SQL

Esercitazioni pratiche

- Per SQL è possibile (e fondamentale) svolgere esercitazioni pratiche
- Verranno anche richieste come condizione per svolgere le prove parziali
- Soprattutto sono utilissime
- Si può utilizzare qualunque DBMS
  - IBM DB2, Microsoft SQL Server, Oracle, PostgreSQL o anche un servizio online (Sqliteonline)
- A lezione utilizziamo PostgreSQL e Sqliteonline

Nota

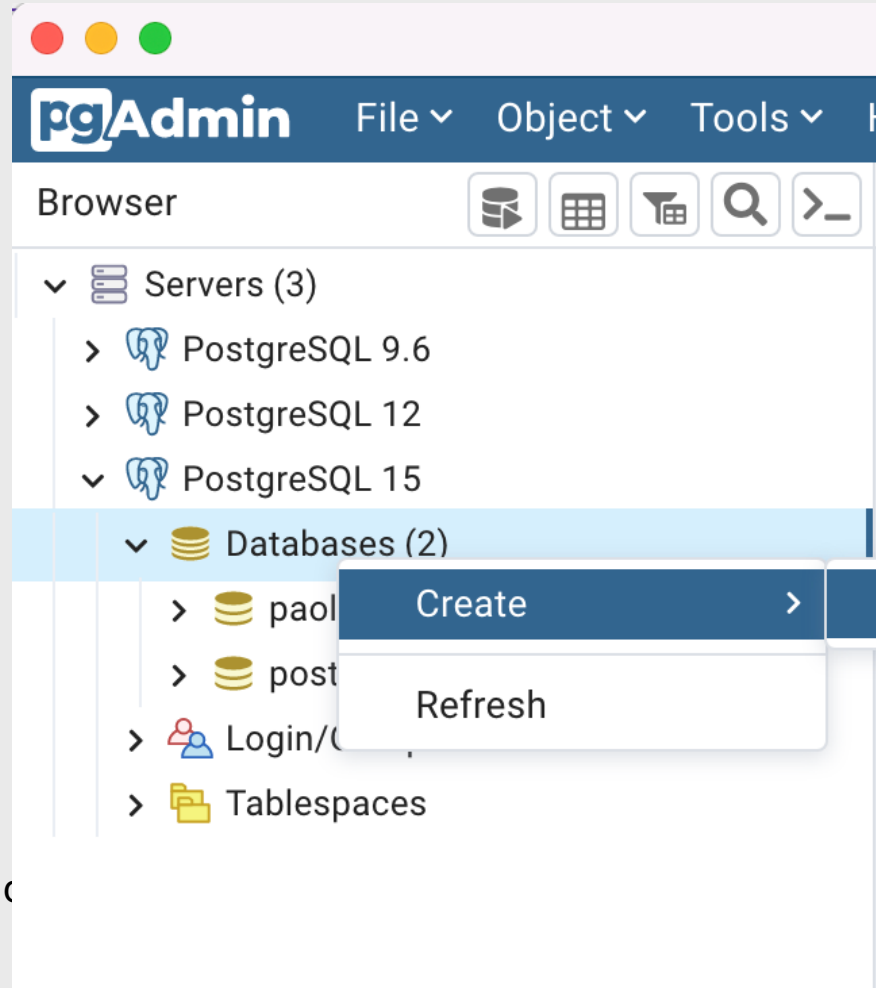
- Un dbms è un programma eseguito in background (un "demone"), cioè senza che sia sotto il controllo dell'utente
- Rimane in ascolto su una porta TCP, attraverso la quale interagisce con uno o piu' client
  - Tipicamente, un DBMS riceve in input comandi SQL e produce in output dati relazionali

Postgres

- Usiamo il server dbms (postgres) e un client (PgAdmin)
- Scaricare Postgres https://www.postgresql.org/download/
- L'installer include
  - il server (postgres)
  - un client (PgAdmin), che usiamo per interagire con il server (mandare istruzioni SQL, visualizzare i risultati)
- La porta TCP di default usata Postgres è la 5432
  - Manteniamo il default
- Durante l'installazione, il server ci chiede di impostare uno username (e relativa password): è l'utente autorizzato ad interagire con il server
  - Consiglio: username=postgres, password=postgres

