Atzeni, Ceri, Paraboschi, Torlone

Basi di dati

McGraw-Hill, 1999

Capitolo 4: SQL

15/10/1999

SQL

- Structured Query Language
- è un linguaggio con varie funzionalità:
  - contiene sia il DDL sia il DML;
- esistono varie versione dell'SQL;
- vediamo gli aspetti essenziali non i dettagli
- "storia":
  - prima proposta SEQUEL (IBM Reserach, 1974);
  - prima implementazione in SQL/DS (IBM, 1981);
  - dal 1983 ca., "standard di fatto"
  - standard (1986, poi 1989 e infine 1992) recepito in parte;
  - standard SQL3 di prossima approvazione (1999?)

Definizione dei dati in SQL

- Istruzione CREATE TABLE:
  - definisce uno schema di relazione e ne crea un'istanza vuota
- Sintassi

```
create table NomeTabella
 ( NomeAttributo Dominio [ Default ] [ Vincoli ]
   { , NomeAttributo Dominio [ Default ] [ 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)
                   on delete set null
                   on update cascade,
   unique (Cognome,Nome)
 )
```


Domini

- Domini elementari (predefiniti)
- Domini definiti dall'utente (semplici, ma riutilizzabili)

Domini elementari

- Carattere: singoli caratteri o stringhe, anche di lunghezza variabile

```
character [varying ][( Lunghezza ) ]
      [character set NomeFamigliaCaratteri ]
```

- Bit: singoli booleani o stringhe

```
bit [varying ][( Lunghezza ) ]
```

- Numerici, esatti e approssimati:

```
numeric [( Precisione [ , Scala ) ]]
integer
float [( Precisione )]
double precision
```

Domini elementari, 2

- Data, ora, intervalli:
  ```
  date
  time [( Precisione )][with time zone]
  timestamp [( Precisione )][with time zone]
  interval UnitàDiTempo [to UnitàDiTempo]
  ```


Definizione di domini

- Istruzione CREATE DOMAIN:
  - definisce un dominio (semplice), utilizzabile in definizioni di relazioni
- Sintassi
  ```
  create domain NomeDominio as Tipo
        [ Default ]
        [ Vincoli ]
  ```
- Default (utilizzabili anche nella CREATE TABLE)
  ```
  default < Valore | user | null >
  ```
- Esempio
  ```
  create domain Voto as smallint default null
        check ( value >=18 and value <= 30 )
  ```


Vincoli intrarelazionali

- `not null` (su singoli attributi)
- `unique`: permette di definire chiavi; sintassi:
  - per singoli attributi:
    `unique` dopo il dominio
  - chiavi formate da più attributi:
    `unique ( Attributo { , Attributo } )`
- `primary key`: definizione della chiave primaria (una sola, implica `not null`); sintassi, come per `unique`
- `check`, vedremo più avanti

Vincoli intrarelazionali, esempi

```
Nome      character(20) not null,
Cognome   character(20) not null,
unique (Cognome,Nome)
```

- è diverso da:

```
Nome      character(20) not null unique,
Cognome   character(20) not null unique
```


Vincoli interrelazionali

- `check`, vedremo più avanti
- `references` e `foreign key` permettono di definire vincoli di integrità referenziale; sintassi:
  - per singoli attributi:
    `references` dopo il dominio
  - riferimenti su più attributi:
    `foreign key( Attributo { , Attributo } )`
    `references ...`
- è possibile associare politiche di reazione alla violazione dei vincoli (causate da modifiche sulla tabella esterna, cioè quella cui si fa riferimento)

Vincoli interrelazionali, esempio

```
create table Infrazioni(
  Codice     character(6) not null primary key,
  Data       date not null,
  Agente     integer not null
               references Agenti(Matricola),
  Provincia  character(2),
  Numero     character(6) ,
  foreign key(Provincia, Numero) references
        Automobili(Provincia, Numero)
 )
```

