vista la sua definizione

- Pertanto, l'uso delle viste non influisce sull'efficienza delle interrogazioni

Viste

- Le viste in SQL possono servire per formulare interrogazioni che sarebbero altrimenti non esprimibili o comunque di difficile espressione.

- Per esempio, con riferimento alla base dati…

  Impiegato**(<u>Nome</u>, <u>Cognome</u>, Dipart, Ufficio, Stipendio, Citta)**

  …estrarre il dipartimento che spende il massimo in stipendi, cioè il dipartimento per cui è massima la somma degli stipendi dei suoi affiliati

Viste

- Definendo la seguente vista…

```
create view BudgetStipendi(Dip,TotaleStipendi)
as select Dipart, sum(Stipendio)
     from Impiegato
     group by Dipart
```

…estrarre il dipartimento che spende il massimo in stipendi:

```
select Dip
from BudgetStipendi
where TotaleStipendi = ( select max(TotaleStipendi)
                         from BudgetStipendi )
```

Viste

- L'interrogazione equivalente, senza utilizzare la vista, avrebbe richiesto una forma del tipo:

```
select Dipart
from Impiegato
group by Dipart
having sum(Stipendio) >= all
    select sum(Stipendio)
    from Impiegato
    group by Dipart
```

Viste

- Tuttavia quest'ultima espressione potrebbe—pur essendo conforme alla sintassi di SQL—non essere riconosciuta da alcuni DBMS che richiedono che la condizione espressa nella clausola having sia una condizione semplice di confronto con un attributo o una costante e non il risultato dell'esecuzione di una query nidificata.

Viste

- Le viste in SQL sono necessarie per formulare interrogazioni che prevedono il calcolo di operatori aggregati come funzione di altri operatori aggregati.

- Per esempio, con riferimento alla base dati…

  Impiegato**(<u>Nome</u>, <u>Cognome</u>, Dipart, Ufficio, Stipendio, Citta)**

  …estrarre il valor medio del numero di impiegati affiliati ad un dipartimento

Viste

```
create view numeroImpiegati(Dip,numero)
    as select Dipart, count(*)
        from Impiegato
        group by Dipart

Select avg(numero)
from numeroImpiegati
```

Infatti, **select avg(count(*)) from impiegato group by Dipart**

È SBAGLIATA

Impiegato(**Nome**, **Cognome**, **Dipart, Ufficio, Stipendio, Citta)**

Viste, motivazioni

- L'uso delle viste può essere vantaggioso per diversi motivi:
  - Ogni utente (o applicazione) può vedere solo ciò che gli interessa e che è autorizzato a vedere
  - Si può semplificare la scrittura di interrogazioni complesse attraverso sottoespressioni
  - In occasione di ristrutturazioni della base dati, le relazioni eliminate possono essere sostituite con delle viste, senza dover alterare i programmi che accedevano alla base dati. Ad esempio, se uno schema R(A,B,C) viene sostituito con due schemi R'(A,B) ed R''(B,C) è possibile definire una vista R=R'JOIN R'' per mantenere inalterate le applicazioni che accedevano al vecchio schema

Viste ed aggiornamenti

- L’utente o applicazione che accede ad una vista può credere sia una normale tabella
  - Può quindi provare a fare operazioni di INSERT o UPDATE sulla vista.
- E’ questa una cosa possibile?

Aggiornabilità delle viste

- Di fatto, una vista è una funzione che calcola un risultato y a partire da un'istanza di base dati r:

$$y = F(r)$$

- L'aggiornamento di una vista che trasforma y in y' può essere gestito dal DBMS solo se è univocamente definita la nuova istanza r' tale che y' = F(r')

Vale a dire se la vista è **invertibile**

Aggiornabilità delle viste

- Data la complessità del problema, di fatto ogni DBMS impone dei vincoli sull'aggiornamento delle viste.

- Le più comuni restrizioni riguardano la non aggiornabilità di viste in cui il blocco più esterno della query di definizione contiene una delle seguenti funzioni:

  group by

  funzioni aggregate

  distinct

  join, sia espliciti che impliciti (la query più esterna deve avere una sola tabella nella clausola from)

Aggiornabilità delle viste

- La restrizione si applica quando tali funzioni vengono applicate nel blocco più esterno della query di definizione.

