Definizione dei dati in SQL

Nicola Vitacolonna

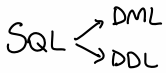
Corso di Sistemi di Elaborazione delle Informazioni
Università degli Studi di Udine

2 novembre 2015

Un po' di storia di SQL

- In origine chiamato SEQUEL (*Structured English QUEry Language*)
- Realizzato da IBM Research come interfaccia per System R
- 1986: primo standard ANSI/ISO (SQL-86)
- 1992: standard revisionato (SQL-92)
- 1999: ulteriore standard (SQL-99)
- Altre revisioni nel 2003, nel 2008 e nel 2011
- Lo standard può essere acquistato dall'ISO (i draft possono essere scaricati gratuitamente)
- *Core* + *package* opzionali (data mining, dati spaziali, dati temporali, XML, etc…)

Caratteristiche di SQL



- Carattere dichiarativo
- *Case insensitive*
- *Data Definition Language* (DDL)
  - Dati
  - Vincoli d'integrità
  - Viste
  - Gestione utenti
  - Autorizzazioni e permessi
- *Data Manipulation Language* (DML)
  - Inserimenti, aggiornamenti, cancellazioni
  - Interrogazioni
  - Transazioni
  - Stored procedure e trigger
  - Embedding in linguaggi OO o procedurali

Principali difetti

- Non proprio dichiarativo…
- Specifica molto ampia
  - Lo standard occupa piú di 3000 pagine (per confronto, ISO C++11 è di circa 1200 pagine)
- Implementazioni non conformi, numerosi dialetti
- Modalità di trattamento dell'assenza d'informazione criticabile

Caveat

Le slide seguenti descrivono il comportamento conforme allo standard. Le implementazioni (in particolare, PostgreSQL) possono differire. Si faccia riferimento al manuale del DBMS per i dettagli.

Tipi di dato: stringhe

`character``(n)` oppure **`char`**`(n)`

- Stringhe di esattamente n caratteri
- A stringhe piú corte sono aggiunti spazi in coda
- Nei confronti, gli spazi in coda sono ignorati
- **`character`** o **`char`** $\equiv$ **`char`**`(1)`

`character varying``(n)` oppure **`varchar`**`(n)`

- Stringhe di al piú n caratteri
- Spazio occupato su disco variabile

- Le stringhe sono racchiuse tra apici singoli
- I confronti tra stringhe sono *case sensitive*

Tipi di dato: valori booleani

`boolean`

- Tre valori di verità: vero, falso, indeterminato
- Il valore indeterminato è rappresentato da **`null`**
- Diversi modi di specificare i valori vero e falso:
  - `'t'`, `'f'`
  - **`true`**, **`false`**
  - `'yes'`, `'no'`
  - `'y'`, `'n'`
  - `'1'`, `'0'`

Tipi di dato numerici

`smallint`

- 2 byte, range $[-2^{15}, 2^{15} - 1]$

`integer` oppure **`int`**

- 4 byte, range $[-2^{31}, 2^{31} - 1]$

`real`

- Range tipico $[10^{-37}, 10^{37}]$
- Almeno 6 cifre decimali corrette

`double precision`

- Range tipico $[10^{-307}, 10^{307}]$
- Almeno 15 cifre decimali corrette

Tipi numerici a precisione (quasi) arbitraria

numeric(prec,scala)

- Per calcoli esatti fino a 1000 cifre significative
- *scala*: numero di cifre dopo la virgola
- *prec*: numero di cifre significative
- Esempio: 26.3456 ha precisione 6 e scala 4
- Esempio: **decimal**(5,2): valori tra -999.99 e 999.99
- Gli interi hanno scala 0
- **numeric** (senza parametri): precisione massima, scala 0

decimal(prec,scala)

- come **numeric**(prec,scala), ma assume che *prec* sia un limite inferiore alla precisione

Tipi di dato temporali

timestamp(prec)

- Esempio: '10-may-2004 14:30:10'

date

- Esempio (formato raccomandato): **date** '2004-05-13'

interval(prec)

- Intervalli di tempo relativi
- Esempio: '1 day 12 hours 59 min 10 sec ago'
- *prec*: valore opzionale che indica il numero di cifre frazionarie dopo i secondi

