



SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ing. dell'Informazione, Informatica e Statistica
Laurea in Informatica

Basi di Dati, Modulo 2

Prof. Toni Mancini
Dipartimento di Informatica
<http://tmancini.di.uniroma1.it>

Slides B.2.2 (S.B.2.2)

Basi di dati
Sistemi di Gestione di Basi di Dati Relazionali
Il Linguaggio SQL

Indice

Queste slide sono composte dalle seguenti sottounità:

S.B.2.2.1. Introduzione

S.B.2.2.2. Data Definition Language

S.B.2.2.2.1. Creazione di Database, Schemi e Tabelle

S.B.2.2.2.2. Domini Definiti dall'Utente

S.B.2.2.2.3. Generazione di Valori Progressivi

S.B.2.2.2.4. Viste

S.B.2.2.2.5. Controllo dell'Accesso

S.B.2.2.2.6. Indici

S.B.2.2.2.7. Vincoli di Integrità

S.B.2.2.3. Data Manipulation Language

S.B.2.2.3.1. Inserimento, Cancellazione e Modifica (Comandi Base)

S.B.2.2.3.2. Interrogazioni

S.B.2.2.3.2.1. Interrogazioni su Singola Tabella

Indice (2)

- S.B.2.2.3.2.2. Interrogazioni su Tabelle Multiple
- S.B.2.2.3.2.3. Funzioni Aggregate
- S.B.2.2.3.2.4. Raggruppamenti
- S.B.2.2.3.2.5. Operatori Insiemistici
- S.B.2.2.3.2.6. Interrogazioni nella Clausola From
- S.B.2.2.3.2.7. Interrogazioni Annidate
- S.B.2.2.3.2.8. Join Espliciti
- S.B.2.2.3.2.9. Outer Join
- S.B.2.2.3.3. Espressioni
- S.B.2.2.3.4. Inserimento, Cancellazione e Modifica (Comandi Avanzati)
- S.B.2.2.3.5. Transazioni



SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ing. dell'Informazione, Informatica e Statistica
Laurea in Informatica

Basi di Dati, Modulo 2

Prof. Toni Mancini
Dipartimento di Informatica
<http://tmancini.di.uniroma1.it>

Slides B.2.2.1 (S.B.2.2.1)

Basi di dati
Sistemi di Gestione di Basi di Dati Relazionali
Il Linguaggio SQL
Introduzione

Il linguaggio SQL

- ▶ Linguaggio di riferimento per le basi di dati relazionali
- ▶ Originariamente **Structured Query Language**, ora **nome proprio**
- ▶ **1974**: prima versione come linguaggio di interrogazione di System R (IBM)
- ▶ **1983**: standard **de-facto**
- ▶ **1986**: standard ANSI con costrutti base (**SQL-86**)
- ▶ **1989**: aggiunta dei costrutti per la gestione di **integrità referenziale** (**SQL-89**)

Il linguaggio SQL (2)

- ▶ 1992: introduzione di un grande numero di funzionalità (SQL-92 o SQL-2)
 - ▶ linguaggio ricco e complesso
 - ▶ **non ancora** (!) supportato totalmente dai DBMS commerciali
 - ▶ tre livelli di supporto: **Entry** (simile a SQL-89), **Intermediate SQL** (gran parte implementata da DBMS), **Full SQL** (include molte caratteristiche non implementate)
 - ▶ sebbene DBMS si definiscano SQL-compliant, spesso **non** aderiscono completamente allo standard (!)

Il linguaggio SQL (3)

- ▶ 1999 – 2011: SQL:1999, SQL:2003, SQL:2006, SQL:2008, SQL:2011
 - ▶ compatibile con SQL-2, ma offre molte funzionalità in più
 - ▶ suddiviso in parti: SQL/XML (gestione XML), etc.
 - ▶ ancora lontano dall'essere adottato dai DBMS commerciali

Noi vedremo **SQL-2** (SQL-92)

Architettura standard a 3 livelli per i DBMS

Standard ANSI/SPARC

1. **Livello interno**: implementa le strutture fisiche di memorizzazione (file sequenziali, file hash, indici, etc.)
2. **Livello logico**: fornisce un **modello logico** dei dati, indipendente da come sono memorizzati fisicamente: \implies **modello relazionale**
3. **Livello esterno**: fornisce una o più descrizioni di porzioni di interesse della base dati, indipendenti dal modello logico. Può prevedere organizzazioni dei dati alternative e diverse rispetto a quelle utilizzate nello schema logico.

Architettura standard a 3 livelli per i DBMS (2)

Esempio:

Livello interno: strutture interne di memorizzazione

Livello logico: modello relazionale dei dati

Docente

| mat | cognome | nome | email |
|------|---------|--------|---------------------|
| 1101 | Rossi | Mario | rossi@unimia.it |
| 1102 | Bianchi | Anna | a.bianchi@unimia.it |
| 1103 | Verdi | Giulia | giulia.v@unimia.it |

Corso

| codice | nome | aula |
|--------|--------------|------|
| 590 | Basi di dati | 1C |
| 591 | Ing. SW | 3F |
| 592 | Algebra | 1C |
| 593 | Geometria | 3F |

Incarico

| docente | corso |
|---------|-------|
| 1101 | 590 |
| 1101 | 591 |
| 1103 | 593 |
| 1103 | 592 |

Architettura standard a 3 livelli per i DBMS (3)

Livello esterno: viste sui dati

InfoCorsi

| corso | aula | cognome doc. | nome doc. | email doc. |
|--------------|------|--------------|-----------|--------------------|
| Basi di dati | 1C | Rossi | Mario | rossi@unimia.it |
| Ing. SW | 3F | Rossi | Mario | rossi@unimia.it |
| Algebra | 1C | Verdi | Giulia | giulia.v@unimia.it |
| Geometria | 3F | Verdi | Giulia | giulia.v@unimia.it |

Viste **diverse** per utenti **diversi**

SQL e livelli dell'architettura dei DBMS

Il linguaggio SQL fornisce costrutti per operare:

- ▶ a livello logico:
 - ▶ Data Definition Language (DDL) per creare schemi di relazioni
 - ▶ Data Manipulation Language (DML) per interrogare ed aggiornare i dati
- ▶ a livello esterno: costrutti DDL per creare viste della base dati



SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ing. dell'Informazione, Informatica e Statistica
Laurea in Informatica

Basi di Dati, Modulo 2

Prof. Toni Mancini
Dipartimento di Informatica
<http://tmancini.di.uniroma1.it>

Slides B.2.2.2 (S.B.2.2.2)

Basi di dati
Sistemi di Gestione di Basi di Dati Relazionali
Il Linguaggio SQL
Data Definition Language



SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ing. dell'Informazione, Informatica e Statistica
Laurea in Informatica

Basi di Dati, Modulo 2

Prof. Toni Mancini
Dipartimento di Informatica
<http://tmancini.di.uniroma1.it>

Slides B.2.2.2.1 (S.B.2.2.2.1)

Basi di dati
Sistemi di Gestione di Basi di Dati Relazionali
Il Linguaggio SQL
Data Definition Language
Creazione di Database, Schemi e Tabelle

Creazione di Database, Schemi e Tabelle

- ▶ **create** database nome_database [opzioni]
Crea un database.
- ▶ **create** schema nome_schema [opzioni]
Crea uno schema (namespace) all'interno del database corrente.
- ▶ **create table** nome_tabella (...)
Crea una nuova tabella all'interno di uno schema del database corrente

Creazione di tabelle

```
create table [nome_schema.] nome_tabella (  
    nome_attributo dominio [vincoli di dominio],  
    nome_attributo dominio [vincoli di dominio],  
    ...  
    nome_attributo dominio [vincoli di dominio],  
    [altri vincoli intra-relazionali]  
    [vincoli inter-relazionali]  
)
```

nome_schema è opzionale

se omesso \implies schema di default (senza nome).

Creazione di tabelle (2)

Esempio:

```
create table Corso (  
    codice integer not null ,  
    nome character varying (100) not null ,  
    aula character varying (10) not null ,  
    primary key (codice)  
)
```

```
create table Incarico (  
    docente integer not null ,  
    corso integer not null ,  
    primary key (docente, corso),  
    foreign key (docente) references Docente(matr),  
    foreign key (corso) references Corso(codice)  
)
```


Domini SQL predefiniti

- ▶ Domini numerici:
 - ▶ interi: **integer**, **smallint** (e molti altri)
 - ▶ decimali: **numeric**(prec, scala), **decimal**(prec, scala)
 - ▶ approssimati: **float**(prec), real, double precision
- ▶ Stringhe
 - ▶ **character** [**varying**] (lung_max) (abbrev. in **char**/**varchar**)
 - ▶ text
 - ▶ (molti altri)
- ▶ Istanti temporali:
 - ▶ **date** (tipo record con campi per anno, mese, giorno)
 - ▶ **time** (tipo record con campi per ora, min, sec)
 - ▶ **timestamp** (tipo record con campi per anno, . . . , sec)
- ▶ Intervalli temporali: **interval**
- ▶ Valori booleani: Boolean (SQL:1999)
- ▶ Dati non strutturati di grandi dimensioni: **CLOB** e **BLOB**

Creazione di tabelle: valori di default

```
create table Impiegato (  
    nome ...,  
    cognome ...,  
    stipendio integer default 0,  
    ...
```

Se, durante un inserimento o modifica di una ennupla della tabella Impiegato **non** viene indicato un valore per la colonna stipendio **allora** tale valore viene messo a 0.

Creazione di tabelle: vincoli di dominio

```
create table Impiegato (  
    nome varchar(100) not null ,  
    cognome varchar(100) not null ,  
    stipendio integer default 0  
        check (stipendio >= 0) ,  
    ...  
)
```

- ▶ Ogni ennupla nella relazione **deve** soddisfare il vincolo $\text{stipendio} \geq 0$.
- ▶ Il vincolo viene controllato automaticamente **prima** dell'**inserimento** o della **modifica** di ennuple.
- ▶ In caso di **violazione**, l'inserimento o la modifica non ha luogo (e viene generato un errore).

SQL e modello relazionale

Una tabella SQL **non** rappresenta in generale una relazione (!)

In particolare, **può contenere ennuple uguali**

⇒ affinché rappresenti una relazione, è necessario definire **almeno una chiave**

SQL e modello relazionale (2)

```
create table Studente (
  matr integer not null ,
  cognome
    varchar(100) not null ,
  nome
    varchar(100) not null
)
```

| matr | cognome | nome |
|------|---------|--------|
| 1000 | Rossi | Mario |
| 1000 | Rossi | Mario |
| 1001 | Verdi | Giulia |

```
create table Studente (
  matr integer not null
,
  cognome
    varchar(100) not
      null ,
  nome
    varchar(100) not
      null ,
  primary key (matr)
)
```

| <u>matr</u> | cognome | nome |
|-------------|---------|--------|
| 1000 | Rossi | Mario |
| 1001 | Verdi | Giulia |
| 1002 | Bianchi | Anna |

Vincoli di integrità referenziale

Esempio

Officina

| <u>nome</u> | indirizzo |
|-------------|----------------------|
| FixIt | via delle Spighe 4 |
| CarFix | via delle Betulle 32 |
| MotorGo | piazza Turing 1 |

Riparazione

| <u>officina</u> | <u>codice</u> | veicolo |
|-----------------|---------------|-----------|
| FixIt | 1 | HK 243 BW |
| CarFix | 1 | AA 662 XQ |
| FixIt | 2 | HK 243 BW |

$$\pi_{officina}(Riparazione) \subseteq \pi_{nome}(Officina)$$

$$\pi_{veicolo}(Riparazione) \subseteq \pi_{targa}(Veicolo)$$

Veicolo

| <u>targa</u> | tipo |
|--------------|------|
| HK 243 BW | auto |
| AA 662 XQ | auto |
| GF 211 HA | moto |

RicambioRip

| <u>officina</u> | <u>rip</u> | <u>ricambio</u> |
|-----------------|------------|-----------------|
| FixIt | 1 | A755 |
| FixIt | 1 | A788 |
| CarFix | 1 | A991 |
| FixIt | 2 | B332 |

$$\pi_{officina}^{rip}(RicambioRip) \subseteq \pi_{officina}^{codice}(Riparazione)$$

Vincoli di integrità referenziale (2)

| Officina | |
|-------------|-----------|
| <u>nome</u> | indirizzo |

```
create table Officina (  
    nome varchar(100) not null ,  
    indirizzo varchar(500) not null ,  
    primary key (nome)  
)
```


Vincoli di integrità referenziale (3)

Veicolo

| Veicolo | |
|--------------|------|
| <u>targa</u> | tipo |

```
create table Veicolo (  
    targa char(8) not null ,  
    tipo varchar(50) not null ,  
    primary key (targa)  
)
```

Vincoli di integrità referenziale (4)

Riparazione

| | | |
|-----------------|---------------|---------|
| <u>officina</u> | <u>codice</u> | veicolo |
|-----------------|---------------|---------|

```
create table Riparazione (  
    officina varchar(100) not null ,  
    codice integer not null ,  
    veicolo char(8) not null ,  
    primary key (officina , codice) ,  
    foreign key (officina) references Officina(nome) ,  
    foreign key (veicolo) references Veicolo(targa)  
)
```

Sintassi:

foreign key (attributi) references Tabella(attributi chiave)

⇒ Gli attributi di Tabella a cui il vincolo punta **devono** formare una **chiave** (**primary key** o **unique**) di Tabella

Vincoli di integrità referenziale (5)

RicambioRip

| | | |
|-----------------|------------|-----------------|
| <u>officina</u> | <u>rip</u> | <u>ricambio</u> |
|-----------------|------------|-----------------|

```
create table RicambioRip (  
    officina varchar(100) not null ,  
    rip integer not null ,  
    ricambio char(5) not null ,  
    primary key (officina , rip , ricambio) ,  
    foreign key (officina , rip)  
        references Riparazione(nome, codice) ,  
    foreign key (ricambio)  
        references Ricambio(codice)  
)
```

Modifica e cancellazione di tabelle, schemi e database

Modifica

- ▶ **alter table**
 - ▶ **alter table add column**
 - ▶ **alter table drop column**
 - ▶ **alter table alter column**
 - ▶ **alter table add constraint**
 - ▶ **alter table drop constraint**
 - ▶ etc.