Aggiornabilità delle viste

- Consideriamo la base dati sotto mostrata…

Imp

CodImp	Nome	Sede	Ruolo	Stipendio
E001	Rossi	S01	Analista	2000
E002	Verdi	S02	Sistemista	1500
E003	Bianchi	S01	Programmatore	1000
E004	Gialli	S03	Programmatore	1000
E005	Neri	S02	Analista	2500
E006	Grigi	S01	Sistemista	1100
E007	Violetti	S01	Programmatore	1000
E008	Aranci	S02	Programmatore	1200

Sedi

Sede	Responsabile	Citta
S01	Biondi	Milano
S02	Mori	Bologna
S03	Fulvi	Milano

Aggiornabilità delle viste

- Definiamo la vista ImpBO con i dati degli impiegati che lavorano a Bologna:

```
create view ImpBO(CodImp, Nome, Sede, Ruolo, Stipendio)
as select I.*
from Impiegato I join Sedi S on (I.Sede=S.Sede)
where S.Città='Bologna'
```

...tale vista non è aggiornabile !

Imp

CodImp	Nome	Sede	Ruolo	Stipendio
E001	Rossi	S01	Analista	2000
E002	Verdi	S02	Sistemista	1500
E003	Bianchi	S01	Programmatore	1000
E004	Gialli	S03	Programmatore	1000
E005	Neri	S02	Analista	2500
E006	Grigi	S01	Sistemista	1100
E007	Violetti	S01	Programmatore	1000
E008	Aranci	S02	Programmatore	1200

Sedi

Sede	Responsabile	Citta
S01	Biondi	Milano
S02	Mori	Bologna
S03	Fulvi	Milano

Aggiornabilità delle viste

- la seguente vista, del tutto equivalente alla prima, è aggiornabile:

```
create view ImpBO(CodImp, Nome, Sede, Ruolo, Stipendio)
as select I.* from Impiegato I
where I.Sede in
    (select S.Sede from Sedi S where S.Città='Bologna')
```

E' quindi possibile eseguire una istruzione del tipo:

```
insert into ImpBO(CodImp, Nome, Sede, Ruolo, Stipendio)
values ('E010', 'Lilla', 'S02', 'Analista', 1800)
```

Imp

CodImp	Nome	Sede	Ruolo	Stipendio
E001	Rossi	S01	Analista	2000
E002	Verdi	S02	Sistemista	1500
E003	Bianchi	S01	Programmatore	1000
E004	Gialli	S03	Programmatore	1000
E005	Neri	S02	Analista	2500
E006	Grigi	S01	Sistemista	1100
E007	Violetti	S01	Programmatore	1000
E008	Aranci	S02	Programmatore	1200

Sedi

Sede	Responsabile	Citta
S01	Biondi	Milano
S02	Mori	Bologna
S03	Fulvi	Milano

Aggiornabilità delle viste

- Per le viste aggiornabili si presenta un ulteriore problema. Si consideri, per esempio, il seguente inserimento nella vista ImpBO:

```
insert into ImpBO(CodImp, Nome, Sede, Ruolo, Stipendio)
values ('E009', 'Azzurri', 'S03', 'Analista', 1800)
```

Il valore di sede (la sede 'S03' è a Milano e non Bologna) non rispetta la query che definisce la vista.

Imp

CodImp	Nome	Sede	Ruolo	Stipendio
E001	Rossi	S01	Analista	2000
E002	Verdi	S02	Sistemista	1500
E003	Bianchi	S01	Programmatore	1000
E004	Gialli	S03	Programmatore	1000
E005	Neri	S02	Analista	2500
E006	Grigi	S01	Sistemista	1100
E007	Violetti	S01	Programmatore	1000
E008	Aranci	S02	Programmatore	1200

Sedi

Sede	Responsabile	Citta
S01	Biondi	Milano
S02	Mori	Bologna
S03	Fulvi	Milano

Aggiornabilità delle viste

- Di conseguenza, una successiva visualizzazione del contenuto della vista ImpBO (**`select * from ImpBO`**) non restituirebbe nel risultato la tupla appena inserita
  - I dati sarebbero comunque presenti nella tabella **Imp**
- Per evitare situazioni di questo tipo, all'atto della creazione di una vista si può specificare la clausola **`with check option`**, che garantisce che ogni tupla inserita nella vista rispetti la query di definizione della vista
- L'eventuale tentativo di inserire una tupla che non rispetti la query di definizione causa un errore:

```
********** Error **********

ERRORE: la nuova riga viola l'opzione di controllo della vista
```

Imp

CodImp	Nome	Sede	Ruolo	Stipendio
E001	Rossi	S01	Analista	2000
E002	Verdi	S02	Sistemista	1500
E003	Bianchi	S01	Programmatore	1000
E004	Gialli	S03	Programmatore	1000