Modifiche degli schemi

- `alter domain`
- `alter table`
- `drop domain`
- `drop table`
- `...`

Definzione degli indici

- è rilevante dal punto di vista delle prestazioni
- ma è a livello più basso (fisico e non logico)
- non ne parliamo qui
- in passato era importante perché in alcuni sistemi era l'unico mezzo per definire chiavi

SELECT, sintassi

```
select AttrExpr [ [as] Alias ] { , AttrExpr [ [as] Alias ] }
from Tabella [ [as] Alias ] { , Tabella [ [as] Alias ] }
[ where Condizione ]
```

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

persone

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

maternita

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

paternita

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

Selezione e proiezione

"Nome e reddito delle persone con meno di trenta anni"

$\pi_{Nome, Reddito}(\sigma_{Eta<30}(persone))$

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


SELECT, abbreviazioni

- data una relazione R su A e B

```
select  *
from R
```

- equivale (intutivamente) a

```
select X.A AS A, X.B AS B
from R X
where true
```

Impiegato

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

Dipartimento

Nome	Indirizzo	Citta
Amministrazione	Via Tito Livio	Milano
Produzione	P.zza Lavater	Torino
Distribuzione	Via Segre	Roma
Direzione	Via Tito Livio	Milano
Ricerca	Via Morone	Milano

Selezione, senza proiezione

```
select *
from Impiegato
where Cognome = 'Rossi'
```

$\pi_{Nome, Reddito}(\sigma_{Eta<30}(persone))$

Nome	Cognome	Dipart	Ufficio	Stipendio
Mario	Rossi	Amministrazione	10	45
Carlo	Rossi	Direzione	14	80

Espressioni nella target list

```
select Stipendio/12 as StipendioMensile
from Impiegato
where Cognome = 'Bianchi'
```

StipendioMensile
3.00

Disgiunzione

```
select Nome, Cognome
from Impiegato
where Dipart = 'Amministrazione' or
      Dipart = 'Produzione'
```

Nome	Cognome
Mario	Rossi
Carlo	Bianchi
Giuseppe	Verdi
Paola	Borroni
Marco	Franco

Condizione complessa

```
select Nome
from Impiegato
where Cognome = 'Rossi' and
      (Dipart = 'Amministrazione' or
       Dipart = 'Produzione')
```

Nome
Mario

Condizione "LIKE"

"Gli impiegati che hanno un cognome che ha una 'o' in seconda posizione e finisce per 'i'."

```
select *
from Impiegato
where Cognome like '_o%i'
```

Nome	Cognome	Dipart	Ufficio	Stipendio
Mario	Rossi	Amministrazione	10	45
Carlo	Rossi	Direzione	14	80
Paola	Borroni	Amministrazione	75	40

Gestione dei valori nulli

"Gli impiegati che hanno o potrebbero avere uno stipendio minore di 50 milioni"

Nome	Cognome	Dipart	Ufficio	Stipendio
Mario	Rossi	Amministrazione	10	45
Carlo	Rossi	Direzione	14	80
Paola	Borroni	Amministrazione	75	NULL

```
select *
from Impiegato
where Stipendio < 50 or Stipendio is null
```

Nome	Cognome	Dipart	Ufficio	Stipendio
Mario	Rossi	Amministrazione	10	45
Paola	Borroni	Amministrazione	75	NULL

