Università di Roma "La Sapienza"
Laurea in Ingegneria Informatica

Basi di Dati

Anno Accademico 2003/2004
Canale M-Z

Diego Calvanese

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
Università di Roma "La Sapienza"

http://www.dis.uniroma1.it/~calvanese/didattica/03-04-basididati/

3. Il Linguaggio SQL

3.1 Definizione dei dati

1. **definizione dei dati**
2. interrogazioni
3. manipolazione dei dati
4. ulteriori aspetti

SQL

- originariamente "**S**tructured **Q**uery **L**anguage", ora "nome proprio“
- è un linguaggio con varie funzionalità:
  - contiene sia il DDL sia il DML
- ne esistono varie versioni
- analizziamo gli aspetti essenziali non i dettagli
- “storia”:
  - prima proposta **SEQUEL** (IBM Research, 1974);
  - prime implementazioni in SQL/DS (IBM) e Oracle (1981);
  - dal 1983 ca., “standard di fatto”
  - standard (1986, poi 1989, poi **1992**, e infine 1999):
    recepito solo in parte

SQL-92

- è un linguaggio ricco e complesso
- ancora nessun sistema mette a disposizione tutte le funzionalità del linguaggio
- 3 livelli di aderenza allo standard:
  - **Entry SQL**: abbastanza simile a SQL-89
  - **Intermediate SQL**: caratteristiche più importanti per le esigenze del mercato; supportato dai DBMS commerciali
  - **Full SQL**: funzioni avanzate, in via di inclusione nei sistemi
- i sistemi offrono funzionalità non standard
  - incompatibilità tra sistemi
  - incompatibilità con i nuovi standard (es. trigger in SQL:1999)

Definizione dei dati in SQL

- Oltre alla istruzione **`create schema`** (che serve a dichiarare uno schema), l'istruzione più importante del DDL di SQL è

  `create table`

  – definisce uno schema di relazione (specificando attributi e vincoli)
  – crea un'istanza vuota dello schema

- 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)
)
```

SQL e modello relazionale

- **Attenzione**: una tabella in SQL è definita come un multiinsieme di ennuple.

- In particolare, se una tabella non ha una primary key o un insieme di attributi definiti come unique, allora potranno comparire due ennuple uguali nella tabella.

  Ne segue che una tabella SQL **non** è in generale una relazione.

- Se invece una tabella ha una primary key o un insieme di attributi definiti come unique, allora non potranno mai comparire nella tabella due ennuple uguali. Per questo, è consigliabile definire almeno una primary key per ogni tabella.

Domini

- **Domini elementari** (predefiniti)
  - **Carattere**: singoli caratteri o stringhe, anche di lunghezza variabile
  - **Bit**: singoli booleani o stringhe
  - **Numerici**: esatti e approssimati
  - **Data**, **ora**, **intervalli di tempo**
  - Introdotti in SQL:1999
    - **Boolean**
    - **BLOB**, **CLOB** (binary/character large object): per grandi immagini e testi
- **Domini definiti dall'utente** (semplici, ma riutilizzabili)

Definizione di domini

- L‘istruzione

  `create domain`

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

- Sintassi

  `create domain` *NomeDominio*
  `as` *Tipo* [*Default*] [*Vincoli*]

- *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 attributi che identificano la tupla:
  - singolo attributo:
    `unique` dopo la specifica del dominio
  - più attributi:
    `unique (`*Attributo*`, ..., `***Attributo*`)`**

- **`primary key`**: definizione della chiave primaria (una sola, implica **`not null`**); sintassi come per **`unique`**

- **`check`**, per vincoli complessi (più avanti)

Vincoli intrarelazionali, esempi

```
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)
)
```

primary key, alternative

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

oppure

```
create table Impiegato (
   Matricola character(6),
   ...
   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 **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 (**`primay 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).

