Tipi di linguaggi per basi di dati

Si distinguono due categorie:

DDL **Linguaggi di definizione dei dati** o **data definition languages**: utilizzati per definire gli schemi logici, esterni e fisici e le autorizzazioni degli utenti

DML **Linguaggi di manipolazione dei dati** o **data manipulation languages**: utilizzati per l'interrogazione e l'aggiornamento delle istanze di basi di dati

L'algebra relazionale è un DML

Structured **Q**uery **L**anguage

pron: “esse-qu-elle”, “es-que-el” o “sequel”

Letteralmente: “Linguaggio Strutturato di Interrogazione”

E’ sia un DDL che un DML

Ne esistono varie versioni

Fu proposto (come SEQUEL) dall’IBM Research nel 1974

La prima implementazione risale al 1981 (IBM SQL/DS)

Dal 1983 circa è uno standard di fatto

E’ standard ANSI 1986, 1989, 1992 (quest’ultimo denominato SQL-2)

Convenzioni di specifica della sintassi

“`courier`”	esattamente i caratteri specificati dalla stringa Es.: “`where`” indica la sequenza “`w`”, “`h`”, “`e`”, “`r`”, “`e`”
neretto	un valore non specificato, ma il cui tipo (stringa, intero, booleano, ecc) è evidente dal contesto Es.: **NomeRelazione** può rappresentare la stringa “`Impiegati`”

Convenzioni di specifica della sintassi (2)

[...]	tutto ciò che è tra parentesi quadre è opzionale Es.: [“abc”] può indicare tanto “abc” quanto “” (la stringa vuota)
{ ... }	tutto ciò che è tra parentesi graffe può essere ripetuto zero o più volte Es.: [“abc”] può indicare “”, “abc”, “abc abc”, ...
< ... | ... >	tutto ciò che è tra parentesi angolate e separato da barre verticali è alternativo Es.: < “a” | “b” | “c” > può indicare “a”, “b” o “c”

create table

Scopo

Definisce uno schema di relazione e ne crea un'istanza vuota

Sintassi

"create table" **NomeTabella**
"(" **NomeAttributo Dominio**
[Default] [Vincoli]
{ "," **NomeAttributo Dominio**
[Default] [Vincoli] }
["," **AltriVincoli**]
")"

Esempio

```
create table Dipartimento
(
   Nome      char (20) primary key,
   Indirizzo  char (50),
   Citta      char (20)
)
```

Esempio di **`create table`**

```
create table Impiegato (

  Matricola     char (6) primary key,
  Nome          char (20) not null,
  Cognome       char (20) not null,
  Dipart        char (15),
  Stipendio     numeric (9) default 0,
  Citta         char (15),

  foreign key (Dipart)
        refereces Dipartimento (NomeDip)
        on delete set null
        on update cascade,
  unique (Cognome, Nome)
)
```

Altro esempio di **`create table`**

Incidenti

codice	prov1	num1	prov2	num2
00001	RM	034524	PG	982453
00002	BO	823507	MI	827283

```
create table Incidenti (

  codice        numeric (5) primary key,
  prov1         char (2),
  num1          numeric (6),
  prov2         char (2),
  num2          numeric (6)
)
```

Tipi di domini

Esistono due tipi di domini

1) **domini elementari**: sono quelli predefiniti e presenti in tutte le implementazioni di SQL

2) **domini definiti dall'utente**:

- solo domini semplici (derivati cioè da un altro dominio già esistente)
- vengono dichiarati prima del loro utilizzo
- possono essere riutilizzati in più punti

Domino `character`

Scopo

Definisce stringhe e singoli caratteri

Sintassi

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

`char` = `character`
`varchar` = `character varying`

Es.

```
character (20)
```

stringa di 20 caratteri

Es.

```
varchar (1000)
        character set Greek
```

stringa di caratteri dell'alfabeto greco a lunghezza variabile di lunghezza massima 1000

Dominio **`bit`** (SQL-2)

Scopo

Definisce sequenze di valori binari che vengono generalmente utilizzati come *flag*

Sintassi

“`bit`” [“`varying`”]
[“`(`” **Lunghezza** “`)`”]

Es.

bit (5)

sequenza di 5 valori binari

Es.

bit varying (100)

sequenza di valori binari di lunghezza variabile e lunghezza massima 100

Domini numerici esatti

Scopo

Rappresentano valori interi, decimali, o in virgola fissa.