Cancellazione

- ▶ **drop table** <nome tabella>
- ▶ **drop schema** <nome schema>
- ▶ **drop database** <nome database>

Per i dettagli si veda la documentazione di SQL e quella del DBMS in uso.



SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ing. dell'Informazione, Informatica e Statistica
Laurea in Informatica

Basi di Dati, Modulo 2

Prof. Toni Mancini
Dipartimento di Informatica
<http://tmancini.di.uniroma1.it>

Slides B.2.2.2.2 (S.B.2.2.2.2)

Basi di dati
Sistemi di Gestione di Basi di Dati Relazionali
Il Linguaggio SQL
Data Definition Language
Domini Definiti dall'Utente

Domini SQL definiti dall'utente

È possibile per l'utente definire nuovi domini, da usarsi come i domini predefiniti.

In particolare, è possibile definire:

- ▶ Domini specializzazione di altri domini
- ▶ Domini di tipo enumerativo
- ▶ Domini di tipo record

SQL offre due comandi:

create domain ...

e

create type ...

I due comandi offrono funzionalità **diverse**.

Definizione di domini SQL specializzati

Un **dominio specializzato** definisce un insieme di valori che è un **sottoinsieme** dei valori di un dominio esistente.

```
create domain nome_dominio as tipo_base  
    [valore di default]  
    [vincolo]
```

Esempio: Definizione del dominio **voto** (interi tra 18 e 30):

```
create domain voto as integer  
    default 0  
    check (value >= 18 and value <= 30)
```

Il **vincolo** definisce i valori del dominio base che sono anche valori del dominio specializzato. Si usano le parole chiavi **check** e **value**.

Definizione di domini SQL di Tipo Enumerativo

Un **dominio di tipo enumerativo** definisce un insieme **finito**, **piccolo** e **stabile** di valori, ognuno identificato da una **etichetta**.

```
create type nome_dominio as  
    enum ( 'valore 1' , ... , 'valore N' )
```

Esempio: Definizione del dominio enumerativo **genere** (valori: **M** ed **F**):

```
create type genere as enum ( 'M' , 'F' )
```

Esempio: Definizione del dominio enumerativo **continente** (a 6 valori):

```
create type continente as  
    enum ( 'America Nord' , 'America Sud' , 'Africa' ,  
          'Europa' , 'Asia' , 'Oceania' )
```

Nota: le etichette (ad es., **'America Nord'**) **non sono stringhe**, ma **identificatori per gli elementi dell'insieme** (nonostante gli apici!)

Definizione di domini SQL di Tipo Record

I valori di un **dominio di tipo record** (o **dominio composto**) sono record di valori, uno per ogni **campo** del record. Il valore di ogni campo di un record è del rispettivo dominio.

```
create type nome_dominio as (  
    campo1 dominio1, ..., campoN dominioN  
)
```

Esempio: Definizione del dominio composto **indirizzo**:
(una definizione molto semplice, solo a titolo di esempio)

```
create type indirizzo (  
    via varchar(200), città varchar(100)  
)
```

Nota: Il costrutto **create type** è implementato in modo non-standard da alcuni DBMS, e può soffrire di limitazioni (ad es., niente vincoli di dominio mediante **check** e/o **not null**).

Qui mostriamo la sintassi di PostgreSQL. Consultare sempre la documentazione del DBMS in uso!

Modifica e cancellazione di domini

I domini creati dall'utente possono essere **modificati** o **rimossi**:

Modifica:

- ▶ **alter domain** <nome dominio>
- ▶ **alter** type <nome dominio>

Cancellazione:

- ▶ **drop domain** <nome dominio>
- ▶ **drop** type <nome dominio>

Per i dettagli si veda la documentazione di SQL e quella del DBMS in uso.



SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ing. dell'Informazione, Informatica e Statistica
Laurea in Informatica

Basi di Dati, Modulo 2

Prof. Toni Mancini
Dipartimento di Informatica
<http://tmancini.di.uniroma1.it>

Slides B.2.2.2.3 (S.B.2.2.2.3)

Basi di dati
Sistemi di Gestione di Basi di Dati Relazionali
Il Linguaggio SQL
Data Definition Language
Generazione di Valori Progressivi

Generazione di Valori Progressivi

Abbiamo visto come alcune volte sia necessario o conveniente aggiungere un **identificatore artificiale** in una entità e quindi un attributo aggiuntivo nella relazione della base dati che la realizza

Esempio:

| Prenotazione |
|---------------------|
| id : integer {id} |
| istante : timestamp |

Prenotazione (id:integer, istante:timestamp)

È possibile lasciare al DBMS il compito di assegnare valori diversi (tipicamente **progressivi**) per il campo **id** alle diverse ennuple della relazione **Prenotazione**.

Non esiste uno standard.

Generazione di Valori Progressivi in PostgreSQL

PostgreSQL fornisce il costrutto delle **sequenze**.

```
create sequence Prenotazione_id_seq;  
create table Prenotazione (  
    id integer default nextval('Prenotazione_id_seq'  
    ') not null ,  
    istante timestamp not null ,  
    primary key (id)  
);
```

Scorciatoia: **create table** Prenotazione (id serial **not null** , ...)

Inserimento di **ennuple**: sfrutta il fatto che id ha un valore di **default**:

```
insert into Prenotazione(istante)  
    values ( '2011-08-24 13:15:05 ' ) returning id
```

Nota: Il costrutto PostgreSQL (non standard) **returning id** permette al comando **insert** di restituire all'utente il valore per **id** scelto dal DBMS.

Generazione di Valori Progressivi in MySQL

MySQL fornisce il modificatore `auto_increment` per gli attributi.

```
create table Prenotazione (  
    id integer not null auto_increment,  
    istante timestamp not null,  
    primary key (id)  
);
```

Inserimento di ennuple:

```
insert into Prenotazione(istante)  
    values ( '2011-08-24 13:15:05 ' )
```



SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ing. dell'Informazione, Informatica e Statistica
Laurea in Informatica

Basi di Dati, Modulo 2

Prof. Toni Mancini
Dipartimento di Informatica
<http://tmancini.di.uniroma1.it>

Slides B.2.2.2.4 (S.B.2.2.2.4)

Basi di dati
Sistemi di Gestione di Basi di Dati Relazionali
Il Linguaggio SQL
Data Definition Language
Viste

Architettura standard per i DBMS

Architettura standard a 3 livelli per i DBMS:

1. Livello interno

rappresentazione fisica dei dati, dipende dal DBMS.

2. Livello logico

modello logico dei dati, indipendente da come sono memorizzati fisicamente \implies modello relazionale.

3. Livello esterno

fornisce una o più descrizioni di porzioni di interesse della base dati, indipendenti dal modello logico. Può prevedere organizzazioni dei dati alternative e diverse rispetto a quelle utilizzate nello schema logico.

\implies Viste sui dati

Viste sui Dati

Vista: tabella le cui ennuple sono **calcolate** a partire da una interrogazione su altre tabelle (e/o viste).

```
create view <Nome vista> as  
  select ...
```

Esempio:

```
create view Genitori as  
  select mat.figlio as persona ,  
         mat.madre as madre, pat.padre as padre  
from Maternita mat, Paternita pat  
where mat.figlio = pat.figlio
```

Genitori è una tabella **virtuale**: il suo contenuto è calcolato solo quando serve e **non** è memorizzato in modo persistente.

⇒ Non è possibile invocare **insert**, **delete**, **update** su una vista

Esempi

Esempio 1: Definire una vista che mostri il reddito medio delle persone divise per età.

| Persona | | |
|-------------|-----|---------|
| <u>nome</u> | eta | reddito |

```
create view RedditoMedioPerEta as  
  select eta , avg(reddito) as redditoMedio  
  from Persona  
  group by eta
```

Esempi

Esempio 2: Definire una vista che mostri le età delle persone che hanno reddito medio massimo.

| Persona | | |
|-------------|-----|---------|
| <u>nome</u> | eta | reddito |

```
create view EtaRedditoMax as  
  select eta  
  from RedditoMedioPerEta  
  where redditoMedio =  
    (select max(redditoMedio)  
     from RedditoMedioPerEta)
```



SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ing. dell'Informazione, Informatica e Statistica
Laurea in Informatica

Basi di Dati, Modulo 2

Prof. Toni Mancini
Dipartimento di Informatica
<http://tmancini.di.uniroma1.it>

Slides B.2.2.2.5 (S.B.2.2.2.5)

Basi di dati
Sistemi di Gestione di Basi di Dati Relazionali
Il Linguaggio SQL
Data Definition Language
Controllo dell'Accesso

Controllo dell'accesso

- ▶ I DBMS prevedono meccanismi per controllare l'accesso alle risorse:
 - ▶ tabelle
 - ▶ domini
 - ▶ viste
 - ▶ procedure
 - ▶ ...
- ▶ Meccanismo basato su utenti/password e privilegi
- ▶ L'utente che crea una risorsa ne è il proprietario, può modificarla, eliminarla, e dare privilegi ad altri utenti
- ▶ Esiste super-utente (`_system` o `root`).

Creazione di Utenti

Per creare un utente:

```
create user <Nome> with <Opzioni>
```

Tra le opzioni è presente 'password' per definire una password.

Attenzione: molti dettagli sono omessi e i DBMS possono comportarsi in modo diverso. Consultare **sempre** la documentazione del DBMS in uso.

Privilegi Utente

Un **privilegio** è definito da:

- ▶ la **risorsa** alla quale si riferisce
- ▶ l'**utente** che lo **concede**
- ▶ l'**utente** che lo **riceve**
- ▶ l'**azione** permessa sulla risorsa
- ▶ se l'utente che riceve il privilegio può **concederlo** o meno ad altri utenti.

Privilegi Disponibili

I DBMS prevedono (almeno) i seguenti **privilegi**:

- ▶ **insert**: permette di inserire un nuovo oggetto (ad es., **ennupla**) nella risorsa (ad es., **tabella**)
- ▶ **update**: permette di aggiornare il valore di un oggetto (ad es., **ennupla**) della risorsa (ad es., **tabella**)
- ▶ **delete**: permette di eliminare oggetti (ad es., **ennuple**) dalla risorsa (ad es., **tabella**)
- ▶ **select**: permette di leggere la risorsa (ad es., **tabella**)
- ▶ **references**: permette che venga fatto un riferimento alla risorsa (ad es., **tabella**) durante la definizione dello schema di una tabella
- ▶ **usage**: permette di utilizzare la risorsa (ad es., **dominio**) in una definizione (ad es., dello schema di una tabella)
- ▶ **all privileges**: permette qualunque azione sulla risorsa.

Concessione di Privilegi ad Utenti

Una parte del linguaggio SQL è dedicata alla gestione del controllo d'accesso.

Per **concedere** dei privilegi ad un insieme di utenti:

```
grant <Privilegi> on <Risorsa> to <Utenti>  
[with grant option]
```

Fornisce i privilegi < Privilegi > sulla risorsa <Risorsa> agli utenti <Utenti>

Se si specifica with **grant** option gli utenti che ricevono i privilegi possono invocare comandi **grant** per gli stessi privilegi sulla stessa risorsa per concederli ad altri utenti.