Vincoli interrelazionali, esempio

Infrazioni

Codice	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

Matricola	Cognome	Nome
3987	Rossi	Luca
3295	Neri	Piero
9345	Neri	Mario
7543	Mori	Gino

Vincoli interrelazionali, esempio (cont.)

Infrazioni

Codice	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

Prov	Numero	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 e definizione di indici

Modifiche degli schemi:

- **`alter domain`**: permette di modificare un dominio
- **`alter table`**: permette di modificare una tabella
- **`drop domain`**: elimina un dominio
- **`drop table`**: elimina una tabella
- ...

Definizione di indici:

- è rilevante dal punto di vista delle prestazioni
- ma è a livello fisico e non logico
- in passato era importante perché in alcuni sistemi era l'unico mezzo per definire chiavi
- istruzione **`create index`**

Catalogo o dizionario dei dati

Ogni sistema relazionale mette a disposizione delle tabelle già definite che raccolgono tutti i dati relativi a:

- **tabelle**
- **attributi**
- **domini**
- ...

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

- **`Column_Name`**
- **`Table_name`**
- **`Ordinal_Position`**
- **`Column_Default`**
- …

3. Il Linguaggio SQL

3.2 Interrogazioni

1. definizione dei dati
2. **interrogazioni**
3. manipolazione dei dati
4. ulteriori aspetti

Istruzione `select` (versione base)

- L'istruzione di interrogazione in SQL è

  `select`

  che definisce una interrogazione, e restituisce il risultato in forma di tabella

  `select` *Attributo* … *Attributo*
  `from` *Tabella* … *Tabella*
  [**`where`** *Condizione*]

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

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

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

Selezione e proiezione

Nome e reddito delle persone con meno di 30 anni.

$$PROJ_{nome,\ reddito}(SEL_{Eta<30}(persone))$$

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

nome	reddito
Andrea	21
Aldo	15
Filippo	30

Convenzioni sui nomi

- Per evitare ambiguità, ogni nome di attributo è composto da

  NomeRelazione`.`*NomeAttributo*

- Quando l'ambiguità non sussiste, si può omettere la parte

  NomeRelazione`.`

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

si può scrivere come:

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

SELECT, abbreviazioni

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

è un'abbreviazione per:

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

e anche per:

```
select p.nome as nome,  p.reddito as reddito
from   persone p
where  p.eta < 30
```

Proiezione, attenzione

Cognome e filiale di tutti gli impiegati.

impiegati

matricola	cognome	filiale	stipendio
7309	Neri	Napoli	55
5998	Neri	Milano	64
9553	Rossi	Roma	44
5698	Rossi	Roma	64

$$\text{PROJ}_{\text{Cognome, Filiale}}\ (\text{impiegati})$$

Proiezione, attenzione

```
select cognome,
       filiale
from impiegati
```

cognome	filiale
Neri	Napoli
Neri	Milano
Rossi	Roma
Rossi	Roma

```
select distinct cognome,
       filiale
from impiegati
```

cognome	filiale
Neri	Napoli
Neri	Milano
Rossi	Roma

SELECT, uso di “as”

“**as**” nella lista degli attributi serve a specificare esplicitamente un nome per gli attributi del risultato. Quando per un attributo manca “**as**”, il nome è uguale a quello che compare nella lista.

Esempio:

```
select nome as nomePersone, reddito as salario
from   persone
where  eta < 30
```

restituisce come risultato una relazione con due attributi, il primo di nome **nomePersone** ed il secondo di nome **salario**

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

restituisce come risultato una relazione con due attributi, il primo di nome **nome** ed il secondo di nome **reddito**

Esercizio 1

Calcolare la tabella ottenuta dalla tabella **`persone`** selezionando solo le persone con reddito tra 20 e 30 aggiungendo un attributo che ha, in ogni ennupla, lo stesso valore dell'attributo **`reddito`**.

