Definizione e manipolazione dei dati in SQL

Nicola Vitacolonna
Corso di Basi di Dati
Università degli Studi di Udine

19 novembre 2013

Nota bene: nel presente documento, con il termine "macchina virtuale", o "server virtuale", s'intende la macchina virtuale fornita agli studenti del corso (si veda la precedente dispensa).

1 Preliminari: connessione a un server PostgreSQL

In generale, per la connessione a un server è necessario conoscere il valore dei seguenti parametri:

- il nome dell'host o l'indirizzo IP del server;
- la porta presso cui il server è in ascolto (5432 nella configurazione predefinita di PostgreSQL);
- il nome dell'utente della base di dati;
- la corrispondente password;
- il nome della base di dati.

1.1 Connessione locale al server della macchina virtuale

Nella macchina virtuale è già installato il client da linea di comando `psql`. Una volta effettuato il login al sistema (direttamente o via SSH), la connessione al DBMS server può essere effettuata con un comando del tipo:

```
psql -U <nome utente> -p 5432 -h 127.0.0.1 <database>
```

Tutti gli argomenti del comando sono opzionali. Se `-h <host>` è omesso, la connessione avviene tramite socket Unix piuttosto che via TCP/IP (e si applicano le politiche d'accesso specificate dalle clausole `local` nel file `pg_hba.conf` invece di quelle specificate dalle clausole `host`). Se l'opzione `-U` è omessa, la connessione al DBMS avviene usando come nome utente il nome dell'utente di sistema che esegue il comando. Se il nome della base di dati è omesso, `psql` tenta di collegarsi a una base di dati che ha lo stesso nome dell'utente. Infine, se la porta non è specificata, si assume la porta di default (tipicamente, 5432).

In un'installazione di base di PostgreSQL è definito un solo utente, chiamato `postgres`. La prima connessione avviene effettuando il login al sistema con le credenziali di un utente che ha privilegi d'amministrazione (tale utente è `vagrant` se si usa la macchina virtuale del corso) e scrivendo:

```
sudo su postgres
psql
```

La prima operazione da eseguire a questo punto è la creazione di un altro utente. Da qui in avanti assumeremo che sia stato creato un utente con il seguente comando:

```
create role vagrant with login createdb createrole
  encrypted password 'vagrant';
```

La clausola `with` specifica i privilegi da assegnare al nuovo utente:

- `login`: l'utente può effettuare il login al DBMS server;
- `createdb`: l'utente può creare nuove basi di dati;
- `createrole`: l'utente può creare nuovi utenti.

Se non si specifica una password al momento della creazione dell'utente, o se si desidera modificarla, si può usare il seguente comando:

```
alter role vagrant with encrypted password 'nuova password';
```

Si noti che la sintassi del comando `create role` non è completamente conforme allo standard SQL e l'istruzione `alter role` è un'estensione di PostgreSQL.

1.2 Connessione remota al server della macchina virtuale

La porta 5432 del server virtuale è inoltrata verso la porta **55432** nell'host. Perciò, per collegarsi dall'host al DBMS server bisogna usare quest'ultima porta. Ad esempio, per collegarsi al server usando pgAdmin, scegliere l'opzione di menú File → Add Server... e compilare i campi come illustrato in Figura 1.