Esempio: **grant select on** Impiegato **to** www

L'utente **www** può effettuare interrogazioni sulla tabella (o vista) **Impiegato** (ma non può concedere il privilegio ad altri).

Revoca di Privilegi ad Utenti

Per **revocare** dei privilegi ad un insieme di utenti:

```
revoke <Privilegi> on <Risorsa> from <Utenti>  
    [restrict | cascade]
```

Revoca i privilegi < Privilegi > sulla risorsa <Risorsa> agli utenti <Utenti>

Se si specifica **cascade**, vengono revocati anche i privilegi propagati verso altri utenti (default: **restrict**).

SQL:1999 definisce anche i **ruoli** (insiemi di privilegi). Agli utenti si assegnano ruoli che definiscono privilegi.

Attenzione: **molte** dettagli sono omessi e i DBMS possono comportarsi in modo diverso. Consultare **sempre** la documentazione del DBMS in uso.

Principio dei Privilegi Minimali

In questo corso non entriamo in dettagli sugli **aspetti di sicurezza e privacy**, cruciali per applicazioni reali.

Tali aspetti saranno affrontati in corsi più avanzati.

Tuttavia, la **regola d'oro** è:

Principio dei privilegi minimali:

*Concedere ad un utente **solo i privilegi strettamente necessari** per poter eseguire le azioni che deve poter eseguire.*



SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ing. dell'Informazione, Informatica e Statistica
Laurea in Informatica

Basi di Dati, Modulo 2

Prof. Toni Mancini
Dipartimento di Informatica
<http://tmancini.di.uniroma1.it>

Slides B.2.2.2.6 (S.B.2.2.2.6)

Basi di dati
Sistemi di Gestione di Basi di Dati Relazionali
Il Linguaggio SQL
Data Definition Language
Indici

Indici

Strutture dati **ausiliarie** su memoria secondaria che consentono il recupero efficiente delle ennuple di una relazione quando sono dati i valori (o intervalli di valori) di alcuni attributi (**pseudo-chiave**).

Implementati per lo più con strutture **ad albero** a pochi livelli (ma anche **hash**).

Possono essere:

- ▶ **primari**: le ennuple sono memorizzate **nello stesso indice** o, sebbene in un file separato, sono memorizzate ordinate per il valore della **pseudo-chiave**
- ▶ **secondari**: l'ordinamento dell'indice **è indipendente da quello delle ennuple nel file che memorizza la relazione**. L'indice contiene **puntatori alle ennuple**.

⇒ Per dettagli, si veda il corso di “Basi di Dati, Modulo 1”.

Gestione degli indici in SQL

Molti DBMS accettano i comandi seguenti (con varianti e opzioni) per creare ed eliminare indici.

- ▶ **create index** <Nome> **on** <Tabella> (<Attributi>) **using** <Metodo>
- ▶ **drop index** <Nome>

Quest comandi non sono standard SQL.

Creare un indice su una lista di attributi <Attributi> consente:

- ▶ il **recupero più efficiente** delle ennuple di <Tabella> a partire da valori degli attributi <Attributi> (esatti o intervalli di valori)
- ▶ al prezzo di **minore efficienza negli inserimenti, aggiornamenti e cancellazioni di ennuple** (l'impatto negativo è di solito **piccolo**, grazie a tecniche sofisticate di aggiornamento degli indici).

I DBMS di solito creano **automaticamente** un indice sugli attributi **chiave primaria** di ogni tabella.



SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ing. dell'Informazione, Informatica e Statistica
Laurea in Informatica

Basi di Dati, Modulo 2

Prof. Toni Mancini
Dipartimento di Informatica
<http://tmancini.di.uniroma1.it>

Slides B.2.2.2.7 (S.B.2.2.2.7)

Basi di dati
Sistemi di Gestione di Basi di Dati Relazionali
Il Linguaggio SQL
Data Definition Language
Vincoli di Integrità

Vincoli di Integrità

Abbiamo già visto alcune tipologie di **vincoli di integrità** supportati dai DBMS relazionali:

- ▶ Vincoli di **chiave**
primary key e **unique** dentro **create table** e **alter table**
- ▶ Vincoli di **dominio**
check in **create domain** e **alter domain**
check e **not NULL** in **create table** e **alter table**
- ▶ Vincoli di **ennupla**
check in **create table** e **alter table**
- ▶ Vincoli di **foreign key**
foreign key in **create table** e **alter table**

Esempio

Vogliamo creare il seguente **database con vincoli**:

Progetto (nome:varchar(100), inizio:date, fine:date)

VincoloDB **ennupla**: fine \geq inizio

WP (nome:varchar(100), progetto:varchar(100), inizio:date, fine:date)

VincoloDB **foreign key**: progetto references Progetto(nome)

VincoloDB **ennupla**: fine \geq inizio

- Vincolo di ennupla fine \geq inizio in **Progetto**.

Possiamo usare una clausola **check** all'interno di **create table**:

```
create table Progetto (  
    nome varchar(100) not null ,  
    inizio date not null ,  
    fine date not null ,  
    primary key (nome),  
    check (fine >= inizio)  
)
```

Esempio

Vogliamo creare il seguente **database con vincoli**:

Progetto (nome:varchar(100), inizio:date, fine:date)

VincoloDB **ennupla**: fine \geq inizio

WP (nome:varchar(100), progetto:varchar(100), inizio:date, fine:date)

VincoloDB **foreign key**: progetto references Progetto(nome)

VincoloDB **ennupla**: fine \geq inizio

- Vincolo di ennupla **fine \geq inizio** in **WP**. Analogamente:

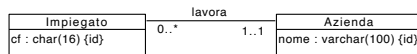
```
create table WP (  
    nome varchar(100) not null ,  
    progetto varchar(100) not null ,  
    inizio date not null ,  
    fine date not null ,  
    primary key (nome),  
    foreign key progetto references Progetto(  
        nome),  
    check (fine >= inizio)  
)
```

Vincoli di Integrità e Transazioni

I vincoli di integrità, per default, vengono valutati al termine di **ogni** comando. Questo può rendere alcuni inserimenti e modifiche **impossibili**.

Esempio

Diagramma ER ristrutturato:



Schema relazionale (senza accorpamenti):

Impiegato (cf: char(16))

VincoloDB **inclusione**: $cf \subseteq \text{lavora}(\text{impiegato})$
 \Rightarrow **foreign key**: cf references lavora(impiegato)

Azienda (nome: varchar(100))

lavora (impiegato: char(16), azienda: varchar(100))

VincoloDB **foreign key**: impiegato references Impiegato(cf)
 VincoloDB **foreign key**: azienda references Azienda(nome)

È **impossibile inserire** una ennupla in **Impiegato**!

Vincoli di Integrità e Transazioni (2)

Un vincolo di integrità può essere dichiarato **deferrable**.

- ▶ Per un vincolo **deferrable** è possibile per l'utente decidere **se** valutarlo solo al termine della **transazione corrente**
- ▶ Per default tutti i vincoli sono **not deferrable**, quindi vengono valutati immediatamente.

Vincoli di Integrità e Transazioni (3)

Per l'esempio possiamo quindi scrivere:

```
create table Impiegato (  
    cf char(16) not null ,  
    primary key (cf)  
);  
create table Azienda (  
    nome varchar(100) not null ,  
    primary key (nome)  
);  
  
create table lavora (  
    impiegato char(16) not null ,  
    azienda varchar(100) not null ,  
    primary key (impiegato),  
    foreign key (impiegato) references  
        Impiegato(cf) deferrable ,  
    foreign key (azienda) references  
        Azienda(nome)  
);  
  
alter table Impiegato  
add constraint impiegatoInLavora  
foreign key (cf) references lavora(impiegato)  
    deferrable;
```

Nota: per imporre uno dei vincoli di foreign key utilizziamo **alter table** per evitare di menzionare (in **create table** Impiegato) il nome della tabella **lavora**, non ancora creata.

Vincoli di Integrità e Transazioni (4)

L'inserimento di dati può ora avvenire in una **transazione**:

```
insert into azienda(nome) values ('az');  
  
begin transaction;  
set constraints all deferred;  
insert into impiegato(cf) values ('111');  
insert into lavora(impiegato, azienda) values ('111'  
    ', 'az');  
commit work;
```

Il comando **set constraints all deferred** ; richiede esplicitamente al DBMS di verificare i vincoli dichiarati **deferrable** solo al termine della transazione.

Vincoli di Integrità Generici

Esempio

Vogliamo creare il seguente database con vincoli:

Progetto (nome:varchar(100), inizio:date, fine:date)

VincoloDB **ennupla**: fine \geq inizio

WP (nome:varchar(100), progetto:varchar(100), inizio:date, fine:date)

VincoloDB **foreign key**: progetto references Progetto(nome)

VincoloDB **ennupla**: fine \geq inizio

Supponiamo di voler imporre il seguente **ulteriore vincolo**: **inizio e fine di ogni WP devono essere comprese tra inizio e fine del corrispondente Progetto.**

Vincoli di Integrità Generici

Esempio

Supponiamo di voler imporre il seguente **ulteriore vincolo**: inizio e fine di ogni **WP** devono essere comprese tra inizio e fine del corrispondente **Progetto**.

Potremmo pensare di procedere così:

```
create table WP (  
  nome varchar(100) not null ,  
  progetto varchar(100) not null ,  
  inizio date not null ,  
  fine date not null ,  
  primary key (nome),  
  foreign key progetto references Progetto(nome),  
  check (fine >= inizio),  
  check (inizio between  
    date (select inizio from Progetto where  
      nome=progetto)  
    and date (select fine from Progetto  
      where nome=progetto)),  
  check (fine between  
    date (select inizio from Progetto where  
      nome=progetto)  
    and date (select fine from Progetto  
      where nome = progetto))  
)
```


Vincoli di Integrità Generici

I vincoli non traducibili con le tecniche viste vanno imposti nel DBMS con **approcci più complessi**:

1. **asserzioni** in standard SQL (almeno teoricamente. . .)
2. **trigger** in linguaggi non standard

Asserzioni

Lo **standard SQL** offre un **costrutto generale** per esprimere vincoli di integrità **arbitrariamente complessi**:

```
create assertion <nome vincolo>  
  check ( <condizione> )
```

Esempio: inizio e fine di ogni **WP** devono essere comprese tra inizio e fine del corrispondente **Progetto**.

Potremmo pensare di procedere così:

```
create assertion check_date_wp  
check (  
  not exists(  
    select * from WP w, Progetto p  
    where w.progetto = p.nome  
    and ( w.inizio < p.inizio or w.fine > p.  
        fine)  
  )  
)
```

Vincolo ignorato! Nessun DBMS commerciale oggi implementa le asserzioni SQL (sarebbe troppo inefficiente!)

Trigger in PostgreSQL

Sintassi

```
create [ constraint ] trigger <nome>
{ before | after | instead of } {<operaz.
  intercettata> [ or ... ]}
on <tabella>
[ from referenced_table_name ]
[ not deferrable | [ deferrable ]
  { initially immediate | initially deferred
  } }
[ for [ each ] { row | statement } ]
[ when ( <condizione> ) ]
execute procedure <nome funzione> ( <argomenti> )
```

- ▶ operazione intercettata: **insert**, **update**, **delete**
- ▶ istante dell'invocazione: **prima** o **dopo** o **invece** dell'operazione (instead of vale solo per le viste)
- ▶ **deferrable** (solo se di tipo **constraint** e after)
- ▶ **when**: se falsa, la funzione non viene eseguita

Trigger in PostgreSQL

Sintassi

```

create [ constraint ] trigger <nome>
  { before | after | instead of } {<operaz.
    intercettata> [ or ... ]}
on <tabella>
  [ from referenced_table_name ]
  { not deferrable | [ deferrable ]
    { initially immediate | initially deferred
      } }
  [ for [ each ] { row | statement } ]
  [ when ( <condizione> ) ]
execute procedure <nome funzione> ( <argomenti> )
  
```

- ▶ **for each row**: la funzione viene invocata una volta per ogni ennupla impattata dall'operazione
- ▶ **for statement**: la funzione viene invocata una volta per comando (ad es., **insert**).

PostgreSQL supporta la definizione di funzioni scritte nel linguaggio procedurale chiamato **PL/pgSQL** (proviene dal più famoso PL/SQL Oracle), ma anche in **altri linguaggi** (ad es. **Python**).