SELECT e algebra relazionale

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

- prodotto cartesiano delle relazioni nella clausola `from` selezione con la condizione nella clausola `where`
- proiezione come nella target list

$$\pi_{A1,A2} (\sigma_{A2=A3} (R1 \bowtie R2))$$

SELECT e algebra relazionale, ancora

- possono essere necessarie ridenominazioni
  - nel prodotto cartesiano
  - nella target list

```
select X.A1 AS B1, X.A2 AS B2, Y.A4 AS B3
from   R1 X, R2 Y, R1 Z
where  X.A2 = Y.A3 AND Y.A4 = Z.A1
```

$$\rho_{B1,B2,B3 \leftarrow A1,A2,A4} (\pi_{A1,A2,A4} (\sigma_{A2 = A3 \wedge A4 = C1} ($$

$$R1 \bowtie R2 \bowtie \rho_{C1,C2 \leftarrow A1,A2} (R1))))$$

SELECT e calcolo relazionale

```
select X.A1 AS B1, X.A2 AS B2, Y.A4 AS B3
from   R1 X, R2 Y, R1 Z
where  X.A2 = Y.A3 AND Y.A4 = Z.A1
```

$$\{B_1: X.A_1, B_2: X.A_2, B_3: Y.A_4 \mid$$
$$X(R_1), Y(R_2), Z(R_1) \mid$$
$$X.A_2 = Y.A_3 \wedge Y.A_4 = Z.A_1 \}$$

Selezione, proiezione e join

"I padri di persone che guadagnano più di venti milioni"

$$\pi_{Padre}(\text{paternita} \bowtie_{Figlio = Nome} (\sigma_{Reddito>20} (\text{persone})))$$

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


Proiezione, senza selezione

```
select Nome, Cognome
from Impiegato
```

Nome	Cognome
Mario	Rossi
Carlo	Bianchi
Giuseppe	Verdi
Franco	Neri
Carlo	Rossi
Lorenzo	Lanzi
Paola	Borroni
Marco	Franco

Proiezione: duplicati

```
select Cognome
from Impiegato
```

Cognome
Rossi
Bianchi
Verdi
Neri
Rossi
Lanzi
Borroni
Franco

```
select distinct Cognome
from Impiegato
```

Cognome
Rossi
Bianchi
Verdi
Neri
Lanzi
Borroni
Franco

Join naturale

"Padre e madre di ogni persona"

paternita $\bowtie$ maternita

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

$\pi_{Figlio,Padre,Madre}$(paternita $\bowtie_{Figlio = Nome}$ ($\rho_{Nome \leftarrow Figlio}$(maternita)))

Join di una relazione con se stessa

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

$\pi_{Nome, Reddito, RP}$ ($\sigma_{Reddito>RP}$ ($\rho_{NP,EP,RP \leftarrow Nome,Eta,Reddito}$(persone) $\bowtie_{NP=Padre}$ (paternita $\bowtie_{Figlio =Nome}$ 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
```


Ridenominazione del risultato

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

$\pi_{Figlio, Reddito, RP}$ ($\sigma_{Reddito>RP}$ ($\rho_{NP,EP,RP \leftarrow Nome,Eta,Reddito}$(persone) $\bowtie_{NP=Padre}$ (paternita $\bowtie_{Figlio =Nome}$ persone)))

```
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
```


SELECT, con join esplicito, sintassi

`select` AttrExpr [[as] Alias] {,AttrExpr [[as] Alias] }
`from` Tabella [[`as`] Alias]
{[TipoJoin] `join` Tabella [[`as`] Alias] on CondDiJoin }, ...
[`where` AltraCondizione]

Join esplicito

"Padre e madre di ogni persona"

paternita $\bowtie$ maternita

$\pi_{Figlio,Padre,Madre}$(paternita $\bowtie_{Figlio = Nome}$ $\rho_{Nome \leftarrow Figlio}$(maternita))

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

Ulteriore estensione: join naturale

"Padre e madre di ogni persona"

paternita ⋈ maternita

$\pi_{Figlio,Padre,Madre}(paternita \bowtie_{Figlio = Nome} \rho_{Nome \leftarrow Figlio}(maternita))$

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

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


Join

maternita

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

paternita

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

Madre	Figlio	Padre
Anna	Olga	Luigi
Anna	Filippo	Luigi
Maria	Andrea	Franco
Maria	Aldo	Franco

Outer join sinistro

maternita

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

paternita

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

Madre	Figlio	Padre
Luisa	Maria	NULL
Luisa	Luigi	NULL
Anna	Olga	Luigi
Anna	Filippo	Luigi
Maria	Andrea	Franco
Maria	Aldo	Franco

Outer join completo

maternita

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

paternita

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

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

Outer join

"Padre e, se nota, madre di ogni persona"

paternita $\overset{LEFT}{\bowtie}$ maternita

$\pi_{Figlio,Padre,Madre}(paternita \overset{Figlio = Nome}{\underset{LEFT}{\bowtie}} \rho_{Nome \leftarrow Figlio}(maternita))$

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

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


Join esplicito