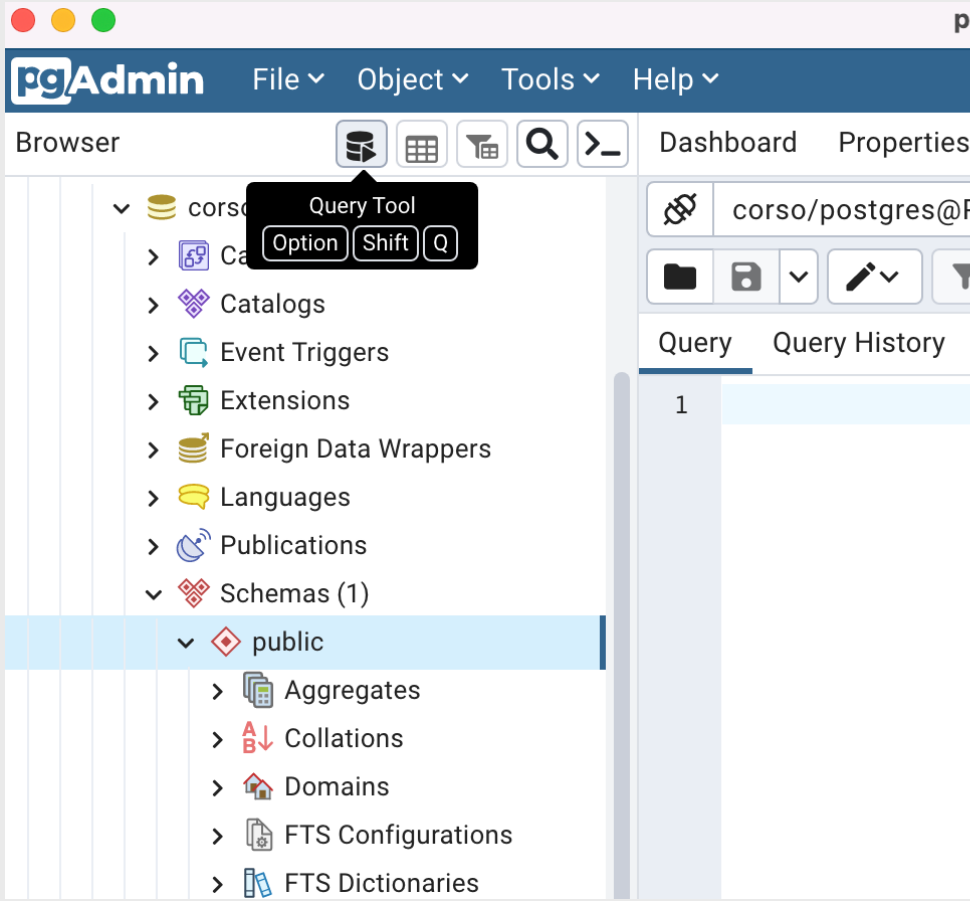
Come usare PgAdmin (per interagire con Postgres)

- Lanciare pgAdmin
- Creare un database (ad esempio: corso)



Come usare PgAdmin (per interagire con Postgres)

- Espandere l'albero a sinistra fino a "Schemas"
- Possiamo creare uno schema o usare lo schema pubblico (noi usiamo lo schema pubblico)
- E poi
  - lavorare sugli elementi dello schema
  - oppure lanciare "Query tool" (SQL interattivo): scelta consigliata



Come usare Sqliteonline

https://sqliteonline.com/

- Ne abbiamo già parlato (vedi lucidi sul modello relazionale)

CREATE TABLE, esempi

```
CREATE TABLE corsi (
    codice numeric NOT NULL PRIMARY KEY,
    titolo character(20) NOT NULL,
    cfu numeric NOT NULL);

CREATE TABLE esami (
    corso numeric REFERENCES corsi (codice),
    studente numeric REFERENCES studenti,
    data date NOT NULL,
   voto numeric NOT NULL,
   PRIMARY KEY (corso, studente));
```