Esempio: Disgiunzione

Diagramma ER concettuale

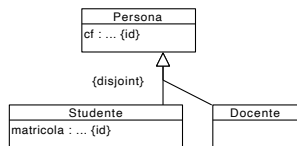
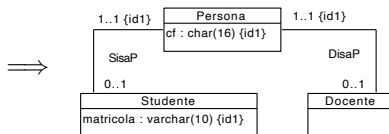


Diagramma ER ristrutturato



$\forall p$ Persona(p):

\rightarrow

$[(\exists s \text{ SisaP}(s, p)) \rightarrow \neg(\exists d \text{ DisaP}(d, p))]$

Schema Relazionale

Persona(cf:char(16))

Studente(persona:char(16), matricola:varchar(10))

VincoloDB **foreign key**: persona references Persona(cf)

VincoloDB **altra chiave**: matricola

Docente(persona:char(16))

VincoloDB **foreign key**: docente references Persona(cf)

~~Persona~~

Esempio: Disgiunzione (2)

Trigger per V.Persona.isa.disj:

- ▶ **Operazioni:** inserimento o modifica in **Studente** o **Docente**
- ▶ **Istante** di invocazione: **prima** dell'operazione intercettata
- ▶ **Funzione:**
 1. Sia **isError** = **FALSE**;
 2. Sia **new** l'ennupla che si sta inserendo oppure l'ennupla risultato della modifica;
 3. Se si sta inserendo o modificando una ennupla in **Studente**:

```
isError := exists (select * from Docente d
                  where d.persona = new.persona);
```
 4. Altrimenti (inserimento/modifica di una ennupla in **Docente**):

```
isError := exists (select * from Studente s
                  where s.persona = new.persona);
```
 5. Se **isError** = **TRUE** **blocca** l'operazione;
 6. Altrimenti **permetti** l'operazione.

Esempio: Disgiunzione (3)

Codice PL/SQL per la funzione del trigger V.Persona.isa.disj:

```
create function V_Persona_isa_disj()
    returns trigger as $V_Persona_isa_disj$
declare isError boolean := false; — var. locale
begin
    case — TG_TABLE_NAME: nome tabella relativa all'operaz.
        when TG_TABLE_NAME ilike 'Studiante' then
            isError := exists (
                select * from Docente d where d.persona = new.persona);
        when TG_TABLE_NAME ilike 'Docente' then
            isError = exists (
                select * from Studiante s where s.persona = new.persona);
        else raise exception 'La funzione non
            puo'' essere invocata sulla tabella %', TG_TABLE_NAME;
    end case;
    if (isError) then raise exception 'Vincolo V.SDisaP.disjoint
        violato da Persona.id = %', new.persona;
    end if;
    return new; — se OK, il trigger 'before' deve restituire 'new'
end;
$V_Persona_isa_disj$ language plpgsql;
```

Esempio: Disgiunzione (4)

Definizione del **trigger** `V.Persona.isa.disj` su `Studente` e `Docente`:

```
create trigger V_Persona_isa_disj_trigger_Studente  
before insert or update on Studente  
for each row execute procedure V_Persona_isa_disj();
```

```
create trigger V_Persona_isa_disj_trigger_Docente  
before insert or update on Docente  
for each row execute procedure V_Persona_isa_disj();
```




SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ing. dell'Informazione, Informatica e Statistica
Laurea in Informatica

Basi di Dati, Modulo 2

Prof. Toni Mancini
Dipartimento di Informatica
<http://tmancini.di.uniroma1.it>

Slides B.2.2.3 (S.B.2.2.3)

Basi di dati
Sistemi di Gestione di Basi di Dati Relazionali
Il Linguaggio SQL
Data Manipulation Language



SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ing. dell'Informazione, Informatica e Statistica
Laurea in Informatica

Basi di Dati, Modulo 2

Prof. Toni Mancini
Dipartimento di Informatica
<http://tmancini.di.uniroma1.it>

Slides B.2.2.3.1 (S.B.2.2.3.1)

Basi di dati
Sistemi di Gestione di Basi di Dati Relazionali
Il Linguaggio SQL
Data Manipulation Language
Inserimento, Cancellazione e Modifica (Comandi Base)

Inserimento di Ennuple

L'istruzione per inserire $M \geq 1$ ennuple in una tabella è **insert**

```
insert into tabella (attributo1 , ... , attributoN)
values
    (valore1_1 , ... , valore1_N) ,
    ...
    (valoreM_1 , ... , valoreM_N)
```

Esempio

| Officina | |
|-------------|----------------------|
| <u>nome</u> | indirizzo |
| FixIt | via delle Spighe 4 |
| CarFix | via delle Betulle 32 |

Per inserire la nuova ennupla ('MotorGo', 'piazza Turing 1')...

Inserimento di Ennuple (2)

Per inserire la nuova ennupla ('MotorGo', 'piazza Turing 1')

```
insert into Officina(nome, indirizzo)  
values ('MotorGo', 'piazza Turing 1')
```

oppure

```
insert into Officina(indirizzo, nome)  
values ('piazza Turing 1', 'MotorGo')
```

Officina

| <u>nome</u> | indirizzo |
|-------------|----------------------|
| FixIt | via delle Spighe 4 |
| CarFix | via delle Betulle 32 |
| MotorGo | piazza Turing 1 |

Cancellazione di Ennuple

L'istruzione per cancellare ennuple in una tabella è **delete**

delete from tabella
where condizione

Vengono cancellate tutte le ennuple di tabella che soddisfano condizione

Esempio

| Officina | |
|-------------|----------------------|
| <u>nome</u> | indirizzo |
| FixIt | via delle Spighe 4 |
| CarFix | via delle Betulle 32 |
| MotorGo | piazza Turing 1 |

Per cancellare la ennupla ('MotorGo', 'piazza Turing 1')...

Cancellazione di Ennuple (2)

Per cancellare la ennupla ('MotorGo', 'piazza Turing 1'):

```
delete from Officina  
where nome = 'MotorGo'  
      and indirizzo = 'piazza Turing 1'
```

oppure, sfruttando il fatto che l'attributo **nome** è chiave

```
delete from Officina  
where nome = 'MotorGo'
```

Officina

| <u>nome</u> | indirizzo |
|-------------|----------------------|
| FixIt | via delle Spighe 4 |
| CarFix | via delle Betulle 32 |

Cancellazione di Ennuple (3)

Attenzione: omettere la clausola **where** equivale a scrivere **where true**
 \implies l'istruzione

delete from Officina

elimina tutte le ennuple della tabella Officina !

Modifica di Ennuple

L'istruzione per modificare ennuple in una tabella è **update**

update Tabella

set attributo1 = espr1 , ... , attributoN = esprN
[**where** condizione]

In **tutte** le ennuple di tabella che **soddisfano** condizione, i valori degli attributi attributo1 , ... , attributoN vengono **modificati** in, rispettivamente, espr1 , ... , esprN.

Esempio

| Officina | |
|-------------|----------------------|
| <u>nome</u> | indirizzo |
| FixIt | via delle Spighe 4 |
| CarFix | via delle Betulle 32 |
| MotorGo | piazza Turing 1 |

Per modificare il valore dell'attributo indirizzo dell'officina '**MotorGo**' in '**viale Einstein 25**'...

Modifica di Ennuple (2)

Per modificare il valore dell'attributo indirizzo dell'officina 'MotorGo' in 'viale Einstein 25'...

```
update Officina  
  set indirizzo = 'viale Einstein 25'  
where nome = 'MotorGo'  
  and indirizzo = 'piazza Turing 1'
```

oppure, sfruttando il fatto che l'attributo **nome** è chiave

```
update Officina  
  set indirizzo = 'viale Einstein 25'  
where nome = 'MotorGo'
```

Officina

| <u>nome</u> | indirizzo |
|-------------|----------------------|
| FixIt | via delle Spighe 4 |
| CarFix | via delle Betulle 32 |
| MotorGo | viale Einstein 25 |

Modifica di Ennuple (3)

Attenzione: omettere la clausola **where** equivale a scrivere **where true**
⇒ l'istruzione

```
update Officina  
  set indirizzo = 'viale Einstein 25'
```

modifica il valore dell'attributo indirizzo di **tutte le ennuple** della tabella Officina !

Costrutti Avanzati

SQL mette a disposizione costrutti più avanzati per l'inserimento, la cancellazione e la modifica dei dati.

Vedremo alcuni di questi costrutti più avanti.

Per gli altri, consultare la documentazione dello standard SQL e quella del DBMS in uso.



SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ing. dell'Informazione, Informatica e Statistica
Laurea in Informatica

Basi di Dati, Modulo 2

Prof. Toni Mancini
Dipartimento di Informatica
<http://tmancini.di.uniroma1.it>

Slides B.2.2.3.2 (S.B.2.2.3.2)

Basi di dati
Sistemi di Gestione di Basi di Dati Relazionali
Il Linguaggio SQL
Data Manipulation Language
Interrogazioni



SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ing. dell'Informazione, Informatica e Statistica
Laurea in Informatica

Basi di Dati, Modulo 2

Prof. Toni Mancini
Dipartimento di Informatica
<http://tmancini.di.uniroma1.it>

Slides B.2.2.3.2.1 (S.B.2.2.3.2.1)

Basi di dati
Sistemi di Gestione di Basi di Dati Relazionali
Il Linguaggio SQL
Data Manipulation Language
Interrogazioni
Interrogazioni su Singola Tabella

SQL Data Manipulation Language: interrogazioni

- ▶ L'istruzione di interrogazione è **select**
- ▶ restituisce il risultato in forma di tabella (che potrebbe **non** rappresentare una relazione!)

Interrogazioni su una singola tabella

```
select  tabella.attributo1 ,      — target list
        tabella.attributo2 ,
        ... ,
        tabella.attributoN
from    tabella                  — clausola from
where    condizione              — clausola where
```

Interrogazioni su singola tabella

Esempio: restituire l'indirizzo dell'officina "FixIt"

| Officina | |
|-------------|----------------------|
| <u>nome</u> | indirizzo |
| FixIt | via delle Spighe 4 |
| CarFix | via delle Betulle 32 |
| MotorGo | piazza Turing 1 |

$$\pi_{\text{indirizzo}} (\sigma_{\text{nome}='FixIt'} (\text{Officina}))$$

```

select Officina.indirizzo
from Officina
where Officina.nome = '
      FixIt '
  
```

| Risultato |
|--------------------|
| Officina.indirizzo |
| via delle Spighe 4 |

Interrogazioni su singola tabella (2)

Esempio: restituire gli stipendi e le età delle persone con più di 40 anni e stipendio minore di 45 kEuro/anno

Persona

| cognome | nome | eta | stipendio |
|---------|--------|-----|-----------|
| Rossi | Mario | 47 | 36 |
| Verdi | Giulia | 35 | 36 |
| Bianchi | Anna | 55 | 56 |
| Rossi | Mario | 44 | 42 |

$\pi_{stipendio,eta}(\sigma_{eta > 40 \wedge stipendio < 45}(Persona))$

```
select Persona.stipendio , Persona.eta
from Persona
where Persona.eta > 40 and Persona.stipendio < 45
```


Interrogazioni su singola tabella (3)

```
select Persona.stipendio , Persona.eta  
from Persona  
where Persona.eta > 40 and Persona.stipendio < 45
```

Risultato

| stipendio | eta |
|-----------|-----|
| 36 | 47 |
| 42 | 44 |

Quando non c'è ambiguità nel nome di un attributo (ad es., quando la clausola **from** contiene una sola tabella), questo si può indicare senza farlo precedere dal nome della tabella

⇒ l'interrogazione precedente equivale a:

```
select stipendio , eta  
from Persona  
where eta > 40 and stipendio < 45
```

Select distinct

La tabella restituita da una istruzione **select** potrebbe **non** rappresentare una relazione

Esempio: restituire cognome e nome delle persone che hanno più di 40 anni

Persona

| cognome | nome | eta | stipendio |
|---------|--------|-----|-----------|
| Rossi | Mario | 47 | 36 |
| Verdi | Giulia | 35 | 36 |
| Bianchi | Anna | 55 | 56 |
| Rossi | Mario | 44 | 42 |

$\pi_{cognome, nome} (\sigma_{eta > 40} (Persona))$

```
select cognome , nome
from    Persona
where   eta > 40
```

Select distinct (2)

```
select cognome , nome  
from Persona  
where eta > 40
```

Risultato

| cognome | nome |
|---------|-------|
| Rossi | Mario |
| Bianchi | Anna |
| Rossi | Mario |

```
select distinct cognome , nome  
from Persona  
where eta > 40
```

Risultato

| cognome | nome |
|---------|-------|
| Rossi | Mario |
| Bianchi | Anna |

Select *

Esempio: restituire tutti i dati (cognome, nome, età e stipendio) delle persone che hanno più di 40 anni

Persona

| cognome | nome | età | stipendio |
|---------|--------|-----|-----------|
| Rossi | Mario | 47 | 36 |
| Verdi | Giulia | 35 | 36 |
| Bianchi | Anna | 55 | 56 |
| Rossi | Mario | 44 | 42 |

$$\pi_{\text{cognome, nome, età, stipendio}} (\sigma_{\text{età} > 40} (\text{Persona})) = \sigma_{\text{età} > 40} (\text{Persona})$$

```
select cognome, nome, età, stipendio  
from Persona  
where età > 40
```