Sintassi

“numeric” [“(” **NumCifre** [“,” **NumDecimali**] “)”]

“decimal” [“(” **NumCifre** [“,” **NumDecimali**] “)”]

“integer”

“smallint”

Es.

numeric (6,3)

rappresenta valori compresi tra -999,999 e +999,999

con `decimal` *vengono definiti requisiti minimi, con* `numeric` *i valori di precisione esatti*

Domini numerici approssimati

Scopo

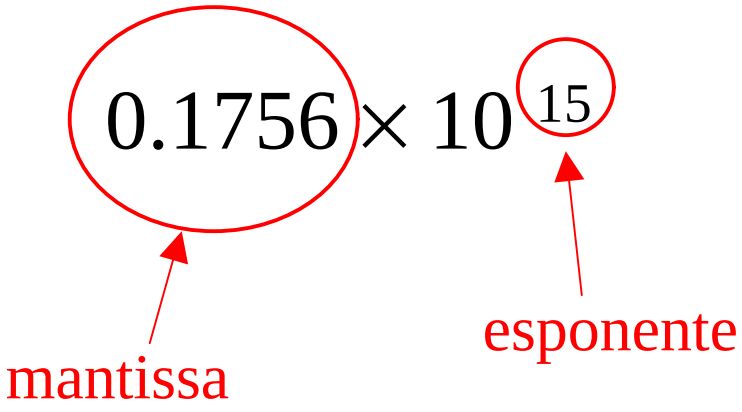
Rappresentano valori approssimati

Sintassi

“`real`”

“`double precision`”

“`float`” [“`(`” **CifreMantissa** “`)`”]



Per `real` *e* `double precision` *il numero di cifre della mantissa e dell’esponente dipendono dall’implementazione*

Data e ora (SQL-2)

Scopo

Definiscono rispettivamente una data e un'ora

Sin.

"`date`"

Es.

date

definisce una data con i campi anno, mese e giorno
(Es.: '`2000-10-18`')

Sin.

"`time`" ["(" **NumDecimali** ")"]
["`with time zone`"]

Es.

time (2) with time zone

definisce una stringa con ore, minuti, secondi, e fuso orario (Es.: '`21:03:04,98+1:00`')

`timestamp` (SQL-2)

Scopo

Definisce un etichettatura temporale (*timestamp*)

Sintassi

```
"timestamp"
        ["(" NumDecimali ")"]
        ["with time zone"]
```

Es.

timestamp (1)

definisce un'etichettatura temporale
(Es.: `'2000-10-18 15:30:45,3'`)

Es.

timestamp with time zone

definisce un'etichettatura temporale.
Un valore può essere il seguente:
`'2000-10-18 15:30:45+5:30'`

`interval` (SQL-2)

Scopo

Definisce un intervallo temporale

Sintassi

“`interval`” **UnitàDiTempo**
[“`to`” **UnitàDiTempo**]

Es.

interval year to month

definisce un intervallo in anni e mesi
(Es.: `‘2-3’` = due anni e tre mesi)

Es.

interval day to second

definisce un intervallo in giorni, ore, minuti e secondi
(Es.: `‘3 20:43:21’` = tre giorni, 20 ore, 43 minuti e 21 secondi)

Non è possibile abbracciare nello stesso intervallo i mesi e i giorni

create domain

Scopo

Definisce un domino (semplice) utilizzabile nelle definizioni delle ralazioni che seguiranno

Sintassi

"create domain" **NomeDominio**
"as" **TipoDiDato**
[**ValoreDiDefault**]
[**Vincoli**]

Esempio

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

L'istruzione create domain aggiunge un alias, un valore di default ed un insieme di vincoli ad un dominio esistente