Domini definiti dall'utente

- È possibile definire domini a partire dai tipi di dato predefiniti:

```
create domain dom_euro as numeric;
create domain dom_provincia as char(2);
create domain dom_pagato as boolean;
```

- È possibile specificare vincoli sui domini:

```
create domain dom_provincia as char(2)
  not null;
create domain dom_voto as integer
  constraint c_voto_valido
  check (value between 18 and 30);
```

Basi di dati e schemi

- Una base di dati si crea con

```
create database <nome>
  [ with owner <utente> ]
  [ encoding <enc> ];
```

- Bisogna avere i privilegi necessari
- Il creatore della base di dati ha tutti i privilegi su di essa
- Le basi di dati sono tra loro indipendenti
- All'interno della base di dati è possibile definire uno o piú schemi:

```
create schema <nome> [ authorization <utente> ];
```

- Gli schemi suddividono la base di dati logicamente

Create table: sintassi essenziale

```
create table R (
    a char(4) primary key, -- chiave primaria
    b char references S, -- chiave esterna
    c boolean not null, -- vincolo di valor non nullo
    d integer unique, -- vincolo d'unicità
    e real check (e > 5.5) -- vincolo generico
);
create table R (
    a char(4),
    b char,
    c boolean,
    d integer,
    e real,
    primary key(a), foreign key(b) references S,
    check (c is not null), unique(d),
    constraint soglia check (e > 5.5)
);
```

Creazione di tabelle: esempio

Esame(*matr*, *corso*, *data*, *voto*, *lode*)

$18 \leq$ *voto* ≤ 30, *lode* $=$ *sì* $\rightarrow$ *voto* $= 30$, *VNN:* $\{$*data*$\}$

```
create table Esame (
  matr integer check (matr > 0),
  corso varchar(256),
  data date not null,
  voto integer check (voto between 18 and 30),
  lode boolean,
  -- Vincoli di tabella
  primary key(matr, corso),
  constraint lode_valida
    check (not lode or voto = 30)
);
```

Cancellazione di oggetti

```
drop <tipo oggetto> <nome>;
```

Esempi:

```
drop domain dom_nat;
drop table Esame;
drop schema UniUd; -- Lo schema dev'essere vuoto
drop database Universitas;
```

- La **drop** fallisce se vi sono oggetti dipendenti da quelli che si vuole cancellare
- Le cancellazioni non richiedono conferma e non sono annullabili
- **drop database** D; è un'istruzione potenzialmente devastante!

Modificazione di oggetti

- Rinomina di tabelle

```
alter table Conto rename to Fattura;
```

- Aggiunta e modificazione di colonne

```
alter table Prodotto
  add column colore varchar(30);

alter table Prodotto
  rename column colore to tinta;
```

- Eliminazione di colonne

```
alter table Prodotto drop column tinta;

alter table Prodotto
  drop column fornitore cascade;
```

Modificazione di oggetti (2)

- Aggiunta di vincoli

```
alter table Prodotto
  add constraint nobianco
  check (colore <> 'bianco');

alter table Prodotto
  add constraint colore_unico
  unique(modello, colore);

alter table Prodotto
  add constraint fk_fornitore
  foreign key(fornitore) references Fornitore
    on update cascade on delete no action;
```

- Rimozione di vincoli

```
alter table Prodotto
  drop constraint fk_fornitore;
```

Aggiornamento del database

$R(\underline{A}: \text{char}, B: \text{int})$

```
create table R(A char primary key, B int);
```

- Inserimenti

```
insert into R(A,B) values ('a',3), ('b',null);
insert into R values ('c');
```

- Aggiornamenti

```
update R set A = 'd', B = null where B < 5;
```

- Cancellazioni

```
delete from R where B is null; -- B = null è un errore!
```

- Attenzione: il sistema non chiede conferma!
  - `delete from R; where A = 'c';` può distruggere terabyte di dati (prima di dare un `syntax error`)

Reazioni ad aggiornamenti

$R(\underline{A}, B)$
$S(\underline{X}, Y)$
CE: $Y \rightarrow R(A)$

R

A	B
f	7
m	5

S

X	Y
123	m
456	null

- Il tentativo di modificare una chiave esterna assegnando un valore non esistente è sempre respinto
- Esempio: `update S set Y = 'd';`
- Ma che fare a fronte di aggiornamenti in R?
- Esempio: `update R set A = 'p' where A = 'm';`

