



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>



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



Definizione dei dati in PostgreSQL:

`create schema`

- La gestione di schemi e databases nei DMBS varia da sistema a sistema. Ricordiamo com'è la situazione in PostgreSQL
- PostgreSQL può gestire vari **databases** (un database si crea con **create database**) e prevede l'istruzione **create schema**, che, contrariamente a quanto suggerito dal nome, non serve a dichiarare uno schema propriamente detto (secondo quanto abbiamo detto finora) per tutto il database, ma un cosiddetto **namespace** all'interno di un database: lo schema totale del database D lo otteniamo giustapponendo tutti gli schemi definiti nell'ambito del database D
- Ad un namespace si possono associare relazioni, vincoli, privilegi per gli utenti, ecc. ed operare sugli stessi in modo unitario. Ogni schema ha un nome, e ricordiamo che la notazione estesa per gli oggetti in esso definiti è:
`<nameschema>.<nomeoggetto>`
- In ogni database si possono quindi definire più schemi, e si possono eseguire operazioni che coinvolgono tabelle di più schemi, a patto di usare il nome esteso della tabelle, ovvero
`<nameschema>.<nometabella>`
- Al contrario, diversi databases non si “parlano”: PostgreSQL non può eseguire operazioni che coinvolgono tabelle di più databases.



Definizione dei dati in SQL: `create table`

- L'istruzione più importante del DDL (data definition language) di SQL è

`create table`

- definisce uno schema di relazione (specificando attributi e vincoli) in uno schema di una base di dati
 - crea un'istanza vuota corrispondente allo schema di relazione
-
- Sintassi: `create table` *NomeTabella* (
 NomeAttributo Dominio [*Vincoli*]

 NomeAttributo Dominio [*Vincoli*]
 [*AltriVincoli*]
)

create table: esempio

```
create table Impiegato (  
  Matricola      character(6) primary key,  
  Nome           character(20) not null,  
  Cognome        character(20) not null,  
  Dipart         character(15),  
  Stipendio      numeric(9) default 0,  
  Citta          character(15),  
  foreign key(Dipart) references  
    Dipartimento(NomeDip),  
  unique (Cognome, Nome)  
)
```

nome
tabella

nome
attributo

vincolo

dominio
(o tipo)

vincolo

vincolo



Domini per gli attributi

• Domini predefiniti

- **Carattere:** singoli caratteri o stringhe, anche di lunghezza variabile
 - `char(n)` o `character(n)` – stringhe di lunghezza fissa
 - `varchar(n)` (o `char varying(n)`) – stringhe di lunghezza variabile
 - `nchar(n)` e `nvarchar(n)` (o `nchar varying(n)`) - come sopra ma UNICODE
- **Numerici:** esatti e approssimati
 - `int` o `integer`, `smallint` - interi
 - `numeric`, (o `numeric(p)`, `numeric(p,s)`) - valori numerici esatti nonnegativi
 - `decimal`, (o `decimal(p)`, `decimal(p,s)`) - valori numerici esatti anche negativi
 - `float`, `float(p)`, `real`, `double precision` - reali
- **Data, ora, intervalli di tempo**
 - `Date`, `time`, `timestamp`
 - `time with timezone`, `timestamp with timezone`
- **Bit:** singoli bit o stringhe di bit
 - `bit(n)`
 - `bit varying(n)`
- **Introdotti in SQL:1999**
 - `boolean`
 - `BLOB`, `CLOB`, `NCLOB` (binary/character large object): per grandi immagini e testi



Definizione dei dati in SQL: `create domain`

- **Domini definiti dagli utenti**

- L'istruzione

`create domain`

definisce un dominio (semplice) con vincoli e valori di default, utilizzabile in definizioni di relazioni

- Sintassi

```
create domain NomeDominio  
as DominioPreesistente [ Default ] [ Vincoli ]
```

- *Esempio:*

```
create domain Voto  
as smallint default null  
check ( value >=18 and value <= 30 )
```

- **Compatibilità:** il nuovo dominio ed il dominio di partenza (quello che compare dopo la "as") sono compatibili, ed inoltre i valori del nuovo dominio devono rispettare i vincoli indicati nella definizione



Vincoli in SQL

Vedremo diversi tipi di vincoli in SQL, sia intrarelazionali, sia interrelazionali.

Ogni vincolo può essere dichiarato con nome esplicito oppure senza nome esplicito (in questo caso il nome viene deciso dal sistema). Per dichiarare un vincolo con il nome occorre fare precedere la sua definizione da

constraint <nome del vincolo>

In queste slide ometteremo spesso di assegnare nomi espliciti il nome dei vincoli, per brevità. Ma nella pratica questa possibilità è molto importante: dare un nome ad un vincolo consente di riferirsi ad esso in modo non ambiguo (utile, ad esempio, nella segnalazione che il sistema fa quando viene violato).

Vincoli intrarelazionali

- not null (su singoli attributi)
- unique: permette di definire un insieme di attributi come superchiave (anche più superchiavi per tabella)
 - singolo attributo:
unique dopo la specifica del dominio
 - più attributi:
unique (*Attributo*, ..., *Attributo*)
- primary key: definizione della chiave primaria (al massimo una chiave primaria per tabella, su uno o più attributi); sintassi come per la unique, ma usando il termine **primary key**. Ricordiamo che implica **not null**. Osservazione importante: **è cura di chi definisce il vincolo di chiave primaria per K assicurarsi che la superchiave K sia minimale, ossia che non esista una parte di K che è a sua volta una superchiave in tutte le istanze.**
- check, per vincoli di tuple o anche più complessi (vedi dopo)