Mostrare il risultato dell'interrogazione.

persone

nome	eta	reddito

Soluzione esercizio 1

```
select nome, eta, reddito,
       reddito as ancoraReddito
from   persone
where  reddito >= 20 and reddito <= 30
```

nome	eta	reddito	ancoraReddito
Andrea	27	21	21
Filippo	26	30	30
Franco	60	20	20

Selezione, senza proiezione

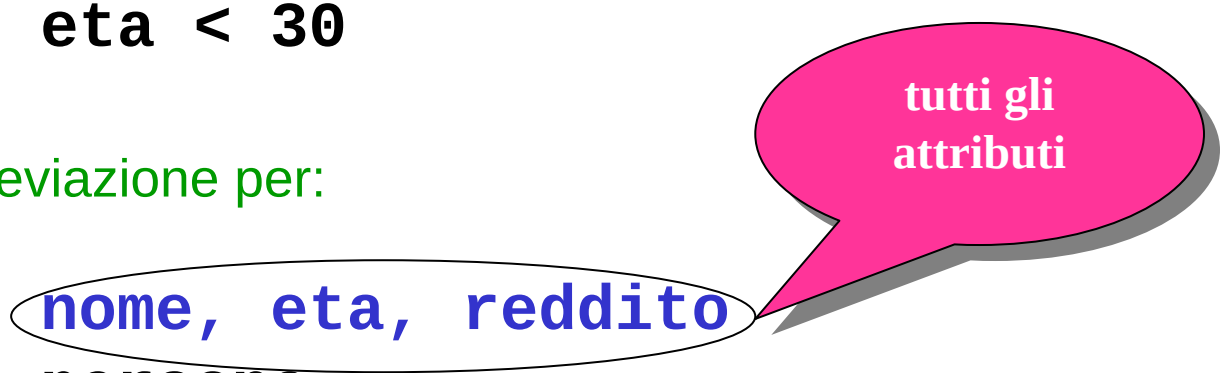
Nome, età e reddito delle persone con meno di trenta anni.

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

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

è un'abbreviazione per:

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



SELECT con asterisco

Data una relazione **R** sugli attributi **A**, **B**, **C**

```
select  *
from    R
where   cond
```

equivale a

```
select A, B, C
from   R
where  cond
```

Proiezione, senza selezione

Nome e reddito di tutte le persone.

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

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

è un'abbreviazione per:

```
select p.nome, p.reddito
from   persone p
where  true
```

Espressioni nella target list

```
select reddito/2 as redditoSemestrale
from   persone
where  nome = 'Luigi'
```

Condizione complessa nella clausola “where”

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

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

Gli impiegati la cui età è o potrebbe essere maggiore di 40.

$$\text{SEL}_{\text{Eta} > 40 \text{ OR Eta IS NULL}} (\text{Impiegati})$$

```
select *
from   impiegati
where  eta > 40 or eta is null
```

Esercizio 2

Calcolare la tabella ottenuta dalla tabella **`impiegati`** selezionando solo quelli delle filiali di Roma e Milano, proiettando i dati sull'attributo **`stipendio`**, ed aggiungendo un attributo che ha, in ogni ennupla, il valore doppio dell'attributo **`stipendio`**.

Mostrare il risultato dell'interrogazione.

impiegati

matricola	cognome	filiale	stipendio

Soluzione esercizio 1

```
select stipendio,
       stipendio*2 as stipendiobis
from   impiegati
where  filiale = 'Milano' or
       filiale = 'Roma'
```

stipendio	stipendiobis
64	128
44	88
64	128

Selezione, proiezione e join

- Istruzioni **`select`** con una sola relazione nella clausola **`from`** permettono di realizzare:
  - selezioni,
  - proiezioni,
  - ridenominazioni

- I **join** (e i prodotti cartesiani) si realizzano indicando due o più relazioni nella clausola **`from`**.

SQL e algebra relazionale

Date le relazioni: R1(A1,A2) e R2(A3,A4)

la semantica della query

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

si può descrivere in termini di
- prodotto cartesiano (**`from`**)
- selezione (**`where`**)
- proiezione (**`select`**)

Attenzione: questo non significa che il sistema calcola davvero il prodotto cartesiano!