Valori di *default*

Scopo

Definisce un valore di default per un dominio

Sintassi

“default”
< **Valore** | “user” | “null” >

Esempio

```
create table Persona (
  Figli    smallint default 0,
...
```

Esempio

```
create table VoceElenco (
   InseritaDa    char (20)
                 default user,
   ...
```

Esempio

```
create domain Voto as smallint
                 default null
...
```

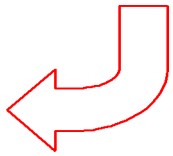
Vincoli

Sintassi

"`create table`" **NomeTabella**
"`(`" **NomeAttributo Dominio**
[**Default**] [**Vincoli**]
{ "`,`" **NomeAttributo Dominio**
[**Default**] [**Vincoli**] }
["`,`" **AltriVincoli**]
"`)`"

i **Vincoli** *compaiono in più punti nella sintassi di* `create table`

```
create table Dipartimento (
   Nome      char (20) primary key,
   Indirizzo  char (50),
   primary key (Nome)
)
```



è equivalente specificare un vincolo dopo l'attributo cui si riferisce o in coda alla definizione

Attenzione!

Esempio uno

```
create table Persona (
   Matricola  integer primary key,
   Nome       char (20),
   Cognome char (20),
   Indirizzo   char (50),
   unique (Cognome,Nome)
)
```

è diverso da...

Esempio due

```
create table Persona (
   Matricola  integer primary key,
   Nome       char (20),
   Cognome char (20),
   Indirizzo   char (50),
   unique (Cognome),
   unique (Nome)
)
```

Quando sono interessati più attributi è d'obbligo specificarli assieme (e dunque necessariamente in coda alla definizione)

Vincoli intrarelazionali

`primary key` elegge l'attributo specificato (o gli attributi specificati) come chiave primaria per la relazione

`not null` impone che l'attributo abbia sempre un valore definito

`unique` impone che non esistano due tuple della relazione con lo stesso valore sull'attributo (o sugli attributi). Gli attributi che sono unique sono anche superchiavi per la relazione

`primary key` *implica il* `not null` *e l'*`unique` *sugli attributi interessati*

Vincoli interrelazionali

Vincoli di integrità referenziali:

- `references` dopo l'attributo

Esempio

```
create table Impiegato (
CF      char(16)
        references Persona(CF),
...
```

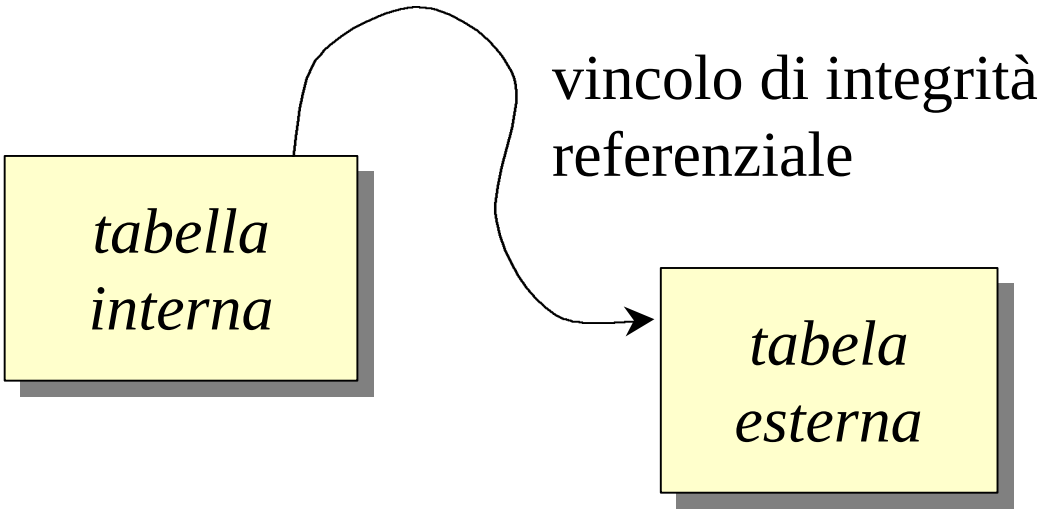
- `foreign key` ... `references`
  per insiemi di attributi (in coda alla create table)