Reazioni a cancellazioni

$R(\underline{A}, B)$
$S(\underline{X}, Y)$
CE: $Y \rightarrow R(A)$

R

A	B
f	7
m	5

S

X	Y
123	m
456	null

- Cancellazioni nelle chiavi esterne sono in generale sempre possibili
  - …se non violano altri vincoli, ovviamente
- Esempio: `delete from S where X = 123;`
- Ma che fare a fronte di cancellazioni in R?
- Esempio: `delete from R;`

Clausole on update e on delete

Possibili reazioni ad aggiornamenti e cancellazioni:

1. Non permettere l'operazione
   - azione di default, o specificata da **`no action`** o **`restrict`**
2. eseguire l'operazione in cascata: **`cascade`**
3. porre a nullo il valore coinvolto: **`set null`**
4. impostare un valore di default: **`set default`**

La specificazione dell'azione da eseguire si pone in una clausola **`on delete`** per le cancellazioni, **`on update`** per gli aggiornamenti. Esempio:

```
create table S (
  X int primary key,
  Y char references R(A)
    on update cascade
    on delete set null
);
```

Esempio

$R(\underline{A})$

$S(\underline{B}, C)$

UNIQUE

UNI: $\{C\}$

CE: $C \rightarrow R(A)$

on update no action

on delete set null

$T(\underline{D}, E)$

CE: $E \rightarrow S(C)$

on update cascade

on delete no action

R
$\underline{A}$
10
20
30
40

S	
$\underline{B}$	C
a	10
b	20
c	30

T	
$\underline{D}$	E
r	20
s	20
t	10

```
delete from R where A = 20;
```
OK

```
delete from S where B = 'a';
```
NO → fallisce perché quando prova ad eliminare in T trova NO ACTION che fa bloccare tutto

```
update S set C = 40 where B = 'c';
```
OK → non da problemi perché se modifico in S poi in T aggiorna (ON UPDATE) in cascade

```
delete from R;
```
OK → non da problemi perché ON DELETE di S è SET NULL (e non NO ACTION)

```
drop table R;
```
NO (occorre `drop table R cascade;`)

Esempio (2)

$R(\underline{A}, B)$

UNI: $\{B\}$

$S(\underline{C}, D)$

CE: $D \rightarrow R(B)$ on update set null

$T(\underline{E}, F)$

CE: $F \rightarrow R(B)$ on update no action

- Si assuma che `'a'` sia un valore di A nell'istanza corrente della base di dati
- Quando ha successo il seguente aggiornamento?

```
update R set B = null where A = 'a';
```

- Ha successo solo nel caso in cui non esista in T un record che faccia riferimento al record cancellato in R

Esempio (3)

$R(\underline{A}, B)$

$S(\underline{C}, D)$

CE: $D \rightarrow R(A)$ on update cascade on delete set null

$T(\underline{E}, F)$

CE: $F \rightarrow S(C)$ on update cascade on delete no action

- Si assuma che `'a'` sia un valore di A nell'istanza corrente della base di dati
- Quando ha successo il seguente aggiornamento?

```
delete from R where A = 'a';
```

- È sempre eseguito con successo

perché quella che potrebbe causare problemi è la chiave esterna S(D) però essendo ON DELETE set null, setterà a NULL tutte le chiavi esterne in S e poi T(F) eseguirà la ON UPDATE

Esempio (4)

$R(\underline{V})$

$S(\underline{W}, X)$

UNI: $\{X\}$

CE: $X \rightarrow R(V)$ on update no action on delete set null

$T(\underline{Y}, Z)$

CE: $Z \rightarrow S(X)$ on update no action on delete cascade

- Qual è il comportamento del DBMS a fronte dell'esecuzione delle seguenti operazioni?

```
delete from R where V = 1;
delete from S where W = 'a';
```

- La prima **`delete`** ha successo se e solo se
  - il valore `1` non compare in `V` o in `X`, oppure
  - il valore `1` compare in `X`, ma non in `Z`
- La seconda **`delete`** ha sempre successo

perché c'è im I(z) on DELETE CASCADE

Vademecum del progettista

Per ogni tabella:

- Definire il predicato associato alla tabella
- Individuare le chiavi
- Scegliere la chiave primaria
- Imporre vincoli di VNN e UNI sulle altre chiavi
- Per ogni altra colonna/insieme di colonne:
  - può essere nullo? (se no, imporre **`not null`**)
  - dev'essere unico? (se sí, imporre **`unique`**)
  - ha un default? (se sí, aggiungere **`default ...`**)
- Per ciascuna chiave esterna:
  - che azione "on delete"? (**`no action`**, **`set null`**, **`set default`**, **`cascade`**, altro)
  - Che azione "on update"? (**`no action`**, **`set null`**, **`set default`**, **`cascade`**, altro)

Transazioni

- Assieme allo schema di una base di dati è necessario definire anche le operazioni ammissibili sui dati
- Solo nei casi più semplici tali operazioni sono riconducibili a singole istruzioni **`insert`**, **`update`** o **`delete`**
- In generale, il passaggio da uno stato consistente della base di dati a un altro stato consistente richiede operazioni più complesse
- **Transazione:** sequenza di operazioni eseguita in modo atomico (tutte le operazioni sono portate a termine con successo ovvero la base di dati non è modificata in alcun modo)
- **Attenzione:** il tema delle transazioni sarà discusso in modo approfondito oltre nel corso: si tenga presente che la definizione appena data è una definizione informale, ma per ora sufficiente

Transazioni: esempio

ContoCorrente(numero, saldo)

VNN: {saldo}

Trasferimento di 1000 € dal conto 9876 al conto 9999 (nella stessa filiale o banca):

```
start transaction;
  update ContoCorrente set saldo = saldo - 1000
    where numero = 9876;
  update ContoCorrente set saldo = saldo + 1000
    where numero = 9999;
commit;
```

Se invece di **commit** si dà il comando **rollback**, la transazione è annullata (la base di dati non è modificata)

Differimento dei vincoli d'integrità

- Normalmente, la verifica dei vincoli d'integrità avviene immediatamente dopo l'esecuzione di ciascuna istruzione SQL, anche all'interno di transazioni
- È possibile tuttavia richiedere al sistema di posticipare al termine di una transazione la verifica dei vincoli d'integrità
- Così facendo, se ne ammette la temporanea violazione
- Non tutti i vincoli d'integrità sono differibili

Il differimento dei vincoli d'integrità in generale richiede:

- che i vincoli siano dichiarati "differibili" in fase di definizione
- che le transazioni dichiarino di voler posticipare la verifica dei vincoli

I VINCOLI
- Se li garantisco con le TRANSAZIONI devo farlo io esplicitamente ogni volta che faccio un'inserimento (posso anche richiamare le UDF) o un'altra operaz.
- Se li garantisco con i TRIGGER allora scattano in automatico ogni volta che faccio un'operaz.

Differimento di vincoli di chiave esterna

```
create table T (
  x int primary key,
  y int references U deferrable initially immediate,
  z int references U deferrable initially deferred
);
```

- **`deferrable`**: il vincolo è differibile
- **`initially immediate`**: il comportamento predefinito è operare una verifica immediata
- **`initially deferred`**: il comportamento predefinito è operare una verifica posticipata

Quando un vincolo è **`deferrable initially immediate`**, le transazioni devono esplicitamente dichiarare:

```
start transaction;
set constraints all deferred;
-- ...
commit;
```

Definizione dei dati: esempio

In un sistema informativo aziendale sussistano i seguenti vincoli:

1. Ogni impiegato afferisce sempre a uno e un solo dipartimento
2. Ogni dipartimento ha sempre uno e un solo manager (che è un impiegato)
3. Un impiegato può dirigere al più un dipartimento
4. Il manager di ciascun dipartimento dev'essere un afferente del dipartimento

Obiettivo: progettare una base di dati che soddisfi tali vincoli

- Si tratta di definire non soltanto la struttura, ma anche le operazioni sui dati
- Per semplicità, consideremo *soltanto* il problema d'inserire un nuovo dipartimento il cui manager non sia un impiegato presente nella base di dati (la definizione di altre operazioni è lasciata per esercizio)

Prima soluzione

Impiegato(<u>cf</u>, nome, dip)

VINCOLO NOT NULL
VNN: {nome, dip}

CHIAVE ESTERNA
CE: dip → Dipartimento(dnumero)