Sedi

Sede	Responsabile	Citta
S01	Biondi	Milano
S02	Mori	Bologna
S03	Fulvi	Milano

Aggiornabilità delle viste

- Se una vista V1 è definita in termini di un'altra vista V2 e si specifica la clausola **`with check option`** per definire V1, il DBMS verifica che la nuova tupla inserita soddisfi sia la condizione di V1 che quella di V2, a prescindere che V2 sia stata o meno definita con la clausola **`with check option`**

- Questo comportamento di default è equivalente a definire la vista con l'opzione **`with cascaded check option`**

Aggiornabilità delle viste

- Per modificare tale comportamento bisogna definire la vista con l'opzione **`with local check option`**

- In questo caso il DBMS verifica che la nuova tupla soddisfi la specifica su V1 e sulle sole viste da cui V1 dipende e per cui sia stata dichiarata la clausola **`with check option`**

Table expressions

- La sintassi SQL prevede che una istruzione SELECT possa comparire
  - Nella clausola where / having (subquery)
  - Nella target list (scalar query)
  - Nella clausola from

- Nell'ultimo caso la clausola from contiene una tabella derivata dinamicamente che prende il nome di table expression (o derived table)

Table expressions

- Calcolare il valor medio del numero di impiegati affiliati ad un dipartimento:

```
SELECT avg(num_affiliati.numero)
FROM (SELECT Dipart, count(*) as numero
      FROM Impiegato
      GROUP BY Dipart) num_affiliati
```

La tabella num_affiliati(Dipart, numero) è derivata dinamicamente come risultato di una SelectSQL all'interno della clausola from

Dipartimento

Nome	Indirizzo	Città

Impiegato

Nome	Cognom	Dipart	Uffici	Stipend	Città

Table expressions

- Per ogni dipartimento, estrarre il nome, il valore dello stipendio massimo e quanti impiegati lo percepiscono (create view):

```
create view SM as
SELECT Dipart, MAX(Stipendio) as Stip
    FROM Impiegato
    GROUP BY Dipart


SELECT SM.Dipart,SM.Stip,COUNT(*)
FROM Impiegato I, SM
WHERE I.Dipart = SM.Dipart AND
           I.Stipendio = SM.Stip
GROUP BY SM.Dipart,SM.Stip
```

Dipartimento

Nome	Indirizzo	Città

Impiegato

Nome	Cognom	Dipart	Uffici	Stipend	Città

Table expressions

- Per ogni dipartimento, estrarre il codice, il valore dello stipendio massimo e quanti impiegati lo percepiscono (tab. dinam.):

```
SELECT SM.Dipart,SM.Stip,COUNT(*)
FROM Impiegato I,(
     SELECT Dipart, MAX(Stipendio) as Stip
     FROM Impiegato
     GROUP BY Dipart) SM
WHERE I.Dipart = SM.Dipart AND
           I.Stipendio = SM.Stip
GROUP BY SM.Dipart,SM.Stip
```

Dipartimento

Nome	Indirizzo	Città

Impiegato

Nome	Cognom	Dipart	Uffici	Stipend	Città

Table expressions

- All'interno della table expression non possono comparire variabili definite esternamente ad essa (correlated derived tables are not ANSI standard)
- Estrarre il codice di ciascun dipartimento ed il numero di affiliati che sono responsabili di progetto:

```
SELECT d.codice, Responsabili.numero
FROM Dipartimento d,(
     SELECT Dipartimento, count(distinct matricola)
                                as numero
     FROM Impiegato, Progetto
     WHERE responsabile = matricola AND
     Dipartimento = d.codice) Responsabili
WHERE d.codice = Responsabili.Dipartimento
```

ERRATA

Table expressions

- All'interno della table expression non possono comparire variabili definite esternamente ad essa (correlated derived tables are not ANSI standard)
- Estrarre il codice di ciascun dipartimento ed il numero di affiliati che sono responsabili di progetto:

```
SELECT d.codice, Responsabili.numero
FROM Dipartimento d ,(
   SELECT Dipartimento, count(distinct matricola) as
                                                numero
   FROM Impiegato, Progetto
   WHERE responsabile = matricola
   group by Dipartimento) Responsabili
WHERE d.codice=Responsabili.dipartimento
```

CORRETTA

Table expressions

- All'interno della table expression non possono comparire variabili definite esternamente ad essa
- Estrarre il codice di ciascun dipartimento ed il numero di affiliati che sono responsabili di progetto:

```
SELECT d.codice, count(distinct matricola) numResp
FROM (Progetto p join Impiegato i on
  matricola=responsabile) right join Dipartimento d
  on d.codice=dipartimento
group by d.codice
```