```
select I.Nome, Cognome, Citta
from Impiegato I join Dipartimento D
   on Dipart = D.Nome
```

Nome	Cognome	Dipart
Mario	Rossi	Milano
Carlo	Bianchi	Torino
Giuseppe	Verdi	Milano
Franco	Neri	Roma
Carlo	Rossi	Milano
Lorenzo	Lanzi	Milano
Paola	Borroni	Milano
Marco	Franco	Torino

Ordinamento del risultato

```
order by AttrDiOrdinamento[asc | desc ]
 {, AttrDiOrdinamento[ asc | desc ]}
```

```
select Cognome, Nome, Stipendio
from Impiegato
where Dipart like 'Amm%'
order by Stipendio desc, Cognome
```

Cognome	Nome	Stipendio
Rossi	Mario	45
Borroni	Paola	40
Verdi	Giuseppe	40

Operatori aggregati

```
select count(*) AS NumeroImpiegati
from Impiegato
where Dipart = 'Produzione'
```

NumeroImpiegati
2

- l'operatore aggregato (`count`) viene applicato al risultato dell'interrogazione:

```
select *
from Impiegato
where Dipart = 'Produzione'
```


COUNT: sintassi

```
count ( < * | [ distinct | all ] ListaAttributi > )
```

```
select count(Stipendio) as NumeroStipendi
from Impiegato
```

NumeroStipendi
8

```
select count(distinct Stipendio)
                    as StipendiDiversi
from Impiegato
```

StipendiDiversi
6

COUNT e valori nulli

Impiegato

Nome	Cognome	Dipart	Ufficio	Stipendio
Mario	Rossi	Amministrazione	10	45
Carlo	Bianchi	Produzione	20	45
Giuseppe	Verdi	Amministrazione	20	NULL

```
select count(*) as NumeroImpiegati
from Impiegato
```

NumeroImpiegati
3

```
select count(Stipendio) as NumeroStipendi
from Impiegato
```

NumeroStipendi
2

Somma, media, massimo, minimo

```
< sum | max | min | avg > ( [ distinct | all ] AttrEspr )
```

Totale degli stipendi del dipartimento amministrazione

```
select sum(Stipendio) as TotaleStipendi
from Impiegato
where Dipart = 'Amministrazione'
```

TotaleStipendi
125

- escludono opportunamente i valori nulli

Join e operatore aggregato

"Il massimo stipendio tra quelli degli impiegati che lavorano in un dipartimento con sede a Milano"

```
select max(Stipendio)
from Impiegato, Dipartimento D
where Dipart = D.Nome and
Citta = 'Milano'
```

80

- Nota: non abbiamo usato la `as` e l'attributo nel risultato non ha nome

Operatori aggregati e target list

- un'interrogazione scorretta:

```
select Cognome, Nome, max(Stipendio)
from Impiegato, Dipartimento
where Dipart = NomeDip and
Citta = 'Milano'
```

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

```
select max(Stipendio), min(Stipendio)
from Impiegato
```


Operatori aggregati e raggruppamenti

"Per ogni dipartimento, la somma degli stipendi"