La chiave primaria viene definita come NOT NULL anche se non lo specifichiamo (in Postgres)

Creazione di tabelle, in pratica

- In molti sistemi si utilizzano strumenti diversi dal codice SQL per definire lo schema della base di dati
  - Ad esempio, in PgAdmin si può creare una tabella attraverso l'interfaccia grafica

```
CREATE TABLE tabella (
    attributo1 tipo vincoli,
    attributo2 tipo vincoli,
    …
    attributon tipo vincoli);
```

SQL, operazioni sui dati

- interrogazione:
  - SELECT
- modifica:
  - INSERT, DELETE, UPDATE

Inserimento

(necessario per gli esercizi)

```
INSERT INTO Tabella [ ( Attributi ) ]
        VALUES( Valori )
```

oppure

```
INSERT INTO Tabella [ ( Attributi )]
          SELECT ...
```

(vedremo più avanti)

```
INSERT INTO Persone VALUES ('Mario',25,52)

INSERT INTO Persone(Nome, Reddito, Eta)
       VALUES('Pino',52,23)

INSERT INTO Persone(Nome, Reddito)
       VALUES('Lino',55)
```

Maternità

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

Paternità

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

Persone

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

Esercizi

- Definire la base di dati per gli esercizi
  - installare un sistema
  - creare lo schema
  - creare le relazioni (CREATE TABLE)
  - inserire i dati

  - eseguire le interrogazioni
    - suggerimento, per chi usa Postgres: usare database diversi

```
create table persone (
  nome char (10) not null primary key,
  eta numeric not null,
  reddito numeric not null);
create table paternita (
  padre char (10) references persone,
  figlio char (10) primary key references persone);
...
insert into Persone values('Andrea',27,21);
...
insert into Paternita values('Sergio','Franco');
...
```

Vedere file:

http://dia.uniroma3.it/~atzeni/didattica/BDN/20222023/protected/BD-04-esempiSQL2022 persone.sql

Vediamo gli esempi

- con RelaX

  http://dbis-uibk.github.io/relax/calc/gist/1362a9bb84dd2e4052561b714613b1de

- con Sqliteonline

  https://sqliteonline.com/

  con i dati

  http://dia.uniroma3.it/~atzeni/didattica/BDN/20222023/protected/BD-04-esempiSQL2022 persone.sql

L'espressione più semplice

- In RelaX potevamo scrivere un'espressione composta da una sola relazione

  Paternita

- In SQL dobbiamo scrivere

  ```
  SELECT Padre, Figlio
  FROM Paternita
  ```

- Oppure

  ```
  SELECT *
  FROM Paternita
  ```

Operatori insiemistici

- Unione
- Intersezione
- Differenza

- Nell'algebra:
  - sugli stessi schemi
- Ma in RelaX …
  - … e pure in SQL

Unione

R(A,B) S(A,B)

```
select A,  B
from R
union
select A , B
from S
```

Unione

Unione "normale"

R(A,B) S(A,B)

```
select A,  B
from R
union
select A , B
from S
```

Unione "forzata"

R(A,B) T(C,D)

```
select A,  B
from R
union
select C, D
from T
```

Richiede ennuple "compatibili"

Quali nomi per il risultato?

Approccio posizionale

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

Padre	Figlio
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo
Luisa	Maria
Luisa	Luigi
Anna	Olga
Anna	Filippo
Maria	Andrea
Maria	Aldo

	Figlio
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo
Luisa	Maria
Luisa	Luigi
Anna	Olga
Anna	Filippo
Maria	Andrea
Maria	Aldo

Operazione non commutativa (in molti sistemi)

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

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

Approccio posizionale!