SQL e algebra relazionale, 2

Date le relazioni: R1(A1,A2) e R2(A3,A4)

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

corrisponde a:

$$\text{PROJ}_{A1,A4}\ (\text{SEL}_{A2=A3}\ (R1\ \text{JOIN}\ R2))$$

SQL e algebra relazionale, 3

Possono essere necessarie ridenominazioni

- nella target list (come nell'algebra relazionale)
- nel prodotto cartesiano (in particolare quando occorre riferirsi due volte alla stessa tabella)

```
select X.A1 as B1, …
from   R1 X, R2 Y, R1 Z
where  X.A2 = Y.A3 and …
```

che si scrive anche

```
select X.A1 as B1, …
from   R1 as X, R2 as Y, R1 as Z
where  X.A2 = Y.A3 and …
```

SQL e algebra relazionale: esempio

```
select X.A1 as B1, Y.A4 as B2
from   R1 X, R2 Y, R1 Z
where  X.A2 = Y.A3 and Y.A4 = Z.A1
```

$$\text{REN}_{B1,B2 \leftarrow A1,A4} (\text{PROJ}_{A1,A4} (\text{SEL}_{A2 = A3 \textbf{ and } A4 = C1} (R1 \text{ JOIN } R2 \text{ JOIN } \text{REN}_{C1,C2 \leftarrow A1,A2} (R1))))$$

SQL: esecuzione delle interrogazioni

- Le espressioni SQL sono dichiarative e noi ne stiamo vedendo la semantica.
- In pratica, i DBMS eseguono le operazioni in modo efficiente, ad esempio:
  - eseguono le selezioni al più presto
  - se possibile, eseguono join e **non** prodotti cartesiani
- La capacità dei DBMS di "**ottimizzare**" le interrogazioni, rende (di solito) non necessario preoccuparsi dell'efficienza quando si specifica un'interrogazione
- È perciò più importante preoccuparsi della chiarezza (anche perché così è più difficile sbagliare …)

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

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

Esercizio 3: selezione, proiezione e join

I padri di persone che guadagnano più di venti milioni.

Esprimere la query sia in algebra relazionale sia in SQL.

Esercizio 3: soluzione

I padri di persone che guadagnano più di venti milioni.

$PROJ_{padre}(paternita\ JOIN_{figlio=nome}\ SEL_{reddito>20}(persone))$

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

Esercizio 4: join

Padre e madre di ogni persona.

Esprimere la query sia in algebra relazionale sia in SQL.

Esercizio 4: soluzione

Padre e madre di ogni persona.

In algebra relazionale si calcola mediante il **join naturale**.

paternita JOIN maternita

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

Esercizio 5: join e altre operazioni

Le persone che guadagnano più dei rispettivi padri, mostrando nome, reddito e reddito del padre.

Esprimere la query sia in algebra relazionale sia in SQL.