Select * (2)

Risultato

| cognome | nome | eta | stipendio |
|---------|-------|-----|-----------|
| Rossi | Mario | 47 | 36 |
| Bianchi | Anna | 55 | 56 |
| Rossi | Mario | 44 | 42 |

L'interrogazione è equivalente a:

```
select *  
from   Persona  
where  eta > 40
```

Select senza where

Se la condizione della clausola **where** è true, la clausola può essere omessa del tutto

Esempio: nome e cognome di tutte le persone

```
select cognome , nome  
from Persona
```

che è equivalente a:

```
select cognome , nome  
from Persona  
where true
```

Condizione like

Esempio: restituire i dati delle persone che hanno un cognome che inizia per 'R'

```
select *  
from Persona  
where cognome like 'R%'
```

- ▶ '%': qualunque stringa
- ▶ '_': qualunque carattere

Esempio: **where** cognome **like** 'R_s%'

⇒ cognome che inizia per 'R' ed ha 's' come terzo carattere (ad es., Rossi)

Condizioni 'is null' e 'is not null'

Esempio: restituire i dati delle persone che hanno più di 40 anni o di cui non si conosce l'età

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

Esempio: restituire i dati delle persone di cui si conosce l'età

```
select *  
from Persona  
where eta is not null
```




SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ing. dell'Informazione, Informatica e Statistica
Laurea in Informatica

Basi di Dati, Modulo 2

Prof. Toni Mancini
Dipartimento di Informatica
<http://tmancini.di.uniroma1.it>

Slides B.2.2.3.2.2 (S.B.2.2.3.2.2)

Basi di dati
Sistemi di Gestione di Basi di Dati Relazionali
Il Linguaggio SQL
Data Manipulation Language
Interrogazioni
Interrogazioni su Tabelle Multiple

Interrogazioni su più tabelle

Esempio: restituire gli indirizzi delle officine dove è stato riparato il veicolo di targa 'HK 243 BW'

| Officina | | Riparazione | | |
|-------------|----------------------|-----------------|---------------|-----------|
| <u>nome</u> | indirizzo | <u>officina</u> | <u>codice</u> | veicolo |
| FixIt | via delle Spighe 4 | FixIt | 1 | HK 243 BW |
| CarFix | via delle Betulle 32 | CarFix | 1 | AA 662 XQ |
| MotorGo | piazza Turing 1 | MotorGo | 2 | HK 243 BW |

$$\begin{aligned}
 &\pi_{\text{indirizzo}} \left(\sigma_{\text{veicolo}='HK 243 BW'} \left(\text{Officina} \bowtie_{\text{nome=officina}} \text{Riparazione} \right) \right) = \\
 &\pi_{\text{indirizzo}} \left(\text{Officina} \bowtie_{\text{nome=officina}} \left(\sigma_{\text{veicolo}='HK 243 BW'} \left(\text{Riparazione} \right) \right) \right)
 \end{aligned}$$

Interrogazioni su più tabelle

Esempio: restituire gli indirizzi delle officine dove è stato riparato il veicolo di targa 'HK 243 BW'

| Officina | | Riparazione | | |
|-------------|----------------------|-----------------|---------------|-----------|
| <u>nome</u> | indirizzo | <u>officina</u> | <u>codice</u> | veicolo |
| FixIt | via delle Spighe 4 | FixIt | 1 | HK 243 BW |
| CarFix | via delle Betulle 32 | CarFix | 1 | AA 662 XQ |
| MotorGo | piazza Turing 1 | MotorGo | 2 | HK 243 BW |

```

select Officina.indirizzo
from Officina , Riparazione
where Officina.nome = Riparazione.officina
and Riparazione.veicolo = 'HK 243 BW'

```

Interrogazioni su più tabelle

Esempio: restituire gli indirizzi delle officine dove è stato riparato il veicolo di targa 'HK 243 BW'

| Officina | | Riparazione | | |
|-------------|----------------------|-----------------|---------------|-----------|
| <u>nome</u> | indirizzo | <u>officina</u> | <u>codice</u> | veicolo |
| FixIt | via delle Spighe 4 | FixIt | 1 | HK 243 BW |
| CarFix | via delle Betulle 32 | CarFix | 1 | AA 662 XQ |
| MotorGo | piazza Turing 1 | MotorGo | 2 | HK 243 BW |

```

select Officina.indirizzo
from Officina , Riparazione
where Officina.nome = Riparazione.officina
and Riparazione.veicolo = 'HK 243 BW'
  
```

Risultato

| Officina.indirizzo |
|--------------------|
| via delle Spighe 4 |
| piazza Turing 1 |

Alias di tabelle

```
select Officina.indirizzo  
from Officina , Riparazione  
where Officina.nome = Riparazione.officina  
and Riparazione.veicolo = 'HK 243 BW'
```

può essere riscritta come:

```
select o.indirizzo  
from Officina as o, Riparazione as r  
where o.nome = r.officina  
and r.veicolo = 'HK 243 BW'
```

oppure:

```
select o.indirizzo  
from Officina o, Riparazione r  
where o.nome = r.officina  
and r.veicolo = 'HK 243 BW'
```

Occorrenze multiple di una tabella

Esempio: restituire le targhe dei veicoli che sono stati riparati in almeno due diverse officine

Riparazione

| <u>officina</u> | <u>codice</u> | veicolo |
|-----------------|---------------|-----------|
| FixIt | 1 | HK 243 BW |
| CarFix | 1 | AA 662 XQ |
| MotorGo | 2 | HK 243 BW |

$$\pi_{v1}(\rho_{officina,veicolo \rightarrow o1,v1} (Riparazione))$$

$$\bowtie_{v1=v2 \wedge o1 \neq o2}$$

$$\rho_{officina,veicolo \rightarrow o2,v2} (Riparazione))$$

```

select distinct r1.veicolo as targa
from Riparazione as r1, Riparazione as r2
where r1.veicolo = r2.veicolo
      and r1.officina <> r2.officina
  
```

Nota: si osservi anche **select** attributo **as** nuovo_nome

Select SQL e algebra relazionale

In generale, una interrogazione SQL del tipo:

```
select attr1 , attr2 , ... , attrN  
from Tabella1 , ... , TabellaM  
where condizione
```

è (quasi) equivalente alla seguente espressione in algebra relazionale:

$$\pi_{attr1, \dots, attrN} (\sigma_{condizione} (Tabella1 \times \dots \times TabellaM))$$

Il risultato della interrogazione SQL potrebbe avere ennuple **duplicate**

\Rightarrow **select distinct** per eliminarli, ma **costoso**

Select SQL e algebra relazionale (2)

La semantica di una interrogazione SQL del tipo

select ... **from** ... **where** è data in termini di **prodotto cartesiano** tra tabelle

Questo **non** significa che il DBMS esegua davvero il prodotto cartesiano (molto costoso)

Prima di essere eseguita, una interrogazione SQL viene pre-processata dal DBMS che ne calcola il **piano di esecuzione**, riscrivendola in modo più **efficiente**

- ▶ esecuzione di selezioni (σ , ovvero parti della clausola **where**) e proiezioni (π , parti della clausola **select**) il più presto possibile
- ▶ esecuzione di **join** invece che dei prodotti cartesiani

L'ottimizzazione di interrogazioni è un argomento abbastanza complesso, che verrà presentato in corsi più avanzati

Per l'utente: privilegiare la **chiarezza** e la **leggibilità**

Select ... order by

Il risultato di una interrogazione SQL può essere **ordinato** (a proposito di scostamenti con il modello relazionale)

```
select *  
from Officina  
where ...  
order by nome asc
```

Clausola **order by** attr1 **asc|desc**, ..., attrN **asc|desc**



SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ing. dell'Informazione, Informatica e Statistica
Laurea in Informatica

Basi di Dati, Modulo 2

Prof. Toni Mancini
Dipartimento di Informatica
<http://tmancini.di.uniroma1.it>

Slides B.2.2.3.2.3 (S.B.2.2.3.2.3)

Basi di dati
Sistemi di Gestione di Basi di Dati Relazionali
Il Linguaggio SQL
Data Manipulation Language
Interrogazioni
Funzioni Aggregate

Funzioni Aggregate nella Target List: Count

```
select target_list from ... where ...
```

Nelle espressioni della target_list possiamo avere **funzioni aggregate**

Calcolano **un singolo valore** a partire da **tutte** le ennuple

⇒ il risultato è una tabella con **una sola ennupla**

Esempio:

Restituire l'elenco delle riparazioni del veicolo 'HK 243 BW':

```
select *
from Riparazione
where veicolo = 'HK 243 BW'
```

| officina | codice | veicolo |
|----------|--------|-----------|
| FixIt | 1 | HK 243 BW |
| FixIt | 2 | HK 243 BW |

Restituire il **numero** di riparazioni del veicolo 'HK 243 BW':

```
select count(*)
from Riparazione
where veicolo = 'HK 243 BW'
```

| count(*) |
|----------|
| 2 |

Funzioni Aggregate nella Target List: Count (2)

Esempio:

Restituire l'elenco di officine che hanno riparato il veicolo 'HK 243 BW':

```
select officina  
from Riparazione  
where veicolo = 'HK 243 BW'
```

| officina |
|----------|
| Fixlt |
| Fixlt |

Restituire la **lunghezza** dell'elenco di officine che hanno riparato il veicolo 'HK 243 BW' (non ha molto senso...):

```
select count(officina)  
from Riparazione  
where veicolo = 'HK 243 BW'
```

| count(officina) |
|-----------------|
| 2 |

Funzioni Aggregate nella Target List: Count (3)

Esempio:

Restituire l'elenco delle officine **distinte** che hanno riparato il veicolo 'HK 243 BW':

```
select distinct officina  
from Riparazione  
where veicolo = 'HK 243 BW'
```

| officina |
|----------|
| FixIt |

Restituire il numero di officine (distinte) che hanno riparato il veicolo 'HK 243 BW':

```
select count(distinct officina)  
from Riparazione  
where veicolo = 'HK 243 BW'
```

| count(officina) |
|-----------------|
| 1 |

Funzioni Aggregate nella Target List: Count (4)

- ▶ **count**(*): numero di ennuple
- ▶ **count**(attributo): numero di valori non **NULL** per l'attributo (con duplicati)
- ▶ **count**(**distinct** attributo): numero di valori non **NULL** e **distinti** per l'attributo

Funzioni Aggregate nella Target List: Count (5)

Esempio

| Persona | | | |
|---------|--------|-----|-----------|
| cognome | nome | eta | stipendio |
| Rossi | Mario | 47 | 36 |
| Verdi | Giulia | 35 | NULL |
| Bianchi | Anna | 55 | 56 |
| Rossi | Mario | 44 | 42 |
| Verdi | Giulia | 35 | 56 |

```
select count(*)
from Persona
```

 count(*)

 5

```
select count(stipendio)
from Persona
```

 count(stipendio)

 4

```
select
    count(distinct stipendio)
from Persona
```

 count(distinct stipendio)

 3

Altre Funzioni Aggregate

- ▶ **sum**(attributo): somma su domini numerici o tempo
- ▶ **avg**(attributo): valore medio su domini numerici o tempo
- ▶ **min**(attributo): valore minimo su domini ordinati
- ▶ **max**(attributo): valore massimo su domini ordinati

I valori **NULL** sono **ignorati**

Altre Funzioni Aggregate (2)

Esempio:

Persona

| cognome | nome | eta | stipendio |
|---------|--------|-----|-----------|
| Rossi | Mario | 47 | 36 |
| Verdi | Giulia | 35 | NULL |
| Bianchi | Anna | 55 | 56 |
| Rossi | Mario | 44 | 42 |
| Verdi | Giulia | 35 | 56 |

select

count(eta), **sum**(eta), **avg**(eta), **min**(eta), **max**(eta),

count(stipendio), **sum**(stipendio), **avg**(stipendio),
min(stipendio), **max**(stipendio)

from Persona

| count(eta) | sum(eta) | avg(eta) | min(eta) | max(eta) | count(stipendio) | sum(stipendio) | avg(stipendio) | min(stipendio) | max(stipendio) |
|------------|----------|----------|----------|----------|------------------|----------------|----------------|----------------|----------------|
| 5 | 216 | 43.2 | 35 | 55 | 4 | 190 | 47.5 | 36 | 56 |

Funzioni Aggregate: Omogeneità nella Target List

| Persona | | | |
|---------|--------|-----|-----------|
| cognome | nome | eta | stipendio |
| Rossi | Mario | 47 | 36 |
| Verdi | Giulia | 35 | NULL |
| Bianchi | Anna | 55 | 56 |
| Rossi | Mario | 44 | 42 |
| Verdi | Giulia | 35 | 56 |

```
select nome, avg(stipendio)
from Persona
```

Errore: di chi sarebbe il **nome**?