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

OK (o quasi)

Approccio posizionale, 2

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

NO!
Funziona, ma produce un risultato indesiderabile

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

OK

Ridenominazione

R(A,B) A1, B1 nomi diversi

```
select A as A1, B as B1
from R
```

Unione con ridenominazione

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

Notazione posizionale, 3

- Anche con le ridenominazioni il problema resta

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

- Corretta:

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

Le tabelle SQL non sono insiemi!

```
select A,  B
from R
union
select A , B
from S
```

```
select A,  B
from R
union all
select A , B
from S
```

Le tabelle SQL non sono insiemi!

```
select *
from paternita
union
select *
from paternita
```

```
select *
from paternita
union all
select *
from paternita
```

Differenza

```
select A,  B
from R
except
select A , B
from S
```

Intersezione

```
select A,  B
from R
intersect
select A , B
from S
```

Selezione

- Nome, età e reddito delle persone con meno di trenta anni

$$SEL_{Eta<30}(Persone)$$

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

Condizione complessa

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

Proiezione

- Nome e reddito di tutte le persone

$$PROJ_{Nome,\ Reddito}(Persone)$$

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

Selezione e proiezione

- Nome e reddito delle persone con meno di trenta anni

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

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

Proiezione, con ridenominazione

- Nome e reddito di tutte le persone

$$REN_{Anni \leftarrow Eta}(PROJ_{Nome,\ Eta}(Persone))$$

```
select nome, eta as anni
from persone
```

Proiezione, attenzione

```
select
      madre
from maternita
```

```
select distinct
      madre
from maternita
```

Ancora l'anomalia degli operatori inseimistici

```
select padre
from paternita
union
select padre
from paternita
```

```
select padre
from paternita
union all
select padre
from paternita
```

Finalmente, il JOIN

Istruzione SELECT (versione base, su più relazioni)

SELECT ListaAttributi
FROM ListaTabelle
[WHERE Condizione]

- "target list"
- clausola FROM
- clausola WHERE

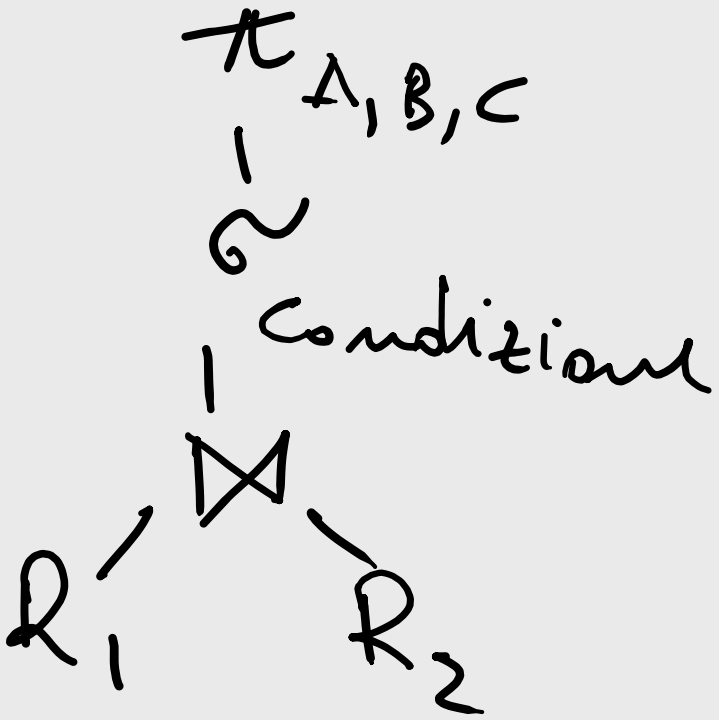
Intuitivamente

SELECT ListaAttributi
FROM ListaTabelle
[WHERE Condizione]

- Prodotto cartesiano di ListaTabelle
- Selezione su Condizione
- Proiezione su ListaAttributi

Intuitivamente

SELECT A, B, C
FROM R1, R2
WHERE Condizione



Una prima interrogazione con selezione, proiezione e join

- I padri di persone che guadagnano più di 20

$$PROJ_{Padre}(paternita \; JOIN_{Figlio = Nome} \; SEL_{Reddito>20}(persone))$$

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

I tre passi

```
select *
from persone, paternita
```

Prodotto cartesiano

```
select *
from persone, paternita
where figlio = nome and reddito > 20
```