Esercizio 5: soluzione

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 t, persone f
where  p.nome = t.padre and
       t.figlio = f.nome and
       f.reddito > p.reddito
```

SELECT, con ridenominazione del risultato

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

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

Join esplicito

Padre e madre di ogni persona.

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

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

SELECT con join esplicito, sintassi

select …
from *Tabella* { **join** *Tabella* **on** *CondDiJoin* }, …
[**where** *AltraCondizione*]

Esercizio 6: join esplicito

Le persone che guadagnano più dei rispettivi padri, mostrando nome, reddito e reddito del padre.

Esprimere la query in SQL usando il join esplicito.

Esercizio 6: soluzione

Le persone che guadagnano più dei rispettivi padri, mostrando nome, reddito e reddito del padre.

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

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

Ulteriore estensione: join naturale (meno diffuso)

$$\text{PROJ}_{\text{Figlio,Padre,Madre}}(\text{paternita JOIN}_{\text{Figlio = Nome}} \text{REN}_{\text{Nome=Figlio}}(\text{maternita}))$$

In algebra: paternita JOIN maternita

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

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

Join esterno: "outer join"

Padre e, se nota, madre di ogni persona.

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

NOTA: "**outer**" è opzionale

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

Outer join, esempi

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

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

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

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

Ordinamento del risultato: `order by`

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

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


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

Ordinamento del risultato: `order by`

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

nome	reddito
Andrea	21
Aldo	15
Filippo	30

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

nome	reddito
Aldo	15
Andrea	21
Filippo	30

Operatori aggregati

Nelle espressioni della target list possiamo avere anche espressioni che calcolano valori a partire da insiemi di ennuple:

- conteggio, minimo, massimo, media, totale

Sintassi base (semplificata):

Funzione **(** [**`distinct`**] *EspressioneSuAttributi* **)**

Operatori aggregati: `count`

Sintassi:

- conta il numero di ennuple:

  `count (*)`

- conta i valori di un attributo (considerando i duplicati):

  `count (`*Attributo*`)`**

- conta i valori distinti di un attributo:

  `count (distinct` *Attributo*`)`**

Operatore aggregato `count`: esempio

Esempio: Quanti figli ha Franco?

```
select count(*) as NumFigliDiFranco
from   paternita
where  padre = 'Franco'
```

Semantica: l'operatore aggregato (**`count`**), che conta le ennuple, viene applicato al risultato dell'interrogazione:

```
select *
from   paternita
where  padre = 'Franco'
```

Risultato di `count`: esempio

paternita

padre	figlio
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo

NumFigliDiFranco
2

count e valori nulli

```
select count(*)
from   persone
```

Risultato = numero di ennuple
= 4

```
select count(reddito)
from   persone
```

Risultato = numero di valori diversi da NULL
= 3

```
select count(distinct reddito)
from   persone
```

Risultato = numero di valori distinti (escluso NULL)
= 2

persone

nome	eta	reddito
Andrea	27	21
Aldo	25	NULL
Maria	55	21
Anna	50	35

Altri operatori aggregati

`sum`, **`avg`**, **`max`**, **`min`**

- ammettono come argomento un attributo o un'espressione (ma non "**`*`**")
- **`sum`** e **`avg`**: argomenti numerici o tempo
- **`max`** e **`min`**: argomenti su cui è definito un ordinamento

Esempio: media dei redditi dei figli di Franco.

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

Operatori aggregati e valori nulli

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

persone

nome	eta	reddito
Andrea	27	30
Aldo	25	NULL
Maria	55	36
Anna	50	36

redditoMedio
34

Operatori aggregati e target list

Un’interrogazione scorretta (di chi sarebbe il nome?):

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

La **target list** deve essere **omogenea**, ad esempio:

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

Operatori aggregati e raggruppamenti

- Le funzioni di aggregazione possono essere applicate a **partizioni delle ennuple** delle relazioni.

- Per specificare le partizioni delle ennuple, si utilizza la clausola **`group by`**:

`group by` *listaAttributi*

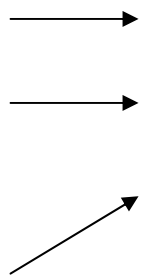
Operatori aggregati e raggruppamenti

Il numero di figli di ciascun padre.

```
select padre, count(*) as NumFigli
from   paternita
group by padre
```

paternita

padre	figlio
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo



padre	NumFigli
Sergio	1
Luigi	2
Franco	2

Semantica di interrogazioni con operatori aggregati e raggruppamenti

1. Si esegue l'interrogazione **ignorando la `group by`** e gli operatori aggregati:

```
select *
from   paternita
```

2. Si raggruppano le **ennuple che hanno lo stesso valore per gli attributi che compaiono nella `group by`**,

   si produce una ennupla del risultato per ogni gruppo, e

   si applica l'operatore aggregato a ciascun gruppo.

Esercizio 7: group by

Massimo dei redditi per ogni gruppo di persone che sono maggiorenni ed hanno la stessa età (indicando anche l'età).

Esprimere la query in SQL.

persone

nome	eta	reddito

Esercizio 7: soluzione

Massimo dei redditi per ogni gruppo di persone che sono maggiorenni ed hanno la stessa età (indicando anche l'età).