```
select Dipart, sum(Stipendio) as SommaStipendi
from Impiegato
group by Dipart
```

Dipart	SommaStipendi
Amministrazione	125
Produzione	82
Distribuzione	45
Direzione	153

Semantica di interrogazioni con operatori aggregati e raggruppamenti

- interrogazione senza `group by` e senza operatori aggregati:

```
select Dipart, Stipendio
from Impiegato
```

Dipart	Stipendio
Amministrazione	45
Produzione	36
Amministrazione	40
Distribuzione	45
Direzione	80
Direzione	73
Amministrazione	40
Produzione	46

Semantica …, 2

- poi si raggruppa e si applica l'operatore aggregato a ciascun gruppo

Dipart	Stipendio
Amministrazione	45
Amministrazione	40
Amministrazione	40
Distribuzione	45
Direzione	80
Direzione	73
Produzione	36
Produzione	46

Dipart	SommaStipendi
Amministrazione	125
Distribuzione	45
Direzione	153
Produzione	82

Raggruppamenti e target list

scorretta

```
select Dipart, count(*), Citta
from Impiegato I join Dipartimento D
        on (I.Dipart = D.Nome)
group by Dipart
```

corretta

```
select Dipart, count(*) as NumeroImp, Citta
from Impiegato join Dipartimento
   on (Impiegato.Dipart = Dipartimento.Nome)
group by Dipart, Citta
```


Condizioni sui gruppi

"I dipartimenti che spendono più di 100 milioni in stipendi"

```
select Dipart,
      sum(Stipendio) as SommaStipendi
from Impiegati
group by Dipart
having sum(Stipendio) > 100
```

Dipart	SommaStipendi
Amministrazione	125
Direzione	153

WHERE o HAVING?

"I dipartimenti per cui la media degli stipendi degli impiegati che lavorano nell'ufficio 20 è superiore a 25 milioni"

```
select Dipart
from Impiegato
where Ufficio = 20
group by Dipart
having avg(Stipendio) > 25
```


Sintassi, riassumiamo

SelectSQL ::=
`select` ListaAttributiOEspressioni
`from` ListaTabelle
[`where` CondizioniSemplici]
[`group by` ListaAttributiDiRaggruppamento]
[`having` CondizioniAggregate]
[`order by` ListaAttributiDiOrdinamento]

Unione, intersezione e differenza

- La `select` da sola non permette di fare unioni; serve un costrutto esplicito:

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

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

- i duplicati vengono eliminati (a meno che si usi `all`); anche dalle proiezioni!

Notazione posizionale!

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

- quali nomi per gli attributi del risultato?
  - nessuno
  - quelli del primo operando
  - …

Notazione posizionale, 2

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

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


Notazione posizionale, 3

- Anche con le ridenominazioni non cambia niente:

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

Intersezione e differenza

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

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

- equivale a

```
select I.Nome
from Impiegato I, Impiegato J
where I.Nome = J.Cognome
```

- vedremo che si può esprimere con `select` nidificate

Interrogazioni nidificate

- le condizioni atomiche permettono anche
  - il confronto fra un attributo (o più, vedremo poi) e il risultato di una sottointerrogazione
  - quantificazioni esistenziali

Interrogazioni nidificate, sintassi

ConfrontoConNidificazione :: =
Scalare OpConfronto [`any` | `all`]
(SelectAttributoSingolo)|
`exists` (SelectStar)

- senza `any` o `all` , il risultato della SelectAttributoSingolo deve essere un solo valore
- `v.A` θ any Select... (risp. `all`) è vero se `v.A` è in relazione θ con almeno uno (risp. con tutti) dei valori del risultato della Select
- `= any` puo essere abbreviato con `in`
- `exists( SelectStar )` è vero se il risultato della sottoespressione non è vuoto.

Interrogazioni nidificate

"Gli impiegati che lavorano in dipartimenti di Roma"

```
select *
from Impiegato
where Dipart = any (select Nome
                    from Dipartimento
                    where Citta = 'Roma')
```


"nome e reddito del padre di Mario"

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