La target-list di una interrogazione **select** ... **from** ... **where** ... **deve** essere **omogenea**: se contiene funzioni aggregate non può contenere attributi e viceversa (con una eccezione, v. seguito).



SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ing. dell'Informazione, Informatica e Statistica
Laurea in Informatica

Basi di Dati, Modulo 2

Prof. Toni Mancini
Dipartimento di Informatica
<http://tmancini.di.uniroma1.it>

Slides B.2.2.3.2.4 (S.B.2.2.3.2.4)

Basi di dati
Sistemi di Gestione di Basi di Dati Relazionali
Il Linguaggio SQL
Data Manipulation Language
Interrogazioni
Raggruppamenti

Funzioni Aggregate e Raggruppamenti: Group by

Le funzioni aggregate possono essere applicate a **partizioni** delle ennuple:

| Persona | | | GenFiglio | |
|-----------|---------|-----------|------------|---------------|
| <u>id</u> | nome | stipendio | <u>gen</u> | <u>figlio</u> |
| 1001 | Luca | 45 | 1001 | 1005 |
| 1002 | Anna | 48 | 1001 | 1003 |
| 1003 | Giulia | 35 | 1004 | 1003 |
| 1004 | Maria | 45 | 1002 | 1005 |
| 1005 | Antonio | 48 | | |

Restituire i nomi delle persone con figli e con stipendio ≥ 45 , con i nomi dei relativi figli:

```
select g.id as gid, g.nome as genitore, f.nome as figlio
from Persona g, GenFiglio gf, Persona f
where g.id = gf.gen and gf.figlio = f.id and g.stipendio >= 45
```

| Risultato | | |
|-----------|----------|---------|
| gid | genitore | figlio |
| 1001 | Luca | Antonio |
| 1001 | Luca | Giulia |
| 1002 | Anna | Antonio |
| 1004 | Maria | Giulia |

Funzioni Aggregate e Raggruppamenti: Group by

Le funzioni aggregate possono essere applicate a **partizioni** delle ennuple:

```
select g.id as gid, g.nome as genitore, count(f.nome) as nFigli
from Persona g, GenFiglio gf, Persona f
where g.id = gf.gen and gf.figlio = f.id and g.stipendio >= 45
group by g.id, g.nome
```

Risultato

| gid | genitore | figlio | | gid | genitore | nFigli |
|------|----------|---------|---|------|----------|--------|
| 1001 | Luca | Antonio | ⇒ | 1001 | Luca | 2 |
| 1001 | Luca | Giulia | | 1002 | Anna | 1 |
| 1002 | Anna | Antonio | | 1004 | Maria | 1 |
| 1004 | Maria | Giulia | | | | |

Restituire i nomi delle persone con figli e con stipendio ≥ 45 , insieme al **numero** dei loro figli.

Nota: gli attributi nella target list **devono** comparire tra gli attributi della clausola **group by**

Semantica delle interrogazioni con **group by** ed aggregati

```
select attrA, attrB, aggr (...) , ..., aggrN (...)  
from Tabella1, ..., TabellaM  
where condizione  
group by attrA, attrB, attrC, ..., attrG
```

1. Si esegue l'interrogazione
select * **from** Tabella1, ..., TabellaM **where** condizione
2. Si partizionano le ennuple risultanti mettendo nello stesso gruppo quelle che coincidono nei valori di tutti gli attributi nella clausola **group by**
3. Per ogni gruppo si calcolano indipendentemente i valori delle funzioni aggregate
4. Si restituisce una ennupla per ogni gruppo, con i valori di (alcuni tra) gli attributi della clausola **group by** e i valori delle funzioni aggregate per il singolo gruppo ...

Semantica delle interrogazioni con **group by** ed aggregati

```
select attrA, attrB, aggr (...) , ..., aggrN (...)  
from Tabella1, ..., TabellaM  
where condizione  
group by attrA, attrB, attrC, ..., attrG  
having condizione_sui_gruppi
```

1. Si esegue l'interrogazione
select * **from** Tabella1, ..., TabellaM **where** condizione
2. Si partizionano le ennuple risultanti mettendo nello stesso gruppo quelle che coincidono nei valori di tutti gli attributi nella clausola **group by**
3. Per ogni gruppo si calcolano indipendentemente i valori delle funzioni aggregate
4. Si restituisce una ennupla per ogni gruppo, con i valori di (alcuni tra) gli attributi della clausola **group by** e i valori delle funzioni aggregate per il singolo gruppo ...
5. ...omettendo le ennuple per i gruppi che non soddisfano la condizione **having**

Group by e condizione having

La condizione **having** esprime una condizione sui **gruppi**
⇒ può contenere **funzioni aggregate**

Esempio: Restituire i nomi delle persone con stipendio ≥ 45 ed almeno 2 figli, insieme al **numero** dei figli.

```
select g.id as gid, g.nome as genitore ,  
       count(f.nome) as nFigli  
from Persona g, GenFiglio gf, Persona f  
where g.id = gf.gen and gf.figlio = f.id  
       and g.stipendio >= 45  
group by g.id, g.nome  
having count(f.nome) >= 2
```

Nota 1: Non è possibile utilizzare nella clausola **having** gli **alias** della target list.

Nota 2: Per maggiore robustezza, è preferibile usare **count(f.id)** invece di **count(f.nome)** (id è chiave primaria di Persona, quindi non può avere valori **NULL**): questo permetterebbe di contare anche i figli con nome **NULL**.



SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ing. dell'Informazione, Informatica e Statistica
Laurea in Informatica

Basi di Dati, Modulo 2

Prof. Toni Mancini
Dipartimento di Informatica
<http://tmancini.di.uniroma1.it>

Slides B.2.2.3.2.5 (S.B.2.2.3.2.5)

Basi di dati
Sistemi di Gestione di Basi di Dati Relazionali
Il Linguaggio SQL
Data Manipulation Language
Interrogazioni
Operatori Insiemistici

Operatori insiemistici: unione

```
queryA  
union [ all | distinct ]  
queryB
```

Restituisce l'**unione** delle ennuple restituite da **queryA** e **queryB**

- ▶ **union distinct**: elimina i duplicati (**default!!**)
- ▶ **union all**: mantiene i duplicati

Operatori insiemistici: unione (2)

Esempio:

| Persona | | | Maternita | | Paternita | |
|-------------|-----|---------|-----------|---------------|-----------|---------------|
| <u>nome</u> | eta | reddito | madre | <u>figlio</u> | padre | <u>figlio</u> |
| Andrea | 27 | 21 | Luisa | Maria | Sergio | Franco |
| Aldo | 25 | 15 | Luisa | Luigi | Luigi | Olga |
| Maria | 55 | 42 | Anna | Olga | Luigi | Filippo |
| Anna | 50 | 35 | Anna | Filippo | Franco | Andrea |
| Filippo | 26 | 30 | Maria | Andrea | Franco | Aldo |
| Luigi | 50 | 40 | Maria | Aldo | | |
| Franco | 60 | 20 | | | | |
| Olga | 30 | 41 | | | | |
| Sergio | 85 | 35 | | | | |
| Luisa | 75 | 87 | | | | |

Operatori insiemistici: unione (3)

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

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

Nomi degli attributi del risultato dal **primo** operando

Operatori insiemistici: unione (4)

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

| genitore | figlio |
|----------|---------|
| Sergio | Franco |
| Luigi | Olga |
| Luigi | Filippo |
| Franco | Andrea |
| Franco | Aldo |
| Maria | Luisa |
| Luigi | Luisa |
| Olga | Anna |
| Filippo | Anna |
| Andrea | Maria |
| Aldo | Maria |

Notazione **posizionale**

Operatori insiemistici: differenza

| | | |
|---------------|--------|--------|
| queryA | | queryA |
| except | oppure | minus |
| queryB | | queryB |

Restituisce le ennuple di queryA che **non** sono in queryB

Esempio:

```
select nome
from Impiegato
minus
select cognome
from Impiegato
```

Operatori insiemistici: intersezione

queryA
intersect
queryB

Restituisce le ennuple comuni a queryA e queryB
⇒ può essere riscritta in modo più efficiente utilizzando un **join**

Esempio:

| | | |
|-----------------------|------------|--------------------------|
| select nome | | select i.nome |
| from Impiegato | | from Impiegato i, |
| intersect | equivale a | Impiegato j |
| select cognome | | where i.nome = j. |
| from Impiegato | | cognome |



SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ing. dell'Informazione, Informatica e Statistica
Laurea in Informatica

Basi di Dati, Modulo 2

Prof. Toni Mancini
Dipartimento di Informatica
<http://tmancini.di.uniroma1.it>

Slides B.2.2.3.2.6 (S.B.2.2.3.2.6)

Basi di dati
Sistemi di Gestione di Basi di Dati Relazionali
Il Linguaggio SQL
Data Manipulation Language
Interrogazioni
Interrogazioni nella Clausola From

Query nella clausola from

La clausola **from** può contenere sotto-query

```
select ...  
from Persona p,  
    (select ... as nome ...  
     from ...  
     where ...  
     group by ...  
     having ...) q  
where p.nome = q.nome and ...
```

Semantica: valutazione **stratificata**

1. viene **calcolata** la query più interna e **mantenuta** in una tabella temporanea
2. viene calcolata la query più esterna trattando q come se fosse una **normale** relazione
3. viene cancellata la tabella temporanea creata al punto 1.

Esempio

| Persona | | | Maternita | | Paternita | |
|-------------|-----|---------|-----------|---------------|-----------|---------------|
| <u>nome</u> | eta | reddito | madre | <u>figlio</u> | padre | <u>figlio</u> |

Restituire la media (sulle madri) dei redditi totali dei loro figli:

```
select avg(q.tot_reddito_figli) as
    avg_reddito_totale_figli
from (select m.nome as madre,
            sum(f.reddito) as tot_reddito_figli
      from Maternita m, Persona f
     where m.figlio = f.nome
    group by m.nome
) as q
```



SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ing. dell'Informazione, Informatica e Statistica
Laurea in Informatica

Basi di Dati, Modulo 2

Prof. Toni Mancini
Dipartimento di Informatica
<http://tmancini.di.uniroma1.it>

Slides B.2.2.3.2.7 (S.B.2.2.3.2.7)

Basi di dati
Sistemi di Gestione di Basi di Dati Relazionali
Il Linguaggio SQL
Data Manipulation Language
Interrogazioni
Interrogazioni Annidate

Query Annidate (nested)

La clausola where può contenere **condizioni** su **sotto-query**:

- ▶ attr = (**select** aggregato() **from** ...)
- ▶ attr > (**select** aggregato() **from** ...)
- ▶ attr >= (**select** aggregato() **from** ...)
- ▶ attr < (**select** aggregato() **from** ...)
- ▶ attr <= (**select** aggregato() **from** ...)
- ▶ attr <> (**select** aggregato() **from** ...)
- ▶ attr =**any** (**select** ...)
oppure attr **in** (**select** ...)
- ▶ attr =**all** (**select** ...)
- ▶ attr <**any** (**select** ...)
- ▶ attr <=**any** (**select** ...)
- ▶ attr >**any** (**select** ...)
- ▶ attr >=**any** (**select** ...)
- ▶ attr <**all** (**select** ...)
- ▶ attr <=**all** (**select** ...)
- ▶ attr >**all** (**select** ...)
- ▶ attr >=**all** (**select** ...)
- ▶ attr <>**all** (**select** ...)
oppure attr **not in** (**select** ...)
- ▶ **exists** (**select** ...)
- ▶ **not exists** (**select** ...)