Vincoli intrarelazionali, esempi

```
create table Impiegato ( RIMANINA SEMPRE PK  
    Matricola character(6) constraint pk1 primary key,  
    Nome character(20) not null,  
    Cognome character(20) not null,  
    Dipart character(15),  
    Stipendio numeric(9) default 0,  
    Citta character(15),  
    foreign key(Dipart) references Dipartimento(NomeDip),  
    constraint un1 unique (Cognome, Nome)  
)
```

nome del
vincolo

nome del
vincolo

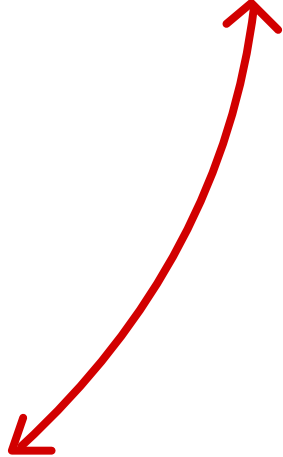


primary key, alternative

```
create table Impiegato (  
    Matricola character(6) constraint pk1 primary key,  
    ...  
)
```

oppure

```
create table Impiegato (  
    Matricola character(6),  
    ...  
    constraint pk1 primary key (Matricola)  
)
```



Chiavi su più attributi, attenzione

```
create table Impiegato ( ...  
    Nome      character(20) not null,  
    Cognome   character(20) not null,  
    unique (Cognome, Nome)  
)
```

è ovviamente **diverso** da:

```
create table Impiegato ( ...  
    Nome      character(20) not null unique,  
    Cognome   character(20) not null unique  
)
```

Vincoli interrelazionali

- check, per vincoli complessi
- references e foreign key permettono di definire vincoli di integrità referenziale

Sintassi:

- per singoli attributi:

references dopo la specifica del dominio

- riferimenti su più attributi:

foreign key (Attributo, ..., Attributo) references ...

Gli attributi referenziati nella tabella di arrivo devono formare una chiave (**primary key** o **unique**). Se mancano, il riferimento si intende alla chiave primaria

Semantica: ogni combinazione (senza NULL) di valori per gli attributi nella tabella di partenza deve comparire nella tabella di arrivo

- È possibile associare politiche di reazione alla violazione dei vincoli (causate da modifiche sulla tabella esterna, cioè quella cui si fa riferimento)

DA UNA TABELLA
AD UN'ALTRA



Vincoli interrelazionali, esempio

Infrazioni

<u>Codice</u>	Data	Vigile	Prov	Numero
34321	1/2/95	3987	MI	39548K
53524	4/3/95	3295	TO	E39548
64521	5/4/96	3295	PR	839548
73321	5/2/98	9345	PR	839548

Vigili

<u>Matricola</u>	Cognome	Nome
3987	Rossi	Luca
3295	Neri	Piero
9345	Neri	Mario
7543	Mori	Gino

Vincoli interrelazionali, esempio (cont.)

Infrazioni

<u>Codice</u>	Data	Vigile	Prov	Numero
34321	1/2/95	3987	MI	39548K
53524	4/3/95	3295	TO	E39548
64521	5/4/96	3295	PR	839548
73321	5/2/98	9345	PR	839548

Auto

<u>Prov</u>	<u>Numero</u>	Cognome	Nome
MI	39548K	Rossi	Mario
TO	E39548	Rossi	Mario
PR	839548	Neri	Luca



Vincoli interrelazionali, esempio

```
create table Infrazioni (  
  Codice    character(6) not null primary key,  
  Data      date not null,  
  Vigile    integer not null  
            references Vigili(Matricola) ,  
  Provincia character(2) ,  
  Numero    character(6) ,  
  foreign key(Provincia, Numero)  
            references Auto(Provincia,Numero)  
)
```


Modifiche degli schemi: **alter table**

alter table: permette di modificare una tabella

Esempio:

```
create table Infrazioni (  
    Codice      character(6) not null primary key,  
    Data        date not null,  
    Vigile      integer not null references Vigili(Matricola),  
    Provincia   character(2),  
    Numero      character(6)  
)
```

```
alter table Infrazioni  
add constraint MioVincolo foreign key(Provincia, Numero)  
references Auto(Provincia,Numero)
```

Nella alter table si possono cambiare (**change**, **modify**) o eliminare (**drop**) elementi presenti o aggiungere (**add**) nuovi elementi.

Modifiche degli schemi: `alter table`

La `alter table` è importante anche per potere definire tabelle in cui sono definiti vincoli di integrità referenziali 'incrociati'.

Se per R1 deve essere definito un vincolo di integrità referenziale verso R2 e, viceversa, per R2 deve essere definito un vincolo di integrità referenziale verso R1, sussiste un problema: siccome all'interno della definizione di R1 deve essere menzionato R2, questo vuol dire che per definire R1 deve essere già definita la tabella R2. Ma allo stesso tempo, siccome all'interno della definizione di R2 deve essere menzionato R1, questo vuol dire che per definire R2 deve essere già definita la tabella R1. Ovviamente queste due esigenze sono inconciliabili.

La soluzione è allora questa:

- si definisce R1 senza inserire il vincolo di integrità referenziale verso R2
- si fornisce la definizione completa di R2, quindi anche con il vincolo di integrità referenziale verso R1 (ora già definita)
- con il comando `alter table` si aggiunge alla definizione di R1 il vincolo di integrità referenziale verso R2 (ormai già definita)



Un'importante applicazione di `alter table`

Vogliamo definire due tabelle: `Persona(cf,cittaNascita)`, `Citta(nome,sindaco)` con un vincolo di integrità referenziale da `cittaNascita` a `Citta[nome]` ed uno da `sindaco` a `Persona[CF]`. Tentiamo di procedere così:

```
create table persona(  
  cf varchar(30) primary key,  
  cittaNascita varchar(30)) references citta;  
  
create table citta(  
  nome varchar(30) primary key,  
  sindaco varchar(100) references persona);
```

Un'importante applicazione di `alter table`

Vogliamo definire due tabelle: `Persona(cf,cittaNascita)`, `Citta(nome,sindaco)` con un vincolo di integrità referenziale da `cittaNascita` a `Citta[nome]` ed uno da `sindaco` a `Persona[CF]`. Tentiamo di procedere così:

```
create table persona(  
  cf varchar(30) primary key,  
  cittaNascita varchar(30)) references citta;
```

```
create table citta(  
  nome varchar(30) primary key,  
  sindaco varchar(100) references persona);
```