1. Ogni impiegato afferisce sempre a uno e un solo dipartimento
2. Ogni dipartimento ha sempre uno e un solo manager (che è un impiegato)
3. Un impiegato può dirigere al più un dipartimento
4. Il manager di ciascun dipartimento dev'essere un afferente del dipartimento

Dipartimento(<u>dnumero</u>, dnome, manager)

VNN: {dnome}

UNI: {manager} → essendo non NOT NULL potrei inserire un dipartimento con un manager NULL

CE: manager → Impiegato(cf)

- L'inserimento di un dipartimento e del suo manager dev'essere fatto in modo atomico per non violare il vincolo 2
- La verifica differita della chiave esterna in **Dipartimento** non è necessaria perché *manager* ammette valori nulli
- La definizione di una delle chiavi esterne dev'essere data in due passi per via della circolarità dei vincoli

Prima soluzione (tabelle)

creo le tabelle

```
create table Impiegato (
  cf dom_cf primary key,
  nome dom_nome not null,
  dip dom_dnum not null
);

create table Dipartimento (
  dnumero dom_dnum primary key,
  dnome dom_dnome not null,
  manager dom_cf unique,
  foreign key (manager) references Impiegato(cf)
    on update cascade on delete restrict
);

alter table Impiegato add foreign key (dip)
  references Dipartimento(dnumero)
  on update cascade on delete restrict;
```

1. Ogni impiegato afferisce sempre a uno e un solo dipartimento
2. Ogni dipartimento ha sempre uno e un solo manager (che è un impiegato)
3. Un impiegato può dirigere al più un dipartimento
4. Il manager di ciascun dipartimento dev'essere un afferente del dipartimento

Prima soluzione (transazione)

Esempio d'inserimento di un dipartimento:

```
start transaction;
```
→ facendo la TRANSAZIONE riesco a rispettare tutti i vincoli

```
  insert into Dipartimento(dnumero, dnome, manager)
    values (1, 'Marketing', null);

  -- Vincolo 2 temporaneamente violato

  insert into Impiegato(cf, nome, dip)
    values ('MRRPML65G07R697A', 'Pamela Murray', 1);

  update Dipartimento set manager = 'MRRPML65G07R697A'
    where dnumero = 1;

  -- Vincolo 2 soddisfatto

commit;
```

1. Ogni impiegato afferisce sempre a uno e un solo dipartimento
2. Ogni dipartimento ha sempre uno e un solo manager (che è un impiegato)
3. Un impiegato può dirigere al più un dipartimento
4. Il manager di ciascun dipartimento dev'essere un afferente del dipartimento

La transazione deve assicurare il rispetto dei vincoli 2 e 4

Seconda soluzione

Impiegato(<u>cf</u>, nome, dip)

VNN: {nome, dip}

CE: dip $\rightarrow$ Dipartimento(dnumero)

Dipartimento(<u>dnumero</u>, dnome, manager)

VNN: {dnome, <u>manager</u>}

↳ manager ora è NOT NULL

UNI: {manager}

CE: manager $\rightarrow$ Impiegato(cf)

1. Ogni impiegato afferisce sempre a uno e un solo dipartimento
2. Ogni dipartimento ha sempre uno e un solo manager (che è un impiegato)
3. Un impiegato può dirigere al più un dipartimento
4. Il manager di ciascun dipartimento dev'essere un afferente del dipartimento

- L'inserimento di un dipartimento e del suo manager dev'essere fatto in modo atomico per non violare il vincolo 2
- La verifica differita della chiave esterna in **Dipartimento** è necessaria perché *manager* non ammette valori nulli
- La definizione di una delle chiavi esterne dev'essere data in due passi per via della circolarità delle chiavi

Seconda soluzione (tabelle)

```
create table Impiegato (
  cf dom_cf primary key,
  nome dom_nome not null,
  dip dom_dnum not null
);

create table Dipartimento (
  dnumero dom_dnum primary key,
  dnome dom_dnome not null,
  manager dom_cf not null unique,
  foreign key (manager) references Impiegato(cf)
    on update cascade on delete restrict
    deferrable initially immediate
);

alter table Impiegato add foreign key (dip)
  references Dipartimento(dnumero)
  on update cascade on delete restrict;
```