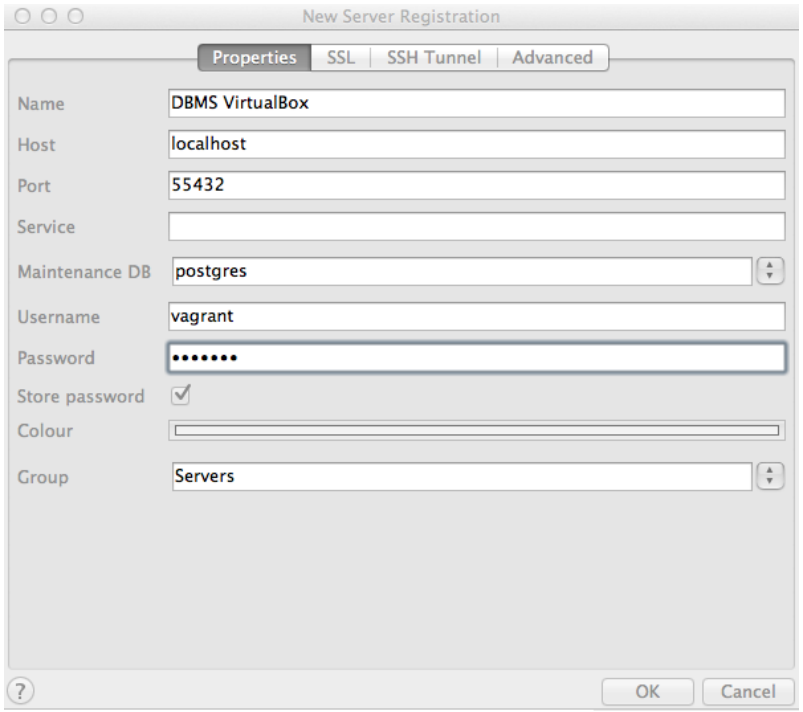

Figura 1: Finestra di configurazione di un DBMS server in PgAdmin III (versione 1.18.1).

1.3 Connessione a un server PostgreSQL su Heroku

Se si usa il servizio Heroku, una volta recuperate le credenziali d'accesso (Figura 2), bisogna inserirle nel proprio client (si vedano le Figure 3 e 4 per un paio di esempi). Nel caso di pgAdmin, bisogna assicurarsi che la connessione avvenga via SSL (Figura 5). Si noti inoltre che il campo *Maintenance DB* va impostato inserendo il nome del database. Conviene inoltre inserire il nome del database anche nel campo *DB restriction* nel tab *Advanced* (Figura 6).

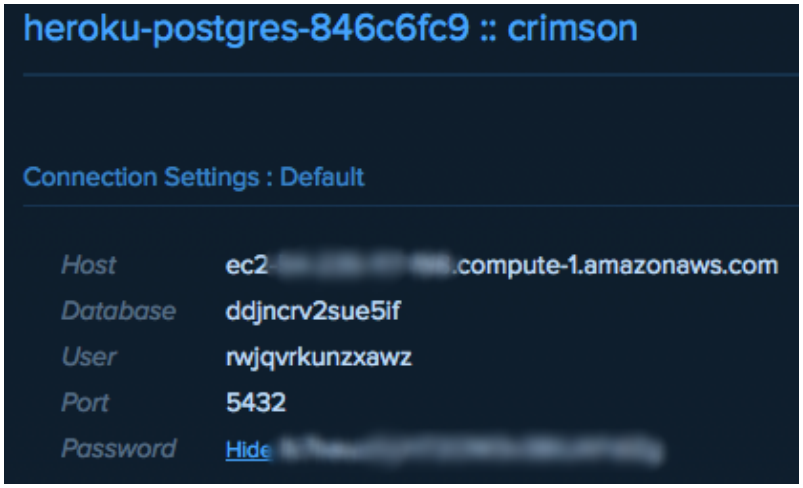

Figura 2: Le impostazioni di connessione a un DBMS presentate da Heroku.