Selezione

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

Proiezione

- Vediamoli in azione

Un commento

- In algebra relazionale

$$PROJ_{Padre}(paternita \; JOIN_{Figlio=Nome} \; SEL_{Reddito>20}(persone))$$

$$PROJ_{Padre}(SEL_{Reddito>20}((paternita \; JOIN_{Figlio=Nome} persone)))$$

o anche

$$PROJ_{Padre}(SEL_{Figlio=Nome \; AND \; Reddito>20}((paternita \; JOIN \; persone)))$$

Algebra e SQL

- In algebra possiamo scrivere un'intrerrogazione in più modi e ci sono differenze nell'efficienza
  - l'algebra è procedurale
- In SQL, possiamo dire che è il sistema che si preoccupa dell'efficienza
  - SQL è, almeno in parte, "dichiarativo"

Versione originaria di SQL

- Si specifica
  - prodotto cartesiano
  - selezione
  - proiezione
- Si esegue
  - selezione
  - join (e ulteriore selezione)
  - proiezione

Qualche anno dopo

- È stato introdotto il join esplicito

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

select distinct padre
from persone join paternita on nome =figlio
where reddito > 20
```

Join esplicito

- Nella clausola FROM:
  - equijoin
    - R1 JOIN R2 ON R1.A = R2.B

Un'osservazione

- Con nomi di attributi diversi, è implicito quale sia la relazione cui appartiene un attributo
- Ma se ci sono attributi con lo stesso nome in relazioni diverse?
  - Serve precisare

Necessità di distinzione

- Per ogni persona per la quale entrambi I genitori sono nella base di dati, mostrarli
- Intuitivamente
  - Join di maternità e paternità
  - Ma "Figlio" compare in entrambe
  - Possiamo "premettere" il nome della relazione

```
SELECT Padre, Madre, Paternita.Figlio
FROM Paternita JOIN Maternita
  ON Paternita.Figlio = Maternita.Figlio
```

Join naturale

- Nella clausola FROM:
  - equijoin
    - R1 JOIN R2 ON R1.A = R2.B
  - equijoin su attributi con lo stesso nome
    - R1 JOIN R2 USING (A)

Join naturale

```
SELECT *
FROM Paternita JOIN Maternita USING (Figlio)
```

Necessità di distinzione, ancora

- Può servire di avere più volte la stessa relazione in una interrogazione ("join di una relazione con se stessa")
- Si utilizzano variabili (chiamate "alias" in SQL)

- Le persone che guadagnano più dei rispettivi padri; mostrare nome, reddito e reddito del padre

$$\text{PROJ}_{\text{Nome, Reddito, RP}}\,(\text{SEL}_{\text{Reddito>RP}}\,(\text{REN}_{\text{NP,EP,RP}\leftarrow\text{Nome,Eta,Reddito}}(\text{persone})\ \text{JOIN}_{\text{NP=Padre}}\ (\text{paternita JOIN}_{\text{Figlio =Nome}}\ \text{persone})))$$

- Le persone che guadagnano più dei rispettivi padri; mostrare nome, reddito e reddito del padre

$$\text{PROJ}_{\text{Nome, Reddito, RP}}\,(\text{SEL}_{\text{Reddito>RP}}\,(\text{REN}_{\text{NP,EP,RP}\leftarrow\text{Nome,Eta,Reddito}}(\text{persone})\ \text{JOIN}_{\text{NP=Padre}}\ (\text{paternita JOIN}_{\text{Figlio =Nome}}\ \text{persone})))$$

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

Meglio con ridenominazione del risultato

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

- Le persone che guadagnano più dei rispettivi padri; mostrare nome, reddito e reddito del padre

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

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

Join esterno: "outer join"

- Padre e, se nota, madre di ogni persona

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

select figlio, padre, madre
from paternita left join maternita using(figlio)

select paternita.figlio, padre, madre
from paternita full join maternita using(figlio)
```