Esempio

```
...
foreign key (Nome,Cognome)
references
Anagrafe(Nome,Cognome)
)
```

SQL richiede che gli attributi della tabella riferita siano dichiarati `unique`

Violazione dei vincoli interrelazionali



Il vincolo può essere violato:

- inserendo o modificando una tupla della tabella interna
- cancellando o modificando una tupla della tabella esterna

Nel primo caso si assume che sia l'applicazione o l'utente a verificare la non violazione del vincolo. Nel secondo caso SQL offre costrutti opportuni

Strategie di reazione

`cascade`: la cancellazione o la modifica nella tupla della tabella esterna si riflette in una analoga cancellazione o modifica nelle tuple corrispondenti della tabella interna

`set null`: le corrispondenti tuple della tabella interna assumono il valore **`null`** sugli attributi interessati

`set default`: le corrispondenti tuple della tabella interna assumono il valore di default sugli attributi interessati

`no action`: non viene presa nessuna particolare precauzione

Sintassi

La strategia di reazione alla violazione viene specificata immediatamente dopo il vincolo con la sintassi seguente

Sintassi

```
"on"  <"delete"|"update">
      <"cascade"
       |"set null"
       |"set default"
       |"no action">
```

Esempio

```
create table Impiegato (
  Nome char(20) not null,
  Cognome char(20) not null,
  Dipart char(15),
  Stipendio numeric(9) default 0,
  primary key (Cognome, Nome),
  foreign key (Dipart) references
        Dipartimento(Nome)
        on delete set null
        on update cascade
)
```

create schema

Scopo

Definisce uno schema in SQL

Sintassi

"create schema"
[**NomeSchema**]
[["autorization"]
UtenteProprietario]
{ **Definizioni** }

Esempio

```
create schema NuovoSchema
  create table Impiegato (
        ...
  )
  create table Dipartimento (
        ...
  )
  ...
```

Esercizio 1

Si definisca in SQL la seguente base di dati

Partenze(Numero, Ora,
Destinazione,Categoria)
Scali(Treno, Stazione, Ora)

- Tra l'attributo `Treno` di **Scali** e la relazione **Partenze** c'è un vincolo di integrità referenziale
- Non si ammette nessun valore nullo
- Non ci sono due treni con lo stesso orario, stessa destinazione e stessa categoria

Esercizio 2

Si definisca in SQL la seguente base di dati

Pazienti(Codice, Cognome, Nome)
Ricoveri(Paziente, Inizio, Fine, Reparto)
Medici(Matr., Cognome, Nome, Reparto)
Reparti(Sigla, Nome, Primario)

Vincoli di integrità referenziale sono:

- tra l'attributo `Paziente` di **Ricoveri** e la relazione **Pazienti**
- tra `Reparto` di **Ricoveri** e **Reparti**
- tra `Primario` di **Reparti** e **Medici**
- tra `Reparto` di **Medici** e **Reparti**

Valori nulli sono ammessi per gli attributi:

- `Cognome` e `Nome` di **Pazienti**
- `Fine` di **Ricoveri**
- `Cognome` e `Nome` di **Medici**
- `Nome` di **Reparti**

Interrogazioni: `select`

Sintassi

> “`select`” **ListaAttributi**
> “`from`” **ListaTabelle**
> [“`where`” **Condizione**]

le tre parti vengono chiamate:

- *target list*
- *clausola* `from`
- *clausola* `where`

Sintassi dettagliata

> “`select`” **AttrExpr** [“`as`” **Alias**]
> { **AttrExpr** [“`as`” **Alias**] }
> “`from`” **Tabella** [“`as`” **Alias**]
> { **Tabella** [“`as`” **Alias**] }
> [“`where`” **Condizione**]

Se la calusola `where` *è assente vengono selezionate tutte le righe delle tabelle*