```
select Nome, Reddito
from Persone, Paternita
where Nome = Padre and
      Figlio = 'Mario'
```


Interrogazioni nidificate, commenti

- La prima versione di SQL prevedeva solo la forma nidificata (o strutturata), con una sola relazione in ogni clausola FROM. Il che è insoddisfacente:
  - la dichiaratività è limitata
  - non si possono includere nella target list attributi di relazioni nei blocchi interni
- La forma nidificata è "meno dichiarativa", ma talvolta più leggibile (richiede meno variabili)
- La forma piana e quella nidificata possono essere combinate
- Le sottointerrogazioni non possono contenere operatori insiemistici ("l'unione si fa solo al livello esterno"); la limitazione non è significativa

"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
               where Figlio =any (select Nome
                                  from Persone
                                  where Reddito > 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
```


Interrogazioni piane o nidificate?

```
select I1.Nome
from Impiegato I1, Impiegato I2
where I1.Nome = I2.Nome and
I2.Dipart = 'Produzione'


select Nome
from Impiegato
where Nome = any (select Nome
                  from Impiegato
                  where Dipart = 'Produzione')
```


"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)
```


Interrogazioni nidificate, commenti, 2

- regole di visibilità:
  - non è possibile fare riferimenti a variabili definite in blocchi più interni
  - se un nome di variabile è omesso, si assume riferimento alla variabile più "vicina"
- nota: in un blocco si può fare riferimento a variabili definite in blocchi più esterni; la semantica (prodotto cartesiano, selezione, proiezione) non funziona più, ne serve una più sofisticata:
  - l'interrogazione interna va ripetuta una volta per ciascun valore della variabile

Necessità della forma nidificata

"Trovare i padri i cui figli guadagnano tutti più di venti milioni"

```
select distinct Padre
from Paternita Z
where not exists (select *
                  from Paternita W, Persone
                  where W.Padre = Z.Padre and
                        W.Figlio = Nome and
                        Reddito <= 20)
```

- anche con `<>all`

Semantica delle espressioni "correlate"

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

Visibilità

- 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)
```


Disgiunzione e unione (ma non sempre)

```
select *
from Persone
where Reddito > 30
union
select F.*
from Persone F, Paternita, Persone P
where F.Nome = Figlio and
      Padre = P.Nome and
      P.Reddito > 30

select *
from Persone F
where Reddito > 30 or
      exists (select *
              from Paternita, Persone P
              where F.Nome = Figlio and
                    Padre = P.Nome and
                    P.Reddito > 30)
```


Differenza e nidificazione

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

select Nome
from Impiegato I
where not exists
      (select *
       from Impiegato
       where Cognome = I.Nome)
```


Massimo e nidificazione

"L'impiegato con lo stipendio massimo"

```
select *
from Impiegato
where Stipendio = (select max(Stipendio)
                   from Impiegato)
```


Operazioni di aggiornamento

- operazioni di
  - inserimento
  - eliminazione
  - modifica
- di una o più ennuple di una relazione
- sulla base di una condizione che può coinvolgere anche altre relazioni
- istruzioni
  - `insert`
  - `delete`
  - `update`

Inserimento di ennuple

- sintassi

  `insert into` NomeTabella [(ListaAttributi)]
  < `values`(ListaDiValori) |
  SelectSQL >

```
insert into Dipartimento(NomeDip,Citta)
  values('Produzione','Torino')

insert into ProdottiMilanesi
  (select codice, descrizione
   from Prodotto
   where LuogoProd = 'Milano')
```

Inserimento di ennuple, 2

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

Eliminazione di ennuple