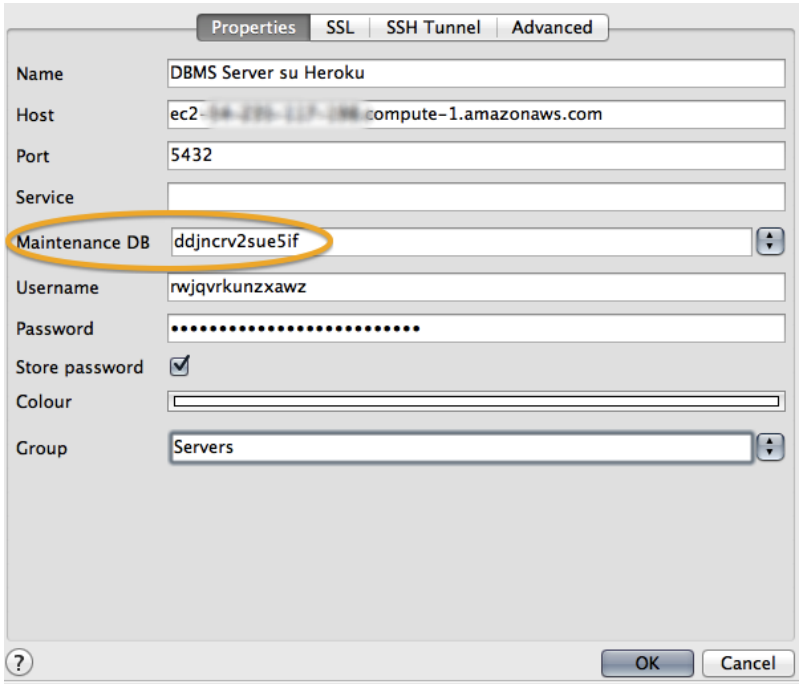

Figura 3: Per collegarsi usando pgAdmin, bisogna inserire i dati forniti da Heroku nella finestra di configurazione di un server (menú File → Add Server...). Si vedano anche le Figure 5 e 6.

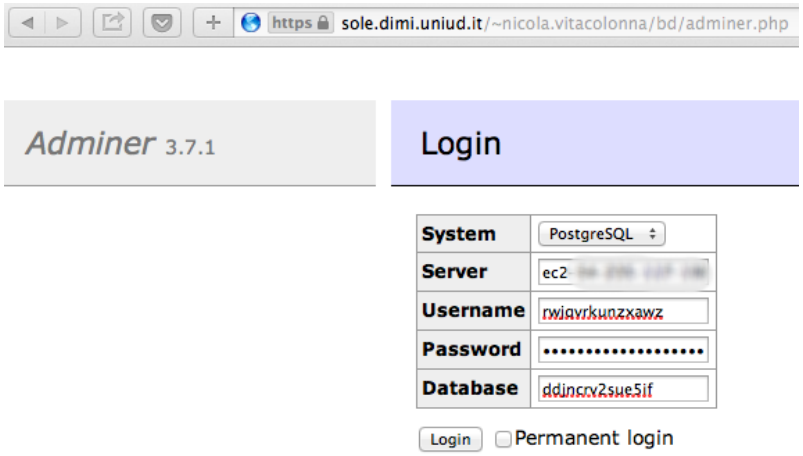

Figura 4: Connessione a un DBMS su Heroku dal client Adminer (disponibile via web agli studenti del corso).