Note:

- left outer, full outer, right outer equivalenti a left, full, right
- sqliteonline non supporta full e right;

Ordinamento del risultato

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

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

Espressioni nella target list

```
select Nome, Reddito/12 as redditoMensile
from Persone
```

Attenzione al tipo – guardatelo da soli (non preoccupatevi, i dettagli non sono importanti ai fini dell'esame)

Condizione “LIKE”

- Le persone che hanno un nome che inizia per 'A' e ha una 'd' come terza lettera

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

Gestione dei valori nulli

Persone

Nome	Età	Reddito
Andrea	27	21
…	…	…
Luisa	75	87
Nicola	43	NULL

- Le persone il cui reddito è o potrebbe essere maggiore di 40

SEL (Reddito > 40) OR (Reddito IS NULL) (Impiegati)

Gestione dei valori nulli

Persone

Nome	Età	Reddito
Andrea	27	21
…	…	…
Luisa	75	87
Nicola	43	NULL

- Le persone il cui reddito è o potrebbe essere maggiore di 40 -- per l'esempio, aggiungiamo la ennupla:

```
insert into persone (Nome, Eta) values ('Nicola',43)
```

```
SELECT * FROM Persone
WHERE Reddito > 40 OR Reddito IS null
```

Operatori aggregati: COUNT

- Il numero di figli di Franco

$$\gamma\ \text{count}(*) \rightarrow \text{NumFigliDiFranco}\ (\sigma\ \text{Padre} = \text{'Franco'}\ (\text{Paternita}))$$

Operatori aggregati: COUNT

- Il numero di figli di Franco

γ count(*) $\rightarrow$ NumFigliDiFranco (σ Padre = 'Franco' (Paternita))

```
select count(*) as NumFigliDiFranco
from Paternita
where Padre = 'Franco'
```

COUNT DISTINCT

```
select count(*) from persone

select count(reddito) from persone

select count(distinct reddito) from persone
```

Altri operatori aggregati

- SUM, AVG, MAX, MIN

- Media dei redditi dei figli di Franco

$$\gamma\ \text{avg(Reddito)} \rightarrow \text{RedditoMedioFigliDiFranco}$$
$$(\sigma\ \text{Padre = 'Franco'}\ (\text{Paternita}) \bowtie \text{Figlio=Nome Persone})$$

Altri operatori aggregati

- SUM, AVG, MAX, MIN

- Media dei redditi dei figli di Franco

$$\gamma\ \text{avg(Reddito)} \rightarrow \text{RedditoMedioFigliDiFranco}$$
$$(\sigma\ \text{Padre = 'Franco'}\ (\text{Paternita}) \bowtie \text{Figlio=Nome Persone})$$

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

Operatori aggregati e valori nulli

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

Operatori aggregati e raggruppamenti

- Il numero di figli di ciascun padre

  γ Padre; count(*) $\rightarrow$ NumFigli (Paternita)

Operatori aggregati e raggruppamenti

- Il numero di figli di ciascun padre

  γ Padre; count(*) → NumFigli (Paternita)

  ```
  select Padre, count(*)  AS NumFigli
  from paternita
  group by Padre
  ```

- Attenzione, è opportuno che gli attributi di raggruppamento siano scritti sia nella group by sia nella target list (altrimenti il risultato non è comprensibile)

Operatori aggregati e raggruppamenti

- Il numero di figli di ciascun padre

  γ Padre; count(*) → NumFigli (Paternita)

  ```
  select Padre, count(*)  AS NumFigli
  from paternita
  group by Padre
  ```

- Gli attributi nella target list (Padre) debbono comparire nella GROUP BY
- Purtroppo in alcuni sistemi (come sqliteonlite) questo non accade

Condizioni sui gruppi

- I padri i cui figli hanno un reddito medio maggiore di 25; mostrare padre e reddito medio dei figli

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

Un errore "classico"

- La persona con il reddito massimo

```
select nome
from persone
```

- NO!! Cerchiamo di mettere insieme una ennupla con una aggregazione
- Proviamo con PostgreSQL