**errore: la
tabella citta
non è definita**



Un'importante applicazione di `alter table`

Vogliamo definire due tabelle: `Persona(cf,cittaNascita)`, `Citta(nome,sindaco)` con un vincolo di integrità referenziale da `cittaNascita` a `Citta[nome]` ed uno da `sindaco` a `Persona[CF]`. Procediamo allora così:

```
create table persona(  
  cf varchar(30) primary key,  
  cittaNascita varchar(30)); -- vincolo su cittaNascita non definito
```

```
create table citta(  
  nome varchar(30) primary key,  
  sindaco varchar(100)  
  constraint vincolo1 foreign key (sindaco) references persona);  
-- vincolo definito su sindaco
```

Ora che abbiamo definito `citta` possiamo aggiungere il vincolo su `cittaNascita` mediante “`alter table`”:

```
alter table persona  
add constraint vincolo2 foreign key(cittaNascita) references citta;
```



Modifiche degli schemi: **drop table**

drop table: elimina una tabella

Sintassi:

```
drop table NomeTabella restrict | cascade
```

Esempio:

```
drop table Infrazioni restrict o semplicemente
```

```
drop table Infrazioni
```

- elimina la tabella solo se non ci sono riferimenti ad essa (con vincoli di integrità referenziali)

```
drop table Infrazioni cascade
```

- elimina la tabella e tutte le tabella (o più in generale tutti gli oggetti del database) che si riferiscono ad essa



Definizione di indici

Definizione di indici:

- è rilevante dal punto di vista delle prestazioni
- riguarda il livello fisico, non quello logico
- in passato era importante perché in alcuni sistemi era l'unico mezzo per definire chiavi
- istruzione **create index**
- Sintassi (semplificata):

**create [unique] index NomeIndice on
NomeTabella Attributo,...,Attributo)**

- *Esempio:*

**create index IndiceIP on
Infrazioni (Provincia)**



Catalogo o dizionario dei dati

Ogni sistema relazionale mette a disposizione delle tabelle già definite che raccolgono tutti i “meta dati”, ossia dati relativi a:

- **tabelle**
- **attributi**
- **altri elementi della base di dati e del suo schema**

Ad esempio, la tabella **Columns** contiene i campi:

- **Column_Name**
- **Table_name**
- **Ordinal_Position**
- **Column_Default**
- ...

Esempio (ipotetico):

select Column_Name from Columns where Table:Name = 'impiegato'

Si rimanda al manuale per i modi in cui si possono consultare i meta dati in PostgreSQL.



3. Il Linguaggio SQL

3.2 Manipolazione dei dati

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



Operazioni di aggiornamento in SQL

- operazioni di
 - inserimento: insert
 - eliminazione: delete
 - modifica: update
- di una o più tuple di una tabella
- sulla base di una condizione che può coinvolgere anche altre tabelle



Inserimento: sintassi

```
insert into Tabella [ ( Attributi ) ]  
values ( Valori )
```

oppure

```
insert into Tabella [ ( Attributi ) ]  
select ...
```

**inserimento di
tuple con i
valori specificati**


**inserimento delle tuple
che sono il risultato
di una select**



Inserimento: esempio

```
insert into persone values ('Mario',25,52)
```

```
insert into persone(nome, eta, reddito)  
values ('Pino',25,52)
```

 **CONTROLLA
PRIMA I
VINCOLI**

```
insert into persone(nome, reddito)  
values ('Lino',55)
```

```
insert into persone (nome)  
select padre  
from paternita  
where padre != 'Giovanni'
```



Inserimento: commenti

- l'ordinamento degli attributi (se presente) e dei valori è significativo
- le due liste di attributi e di valori debbono avere lo stesso numero di elementi
- se la lista di attributi è omessa, si fa riferimento a tutti gli attributi della tabella, secondo l'ordine con cui sono stati definiti nella «create table»
- se la lista di attributi non contiene tutti gli attributi della tabella, per gli altri viene inserito il valore di default o, se il valore di default non è specificato, il valore nullo (che deve essere permesso)



Eliminazione di tuple

Sintassi:

```
delete from Tabella [ where Condizione ]
```

Esempi:

```
delete from persone  
where eta < 35
```

```
delete from paternita  
where figlio != 'Paolo'
```



Eliminazione: commenti

- elimina le tuple che soddisfano la condizione nella clausola **where**
- può causare (se i vincoli di integrità referenziale sono definiti con politiche di reazione **cascade**) eliminazioni da altre relazioni (si veda dopo)
- ricordare: se la **where** viene omessa, si intende **where true**

Modifica di tuple

- **Sintassi:**

update *NomeTabella*

set *Attributo* = < *Espressione* | **select** ... | **null** | **default** >
[**where** *Condizione*]

- **Semantica:** vengono modificate le tuple della tabella che soddisfano la condizione “where” secondo quanto stabilito dalla clausola “set”

- *Esempi:*

```
update persone set reddito = 45  
where nome = 'Piero'
```

```
update persone set reddito = reddito * 1.1  
where eta < 30
```




3. Il Linguaggio SQL

3.2 Manipolazione dei dati

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



Interrogazioni annidate

- Nelle condizioni atomiche (in particolare nella **where**) può comparire una **select** (sintatticamente, deve comparire tra parentesi).
- In particolare, le condizioni atomiche permettono:
 - il confronto fra un attributo (o più attributi) e il risultato di una sottointerrogazione
 - quantificazioni esistenziali



Interrogazioni annidate: esempio

Nome e reddito del padre di Franco.

```
select  nome, reddito
from    persone, paternita
where   nome = padre and figlio = 'Franco'
```

```
select  nome, reddito
from    persone
where   nome = (select padre
                  from    paternita
                  where   figlio = 'Franco')
```