Relazione Impiegato

Impiegato

Nome	Cognome	Dipartimento	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	Franchi	Produzione	20	46

Esempio #1

Domanda

Individuare il salario di tutti gli impiegati di cognome ‘Rossi’

Algebra

$$\rho_{Salario \leftarrow Stipendio} (\pi_{Stipendio} (\sigma_{Cognome='Rossi'} (Impiegato)))$$

SQL

```
select Stipendio as Salario
from Impiegato
where Cognome = ‘Rossi’
```

Risultato

Salario
45
80

Esempio #2

Domanda

Recuperare tutte le informazioni degli impiegati di nome ‘Rossi’

Algebra

$\sigma_{\text{Cognome='Rossi'}}$ (Impiegato)

SQL

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

*Il carattere speciale * (asterisco) rappresenta tutti gli attributi delle tabelle elencate nella clausola from*

Risultato

Nome	Cognome	Dip.	Ufficio	Stip.
Mario	Rossi	Amm.	10	45
Carlo	Rossi	Dir.	14	80

Esempio #3

Domanda

Trovare lo stipendio mensile di ‘Bianchi’

SQL

```
select Stipendio/12 as Mensile
from Impiegato
where Cognome = ‘Bianchi’
```

Nella target list *possono comparire generiche espressioni sui valori degli attributi*

Risultato

Mensile
3,00

Relazione Dipartimento

Dipartimento

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

Esempio #4

Domanda

Trovare i nomi e cognomi degli impiegati e le città in cui lavorano

Algebra

$\pi_{Nome,Cognome,Citta}$ (Impiegato $\bowtie_{Dipartimento = N}$ ($\rho_{N \leftarrow Nome}$ (Dipartimento)))

SQL

```
select  Impiegato.Nome,
        Impiegato.Cognome,
        Dipartimento.Citta
from Impiegato, Dipartimento
where Impiegato.Dipartimento =
        Dipartimento.Nome
```

Risultato esempio #4

Impiegato.Nome	Impiegato.Cognome	Dipartimento.Citta
Mario	Rossi	Milano
Carlo	Bianchi	Milano
Giuseppe	Verdi	Milano
Franco	Neri	Roma
Carlo	Rossi	Milano
Lorenzo	Lanzi	Milano
Paola	Borroni	Milano
Marco	Franchi	Milano

Esempio #5

Domanda

Trovare i nomi e cognomi degli impiegati e le città in cui lavorano

SQL

```
select  I.Nome,
        Cognome,
        Citta
from    Impiegato as I,
        Dipartimento as D
where I.Dipartimento = D.Nome
```

Risultato

I.Nome	I.Cognome	D.Citta
Mario	Rossi	Milano
Carlo	Bianchi	Milano
Giuseppe	Verdi	Milano
Franco	Neri	Roma
Carlo	Rossi	Milano
Lorenzo	Lanzi	Milano
Paola	Borroni	Milano
Marco	Franchi	Milano

Esempio #6

Domanda

Trovare i nomi e cognomi degli impiegati che lavorano nell'ufficio 20 dell'Amministrazione

Algebra

$\pi_{Nome,Cognome}$ (
$\sigma_{Ufficio=20 \wedge Dipartimento = \text{'Amministrazione'}}$
(Impiegato))

SQL

```
select  Nome, Cognome
from    Impiegato
where   Ufficio = 20 and
        Dipartimento =
                'Amministrazione'
```

La clausola where può contenere espressioni booleane con confronti

Risultato

Nome	Cognome
Giuseppe	Verdi

Esempio #7

Domanda

Trovare i nomi e cognomi degli impiegati dell'Amminstrazione e della Distribuzione

Algebra

$\pi_{Nome,Cognome}$ (
$\sigma_{Dipartimento = \text{'Amministrazione'} \vee Dipartimento = \text{'Distribuzione'}}$ (Impiegato))

SQL

```
select Nome, Cognome
from   Impiegato
where Dipartimento =
      'Amministrazione' or
      Dipartimento='Distribuzione'
```