Esempio 1

Restituire nome e reddito dei padri di persone con reddito > 20:

| Persona | | |
|-------------|-----|---------|
| <u>nome</u> | eta | reddito |

| Paternita | |
|-----------|---------------|
| padre | <u>figlio</u> |

```
select distinct p.nome, p.reddito
from Persona p, Paternita pat, Persona f
where p.nome = pat.padre and f.nome = pat.figlio
      and f.reddito > 20
```

oppure

```
select distinct nome, reddito
from Persona p
where p.nome = any (select pat.padre
                    from Paternita pat, Persona f
                    where f.nome = pat.figlio
                      and f.reddito > 20
                    )
```

distinct lo poteva mettere anche qua

Esempio 2

Restituire i dati delle persone con reddito maggiore del reddito di tutte le persone di età < 30

| Persona | | |
|-------------|-----|---------|
| <u>nome</u> | eta | reddito |

```
select nome
from Persona p
where p.reddito > all
      (select reddito from Persona where eta < 30)
```

oppure

```
select nome
from Persona p
where p.reddito >
      (select max(reddito) from Persona where eta <
        30)
```

Esempio 3

Restituire i dati delle persone con almeno un figlio

| Persona | | |
|-------------|-----|---------|
| <u>nome</u> | eta | reddito |

| Paternita | |
|-----------|---------------|
| padre | <u>figlio</u> |

| Maternita | |
|-----------|---------------|
| madre | <u>figlio</u> |

```
select *  
from Persona [p]  
where exists (select *  
              from Paternita  
              where padre = [p.nome])  
or exists (select *  
          from Maternita  
          where madre = [p.nome])
```

Query annidata e **correlata**: l'attributo `[p.nome]` nella query annidata si riferisce alla relazione `[p]` nella clausola **from** della query più esterna

Query Annidate: Semantica

Semantica delle query nidificate:

Per ogni ennupla t definita dalla clausola **from** della query **esterna**:

1. esegui la sotto-query (che può dipendere da t)
2. usa il risultato della sotto-query per valutare se la ennupla t deve far parte del risultato.

Questa è la **semantica** delle query annidate, non l'algoritmo usato dal DBMS

Query Annidate: Efficienza

- ▶ Le query annidate possono porre **problemi di efficienza**: i DBMS **non sono bravi** ad ottimizzarle (in special modo quelle correlate).
- ▶ In particolar modo, i DBMS non sono in grado di ottimizzare a dovere le query annidate e **correlate**.
- ▶ Le query annidate (soprattutto se correlate) andrebbero quindi **evitate** il più possibile, anche se talvolta sono più leggibili.

Query Annidate: Limitazioni

Le sotto-query di una query annidata devono rispettare alcune limitazioni:

- ▶ Le sotto-query non possono contenere operatori insiemistici (**union**, **intersect**, **except** o minus)
- ▶ Sebbene una sotto-query può far riferimento a variabili definite in blocchi più esterni (interrogazioni annidate e **correlate**), non è possibile, in una query, fare riferimento a variabili definite in blocchi più interni (ovviamente!).

Esempio 4

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

Persona

| | | |
|-------------|-----|---------|
| <u>nome</u> | eta | reddito |
|-------------|-----|---------|

Paternita

| | |
|-------|---------------|
| padre | <u>figlio</u> |
|-------|---------------|

Maternita

| | |
|-------|---------------|
| madre | <u>figlio</u> |
|-------|---------------|

```
select nome, eta
from Persona [p], Maternita mat
where p.nome = mat.madre
      and mat.figlio in (select q.nome
                        from Persona q
                        where [p.eta] - q.eta < 20)
```

Esempio 5

Restituire i dati delle persone con il reddito più alto.

| Persona | | |
|-------------|-----|---------|
| <u>nome</u> | eta | reddito |

| Paternita | |
|-----------|---------------|
| padre | <u>figlio</u> |

| Maternita | |
|-----------|---------------|
| madre | <u>figlio</u> |

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

oppure

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

Esempio 6

Restituire i dati delle persone che hanno la coppia (età, reddito) diversa da tutte le altre.

Persona

| <u>nome</u> | eta | reddito |
|-------------|-----|---------|
|-------------|-----|---------|

Paternita

| padre | <u>figlio</u> |
|-------|---------------|
|-------|---------------|

Maternita

| madre | <u>figlio</u> |
|-------|---------------|
|-------|---------------|

```

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



SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ing. dell'Informazione, Informatica e Statistica
Laurea in Informatica

Basi di Dati, Modulo 2

Prof. Toni Mancini
Dipartimento di Informatica
<http://tmancini.di.uniroma1.it>

Slides B.2.2.3.2.8 (S.B.2.2.3.2.8)

Basi di dati
Sistemi di Gestione di Basi di Dati Relazionali
Il Linguaggio SQL
Data Manipulation Language
Interrogazioni
Join Espliciti

Join Esplicito

SQL prevede una sintassi esplicita per l'operazione di **join**:

```
select ...  
from <Tabella A> join <Tabella B>  
      on <condizione di join>  
where <altre condizioni>
```

Equivalente a:

```
select ...  
from <Tabella A>, <Tabella B>  
where <condizione di join> and <altre condizioni>
```

Sintassi non molto usata: il modulo del DBMS dedicato all'ottimizzazione delle query riscrive automaticamente una query in termini di operazioni di **join**, quando possibile

Esempio 1

Restituire madre e padre di ogni persona.

| Persona | | | Paternita | | Maternita | |
|-------------|-----|---------|-----------|---------------|-----------|---------------|
| <u>nome</u> | eta | reddito | padre | <u>figlio</u> | madre | <u>figlio</u> |

```
select mat.figlio as persona ,
       mat.madre as madre, pat.padre as padre
from Maternita mat, Paternita pat
where mat.figlio = pat.figlio
```

ma anche:

```
select mat.figlio as persona ,
       mat.madre as madre, pat.padre as padre
from Maternita mat join Paternita pat
     on mat.figlio = pat.figlio
```


Join Naturale

Esiste anche il costrutto **natural join** che effettua il join naturale.

Join naturale: **equi-join** su **attributi omonimi** di relazioni diverse.

```
select mat.figlio as persona ,  
        mat.madre as madre, pat.padre as padre  
from Maternita mat, Paternita pat  
where mat.figlio = pat.figlio
```

ma anche:

```
select mat.figlio as persona ,  
        mat.madre as madre, pat.padre as padre  
from Maternita mat join Paternita pat  
        on mat.figlio = pat.figlio
```

ma anche:

```
select mat.figlio as persona ,  
        mat.madre as madre, pat.padre as padre  
from Maternita mat natural join Paternita pat
```



SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ing. dell'Informazione, Informatica e Statistica
Laurea in Informatica

Basi di Dati, Modulo 2

Prof. Toni Mancini
Dipartimento di Informatica
<http://tmancini.di.uniroma1.it>

Slides B.2.2.3.2.9 (S.B.2.2.3.2.9)

Basi di dati
Sistemi di Gestione di Basi di Dati Relazionali
Il Linguaggio SQL
Data Manipulation Language
Interrogazioni
Outer Join

Outer Join: Motivazione

In un **join** tra due relazioni R_1 e R_2 :

- ▶ Una ennupla r_1 di R_1 partecipa al risultato solo se esiste una ennupla r_2 di R_2 per cui (r_1, r_2) soddisfa la condizione di join.
- ▶ Una ennupla r_2 di R_2 partecipa al risultato solo se esiste una ennupla r_1 di R_1 per cui (r_1, r_2) soddisfa la condizione di join.

Esempio: restituire madre e padre di ogni persona.

| Persona | | | Paternita | | Maternita | |
|-------------|-----|---------|-----------|---------------|-----------|---------------|
| <u>nome</u> | eta | reddito | padre | <u>figlio</u> | madre | <u>figlio</u> |

```

select mat.figlio as persona ,
mat.madre as madre , pat.padre as padre
from Maternita mat join Paternita pat
on mat.figlio = pat.figlio
  
```

Una ennupla **mat** di **Maternita** partecipa al risultato in una coppia **(mat, pat)** per ogni ennupla **pat** di **Paternita** per cui **(mat, pat)** soddisfa "mat. figlio = pat. figlio".

⇒ Una persona **senza padre** o **senza madre** noti **non** farà parte del risultato.

Left Outer Join

Esempio: restituire madre e, *se noto*, padre di ogni persona.

| Persona | | | Paternita | | Maternita | |
|-------------|-----|---------|-----------|---------------|-----------|---------------|
| <u>nome</u> | eta | reddito | padre | <u>figlio</u> | madre | <u>figlio</u> |

```
select mat.figlio as persona ,
mat.madre as madre , pat.padre as padre
from Maternita mat left outer join Paternita pat
on mat.figlio = pat.figlio
```

In un *left outer join*, *tutte* le ennuple di *Maternita* (relazione a *sinistra*) partecipano al risultato. Per ogni ennupla *mat* di *Maternita*:

- ▶ *mat* partecipa al risultato in una coppia (*mat*, *pat*) per ogni ennupla *pat* di *Paternita* per cui (*mat*, *pat*) soddisfa la condizione di join.
- ▶ Inoltre, *se* non esiste alcuna ennupla *pat* di *Paternita* per cui (*mat*, *pat*) soddisfa la condizione di join, allora *mat* partecipa al risultato sotto forma di (*mat*, **NULL**, ..., **NULL**).

Right Outer Join

Esempio: restituire padre e, *se nota*, madre di ogni persona.

| Persona | | | Paternita | | Maternita | |
|-------------|-----|---------|-----------|---------------|-----------|---------------|
| <u>nome</u> | eta | reddito | padre | <u>figlio</u> | madre | <u>figlio</u> |

```

select mat.figlio as persona ,
mat.madre as madre , pat.padre as padre
from Maternita mat right outer join Paternita pat
on mat.figlio = pat.figlio

```

In un **right outer join**, *tutte* le ennuple di **Paternita** (relazione a *destra*) partecipano al risultato. Per ogni ennupla **pat** di **Paternita**:

- ▶ **pat** partecipa al risultato in una coppia (**mat**, **pat**) per ogni ennupla **mat** di **Maternita** per cui (**mat**, **pat**) soddisfa la condizione di join.
- ▶ Inoltre, *se* non esiste alcuna ennupla **mat** di **Maternita** per cui (**mat**, **pat**) soddisfa la condizione di join, allora **pat** partecipa al risultato sotto forma di (**NULL**, ..., **NULL**, **pat**).

Full Outer Join

Esempio: restituire i genitori noti di ogni persona con almeno un genitore noto.

```
select *  
from Maternita mat full outer join Paternita pat  
on mat.figlio = pat.figlio
```

In un **full outer join**, **tutte** le ennuple di **Maternita** e tutte le ennuple di **Paternita** (**entrambe** le relazioni) partecipano al risultato:

- ▶ Ogni ennupla **mat** di **Maternita** partecipa al risultato in una coppia (**mat**, **pat**) per ogni ennupla **pat** di **Paternita** per cui (**mat**, **pat**) soddisfa la condizione di join.
- ▶ Ogni ennupla **pat** di **Paternita** partecipa al risultato in una coppia (**mat**, **pat**) per ogni ennupla **mat** di **Maternita** per cui (**mat**, **pat**) soddisfa la condizione di join.
- ▶ Le ennuple **mat** di **Maternita** che non sono state incluse, partecipano al risultato sotto forma di (**mat**, **NULL**, ..., **NULL**).
- ▶ Le ennuple **pat** di **Paternita** che non sono state incluse, partecipano al risultato sotto forma di (**NULL**, ..., **NULL**, **pat**).



SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ing. dell'Informazione, Informatica e Statistica
Laurea in Informatica

Basi di Dati, Modulo 2

Prof. Toni Mancini
Dipartimento di Informatica
<http://tmancini.di.uniroma1.it>

Slides B.2.2.3.3 (S.B.2.2.3.3)

Basi di dati
Sistemi di Gestione di Basi di Dati Relazionali
Il Linguaggio SQL
Data Manipulation Language
Espressioni

Espressioni nel Comando Select

La **target list** e la clausola **where** dell'istruzione **select** possono contenere espressioni.

Esempi:

```
select nome, reddito, reddito/12 as stipendioMese  
from Impiegato where eta-2 < 30*reddito
```

```
select eta, count(*)+max(reddito) as nonHaSenso  
from Impiegato group by eta
```

```
select initcap(nome),  
sqrt(eta+sin(reddito))-round(reddito/100) as nonHaSenso  
from Impiegato  
where eta < round(sqrt(850)+length(nome))
```


Espressioni nel Comando Update

Anche i comandi **update** possono contenere **espressioni**.

Esempio: Vogliamo incrementare di 1 il valore dell'attributo voto di tutte le ennuple della tabella **Esame** relative allo studente con matricola 1101 (stando attenti a non superare mai il valore 30):

Esame

| studente | corso | voto |
|----------|-------|------|
| 1101 | 590 | 28 |
| 1101 | 591 | 30 |
| 1103 | 593 | 25 |