Interrogazioni annidate: **semantica**

La semantica di una «select» che nella clausola «where» ha una query annidata non cambia rispetto a quanto già detto: vengono analizzate le tuple del prodotto cartesiano delle tabelle nella «from» e per ognuna di esse viene valutata l'espressione booleana nella «where». Quello che cambia è che adesso per valutare l'espressione booleana nella «where» occorre valutare la «select» annidata! Quindi possiamo assumere che la query annidata venga eseguita per ogni tupla del prodotto cartesiano delle tabelle nella «from» (anche se il motore SQL tenterà di ottimizzare il processo).

```
select  nome, reddito
from    persone
where   nome = (select padre
                from    paternita
                where   figlio = 'Franco')
```

per ogni tupla di persone viene valutata la clausola «where» e questo comporta l'esecuzione di questa query per ognuna delle tuple di persone

Interrogazioni annidate: operatori

Il risultato di una interrogazione annidata può essere oggetto di confronto nella clausola **where** mediante diversi **operatori**: ad esempio, uguaglianza, disuguaglianza, altri operatori di confronto, ecc.

In questo caso il risultato della interrogazione annidata deve essere costituito al massimo da un elemento, considerando che se il risultato è vuoto la condizione nella clausola **where** sarà considerata falsa.

Se non si è sicuri che il risultato sia costituito al massimo da un elemento, si può far precedere l'interrogazione annidata da:

- **any**: la condizione nella clausola **where** sarà vera se la valutazione della espressione di confronto restituisce "true" per **almeno una** delle tuple risultato dell'interrogazione annidata
- **all**: la condizione nella clausola **where** sarà vera se la valutazione della espressione di confronto restituisce "true" per **tutte** le tuple risultato dell'interrogazione annidata

Si può usare

- l'operatore **in**, che è equivalente a **=any**
- l'operatore **not in**, che è equivalente a **<>all**
- l'operatore **exists**

Interrogazioni annidate: esempio

Nome e reddito dei padri di persone che guadagnano più di 20 milioni.

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

```
select nome, reddito
from   persone
where  nome = any (select padre
                   from   paternita, persone
                   where  figlio = nome
                   and reddito > 20)
```

padri di persone che
guadagnano più di 20 milioni



Interrogazioni annidate: esempio

Nome e reddito dei padri di persone che guadagnano più di 20 milioni.

```
select nome, reddito
from persone
where nome in (select padre
               from paternita, persone
               where figlio = nome
               and reddito > 20)
```

padri di persone che guadagnano più di 20 milioni

Interrogazioni annidate: esempio

Nome e reddito dei padri di persone che guadagnano più di 20 milioni.

```
select nome, reddito
from persone
where nome in (select padre
                from persone
                where
                and r
```

padri di persone
che guadagnano più di
20 milioni

persone che
guadagnano più di 20
milioni

```
select nome, reddito
from persone
where nome in (select padre
                from paternita
                where figlio in (select nome
                                from persone
                                where reddito > 20)
                )
```




Interrogazioni annidate: esempio di all

Persone che hanno un reddito maggiore del reddito di tutte le persone con meno di 30 anni.

```
select nome
from   persone
where  reddito > all ( select reddito
                        from persone
                        where eta < 30 )
```

Interrogazioni annidate: esempio di exists

L'operatore **exists** forma una espressione che è vera se il risultato della sottointerrogazione **non è vuota**.

Esempio: le persone che hanno almeno un figlio.

```
select *  
from   persone p  
where  exists (select *  
                from   paternita  
                where  padre = p.nome)  
or  
       exists (select *  
                from   maternita  
                where  madre = p.nome)
```

POSSO USARE
UN'ALIAS ESTERNO
NELLA QUERY INTERNA

Si noti che l'attributo **p.nome** si riferisce alla tabella nella clausola **from** esterna.



Esercizio 9: interrogazioni annidate

Nome ed età delle madri che hanno almeno un figlio minorenni.

Soluzione 1: un join per selezionare nome ed età delle madri, ed una sottointerrogazione per la condizione sui figli minorenni.

Soluzione 2: due sottointerrogazioni e nessun join.

1

```

SELECT P.NOME, P.ETÀ
FROM PERSONA P JOIN MATERNITÀ m ON P.NOME = m.MADRE
WHERE m.FIGLIO IN (SELECT NOME
                    FROM PERSONA
                    WHERE ETÀ < 18)

```

2

```

SELECT P.NOME, P.ETÀ
FROM PERSONA P
WHERE P.NOME IN (SELECT MADRE
                  FROM MATERNITÀ
                  WHERE FIGLIO IN
                    (SELECT NOME
                     FROM PERSONA
                     WHERE ETÀ < 18)
                  )

```



Esercizio 9: soluzione 1

Nome ed età delle madri che hanno almeno un figlio minorenni.

```
select nome, eta
from   persone, maternita
where  nome = madre and
       figlio in (select nome
                  from   persone
                  where  eta < 18)
```



Esercizio 9: soluzione 2

Nome ed età delle madri che hanno almeno un figlio minorenni.

```
select nome, eta
from persone
where nome in (select madre
                from maternita
                where figlio in (select nome
                                from persone
                                where eta<18))
```

Interrogazioni annidate: **semantica e visibilità**

- **Semantica**: l'interrogazione interna viene eseguita una volta **per ciascuna tupla** nel risultato dell'interrogazione esterna

DA SAPERE



- Vale la classica regola di **visibilità** dei linguaggi di programmazione:
 - Una variabile X è visibile in una “select” B se X è definita in B oppure se X è (ricorsivamente) visibile nella “select” in cui B è definita, a meno che X non sia mascherata, ossia in B sia definita una variabile in B con lo stesso nome di X.
 - In altre parole, si può fare riferimento a variabili definite nello stesso blocco o in blocchi più esterni, a meno che esse non siano mascherate da definizioni di variabili di uguale nome. Ovviamente, se un nome di variabile (o tabella) è omesso, si assume il riferimento alla variabile (o tabella) più “vicina”



Interrogazioni annidate: visibilità

Le persone che hanno almeno un figlio.

```
select *  
from persone  
where exists (select *  
              from paternita  
              where padre = nome)  
or  
exists (select *  
        from maternita  
        where madre = nome)
```

si riferisce a

si riferisce a

L'attributo **nome** si riferisce alla tabella **persone** nella clausola **from** più vicina.