Purtroppo

- In alcuni sistemi (es. Sqliteonline) funziona

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

- Ma concettualmente è scorretto: cerca di mettere insieme una ennupla con una aggregazione
- Vediamo una cosa simile:
  - "Le persone con reddito superiore alla media"

Operatori aggregati e target list

- un'interrogazione scorretta:

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

- di chi sarebbe il nome? La target list deve essere omogenea

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

Proviamo …

- Si può fare
  - con i costrutti di SQL che conosciamo
    - con l'aiuto di una vista (concetto che non abbiamo ancora discusso – lo facciamo subito)
  - oppure con le "interrogazioni nidificate"

Proviamo …

- Si può fare
  - **con i costrutti di SQL che conosciamo**
    - **con l'aiuto di una vista (concetto che non abbiamo ancora discusso – lo facciamo subito)**
  - oppure con le "interrogazioni nidificate"

Una soluzione

- "Le persone con reddito superiore alla media"
  - troviamo il reddito medio (vista)
    - Una tabella con un attributo (la media) ed una ennupla (il valore della media)
  - confrontiamo ciascun reddito con il reddito medio (prodotto cartesiano + selezione)

Viste

```
CREATE VIEW V AS
  SELECT …
```

anche (non in tutti i sistemi)

```
CREATE VIEW V AS
 SELECT …
 UNION
 SELECT …
```

```
CREATE OR REPLACE VIEW V AS
  SELECT …
```

I vari passi (persone con reddito superiore alla media)

```
create view  mediaReddito
as select avg(reddito) as redditoMedio
   from persone;
```

Mostrare su sqliteonline

```
select *
from persone, mediaReddito;
```

Mostrare su sqliteonline

```
select nome, reddito, redditomedio
from persone, mediaReddito
where reddito > redditoMedio;
```

Proviamo …

- Si può fare
  - con i costrutti di SQL che conosciamo
    - con l'aiuto di una vista (concetto che non abbiamo ancora discusso – lo facciamo subito)
  - **oppure con le "interrogazioni nidificate"**

Interrogazioni nidificate (nested query o subquery)

- Varie forme di nidificazione
  - nella WHERE
  - nella FROM
  - nella SELECT

- Coerente con i tipi
  - anche Booleano (EXISTS)
    - non lo vediamo quest'anno
  - attenzione al rapporto fra singoli valori e insiemi

Nella WHERE

- La persona che guadagna più di tutte le altre

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

Nella WHERE

- Le persone che guadagnano più della media

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

Correlated subquery

- Per ogni padre, il figlio che guadagna di più

```
select  padre, figlio, reddito
from persone  join paternita p on nome = figlio
where reddito = (    select max(reddito)
                     from persone join paternita
                          on nome = figlio
                     where padre = p.padre )
```

- L'interrogazione interna viene eseguita una volta per ciascuna ennupla della FROM esterna

Correlated subquery

- Per ogni padre, il figlio che guadagna di più

```
select  padre, figlio, reddito
from persone  join paternita p on nome = figlio
where reddito = (    select max(reddito)
                     from persone join paternita
                          on nome = figlio
                     where p.padre = padre )
```

Mostrare su sqliteonline

- L'interrogazione interna viene eseguita una volta per ciascuna ennupla della FROM esterna

Per semplificare, usiamo una vista

```
CREATE VIEW PersoneConPadre
        AS      select *
                from persone join paternita
                        on nome = figlio
```

Correlated subquery

- Per ogni padre, il figlio che guadagna di più

```
select *
from personeconPadre p
where reddito = (select max(reddito)
                 from personeconPadre
                 where p.padre = padre )
```

- L'interrogazione interna viene eseguita una volta per ciascuna ennupla della FROM esterna
- Per ogni ennupla p di personeconPadre viene eseguita

```
select max(reddito)
from personeconPadre
where p.padre = padre
```

dove p.padre è una costante e il valore ottenuto (chiamiamolo m) viene utilizzato nell'interrogazione "esterna"

```
select *
from personeconPadre p
where reddito = m
```