```
update Esame  
set voto = least(voto+1, 30)  
where matricola = '1101'
```

Nota: la funzione SQL **least** (di arità variabile) restituisce il minimo tra i valori nella lista degli argomenti (la funzione **min** è una **funzione aggregata** e serve ad altro).

Espressioni nel Comando Update

Anche i comandi **update** possono contenere **espressioni**.

Esempio: Vogliamo incrementare di 1 il valore dell'attributo voto di tutte le ennuple della tabella **Esame** relative allo studente con matricola 1101 (stando attenti a non superare mai il valore 30):

| Esame | | | | | | Esame | | |
|----------|-------|------|--|--|--|----------|-------|------|
| studente | corso | voto | | | | studente | corso | voto |
| 1101 | 590 | 28 | | | | 1101 | 590 | 29 |
| 1101 | 591 | 30 | | | | 1101 | 591 | 30 |
| 1103 | 593 | 25 | | | | 1103 | 593 | 25 |

update Esame

set voto = least(voto+1, 30)

where matricola = '1101'

Nota: la funzione SQL least (di arità variabile) restituisce il minimo tra i valori nella lista degli argomenti (la funzione **min** è una **funzione aggregata** e serve ad altro).

Espressioni nel Comando Insert

Ovviamente anche i comandi **insert** possono contenere **espressioni**.

Esempio:

| Esame | | |
|----------|-------|------|
| studente | corso | voto |
| 1101 | 590 | 29 |
| 1101 | 591 | 30 |
| 1103 | 593 | 25 |

```
insert into Esame(studente , corso , voto)  
values ( '1101' , '593' , 25+2)
```

Espressioni nel Comando Insert

Ovviamente anche i comandi **insert** possono contenere **espressioni**.

Esempio:

| Esame | | | | | |
|----------|-------|------|--|--|--|
| studente | corso | voto | | | |
| 1101 | 590 | 29 | | | |
| 1101 | 591 | 30 | | | |
| 1103 | 593 | 25 | | | |

⇒

| Esame | | |
|----------|-------|------|
| studente | corso | voto |
| 1101 | 590 | 29 |
| 1101 | 591 | 30 |
| 1103 | 593 | 25 |
| 1101 | 593 | 27 |

```
insert into Esame(studente , corso , voto)
values ( '1101' , '593' , 25+2)
```

Funzioni Disponibili

- ▶ I DBMS mettono a disposizione **molte** funzioni, che possono essere usate nelle espressioni. In particolare, funzioni su:
 - ▶ stringhe
 - ▶ interi
 - ▶ reali
 - ▶ timestamp
 - ▶ etc.
- ▶ Gli utenti possono definire **ulteriori** funzioni.

⇒ Consultare il manuale del DBMS in uso.
(Gli esempi in questo corso sono relativi a PostgreSQL.)

Funzioni su stringhe che ritornano stringhe

Alcuni esempi:

(Consultare il manuale del DBMS per i dettagli e per una lista esaustiva)

- ▶ **upper**(string), **lower**(string), **initcap**(string)
Restituisce la stringa tutta in maiuscolo, tutta in minuscolo, solo con la prima lettera maiuscola rispettivamente.
- ▶ **lpad**(string,length,pad), **rpadd**(string,length,pad)
Restituisce la stringa spostata sulla sinistra o sulla destra fino a length caratteri usando il carattere di riempimento pad (default: ' ', ovvero singolo spazio).
- ▶ **ltrim**(string, trimlist), **rtrim**(string, trimlist)
Restituisce la stringa con la parte più a sinistra o più a destra che fa match con i caratteri in trimlist rimossi (default: singolo spazio).

Funzioni su stringhe che ritornano stringhe (2)

- ▶ **replace**(string, target, replacement)
Restituisce la stringa con tutte le occorrenze di target rimpiazzate con replacement. Se replacement è omissso, tutte le occorrenze di target sono cancellate.
- ▶ **substr**(string, pos, len)
Restituisce la sottostringa di string che comincia alla posizione pos ed è lunga len caratteri. Se pos è negativo, la posizione viene calcolata dalla fine di string. Il primo carattere della stringa ha posizione 1.
- ▶ **translate**(string, fromlist, tolist)
Restituisce la stringa con ogni carattere in fromlist rimpiazzato con il corrispondente carattere in tolist.

Funzioni su stringhe che ritornano stringhe (3)

► **to_char**(number, format)

Formatta il numero **number** secondo la stringa **format** (che deve essere tra singoli apici). Consultare il manuale del DBMS per informazioni sul formato.

Esempio:

select

```
to_char(1250000, '$9 999 999.99') as  
result1;
```

select

```
to_char(date('2013-05-21'), 'DD FMMonth  
YYYY')  
as result2;
```

| result1 |
|-----------------|
| \$ 1 250 000.00 |

| result2 |
|-------------|
| 21 May 2013 |

Funzioni su stringhe che restituiscono interi

Alcuni esempi:

(Consultare il manuale del DBMS per i dettagli e per una lista esaustiva)

- ▶ `position(string, substring)` (in PostgreSQL: `strpos`)
Restituisce la prima posizione della stringa `substring` in `string`. Se in `string` non vi è alcuna occorrenza di `substring`, allora restituisce 0.
- ▶ `length(string)`
Ritorna la lunghezza di `string`.

Funzioni matematiche

Alcuni esempi:

(Consultare il manuale del DBMS per i dettagli e per una lista esaustiva)

- | | | |
|---------------------------------|-----------------------------|----------------------------|
| ▶ <code>abs(value)</code> | ▶ <code>sqrt(value)</code> | ▶ <code>cos(value)</code> |
| ▶ <code>exp(value)</code> | ▶ <code>ceil(value)</code> | ▶ <code>sin(value)</code> |
| ▶ <code>log(value)</code> | ▶ <code>floor(value)</code> | ▶ <code>acos(value)</code> |
| ▶ <code>log(base, value)</code> | ▶ <code>round(value)</code> | ▶ ... |
| ▶ <code>power(base, exp)</code> | ▶ <code>trunc(value)</code> | |

Molte funzioni **non standard** ma implementate in molti DBMS.

Altre Funzioni Disponibili

Altre funzioni disponibili su PostgreSQL:

- ▶ funzioni per pattern matching ([similar](#) per espressioni regolari)
- ▶ funzioni per i tipi [datetime](#), [timestamp](#) etc.
- ▶ funzioni di supporto al tipo [enum](#)
- ▶ funzioni per tipi geometrici
- ▶ funzioni per la manipolazioni di [sequenze](#)
- ▶ funzioni aggregate statistiche
- ▶ ...

Consultare il manuale del DBMS per i dettagli e per una lista esaustiva.



SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ing. dell'Informazione, Informatica e Statistica
Laurea in Informatica

Basi di Dati, Modulo 2

Prof. Toni Mancini
Dipartimento di Informatica
<http://tmancini.di.uniroma1.it>

Slides B.2.2.3.4 (S.B.2.2.3.4)

Basi di dati
Sistemi di Gestione di Basi di Dati Relazionali
Il Linguaggio SQL
Data Manipulation Language
Inserimento, Cancellazione e Modifica (Comandi
Avanzati)

Inserimento da Query

Abbiamo già visto il costrutto:

```
insert into Tabella( attributi ) values (...)
```

per inserire una ennupla in una tabella.

È possibile inserire in una tabella tutte le ennuple risultato di una query:

```
insert into Tabella( attributi ) select ...
```

Esempio:

```
insert into Persona(nome, eta , stipendio)  
  select padre , NULL, NULL  
from Paternita  
  where padre not in ( select nome from Persona )
```

Cancellazione da Query

Abbiamo già visto il costrutto:

delete from Tabella [**where** condizione]

per cancellare le ennuple di una tabella che soddisfano condizione.

La condizione può contenere **query annidate**.

Esempio:

delete from Paternita
where figlio **not in** (**select** nome **from** Persona)

Esistono versioni del comando **delete** che permettono di esprimere condizioni su tabelle **multiple**.

Inserimento di Valori di Default

Abbiamo visto come definire **valori di default** per attributi di tabelle:

```
create table Persona (  
    cf char(16) not null ,  
    nome varchar(100) not null ,  
    cognome varchar(100) not null ,  
    stipendio integer not null default 50,  
    primary key (cf)  
);
```

Esempio di inserimento di una ennupla:

```
insert into Persona(cf, nome, cognome, stipendio)  
    values ( 'xxx .....xxx' , 'Mario' , 'Rossi' , 45)
```

ma anche:

```
insert into Persona(cf, nome, cognome)  
    values ( 'xxx .....xxx' , 'Mario' , 'Rossi' )
```

⇒ Il valore per l'attributo **stipendio** sarà pari al valore di **default**.

Modifica a Valori di Default

Abbiamo visto come definire **valori di default** per attributi di tabelle:

```
create table Persona (  
    cf char(16) not null ,  
    nome varchar(100) not null ,  
    cognome varchar(100) not null ,  
    stipendio integer not null default 50,  
    primary key (cf)  
);
```

Esempio di modifica di una ennupla:

```
update Persona  
    set stipendio = 60 where cf = 'xxx ..... xxx'
```

ma anche:

```
update Persona  
    set stipendio = DEFAULT where cf = 'xxx ..... xxx'
```

⇒ Il valore per l'attributo **stipendio** sarà pari al valore di **default**.



SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ing. dell'Informazione, Informatica e Statistica
Laurea in Informatica

Basi di Dati, Modulo 2

Prof. Toni Mancini
Dipartimento di Informatica
<http://tmancini.di.uniroma1.it>

Slides B.2.2.3.5 (S.B.2.2.3.5)

Basi di dati
Sistemi di Gestione di Basi di Dati Relazionali
Il Linguaggio SQL
Data Manipulation Language
Transazioni

Transazioni

Spesso è necessario:

- ▶ effettuare più **modifiche contestuali** ad un database
- ▶ fare in modo che il database sia utilizzabile **concorrentemente** da più utenti e/o processi.

In queste ed altre situazioni abbiamo bisogno del concetto di **transazione**.

Esempio

```
create table ContoCorrente (  
    id integer not null primary key,  
    saldo real not null,  
    check (saldo >= 0)  
)
```

Esempio: Effettuare un giroconto di EUR 1000 dal conto corrente 111111 al conto corrente 999999.

```
update ContoCorrente  
    set saldo = saldo - 1000 where id = '111111';  
update ContoCorrente  
    set saldo = saldo + 1000 where id = '999999';
```

Cosa succede se:

- ▶ Il conto 111111 non ha saldo sufficiente?
- ▶ Avviene un malfunzionamento dopo il primo comando?

Transazioni: proprietà ACID

Un DBMS si dice **transazionale** se supporta il concetto di **transazione**.

Transazione: insieme di operazioni da considerare **indivisibile**, **corretto** anche in presenza di **concorrenza**, e con effetti **definitivi**.

Le transazioni sono:

- ▶ **Atomiche**
- ▶ **Consistenti**
- ▶ **Isolate**
- ▶ **Durevoli (persistenti)**

Transazioni: proprietà ACID (2)

Le transazioni sono... atomiche

La sequenza di operazioni sulla base dati è **indivisibile**:

- ▶ o vengono resi visibili **tutti** i suoi effetti
- ▶ oppure la transazione non deve avere **alcun** effetto sui dati.

Non è possibile lasciare la base dati in uno stato intermedio attraversato durante l'esecuzione della transazione.

Se si verifica un errore durante l'esecuzione di una transazione, il DBMS deve **ripristinare** lo stato che la base dati aveva al momento dell'avvio della transazione.

Transazioni: proprietà ACID (3)

Le transazioni sono... consistenti

Al termine dell'esecuzione della transazione, tutti i **vincoli** sulla base dati devono essere **soddisfatti**. In caso contrario, il DBMS deve **ripristinare** lo stato che la base dati aveva al momento dell'avvio della transazione.

Nota: durante l'esecuzione della transazione, ci possono essere violazioni dei vincoli (ad es., foreign key, inclusione, esterni) definiti come **deferred**.

Transazioni: proprietà ACID (4)

Le transazioni sono... isolate

L'esecuzione di una transazione deve essere **indipendente** dall'esecuzione contemporanea di altre transazioni (non interferenza).

Il risultato dell'esecuzione concorrente di un insieme di transazioni deve essere analogo all'esecuzione delle stesse transazioni in **sequenza**.

Transazioni: proprietà ACID (5)

Le transazioni sono... durevoli

L'effetto di una transazione completata con **successo** deve essere **persistente** ed essere **mantenuto** in modo **definitivo** nella base dati.

Esecuzione di transazioni in SQL

Comandi principali:

- ▶ start **transaction** (o **begin transaction**)
inizia una transazione
- ▶ **commit** work
termina la transazione con successo (dati salvati)
- ▶ **rollback** work
abortisce una transazione (dati ripristinati, come se nessun comando della transazione fosse stato mai eseguito).

Nota: non tutti i DBMS supportano **transazioni annidate**. Ad esempio, PostgreSQL le supporta solo parzialmente mediante il costrutto **savepoint**.

Esecuzione di transazioni in SQL (2)

Esempio: giroconto di EUR 1000 dal conto corrente 111111 al conto corrente 999999.

```
begin transaction ;  
update ContoCorrente  
    set saldo = saldo - 1000 where id = '111111';  
update ContoCorrente  
    set saldo = saldo + 1000 where id = '999999';  
commit work;
```