Ancora sulla visibilità

Attenzione alle regole di visibilità; questa interrogazione è **scorretta**:

```
select *  
from impiegato  
where dipart in (select nome  
                  from dipartimento D1  
                  where nome = 'Produzione')  
or  
dipart in (select nome  
            from dipartimento D2  
            where D2.citta = D1.citta)
```

impiegato

nome	cognome	dipart
------	---------	--------

dipartimento

nome	indirizzo	citta
------	-----------	-------



Visibilità: variabili in blocchi interni

Nome e reddito dei padri di persone che guadagnano più di 20 milioni,
con indicazione del reddito del figlio.

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

In questo caso l'interrogazione annidata "intuitiva" **non è corretta:**

```
select nome, reddito, f.reddito
from persone
where nome in (select padre
               from paternita
               where figlio in (select nome
                               from persone f
                               where f.reddito > 20) )
```



Interrogazioni annidate e correlate

Può essere necessario usare in blocchi interni variabili (alias) definite in blocchi esterni; si parla in questo caso di interrogazioni annidate e **correlate**.

Esempio: i padri i cui figli guadagnano tutti più di venti milioni.

```
select distinct padre      z È UNA TUPLA
from paternita z
where not exists
  (select *
   from paternita p join persona f on p.figlio = f.nome
   where z.padre = p.padre and f.reddito <= 20)
```



Esercizio 10: interrogazioni annidate e correlate

Nome ed età delle madri che hanno almeno un figlio la cui età differisce meno di 20 anni dalla loro. Usare una query annidata e correlata.



Esercizio 10: soluzione

Nome ed età delle madri che hanno almeno un figlio la cui età differisce meno di 20 anni dalla loro. Usare una query annidata e correlata.

```
select p.nome, p.eta
from   persone p, maternita m
where  p.nome = m.madre and
       m.figlio in (select nome
                    from   persone
                    where  p.eta - eta < 20)
```



Differenza mediante annidamento

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

```
select nome
from impiegato
where nome not in (select cognome
                   from impiegato)
```



Intersezione mediante annidamento

```
select nome from impiegato
intersection
select cognome from impiegato
```

```
select nome
from   impiegato
where  nome in (select cognome
                from impiegato)
```



Esercizio 11: annidamento e funzioni

La persona (o le persone) con il reddito massimo.

Esercizio 11: soluzione

La persona (o le persone) con il reddito massimo.

```
select *  
from persone  
where reddito = (select max(reddito)  
                 from persone)
```

oppure:

```
select *  
from persone  
where reddito >= all (select reddito  
                      from persone)
```

ma non:

```
select *  
from persone  
where reddito = max(reddito)
```

Le funzioni aggregate possono comparire solo nella select, non nella where, perché la condizione where viene valutata per ogni tupla risultante dalla from e applicare una funzione aggregata ad una singola tupla non ha senso



Interrogazioni annidate: condizione su più attributi

Le persone che hanno la coppia (età, reddito) diversa da tutte le altre persone.

Quando vogliamo denotare una tupla
dobbiamo specificare la lista dei suoi elementi
separati da virgola e la lista deve essere
racchiusa in parentesi tonde

```
select *  
from persone p  
where (eta,reddito) not in  
      (select eta, reddito  
       from persone  
       where nome <> p.nome)
```

Interrogazioni annidate nella clausola from

Finora abbiamo parlato di query annidate nella clausola where. Ma anche nella clausola from possono apparire query racchiuse tra parentesi, come ad esempio

```
select p.padre  
from paternita p, (select nome  
                   from persone  
                   where eta > 30) f  
where f.nome = p.figlio
```

“query derivata”
o “vista inline”

Una **vista** è una tabella **le cui tuple sono derivate da altre tabelle mediante una interrogazione.**

La **semantica** è la solita, con la differenza che la tabella il cui alias è **f**, definita come query annidata nella clausola **from**, invece che essere una tabella della base di dati, è una vista calcolata mediante la associata query **select** racchiusa tra parentesi.



Importanza delle “viste inline”

Supponiamo di avere le seguenti relazioni

EsamiTriennale(matricola, corso, voto)

EsamiMagistrale(matricola, corso, voto)

e di volere la media dei voti di tutti gli esami (sia nella triennale sia nella magistrale) dei vari studenti.

```
select v.matricola, avg(v.voto)
from (select matricola, corso, voto
      from EsamiTriennale
      union
      select matricola, corso, voto
      from EsamiMagistrale) v
group by v.matricola
```



Altro modo di definire e usare le “viste inline”

Supponiamo di avere le seguenti relazioni

EsamiTriennale(matricola, corso, voto)

EsamiMagistrale(matricola, corso, voto)

e di volere la media dei voti di tutti gli esami (sia nella triennale sia nella magistrale) degli studenti.

with miavista **as**

```
(select matricola, corso, voto  
from EsamiTriennale  
union  
select matricola, corso voto  
from EsamiMagistrale)
```

```
select miavista.matricola, avg(miavista.voto)  
from miavista  
group by miavista.matricola
```

Interrogazioni annidate nella target list

Finora abbiamo parlato di query annidate nella clausola where o nella clausola from. Ma anche nella target list possono apparire query racchiuse tra parentesi, come ad esempio

```
select p.nome, (select count(*) from persone where eta=p.eta)
from persone p
where reddito > 10
```

“vista inline” nella
target list

La query riportata qui sopra calcola il nome di ogni persona p con reddito maggiore di 10 ed accanto a tale nome mostra anche il numero di persone che hanno la stessa età della persona p

La **semantica** è la solita: per ogni tupla selezionata della tabella calcolata nella clausola from e selezionata dalla clausola where, viene calcolata la target list, e la novità sta nel fatto che questo richiede l'esecuzione di tutte le viste inline che troviamo nella target list.