```
select eta, max(reddito)
from   persone
where  eta > 17
group by eta
```

Raggruppamenti e target list

In una interrogazione che fa uso di **`group by`**, possono comparire nella target list (oltre a funzioni di aggregazione) **solamente** attributi che compaiono nella **`group by`**.

Esempio: **Scorretta**: redditi delle persone, raggruppati per età.

```
select eta, reddito
from persone
group by eta
```

Potrebbero esistere più valori dell'attributo per lo stesso gruppo.

Corretta: media dei redditi delle persone, raggruppati per età.

```
select eta, avg(reddito)
from persone
group by eta
```

Raggruppamenti e target list, 2

La restrizione sintattica sugli attributi nella **`select`** vale anche per interrogazioni che semanticamente sarebbero corrette (ovvero, per cui esiste un solo valore dell'attributo per ogni gruppo).

Esempio: i padri col loro reddito, e con reddito medio dei figli.

Scorretta:

```
select padre, avg(f.reddito), p.reddito
from   persone f join paternita on figlio = nome
       join persone p on padre = p.nome
group by padre
```

Corretta:

```
select padre, avg(f.reddito), p.reddito
from   persone f join paternita on figlio = nome
       join persone p on padre = p.nome
group by padre, p.reddito
```

Condizioni sui gruppi

Si possono anche imporre le condizioni di **selezione sui gruppi**.

La selezione sui gruppi è **ovviamente diversa** dalla condizione che seleziona le tuple che devono formare i gruppi (clausola **`where`**). Per effettuare la selezione sui gruppi si usa la clausola **`having`**, che deve apparire dopo la "**`group by`**"

Esempio: i padri i cui figli hanno un reddito medio maggiore di 25.

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

Esercizio 8: `where` o `having`?

I padri i cui figli sotto i 30 anni hanno un reddito medio maggiore di 20.

Esercizio 8: soluzione

I padri i cui figli sotto i 30 anni hanno un reddito medio maggiore di 20.

```
select padre, avg(f.reddito)
from   persone f join paternita
       on figlio = nome
where  f.eta < 30
group by padre
having avg(f.reddito) > 20
```

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:

```
select ...
union [all]
select ...
```

Con **union**, i duplicati vengono eliminati (anche in presenza di proiezioni).

Con **union all** vengono mantenuti i duplicati.

Notazione posizionale

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

Quali nomi per gli attributi del risultato?

- quelli del primo operando

Risultato dell'unione

padre	figlio
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo
Luisa	Maria
Luisa	Luigi
Anna	Olga
Anna	Filippo
Maria	Andrea
Maria	Aldo

Notazione posizionale: esempio

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

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

Ancora sulla notazione posizionale

Con le ridenominazioni non cambia niente:

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

Corretta (se vogliamo trattare i padri e le madri come i genitori):

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

Differenza

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

Vedremo che la differenza si può esprimere con `select` nidificate.

Intersezione

```
select nome
from   impiegato
intersect
select cognome as nome
from   impiegato
```

equivale a