```
delete from NomeTabella [ where Condizione ]

delete from Dipartimento
       where NomeDip = 'Produzione'

delete from Dipartimento
       where Nome not in (select Dipart
                          from Impiegato)
```


Eliminazione di ennuple

- elimina le ennuple che soddisfano la condizione
- può causare (se i vincoli di integrità referenziale sono definiti con politiche di reazione `cascade`) eliminiazioni da altre relazioni
- ricordare: se la `where` viene omessa, si intende `where true`
- cancelliamo tutte le ennuple (ma manteniamo lo schema):
  `delete from Dipartimento`
- per cancellare anche lo schema
  `drop Dipartimento`

Modifica di ennuple

```
update NomeTabella
   set Attributo = < Espressione | SelectSQL | null | default >
   {, Attributo = < Espressione | SelectSQL | null | default > }
   [ where Condizione ]

update Dipendente set Stipendio = 45
where Matricola = 'M2047'

update Impiegato set Stipendio = Stipendio * 1.1
where Dipart = 'Amministrazione'
```


- attenzione all'approccio insiemistico:

```
update Impiegato set Stipendio = Stipendio * 1.1
where Stipendio <= 30

update Impiegato set Stipendio = Stipendio * 1.15
where Stipendio > 30
```


Vincoli di integrità generici: check

- Specifica di vincoli di ennupla (e anche vincoli più complessi)

```
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)
)
```

Vincoli di integrità generici: asserzioni

- Specifica vincoli a livello di schema

```
create assertion NomeAsserzione check ( Condizione )

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


Viste

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

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

create view ImpiegatiAmminPoveri as
    select *
    from ImpiegatiAmmin
    where Stipendio < 50
    with check option
```


Un'interrogazione non standard

- La nidificazione nella `having` non è ammessa

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


Soluzione con le viste

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

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


Ancora sulle viste

- Interrogazione scorretta

```
select avg(count(distinct Ufficio))
from Impiegato
group by Dipart
```

- Con una vista

```
create view DipartUffici(NomeDip,NroUffici) as
  select Dipart, count(distinct Ufficio)
  from Impiegato
  group by Dipart

select avg(NroUffici)
from DipartUffici
```


Access control

- Every component of the schema can be protected (tables, attributes, views, domains, etc.)
- The owner of a resource (the creator) assigns privileges to the other users
- A predefined user `_system` represents the database administrator and has complete access to all the resources
- A privilege is characterized by:
  - the resource
  - the user who grants the privilege
  - the user who receives the privilege
  - the action that is allowed on the resource
  - whether or not the privilege can be passed on to other users

Types of privilege

- SQL offers six types of privilege
  - `insert`: to insert a new object into the resource
  - `update`: to modify the resource content
  - `delete`: to remove an object from the resource
  - `select`: to access the resource content in a query
  - `references`: to build a referential integrity constraint with the resource (may limit the ability to modify the resource)
  - `usage`: to use the resource in a schema definition (e.g., a domain)

`grant` and `revoke`

- To grant a privilege to a user:
  `grant` < *Privileges* | `all privileges` > `on` *Resource* `to` *Users* `[with grant option]`
  - `grant option` specifies whether the privilege of propagating the privilege to other users must be granted
- E.g.:
  `grant select on Department to Stefano`
- To take away privileges:
  `revoke` *Privileges* `on` *Resource* `from` *Users* `[restrict|cascade]`

SQL immerso in linguaggio ospite "embedded SQL"

- in applicazioni complesse, è spesso necessario "immergere" istruzioni SQL in programmi tradizionali (Pascal, COBOL, C)
- i programmi con SQL immerso sono analizzati da precompilatori che traducono le istruzioni SQL
- le variabili del programma possono essere usate come parametri nelle istruzioni SQL (precedute da ':')
- `select` che producano una sola ennupla e operazioni di aggiornamento possono essere immerse senza problemi (con opportuno meccanismo di trasmissione dei risultati)
- esiste di solito una variabile di sistema `sqlcode` che dopo l'esecuzione di un'operazione assume valore zero se essa ha avuto successo e valore diverso altrimenti

Relazioni e record: cursori

- problema fondamentale: "disadattamento di impedenza"
  - i linguaggi tradizionali gestiscono record uno alla volta
  - SQL produce insiemi di ennuple
- soluzione: cursore
  - accede a tutte le ennuple di una interrogazione in modo globale
  - le trasmette al programma una alla volta

Operazioni sui cursori

Definizione del cursore
`declare` NomeCursore `[scroll]cursor for` SelectSQL
`[for` < `read only` | `update [of` Attributo {`,` Attributo} `] >]`

Esecuzione dell'interrogazione
`open` NomeCursore

Utilizzo dei risultati (una ennupla alla volta)
`fetch` NomeCursore `into` ListaVariabili

Disabilitazione del cursore
`close cursor` NomeCursore

Accesso alla ennupla corrente (di un cursore su singola relazione a fini di aggiornamento)
`current of` NomeCursore
nella clausola where

Frammento di programma con SQL immerso