Risultato

Nome	Cognome
Mario	Rossi
Giuseppe	Verdi
Franco	Neri
Paola	Borroni

Esempio #8

Domanda

Trovare i nomi degli impiegati di cognome ‘Rossi’ che lavorano nell’Amminstrazione e nella Distribuzione

Algebra

$\pi_{\text{Nome}}($
$\sigma_{(\text{Dipartimento} = \text{‘Amministrazione’} \vee \text{Dipartimento} = \text{‘Distribuzione’}) \wedge \text{Cognome}=\text{‘Rossi’}}$
(Impiegato))

SQL

```
select  Nome
from    Impiegato
where Cognome = ‘Rossi’ and
(Dipartimento =
     ‘Amministrazione’ or
     Dipartimento=‘Distribuzione’)
```

Risultato

Nome
Mario

Operatore `like`

Nella clausola `where`, oltre ai normali operatori di confronto (`=`, `<>`, `<`, `>`, `<=`, `>=`) si può usare anche l'operatore `like`, con i due caratteri speciali:

`_` rappresenta un carattere qualsiasi

`%` rappresenta una stringa di caratteri arbitrari (anche la stringa vuota)

Es.

```
Nome like 'ab%ba_'
```

è verificato sia dalla stringa 'abcdedcbac' che dalla stringa 'abbaf'

Esempio #9

Domanda

Trovare gli impiegati che hanno il cognome con una ‘o’ in seconda posizione e terminano per ‘i’

SQL

```
select *
from   Impiegato
where Cognome like ‘_o%i’
```

Domanda

Trovare gli impiegati il cui nome contiene due ‘a’ oppure due ‘e’

SQL

```
select *
from   Impiegato
where (Nome like ‘%a%a%’) or
      (Nome like ‘%e%e%’)
```

Risultati esempio #9

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

Nome	Cognome	Dipartimento	Ufficio	Stipendio
Giuseppe	Verdi	Amministrazione	20	40
Paola	Borroni	Amministrazione	75	40

Interpretazione algebrica

SQL

```
select T1.Attributo1, ...
       Th.Attributoh
from T1, ..., Th
where Condizione
```

Una interrogazione SQL può essere tradotta in una espressione dell'algrebra relazionale come segue

Algebra

$$\pi_{T_1.Attributo_1, \ldots, T_h.Attributo_h} (\sigma_{Condizione} (T_1 \bowtie T_2 \bowtie \ldots T_h))$$

dove $\bowtie$ *simboleggia un prodotto cartesiano (eventuali rinominazioni che si rendessero necessarie sono state omesse)*

Gestione dei duplicati

Devono essere eliminati esplicitamente

SQL 1

```
select Cognome
from Impiegato
```

SQL 2

```
select distinct Cognome
from Impiegato
```

Risultato 1

Cognome
Rossi
Bianchi
Verdi
Neri
Rossi
Lanzi
Borroni
Franchi

Risultato 2

Cognome
Rossi
Bianchi
Verdi
Neri
Lanzi
Borroni
Franchi

Esempio #10

Con riferimento allo schema:

Persone(<u>Nome</u>, Età, Reddito)
Paternità(Padre, <u>Figlio</u>)
Maternità(Madre, <u>Figlio</u>)

Domanda

Trovare i padri di persone che guadagnano più di 20 milioni

Algebra

$\pi_{Padre}(Paternità \bowtie_{Figlio=Nome} (\sigma_{Reddito>20}(Persone)))$

SQL

```
select distinct Padre
from Persone Paternità
where Figlio = Nome and
        Reddito > 20
```

Esempio #11

Domanda

Trovare padre e madre di ogni persona

Algebra

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

SQL

```
select  Paternita.Figlio,
        Padre, Madre
from Maternita Paternità
where Paternita.Figlio =
        Maternita.Figlio
```

Esempio #12

Domanda

Trovare le persone che guadagnano più dei rispettivi padri, mostrandone nomi, redditi, e nomi dei padri

Algebra

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

SQL

```
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
```