```
select i.nome
from   impiegato i, impiegato j
where  i.nome = j.cognome
```

Interrogazioni nidificate

- Nelle condizioni atomiche 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 nidificate: 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 nidificate: operatori

Il risultato di una interrogazione nidificata può essere messo in relazione nella clausola **`where`** mediante diversi **operatori**:

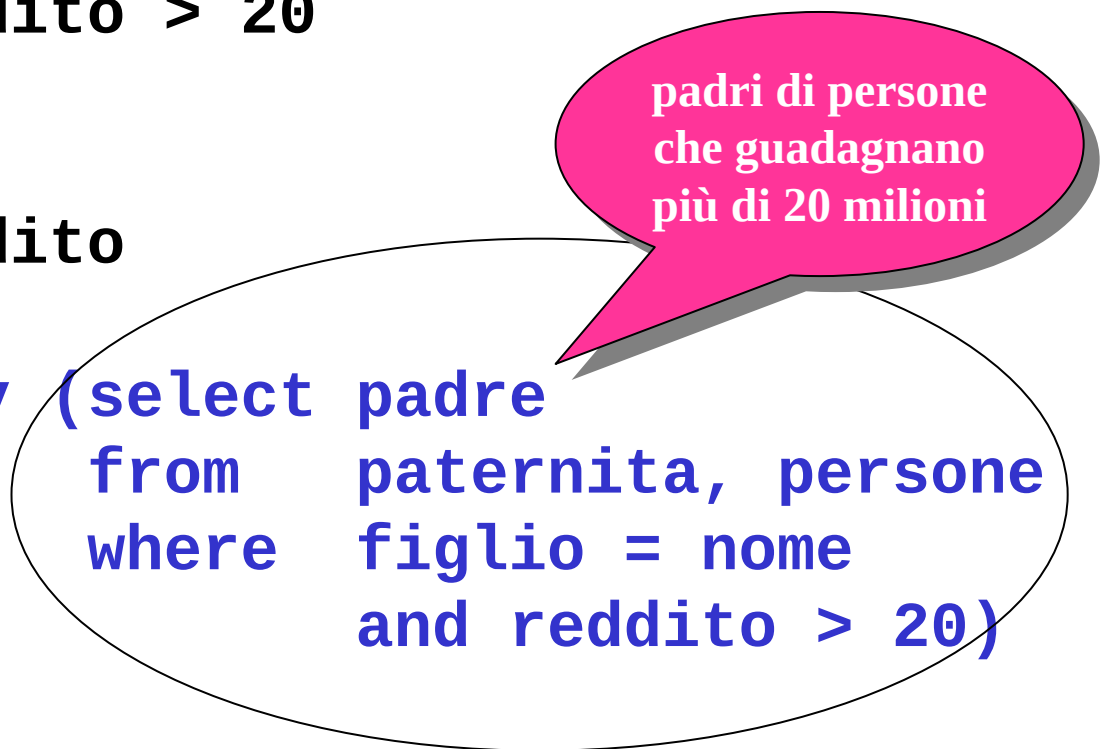
- uguaglianza o altri operatori di confronto (il risultato della interrogazione nidificata deve essere unico)
- se non si è sicuri che il risultato sia unico, si può far precedere l'interrogazione nidificata da:
  - **`any`**: vero, se il confronto è vero per **una qualunque** delle tuple risultato dell'interrogazione nidificata
  - **`all`**: vero, se il confronto è vero per **tutte** le tuple risultato dell'interrogazione nidificata
- l'operatore **`in`**, che è equivalente a **`=any`**
- l'operatore **`not in`**, che è equivalente a **`<>all`**
- l'operatore **`exists`**

Interrogazioni nidificate: esempio

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

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



Interrogazioni nidificate: 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   
                where
                and r
```

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

Interrogazioni nidificate: 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 nidificate: 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)
```

Si noti che l'attributo **nome** si riferisce alla relazione nella clausola **`from`**.

Esercizio 9: interrogazioni nidificate

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

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.

Esercizio 9: soluzione 1

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

```
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 minorenne.

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

Interrogazioni nidificate, commenti

- La forma nidificata può porre problemi di efficienza (i DBMS non sono bravissimi nella loro ottimizzazione), ma talvolta è più leggibile.

- Le sottointerrogazioni non possono contenere operatori insiemistici (“l’unione si fa solo al livello esterno”), ma la limitazione non è significativa.

Interrogazioni nidificate, commenti

- Regole di **visibilità**:
  - non è possibile fare riferimenti a variabili definite in blocchi più interni
  - se un nome di variabile (o tabella) è omesso, si assume riferimento alla variabile (o tabella) più “vicina”

- In un blocco si può fare riferimento a variabili definite nello stesso blocco o in blocchi più esterni.