Altre nidificazioni nella WHERE

- La motivazione originaria per la nidificazione

- nome e reddito del padre di Franco

```
select Nome, Reddito
from Persone join Paternita on Nome = Padre
where Figlio = 'Franco'
```

```
select Nome, Reddito
from Persone
where Nome =     (   select Padre
                     from Paternita
                     where Figlio = 'Franco')
```

- Nome e reddito dei padri di persone che guadagnano più di 20

```
select distinct 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 in (select Padre
               from Paternita
               where Figlio = any (select Nome
                                   from Persone
                                   where Reddito > 20))
```

notare la distinct

- Nome e reddito dei padri di persone che guadagnano più di 20

```
select distinct 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 in (select Padre
               from Paternita, Persone
               where Figlio = Nome
               and Reddito > 20)
```

Altre forme di nidificazione

- Le vediamo più per curiosità che per altro

Nidificazione nella FROM

- Per ogni padre, il figlio che guadagna di più

```
select p.*
from personeconPadre p join
          (  select padre, max(reddito) as maxreddito
             from personeconPadre
             group by padre
           ) as m on p.padre = m.padre
where reddito = maxreddito
```

Ancora nella FROM

- Tutte le persone, con il reddito massimo dei figli dello stesso padre

```
select p.*, maxreddito
from personeconPadre p ,
            (  select padre, max(reddito) as maxreddito
               from personeconPadre
               group by padre
            ) as m
where p.padre = m.padre
```

Nidificazione nella SELECT

- Calcolo di valori con la nidifcazione
- Per ogni padre, tutti i dati e il reddito massimo dei suoi figli (correlazione)

```
select distinct padre, (select max(reddito)
                        from paternita join persone
                             on figlio = nome
                        where padre = p.padre)
from paternita p join persone on padre =nome
```

Operazioni di aggiornamento

- Vediamo solo aspetti base, per sapere operare, i dettagli non interessano e ci sono ben pochi concetti importanti

```
INSERT INTO Persone VALUES ('Mario',25,52)

INSERT INTO Persone(Nome, Reddito, Eta)
       VALUES('Pino',52,23)

INSERT INTO Persone(Nome, Reddito)
       VALUES('Lino',55)

INSERT INTO Persone ( Nome )
SELECT Padre
FROM Paternita
WHERE Padre NOT IN (SELECT Nome
                    FROM Persone)
```

Eliminazione di ennuple

```
DELETE FROM Tabella
   [ WHERE Condizione ]
```

```
DELETE FROM Persone
WHERE Eta < 35

DELETE FROM Paternita
WHERE Figlio NOT in (       SELECT Nome
                            FROM Persone)

DELETE FROM Paternita
```

Modifica di ennuple

```
UPDATE NomeTabella
SET  Attributo = < Espressione |
                                SELECT … |
                                NULL |
                                DEFAULT >
       [ WHERE Condizione ]
```

```
UPDATE Persone SET Reddito = 45
   WHERE Nome = 'Piero'

UPDATE Persone
   SET Reddito = Reddito * 1.1
   WHERE Eta < 30
```

Altre operazioni DDL

- Aggiungere vincoli, con verifica (provate)

```
alter table persone
    add constraint redditononnegativo
        check (reddito >=0)

alter table persone
    add constraint redditoesagerato
        check (reddito >=100)
```

Esercizi con supporto alla correzione

- prova parziale del 18/11/2019, compito C, esercizi 2, 3, 4, 5
  - script SQL per creare e popolare la base di dati per questi esercizi
  - esercitazione su Moodle con correzione automatica

Note su sqliteonline e sulle soluzioni

```
select cod, nome
from Partiti join Risultati on cod = partito join elezioni on  elezione = id
where tipoelezione = 'Europee'
group by partito
```

```
SELECT ID, Anno, TipoElezione, count(Cod) as NumeroPartiti
FROM Risultati JOIN Partiti ON (Partito = Cod) JOIN Elezioni ON Elezione = ID
GROUP BY ID
```