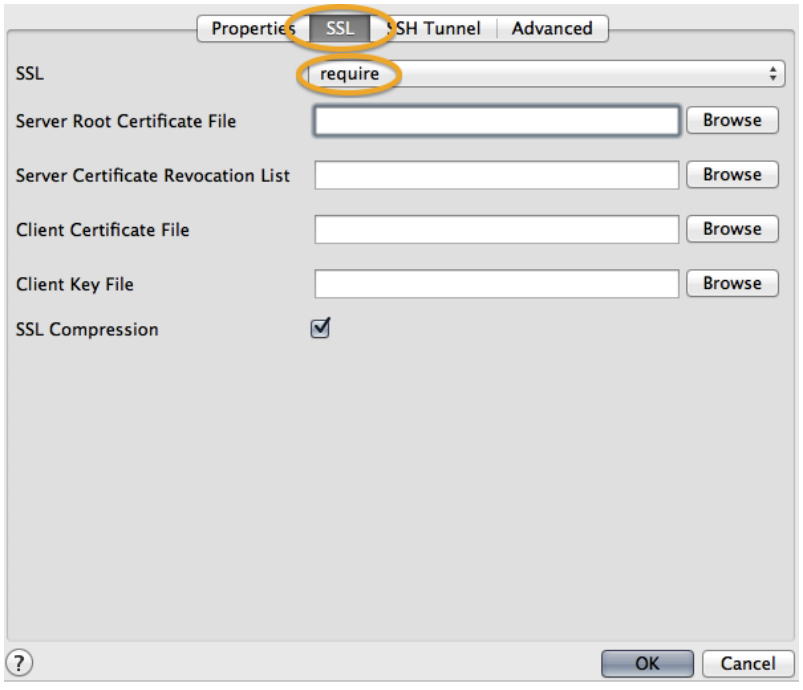

Figura 5: Per collegarsi usando pgAdmin, bisogna assicurarsi che la connessione avvenga via SSL.

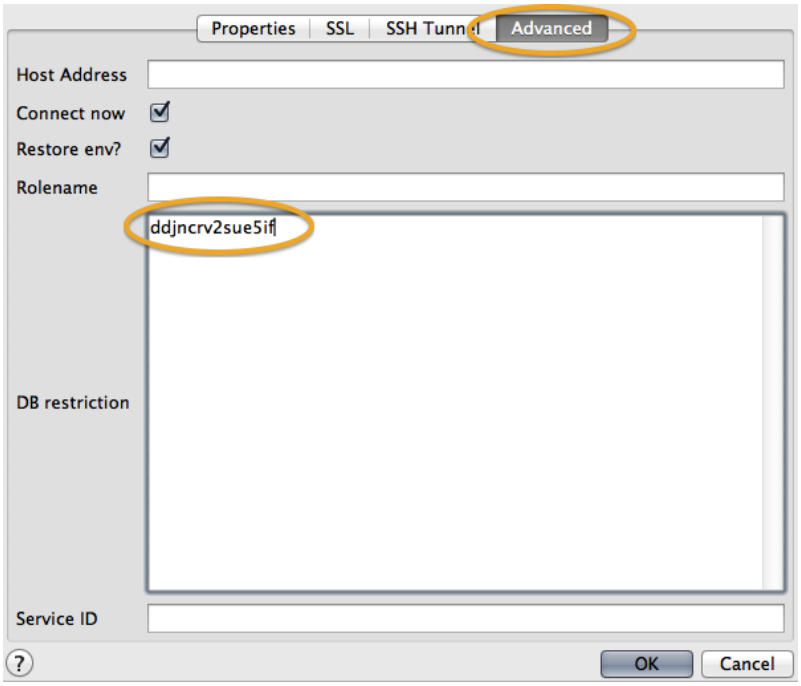

Figura 6: Per evitare di visualizzare l'elenco di tutti i database presenti nel server di Heroku (cui peraltro non si può accedere), conviene aggiungere un filtro nel campo *DB restriction* nel tab *Advanced*.

2 Definizione dei dati in SQL

Nota: per approfondire gli argomenti relativi alla definizione dei dati in SQL, si consiglia la lettura del Cap. 8 (*Data Types*) e del Cap. 5 (*Data Definition*) del manuale di PostgreSQL.

Nota: SQL è un linguaggio *case-insensitive*, ossia non distingue tra caratteri maiuscoli e caratteri minuscoli (tranne che all'interno di stringhe). Ad esempio, le due seguenti istruzioni sono del tutto equivalenti:

select ∗ **from** QualcheTabella;
SELECT ∗ **FROM** qualcheTABELLA;

Tutti i comandi SQL terminano con un punto e virgola. Se si usa pgAdmin, il punto e virgola può essere omesso, perché viene automaticamente aggiunto dall'applicazione prima di sottomettere il comando al server.

2.1 Domini predefiniti

La seguente tabella riporta i principali *domini* per stringhe, valori numerici, valori logici e dati temporali definiti dallo standard SQL, nonché alcune estensioni usate da PostgreSQL.