3. Il Linguaggio SQL

3.4 Ulteriori aspetti

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

Clausola CASE

Nella target list può essere utile produrre un valore che dipende da una serie di condizioni. A questo scopo si può usare, negli elementi della target list, la clausola CASE, la cui sintassi è:

```
case <espressione>
  when <condizione> then <espressione>
  ...
  when <condizione> then <espressione>
  [ else <espressione> ]
end
```

e la semantica è tale per cui l'espressione generata in uscita per un elemento della target list con **case** è

- quella specificata accanto alla prima condizione dopo **when** che viene valutata «vera», oppure
- quella accanto alla **else**, se nessuna condizione dopo **when** è vera e la clausola **else** compare, oppure ancora
- **null** se nessuna condizione dopo **when** è vera e la clausola **else** non compare.



Clausola CASE

Consideriamo ad esempio la relazione Persona(nome, età, reddito) e supponiamo di volere produrre una relazione formata dalle tuple ottenute da quelle di Persona con età maggiore di 10 aggiungendo un attributo «fasciaeta» i cui possibili valori sono 'ragazzo' (se età ≤ 20), 'adulto' (se età > 20 e ≤ 50), 'anziano' (altrimenti).

La query è:

clausola case

```
select case
    when eta <= 20 then 'ragazzo'
    when eta > 20 and eta <= 50 then 'adulto'
    else 'anziano'
end
    as fasciaeta, reddito
from persona
where eta > 10
```

Vincoli di integrità generici: check

Per specificare vincoli di tupla o vincoli più complessi su una sola tabella:

check (*Condizione*)

```
create table impiegato
( matricola character(6),
  cognome character(20),
  nome character(20),
  sesso character not null check (sesso in ('M', 'F'))
  stipendio integer,
  superiore character(6),
  check (stipendio <= (select stipendio
                        from   impiegato j
                        where  superiore = j.matricola))
)
```

Purtroppo, gli attuali DBMS (compreso PostgreSQL) accettano “check” nella “create table” solo se esse non contengono query annidate. Ne segue che la seconda “check” dell’esempio non viene accettata da PostgreSQL. Altri DBMS, come MySQL, accettano la clausola “check”, ma la ignorano!

Vincoli di integrità generici: asserzioni

Specifica vincoli a livello di schema. Sintassi:

```
create assertion NomeAss check ( Condizione )
```

Esempio:

```
create assertion AlmenoUnImpiegato  
check (1 <= (select count(*)  
             from impiegato))
```

Purtroppo, gli attuali DBMS (compreso PostgreSQL) non accettano istruzioni di tipo “create assertion”.

Viste

- Come abbiamo già detto, una vista è una tabella **la cui istanza è derivata da altre tabelle mediante una interrogazione**. Per definire una vista nella base di dati:

`create view NomeVista [(ListaAttributi)] as SelectSQL`

- Le viste sono tabelle virtuali: solo quando vengono utilizzate (ad esempio in altre interrogazioni) la loro istanza viene calcolata.

- *Esempio:*

```
create view ImpAmmin(Mat,Nome,Cognome,Stip) as
select Matricola, Nome, Cognome, Stipendio
from    Impiegato
where   Dipart = 'Amministrazione' and
        Stipendio > 10
```



Un'interrogazione con annidamento nella having

- Voglio sapere l'età delle persone cui corrisponde il massimo reddito (come somma dei redditi delle persone che hanno quella età).
- Assumendo che non ci siano valori nulli in reddito, usando l'annidamento nella having, otteniamo questa soluzione:

```
select eta
from   persone
group by eta
having sum(reddito) >= all (select sum(reddito)
                           from persone
                           group by eta)
```

- Un'altro metodo è definire una vista.



Soluzione con le viste

```
create view etaReddito(eta,totaleReddito) as  
  select eta, sum(reddito)  
  from   persone  
  group by eta
```

```
select eta  
from   etaReddito  
where  totaleReddito = (select max(totaleReddito)  
                        from etaReddito)
```



Tabelle e viste temporanee

- In molti DBMS basati su SQL è possibile specificare che una tabella o una vista che stiamo creando è temporanea, ovvero che sparirà alla fine della sessione di connessione con il sistema
- Ad esempio: create temporary table MiaTabella ... oppure create temporary view MiaVista ...
- Possiamo anche cancellare con la drop table o drop view la tabella o la vista così creata prima della fine della sessione, ma in ogni caso essa sarà eliminata alla fine della sessione, se non l'abbiamo fatto prima.
- Le tabelle o viste temporanee possono essere utili per salvare nella base di dati i risultati di una query in modo temporaneo.



Privilegi

- Un privilegio è caratterizzato da:
 - la risorsa cui si riferisce
 - l'utente che concede il privilegio
 - l'utente che riceve il privilegio
 - l'azione che viene permessa
 - la trasmissibilità del privilegio
- Tipi di privilegi
 - **insert**: permette di inserire nuovi oggetti (tuple)
 - **update**: permette di modificare il contenuto
 - **delete**: permette di eliminare oggetti
 - **select**: permette di leggere la risorsa
 - **references**: permette la definizione di vincoli di integrità referenziale verso la risorsa (può limitare la possibilità di modificare la risorsa)
 - **usage**: permette l'utilizzo in una definizione (per esempio, di un dominio)

grant e revoke

- **Concessione** di privilegi:

`grant < Privileges | all privileges > on
Resource to Users [with grantOption]`

- *grantOption* specifica se il privilegio può essere trasmesso ad altri utenti

`grant select on Dipartimento to Giuseppe`

- **Revoca** di privilegi:

`revoke Privileges on Resource from Users
[restrict | cascade]`



Transazione

- Insieme di operazioni da considerare indivisibile (“atomico”), corretto anche in presenza di concorrenza, e con effetti definitivi.
- Proprietà (“**ACIDE**”):
 - Atomicità
 - Consistenza
 - Isolamento
 - Durabilità (persistenza)



Le transazioni sono ... atomiche

- La sequenza di operazioni sulla base di dati viene eseguita per intero o per niente.