CORRETTA

codice	numResp
D001	1
D002	0
D003	1
D004	1
D005	0

Esercizio SQL - Azienda

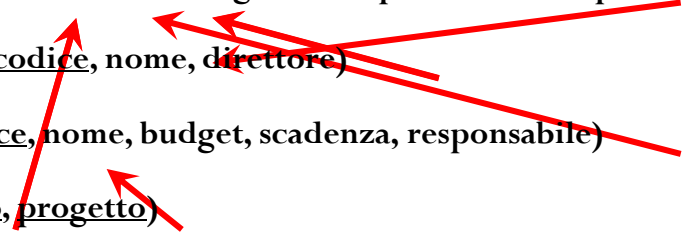
- Il modello relazionale sotto riportato organizza i dati di interesse su dipendenti e progetti di una azienda. In particolare,
  - Per ciascun dipendente sono memorizzate matricola, nome, cognome, dipartimento di affiliazione e stipendio
  - Per ciascun dipartimento sono memorizzati il codice, nome, e matricola del direttore (che è uno dei dipendenti dell'azienda)
  - Nell'azienda vengono portati avanti un certo numero di progetti, per ciascuno dei quali è noto un codice di identificazione, un nome, un budget complessivo, una data di scadenza e la matricola del responsabile di progetto (che è uno dei dipendenti dell'azienda).
  - La relazione PP tiene traccia dei progetti a cui partecipano gli impiegati. Essere responsabile di un progetto non equivale a parteciparvi.

Impiegato **(matricola, nome, cognome, dipartimento, stipendio)**

Dipartimento **(codice, nome, direttore)**

Progetto **(codice, nome, budget, scadenza, responsabile)**

PP **(impiegato, progetto)**



Esercizio SQL

Interrogazioni:

1. Selezionare il numero medio di impiegati che partecipano ad un progetto
2. Selezionare la matricola, nome e cognome di ciascun responsabile di progetto e la media dei budget dei progetti che supervisiona
3. Per ciascun dipartimento selezionare il codice, il nome, il numero di impiegati, il numero di impiegati che è responsabile di progetto, la somma dei budget di progetto per cui è responsabile un impiegato del dipartimento

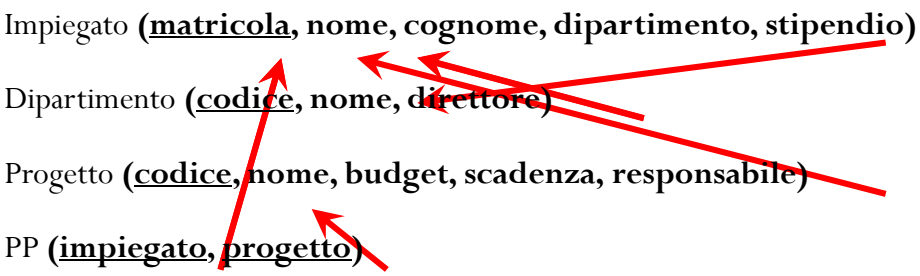


Table expressions

- Vantaggi:
  - la vista è temporanea, non viene memorizzata dal DBMS
- Svantaggi:
  - La definizione nella clausola from la rende poco leggibile

Common table expressions (SQL-99)

- Le table expressions possono anche essere usate con il costrutto WITH per definire delle viste temporanee da essere usate all'interno di interrogazioni come fossero a tutti gli effetti delle viste.

- Vengono definite esternamente al corpo della select, migliorando quindi la leggibilità

- Si parla in questo caso di common table expressions.

Common table expressions (SQL-99)

- Selezionare il dipartimento che spende il massimo in stipendi dei dipendenti

```
with SediStip(Dipart, TotStip)
  as (select Dipart, sum(Stipendio)
        from Impiegato
        group by Dipart)
   (select Dipart
    from SediStip
    where TotStip =
        select max(TotStip)
        from SediStip)
```

Dipartimento

Nome	Indirizzo	Città

Impiegato

Nome	Cognome	Dipart	Ufficio	Stipendio	Città

Common table expressions (SQL-99)

- Esempio: data la relazione Genitori(Figlio, Genitore) sotto mostrata, trovare tutti gli antenati (genitori, nonni, bisnonne, ecc.) di Anna (in realtà al posto dei nomi andrebbero usati i CF).