Dominio	Descrizione
char(n)	Stringhe di n caratteri (lunghezza fissa). A stringhe piú corte sono aggiunti spazi in coda.
char	Sinonimo di **char**(1).
varchar(n)	Stringhe di al piú n caratteri (lunghezza variabile).
varchar	**Estensione PostgreSQL** Stringhe di lunghezza arbitraria.
text	**Estensione PostgreSQL** Stringhe di lunghezza arbitraria.
smallint	Intero (tipicamente, 2 byte, range $[-2^{15}, 2^{15}-1]$).
integer	Intero (tipicamente, 4 byte, range $[-2^{31}, 2^{31}-1]$).
real	Numeri in virgola mobile (tipicamente, nel range $[10^{-37}, 10^{37}]$ con almeno 6 cifre decimali corrette).
double precision	Numeri in virgola mobile (tipicamente, nel range $[10^{-307}, 10^{307}]$ con almeno 15 cifre decimali corrette).
numeric(prec,scala)	Per calcoli esatti fino a 1000 cifre significative; *scala* è il numero di cifre dopo la virgola; *prec* è il numero di cifre significative. Es.: 26.3456 ha precisione 6 e scala 4. Gli interi hanno scala 0.
numeric	Memorizza numeri decimali fino alla precisione massima consentita dall'implementazione.
timestamp	Data e ora. Es.: `'10-may-2004 14:30:10'`.
date	Data. Es.: (formato raccomandato) `'2004-05-13'` (anno-mese-giorno).
time	Ora. Es.: `'14:30:23.80'`, `'4:25 pm'`.
interval	Intervalli di tempo. Es: `'1 day 12 hours 59 min 10 sec ago'`.
boolean	Tipo booleano. Esempi di valori booleani: `TRUE`, `FALSE` (preferibili); `'t'`, `'f'`, `'true'`, `'false'`, `'y'`, `'n'`, `'yes'`, `'no'`, `'1'`, `'0'`.

2.2 Domini definiti dall'utente

L'utente può definire i propri domini con l'istruzione **create domain**. Si consideri il seguente esempio:

create domain nat_pari **as integer**
  constraint positivo **check** (**value** >= 0)
  constraint pari **check** (**value** % 2 = 0);

L'istruzione crea un dominio chiamato nat_pari, definito come il sottoinsieme degli **integer** che soddisfa due vincoli: ciascun vincolo è specificato mediante una clausola **constraint**...**check**. A ciascun vincolo è attribuito un nome ("positivo"e "pari" nell'esempio), che consente di riferirsi al vincolo in successive operazioni di modificazione. La condizione da verificare è posta tra parentesi, e usa la parola chiave **value** per fare riferimento a un generico valore del dominio. Dopo aver creato un dominio, lo si può usare come fosse un dominio predefinito.

Altri esempi di istruzioni per la creazione di domini:

create domain provincia **as char**(2) **not null**;

create domain voto **as integer**
  check (**value between** 18 **and** 30);

create domain voto **as integer**
  check (**value** > 17 **and value** <= 30)

Le ultime due definizioni sono equivalenti.

Per vedere quali sono i domini definiti dall'utente in una determinata base di dati si può usare il comando `\dD` in `psql`. La lista dei domini definiti all'interno di uno schema S si ottiene con `\dD` S`.*` (comando `\dD`, spazio, nome dello schema, punto, asterisco). In pgAdmin, è possibile accedere ai domini definiti dall'utente espandendo l'elemento corrispondente alla base di dati nell'*Object browser*, e seguendo le voci Schemas → ⟨*schema*⟩ → Domains.