1. Ogni impiegato afferisce sempre a uno e un solo dipartimento
2. Ogni dipartimento ha sempre uno e un solo manager (che è un impiegato)
3. Un impiegato può dirigere al più un dipartimento
4. Il manager di ciascun dipartimento dev'essere un afferente del dipartimento

Seconda soluzione (transazione)

Esempio d'inserimento di un dipartimento:

1. Ogni impiegato afferisce sempre a uno e un solo dipartimento
2. Ogni dipartimento ha sempre uno e un solo manager (che è un impiegato)
3. Un impiegato può dirigere al più un dipartimento
4. Il manager di ciascun dipartimento dev'essere un afferente del dipartimento

```
start transaction;

  set constraints all deferred;

  insert into Dipartimento(dnumero, dnome, manager)
    values (1, 'Marketing', 'MRRPML65G07R697A');

  -- Vincolo 2 temporaneamente violato

  insert into Impiegato(cf, nome, dip)
    values ('MRRPML65G07R697A', 'Pamela Murray', 1);

  -- Vincolo 2 soddisfatto

commit; -- La verifica dei vincoli avviene a questo punto
```

La transazione deve anche assicurare il rispetto del vincolo 4

Terza soluzione

1. Ogni impiegato afferisce sempre a uno e un solo dipartimento
2. Ogni dipartimento ha sempre uno e un solo manager (che è un impiegato)
3. Un impiegato può dirigere al più un dipartimento
4. Il manager di ciascun dipartimento dev'essere un afferente del dipartimento

Dipartimento(<u>dnumero</u>, dnome)

VNN: {dnome}

Impiegato(<u>cf</u>, nome, dip, è_manager : *boolean*)

VNN: {nome, dip, è_manager}

CE: dip $\rightarrow$ Dipartimento(dnumero)

- Schema ristrutturato per rimuovere la circolarità dei vincoli d'integrità referenziale
- L'inserimento di un dipartimento e del suo manager dev'essere fatto in modo atomico per non violare il vincolo 2
- Nessun differimento di vincoli è necessario

Terza soluzione (tabelle)

1. Ogni impiegato afferisce sempre a uno e un solo dipartimento
2. Ogni dipartimento ha sempre uno e un solo manager (che è un impiegato)
3. Un impiegato può dirigere al più un dipartimento
4. Il manager di ciascun dipartimento dev'essere un afferente del dipartimento

```
create table Dipartimento (
  dnumero dom_dnum primary key,
  dnome dom_dnome not null
);

create table Impiegato (
  cf dom_cf primary key,
  nome dom_nome not null,
  dip dom_dnum not null references Dipartimento(dnumero)
    on update cascade on delete restrict,
  è_manager boolean not null
);
```

Terza soluzione (transazione)

1. Ogni impiegato afferisce sempre a uno e un solo dipartimento
2. Ogni dipartimento ha sempre uno e un solo manager (che è un impiegato)
3. Un impiegato può dirigere al più un dipartimento
4. Il manager di ciascun dipartimento dev'essere un afferente del dipartimento

```
start transaction;

  insert into Dipartimento(dnumero, dnome)
    values (1, 'Marketing');

  -- Vincolo 2 temporaneamente violato

  insert into Impiegato(cf, nome, dip, è_manager)
    values ('MRRPML65G07R697A', 'Pamela Murray', 1, true);

  -- Vincolo 2 soddisfatto

commit;
```

La transazione deve farsi carico del rispetto dei vincoli 2 e 4

Con questa soluzione potrebbe essere che avrei due impiegati che sono manager dello stesso dipartimento

Quarta soluzione

Impiegato(<u>cf</u>, nome, dip)

VNN: {nome, dip}

CE: dip $\rightarrow$ Dipartimento(dnumero)

Dipartimento(<u>dnumero</u>, dnome)

VNN: {dnome}

Manager(<u>cf</u>, dip)

CE: (cf,dip) $\rightarrow$ Impiegato(cf, dip)

UNI: {dip}

1. Ogni impiegato afferisce sempre a uno e un solo dipartimento
2. Ogni dipartimento ha sempre uno e un solo manager (che è un impiegato)
3. Un impiegato può dirigere al più un dipartimento
4. Il manager di ciascun dipartimento dev'essere un afferente del dipartimento