- **Semantica**: l’interrogazione interna viene eseguita una volta **per ciascuna ennupla** dell’interrogazione esterna

Interrogazioni nidificate: 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)
```

L'attributo **nome** si riferisce alla relazione **persone** nella clausola **from**.

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 nidificata "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 nidificate e correlate

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

Esempio: 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)
```

Esercizio 10: interrogazioni nidificate e correlate

Nome ed età delle madri che hanno almeno un figlio la cui età differisce meno di 20 anni dalla loro.

Esercizio 10: soluzione

Nome ed età delle madri che hanno almeno un figlio la cui età differisce meno di 20 anni dalla loro.

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

Differenza mediante nidificazione

```
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 nidificazione

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

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

Esercizio 11: nidificazione 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)
```

Interrogazioni nidificate: condizione su più attributi

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

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

3. Il Linguaggio SQL

3.3 Manipolazione dei dati

1. definizione dei dati
2. interrogazioni
3. **manipolazione dei dati**
4. ulteriori aspetti

Operazioni di aggiornamento in SQL

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

Inserimento: sintassi

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

oppure

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

Inserimento: esempio

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


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


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


insert into persone (nome)
   select padre
   from paternita
   where  padre not in (select nome from persone)
```

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 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

Sintassi:

delete from *Tabella* [**where** *Condizione*]

Esempi:

```
delete from persone
where eta < 35

delete from paternita
where figlio not in (select nome from persone)
```

Eliminazione: commenti

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

Modifica di ennuple

- **Sintassi**:

update *NomeTabella*
set *Attributo* **=** < *Espressione* | **select** … | **null** | **default** >
[**where** *Condizione*]

- **Semantica**: vengono modificate le ennuple della tabella che soddisfano la condizione “where”

- *Esempi*:

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

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

3. Il Linguaggio SQL

3.4 Ulteriori aspetti

1. definizione dei dati
2. interrogazioni
3. manipolazione dei dati
4. **ulteriori aspetti**

Vincoli di integrità generici: check

Per specificare vincoli di ennupla 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))
)
```

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))
```

Viste

- Una vista è una tabella **la cui istanza è derivata da altre tabelle mediante una interrogazione**.

  `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 non standard

- Voglio sapere l’età delle persone cui corrisponde il massimo reddito (come somma dei redditi delle persone che hanno quella età).

- La nidificazione nella having **non** è ammessa, e perciò questa soluzione è sbagliata:

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

- La soluzione 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)
```

Controllo dell'accesso

- In SQL è possibile specificare chi (utente) e come (lettura, scrittura, …) può utilizzare la base di dati (o parte di essa).
- Oggetto dei **privilegi** (diritti di accesso) sono di solito le tabelle, ma anche altri tipi di **risorse**, quali singoli attributi, viste o domini.
- Un utente predefinito `_system` (amministratore della base di dati) ha tutti i privilegi.
- Il creatore di una risorsa ha tutti i privilegi su di essa.

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 (ennuple)
  - **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 Dipartmento 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 (“**ACID**e”):
  - **A**tomicità
  - **C**onsistenza
  - **I**solamento
  - **D**urabilità (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 ci possono essere violazioni, ma se restano 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

Istruzioni fondamentali

- **`begin transaction`**: specifica l'inizio della transazione (le operazioni non vengono eseguite sulla base di dati)
- **`commit work`**: le operazioni specificate a partire dal **`begin transaction`** vengono eseguite
- **`rollback work`**: si rinuncia all'esecuzione delle operazioni specificate dopo l'ultimo **`begin transaction`**

Esempio di transazione in SQL

```
begin transaction;
    update ContoCorrente
       set Saldo = Saldo – 10
       where NumeroConto = 12345;
    update ContoCorrente
       set Saldo = Saldo + 10
       where NumeroConto = 55555;
commit work;
```