Esempio: trasferimento di fondi da un conto A ad un conto B:
o si fa sia il prelevamento da A sia il versamento su B, o nessuno dei due.



Le transazioni sono ... consistenti

- Al termine dell'esecuzione di una transazione, i vincoli di integrità debbono essere soddisfatti.
- “Durante” l'esecuzione si può chiedere di accettare violazioni di vincoli (si veda più avanti il comando SET CONSTRAINTS DEFERRED), in particolare per quei vincoli definiti “deferrable”.
- Se anche una violazione rimane alla fine, allora la transazione deve essere annullata per intero (“abortita”) .



Le transazioni sono ... isolate

- L'effetto di transazioni concorrenti deve essere coerente (ad esempio “equivalente” all'esecuzione separata).

Esempio: se due assegni emessi sullo stesso conto corrente vengono incassati contemporaneamente si deve evitare di trascurarne uno.



I risultati delle transazioni sono durevoli

- La conclusione positiva di una transazione corrisponde ad un impegno (in inglese **commit**) a mantenere traccia del risultato in modo definitivo, anche in presenza di guasti e di esecuzione concorrente.



Transazioni in SQL

Ogni istruzione SQL non eseguita all'interno di una transazione definita esplicitamente è considerata una transazione.

Ma si possono definire transazioni esplicite mediante le seguenti istruzioni fondamentali

- begin (o **begin transaction**): specifica l'inizio della transazione (le operazioni non vengono eseguite sulla base di dati)
- commit (o **commit work**, o **end**, o **end transaction**): le operazioni specificate a partire dal **begin** vengono rese permanenti sulla base di dati
- rollback (o **rollback work**): si disfano gli effetti delle operazioni specificate dall'ultimo **begin**



Esempio di transazione in SQL

```
begin;  
    update ContoCorrente  
        set Saldo = Saldo - 10  
        where NumeroConto = 12345;  
    update ContoCorrente  
        set Saldo = Saldo + 10  
        where NumeroConto = 55555;  
commit;
```




Vincoli di foreign key: reazioni ad aggiornamenti

SQL dà la possibilità di definire le cosiddette “politiche di reazione alla violazione di vincoli di integrità referenziali”, cioè delle azioni da eseguire quando, aggiornando la base di dati, si violano i vincoli di foreign key.

Nel seguito, se in una tabella $T1$ è definito un vincolo V di integrità referenziale verso la tabella $T2$, allora la tabella $T1$ viene detta tabella “figlia” per il vincolo V e la tabella $T2$ viene detta tabella “padre” (perché ha tutti i valori della tabella figlia) per V .

Ad esempio, se consideriamo:

```
create table persona(cf varchar(100) primary key, cittanascita varchar(30));  
create table citta(nome varchar(30) primary key, sindaco varchar(100)  
                constraint vincolo1 references persona(cf));
```

la tabella persona è la tabella “padre” per il vincolo **vincolo1** e la tabella citta è la tabella “figlia” per il vincolo **vincolo1**.

Analogamente, la tupla ('2000','Genova') nella tabella persona viene detta tupla “padre” rispetto alla tupla ('Savona','2000') nella tabella citta, perché ('2000','Genova') è la tupla che contiene nella chiave il valore referenziato dalla tupla ('Savona','2000') secondo il vincolo di foreign key. Corrispondentemente, la tupla ('Savona','2000') nella tabella citta viene detta tupla “figlia” rispetto alla tupla ('2000','Genova').



Vincoli di foreign key: reazioni ad aggiornamenti

Specifica di vincolo di foreign key nella tabella T1 (tabella “figlia”) verso la tabella T2 (tabella “padre”):

FOREIGN KEY (Attributi) REFERENCES Tabella [(Attributo)]

[ON DELETE (NO ACTION | CASCADE | RESTRICT | SET DEFAULT | SET NULL)]

[ON UPDATE (NO ACTION | CASCADE | RESTRICT | SET DEFAULT | SET NULL)]

- **no action**: se si tenta di cancellare/aggiornare la tupla “padre” (quella nella tabella padre T2) di una tupla della tabella figlia T1, si genera un errore al momento della verifica del vincolo (dopo l’azione stessa a tutte le azioni ad essa collegate). Si noti che, se il vincolo è deferred, questo momento è la fine della transazione. Si noti anche che **no action** è il default: cioè è l’opzione che vale se non si specifica nulla.
- **restrict**: se si tenta di cancellare/aggiornare la tupla “padre”, si genera un errore immediato (prima dell’azione; questo significa che se il comando è all’interno della transazione, non si aspetta il momento la fine della transazione, nemmeno se il vincolo è “deferred”)
- **cascade**: quando si cancella/aggiorna la tupla “padre”, si cancella/aggiorna anche la tupla della tabella T1 che riferenzia la tupla “padre”
- **set default**: quando si cancella/aggiorna la tupla “padre”, si memorizza il valore di default nell’attributo di T1 che è definito come foreign key
- **set null**: quando si cancella/aggiorna la tupla “padre”, si memorizza il valore NULL nell’attributo di T1 che è definito come foreign key



Transazioni in SQL: vincoli “deferred”

- Un vincolo “deferrable” è un vincolo che si può definire “deferred” (opposto di IMMEDIATE) all’interno di una transazione. Quando un vincolo è definito deferred in una transazione, esso viene controllato alla fine della transazione, invece che immediatamente.
- Per specificare un vincolo “deferrable” si deve aggiungere alla definizione di vincolo la parola DEFERRABLE (aggiungendo, se vogliamo, anche INITIALLY DEFERRED oppure INITIALLY IMMEDIATE – considerando che il default è INITIALLY IMMEDIATE). Possiamo anche usare NOT DEFERRABLE, che è il default.
- Se un vincolo di nome <nome> è definito come “deferrable”, allora si può dare il comando all’interno di una transazione:
SET CONSTRAINTS <nome> DEFERRED
ed il vincolo di nome <nome> verrà controllato solo alla fine della transazione. Al posto di <nome> si può specificare ALL, se vogliamo che tutti i vincoli DEFERRABLE siano deferred.
- Se vogliamo tornare alla situazione IMMEDIATE, possiamo dare il comando SET CONSTRAINTS <nome> IMMEDIATE, ma attenzione: quando il sistema esegue questa istruzione controlla i vincoli specificati, senza aspettare la fine della transazione.