Genitori

Figlio	Genitore
Anna	Luca
Luca	Carla
Luca	Giovanni
Carla	Luisa
Giovanni	Augusto
Giorgio	Susanna

Avo
Luca
Carla
Giovanni
Luisa
…

Common table expressions (SQL-99)

- Se fossero richiesti solo gli antenati di Anna sino al secondo grado…

Genitori

Figlio	Genitore
Anna	Luca
Luca	Carla
Luca	Giovanni
Carla	Luisa
Giovanni	Augusto
Giorgio	Susanna

```
Select Genitore
From Genitori
Where Figlio = 'Anna'

Union

Select G2.Genitore
From Genitori G1, Genitori G2
Where G1.Genitore = G2.Figlio
        AND G1.Figlio='Anna'
```

Common table expressions (SQL-99)

- In questi casi, le common table expressions possono costituire una soluzione attraverso le **viste temporanee ricorsive**
- Nell'esempio, si può definire la vista ricorsiva Antenati(Persona, Avo) che risulta dall'unione di:
  - una **sub-query di inizializzazione** (non ricorsiva)
  - una **sub-query ricorsiva** che ad ogni iterazione aggiorna la vista Antenati(Persona, Avo)

Common table expressions (SQL-99)

```
with Antenati(Persona, Avo) as(
    select Figlio, Genitore
    from Genitori
    union all
    select G.Figlio, A.Avo
    from Genitori G, Antenati A
    where G.Genitore=A.Persona)
select Avo
from Antenati
where Persona='Anna'
```

Inizializzazione: subquery di base

Ricorsione: subquery ricorsiva

Genitori

Figlio	Genitore
Anna	Luca
Luca	Carla
Luca	Giovanni
Carla	Luisa
Giovanni	Augusto
Giorgio	Susanna

Common table expressions (SQL-99)

- Sembra che ad ogni iterazione venga fatto il join tra Genitori ed Antenati con Genitore=Persona.
- Tuttavia questo non è vero: infatti, il join in questione non coinvolge l'intera tabella Antenati, ma solo quella parte di tale tabella che è stata aggiunta nell'ultima iterazione.

```
with Antenati(Persona, Avo) as(
    select Figlio, Genitore
    from Genitori
    union all
    select G.Figlio, A.Avo
    from Genitori G, Antenati A
    where G.Genitore=A.Persona)
select Avo
from Antenati
where Persona='Anna'
```

Genitori

Figlio	Genitore
Anna	Luca
Luca	Carla
Luca	Giovanni
Carla	Luisa
Giovanni	Augusto
Giorgio	Susanna

Common table expressions (SQL-99)

- In questo modo, il contenuto della tabella Antenati viene così costruito:

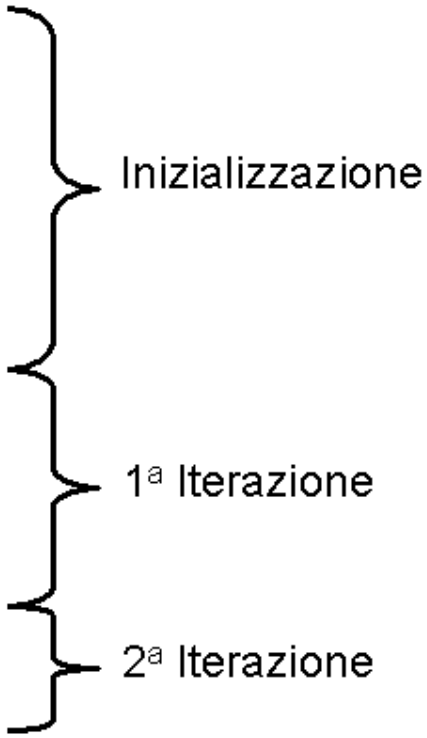
```
with Antenati(Persona, Avo) as(
    select Figlio, Genitore
    from Genitori
    union all
    select G.Figlio, A.Avo
    from Genitori G, Antenati A
    where G.Genitore=A.Persona)
```

Genitori

Figlio	Genitore
Anna	Luca
Luca	Carla
Luca	Giovanni
Carla	Luisa
Giovanni	Augusto
Giorgio	Susanna

Antenati

Persona	Avo
Anna	Luca
Luca	Carla
Luca	Giovanni
Carla	Luisa
Giovanni	Augusto
Giorgio	Susanna
Anna	Carla
Anna	Giovanni
Luca	Luisa
Luca	Augusto
Anna	Luisa
Anna	Augusto



Alla 3ª iterazione la parte ricorsiva della query restituisce un risultato vuoto e quindi la ricorsione termina

Common table expressions (SQL-99)

- Per le common table expressions valgono le seguenti restrizioni:
  - possono essere usate solo nel blocco più esterno di una query
  - nella subquery ricorsiva non si possono usare funzioni aggregate: group by, select distinct

- Le common table expressions non sono supportate in MySQL ma lo sono in PostgreSQL
  - http://stackoverflow.com/questions/1382573/how-do-you-use-the-with-clause-in-mysql