```
write('nome della citta''?');
readln(citta);
EXEC SQL DECLARE P CURSOR FOR
   SELECT NOME, REDDITO
   FROM PERSONE
   WHERE CITTA = :citta ;
EXEC SQL OPEN P ;
EXEC SQL FETCH P INTO :nome, :reddito ;
while SQLCODE = 0
do begin
       write('nome della persona:', nome, 'aumento?');
       readln(aumento);
       EXEC SQL UPDATE PERSONE
                    SET REDDITO = REDDITO + :aumento
                    WHERE CURRENT OF P
       EXEC SQL FETCH P INTO :nome, :reddito
   end;
EXEC SQL CLOSE CURSOR P
```

Altro esempio

```
void VisualizzaStipendiDipart(char NomeDip[])
{
   char Nome[20], Cognome[20];
   long int Stipendio;
   $ declare ImpDip cursor for
        select Nome, Cognome, Stipendio
        from Impiegato
        where Dipart = :NomeDip;
   $ open ImpDip;
   $ fetch ImpDip into :Nome, :Cognome, :Stipendio;
   printf("Dipartimento %s\n",NomeDip);
   while (sqlcode == 0)
   {
        printf("Nome e cognome dell'impiegato: %s
                %s",Nome,Cognome);
        printf("Attuale stipendio: %d\n",Stipendio);
        $ fetch ImpDip into :Nome, :Cognome, :Stipendio;
   }
   $ close cursor ImpDip;
}
```


Dynamic SQL

- When applications do not know at compile-time the SQL statement to execute, they need dynamic SQL
- Major problem: managing the transfer of parameters between the program and the SQL environment

- For direct execution:
  `execute immediate` *SQLStatement*
- For execution preceded by the analysis of the statement:
  `prepare` *CommandName* `from` *SQLStatement*
  - followed by:
    `execute` *CommandName* [`into` *TargetList*] [`using` *ParameterList*]

Procedures

- SQL-2 allows for the definition of procedures, also known as stored procedures
- Stored procedures are part of the schema

```
procedure AssignCity(:Dep char(20),
                     :City char(20))
update Department
set City = :City
where Name = :Dep
```

- SQL-2 does not handle the writing of complex procedures
- Most systems offer SQL extensions that permit to write complex procedures (e.g., Oracle PL/SQL)

Procedure in Oracle PL/SQL

```
Procedure Debit(ClientAccount char(5),Withdrawal integer) is
  OldAmount integer;
  NewAmount integer;
  Threshold integer;
begin
  select Amount, Overdraft into OldAmount, Threshold
    from BankAccount
    where AccountNo = ClientAccount
    for update of Amount;
  NewAmount := OldAmount - WithDrawal;
  if NewAmount > Threshold
    then update BankAccount
           set Amount = NewAmount
           where AccountNo = ClientAccount;
    else
      insert into OverDraftExceeded
        values(ClientAccount,Withdrawal,sysdate);
  end if;
end Debit;
```