- Lo schema è ristrutturato in modo da rimuovere la circolarità dei vincoli d'integrità referenziale
- L'inserimento di un dipartimento e del suo manager dev'essere fatto in modo atomico per non violare il vincolo 2
- Nessun differimento di vincoli è necessario
- Per inciso, questo schema non si può ottenere da un diagramma E-R (con i costrutti visti a lezione)

Quarta soluzione (tabelle)

```
create table Dipartimento (
  dnumero dom_dnum primary key,
  dnome dom_dnome not null
);

create table Impiegato (
  cf dom_cf primary key,
  nome dom_nome not null,
  dip dom_dnum not null references Dipartimento(dnumero)
    on update cascade on delete restrict,
  unique (cf, dip) -- necessario per la CE in Manager
);

create table Manager (
  cf dom_cf primary key,
  dip dom_dnum not null unique,
  foreign key (cf, dip) references Impiegato(cf, dip)
    on update cascade on delete restrict
);
```

1. Ogni impiegato afferisce sempre a uno e un solo dipartimento
2. Ogni dipartimento ha sempre uno e un solo manager (che è un impiegato)
3. Un impiegato può dirigere al più un dipartimento
4. Il manager di ciascun dipartimento dev'essere un afferente del dipartimento

Quarta soluzione (transazione)

1. Ogni impiegato afferisce sempre a uno e un solo dipartimento
2. Ogni dipartimento ha sempre uno e un solo manager (che è un impiegato)
3. Un impiegato può dirigere al più un dipartimento
4. Il manager di ciascun dipartimento dev'essere un afferente del dipartimento

Esempio d'inserimento di un dipartimento:

```
start transaction;
  insert into Dipartimento(dnumero, dnome)
    values (1, 'Marketing');

  -- Vincolo 2 temporaneamente violato

  insert into Impiegato(cf, nome, dip)
    values ('MRRPML65G07R697A', 'Pamela Murray', 1);

  insert into Manager(cf, dip)
    values ('MRRPML65G07R697A', 1);

  -- Vincolo soddisfatto
commit;
```

Osservazioni finali

- Esistono criteri formali che aiutano a valutare la “bontà” di uno schema (teoria della normalizzazione), ma non sono gli unici che si possono/devono usare
- Esistono altri modi di definire vincoli d’integrità arbitrari in SQL (ad esempio, mediante **trigger**, che studieremo oltre nel corso)
- Le soluzioni proposte non sono le uniche possibili
- Ad esempio, nella quarta soluzione si potrebbe definire uno schema **Manager**(<u>cf</u>) e usare trigger per la verifica del vincolo d’integrità interrelazionale seguente:

  $$|\textbf{Manager}| = |\textbf{Dipartimento}| = |\pi_{\text{dip}}(\textbf{Manager} \bowtie \textbf{Impiegato})|$$

  (È lasciato per esercizio verificare che questa condizione è equivalente ai vincoli 2 e 3)

Eseguire postgreSQL
>> psql -U postgres

Per uscire
>> \q

Lista di tutti i database
>> \l

Creare un database
>> create database basididati2021;

Eliminare database
>> DROP DATABASE basididati2021;

Connettersi ad un database
>> \c database_name

Lista di tutte le tabelle nel DB corrente
>> \dt

Toggle "tuples only"
>> \t

Nome del database corrente
>> SELECT current_database();

Esempio:

R: (A pk, B)
S: (X pk, Y)
FK: Y -> R(A)

Creazione tabella
```
>> create table R(
     A char primary key,
     B int);
>> create table S(
     X int primary key,
     Y char references R);
```

Eliminare le tabelle:
>> drop table R,S;

Inserimenti:
>> insert into R(A,B) values (’a’,3), (’b’,null);
>> insert into R values (’c’);

Aggiornamenti:
>> update R set A='d', B=null where B < 5;

Cancellazioni:
>> delete from R where B is null;
---
nota: occhio a *non* scrivere B=null

Esempio:

Tentativo di modifichiare chiave esterna verso un valore non esistente:
>> update S set y='d'; -- errore!

L'eliminazione di una chiave esterna è possibile, se non ci sono altri
vincoli di chiave esterna su di essa:
>> delete from S where x=123;