2.3 Creazione di schemi e tabelle

Uno schema permette di dividere logicamente una base di dati in parti separate, che possono comunicare tra loro. Uno schema può contenere definizioni di domini, tabelle, viste, privilegi, vincoli e funzioni. La seguente istruzione SQL crea uno schema chiamato *Ditta*:

create schema Ditta;

Dopo aver creato uno schema è possibile aggiungere oggetti al suo interno, qualificando il nome di tali oggetti con il nome dello schema (seguito da un punto), come mostrano gli esempi seguenti:

```
create domain Ditta.dom_stipendio as numeric(8,2)
    check (value > 900); -- Stipendi troppo bassi sono certamente un errore

create domain Ditta.dom_cod_impiegato as varchar(4)
    check (value like 'I%'); -- Il codice impiegato è una stringa che comincia con 'I'.

create table Ditta.Impiegato (
    cod Ditta.dom_cod_impiegato primary key,
    nome varchar(40) not null,
    stipendio Ditta.dom_stipendio
);
```

In generale, un qualunque elemento all'interno di una base di dati è identificato dalla notazione

schema.elemento

In PostgreSQL, esiste uno schema predefinito dal sistema chiamato *public*. Quando un elemento della base di dati è creato o riferito senza la qualificazione di uno schema, il sistema assume che lo schema sia *public*. As esempio, l'istruzione

create table R (a **char primary key**, b **char**);

equivale a

create table public.R (a **char primary key**, b **char**);

Per cancellare dalla base di dati lo schema precedente si possono usare le istruzioni seguenti:

```
drop table Ditta.Impiegato; -- Cancella la tabella Impiegato
drop domain Ditta.dom_cod_impiegato; -- Cancella il dominio
drop domain Ditta.dom_stipendio; -- Idem
drop schema Ditta; -- Cancella lo schema
```

In alternativa, si può eliminare uno schema non vuoto in un'unica istruzione che specifica una cancellazione "in cascata": l'istruzione seguente ha lo stesso effetto delle quattro istruzioni precedenti:

drop schema Ditta **cascade**; -- *Cancella lo schema e il suo contenuto*

Senza l'opzione **cascade** si ottiene un errore.

> **Importante!** Il sistema di gestione di basi di dati **non** richiede alcuna conferma prima di effettuare una cancellazione!

La parola chiave **cascade** può essere posposta anche a istruzioni **drop table**, ed è necessaria per cancellare tabelle riferite da chiavi esterne. Ad esempio, si supponga che la base di dati contenga due tabelle R e S basate sullo schema:[1]

[1] La notazione CE: $d \to R(a)$ denota una chiave esterna dall'attributo d della relazione corrispondente all'attributo a di R.

R($\underline{a}$, b)
S($\underline{c}$, d)
CE: $d \rightarrow R(a)$

create, ad esempio, con le istruzioni:

create table R(a **integer primary key**, b **integer**);
create table S(c **integer primary key**, d **integer references** R);

L'istruzione **drop table** R; produce un errore a causa della dipendenza della chiave esterna di S da R. L'istruzione **drop table** R **cascade**; invece elimina la tabella R e la chiave esterna in S (ma *non* la tabella S!).

I vincoli d'integrità associati a una tabella possono essere esaminati in `psql` con il comando

`\d` *schema.tabella*

In pgAdmin, si può cliccare col tasto destro sulla tabella nell'*Object browser*, selezionare Properties..., infine cliccare sul tab Constraints.

2.4 Vincoli di integrità

La seguente istruzione illustra la sintassi SQL per la specificazione dei vincoli di chiave primaria, chiave esterna, valore non nullo, unicità e altri vincoli di dominio mediante *vincoli di colonna*:

create table R (
    a **char**(4) **primary key**,
    b **char references** S,
    c **boolean not null**,
    d **integer unique**,
    e **real constraint** soglia **check** (e > 5.5)
);

I vincoli possono essere scritti anche come *vincoli di tabella*:

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)
);

L'uso di vincoli di tabella è obbligatorio per vincoli che coinvolgono piú di un attributo.

2.5 Inserimento, modificazione, cancellazione di dati

Nota: per approfondire le istruzioni di manipolazione dei dati di SQL si consiglia la lettura del Cap. 6 (*Data Manipulation*) del manuale di PostgreSQL.