Esempio

Definiamo due relazioni con vincoli di foreign key definiti mutuamente:

```
create table persona(cf varchar(100) primary key, cittanascita varchar(30));
```

```
create table citta(nome varchar(30) primary key, sindaco varchar(100)  
                constraint vincolo1 references persona(cf));
```

```
alter table persona add constraint vincolo2 foreign key(cittanascita)  
                references citta(nome);
```

Si noti che per i due vincoli di integrità referenziale vale il default “**no action**”. Si noti che né cittanascita né sindaco è NOT NULL. Inseriamo ora due città ed una persona. Possiamo usare NULL per non violare i vincoli:

```
insert into citta values('Roma',null), ('Milano',null);
```

```
insert into persona values('100','Roma');
```



Esempio

Ma attenzione: se eseguiamo la cancellazione della città di nome 'Roma', otteniamo un errore:

```
delete from citta where nome = 'Roma';
```

ERROR: update or delete on table "citta" violates foreign key constraint "vincolo2" on table "persona"

DETAIL: Key (nome)=(Roma) is still referenced from table "persona".

Come risolviamo? Ci sono almeno tre metodi:

- 1) eseguiamo l'update delle tuple di persona che referenziano Roma, mettendo NULL al posto di Roma. In questo modo evitiamo che rimangano tuple figlie orfane di padre, ossia che referenziano Roma quando quest'ultimo valore viene eliminato dalla tabella citta
- 2) potevamo aver definito nella create table "persona" il vincolo di foreign key come cascade: in questo modo una volta che cancelliamo la tupla della città di nome 'Roma' vengono cancellate automaticamente le tuple figlie in persona, ossia tuple corrispondenti a persone nate a Roma
- 3) se nella create table "persona" il vincolo di foreign key è definito come "deferrable", allora per eseguire la cancellazione di Roma possiamo usare una transazione dentro la quale dichiariamo il vincolo "deferred", per cui la cancellazione della tupla della città di nome 'Roma' non causa immediatamente una violazione, visto che la verifica del vincolo sarà effettuata alla fine della transazione. Ovviamente, prima della fine della transazione dobbiamo «mettere a posto» le tuple di persona che referenziavano Roma al fine di eliminare le violazioni di vincoli



Esempio: metodo 1)

Per eseguire la cancellazione della città Roma:

```
update persona set cittanascita = null where cittanascita = 'Roma';
```

```
delete from citta where nome = 'Roma';
```

Ovviamente questa soluzione presuppone che il valore null sia accettabile nel campo cittanascita della tabella persona



Esempio: metodo 2)

Sostituiamo a quelle di prima le seguenti definizioni delle create table, che associano la politica «on delete cascade» al vincolo di foreign key che appare nella tabella persona:

```
create table persona(cf varchar(100) primary key, cittanascita varchar(30));  
create table citta(nome varchar(30) primary key, sindaco varchar(100)  
                constraint vincolo1 references persona(cf);  
alter table persona add constraint vincolo2 foreign key(cittanascita)  
                references citta(nome) on delete cascade;  
insert into citta values('Roma',null), ('Milano',null);  
insert into persona values('100','Roma');
```

Ora cancelliamo la città di Roma, con una semplice «delete»:

```
delete from citta where nome = 'Roma';
```

e non otteniamo un errore, ma otteniamo la cancellazione delle persone che avevano la città di nascita pari a 'Roma'.



Esempio: metodo 3)

Sostituiamo a quelle di prima le seguenti definizioni delle create table, che ora dichiarano i vincoli di foreign key «deferrable»:

```
create table persona(cf varchar(100) primary key, cittanascita varchar(30));
create table citta(nome varchar(30) primary key, sindaco varchar(100)
    constraint vincolo1 references persona(cf) deferrable);
alter table persona add constraint vincolo2 foreign key(cittanascita)
    references citta(nome) deferrable;
insert into citta values('Roma',null), ('Milano',null);
insert into persona values('100','Roma');
```

Per cancellare la città di Roma, usiamo una transazione in cui definiamo vincolo2 come deferred (che quindi verrà controllato alla fine della transazione):

```
begin;
SET CONSTRAINTS vincolo2 DEFERRED;
delete from citta where nome = 'Roma';
<< mettiamo a posto le tuple di persona che referenziano Roma>>;
end;
```


MARITO

<u>ID</u>	CONSORTE
10	100

MOGLIE

<u>ID</u>	CONSORTE
100	10



```
CREATE TABLE MOGLIE (
    ID INT PRIMARY KEY
    CONSORTE INT NOT NULL
)
```

```
CREATE TABLE MARITO (
    ID INT PRIMARY KEY
    CONSORTE INT NOT NULL, CONSTRAINT FK1 MOGLIE FOREIGN KEY (CONSORTE),
    REFERENCES MOGLIE DEFERRABLE
)
```

```
ALTER TABLE MOGLIE ADD CONSTRAINT FK MARITO
FOREIGN KEY (CONSORTE) REFERENCES MARITO DEFERRABLE;
```

```
BEGIN;
SET CONSTRAINTS FK MARITO DEFERRED;
SET CONSTRAINTS FK MOGLIE DEFERRED;
INSERT INTO MARITO VALUES ('10', '100');
INSERT INTO MOGLIE VALUES ('100', '10');
END;
```