Vediamo come inserire, modificare e cancellare dati in una tabella con schema:

PRODOTTO($\underline{\text{codice}}$, nome, prezzo)

che può essere implementata in SQL, ad esempio, come segue:

```
create table Prodotto (
  codice integer primary key,
  prezzo numeric(8,2) not null,
  nome varchar(20)
);
```

Ecco alcuni esempi:

```
insert into Prodotto(codice, prezzo, nome) values (123, 459.49, 'pc');
```

L'ordine in cui si specificano i nomi dei campi non è importante. Se alcuni campi sono omessi nell'istruzione d'inserimento, a essi è assegnato il valor nullo[2]. Ad esempio,

```
insert into Prodotto(prezzo, codice) values (199.29, 321);
```

equivale a

```
insert into Prodotto(prezzo, codice, nome) values (199.29, 321, null);
```

È possibile inserire più record nella medesima istruzione, separandoli con una virgola:

```
insert into Prodotto(codice, nome, prezzo)
  values (100, 'tablet', 299.99), (101, 'phone', 199.99), (102, 'phablet', 159.69);
```

È possibile cancellare uno o più record con l'istruzione **delete from**:

```
delete from Prodotto where nome = 'pc';
delete from Prodotto where prezzo < 10;
```

La clausola **where** consente di specificare una *condizione* che le tuple da eliminare devono soddisfare. Ad esempio, la prima delle due istruzioni precedenti cancella i prodotti il cui nome è "pc", e la seconda cancella tutti i prodotti il cui prezzo sia inferiore a 10.

Infine, è possibile aggiornare i record di una tabella avviene con l'istruzione **update**:

```
update Prodotto set nome = 'mac' where nome = 'pc';
update Prodotto set prezzo = 29.49 where nome = 'pc' and prezzo < 29;
update Prodotto set (nome, prezzo) = ('pc', 99.99) where nome = 'pc' or prezzo > 99.99;
```

2.6 Reazione a cancellazioni e aggiornamenti

Il tentativo di modificare una chiave esterna assegnando un valore non esistente nel/i campo/i destinazione della chiave è sempre respinto. Ma che accade quando il record cui una chiave esterna fa riferimento è cancellato oppure il campo riferito dalla chiave è modificato? Le possibili azioni sono:

1. non permettere l'operazione: azione di default, o specificata da **no action** o **restrict**;
2. effettuare la cancellazione o l'aggiornamento 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 di seguito alla clausola **on delete** per le cancellazioni, e di seguito a **on update** per gli aggiornamenti. Ad esempio, si consideri la seguente definizione di tabella:

```
create table Iscrizione (
  utente dom_utente references Utente(uid) on update cascade on delete cascade,
  corso dom_corso references Corso(cid) on update cascade on delete restrict,
  primary key(utente, corso)
);
```

[2] Oppure, un valore predefinito, se questo è stato incluso nella definizione della tabella mediante una clausola **default**.

Allora, l'istruzione:

update Utente **set** uid = 2 **where** uid = 1;

aggiorna un record della tabella Utente *e tutti i record* della tabella Iscrizione che assumono il valore 1 in corrispondenza del campo "utente" (in virtù della clausola **on update cascade** associata al campo "utente"). Invece, l'istruzione:

delete from Corso **where** cid = 200;

è respinta dal sistema *se* esiste almeno un record di Iscrizione che assume il valore 200 in corrispondenza del campo "corso" (in virtù della clausola **on delete restrict**).

2.7 Esercizio: reazioni a catena

Si consideri il seguente schema di base di dati:

```
create table T1 (
  w char primary key
);

create table T2 (
  x char primary key,
  y char unique references T1(w)
                on update cascade
                on delete set null
);

create table T3 (
  z char primary key references T2(y)
                    on delete cascade
);
```

e si supponga che siano state eseguite le seguenti istruzioni:

```
insert into T1 values ('a');
insert into T2(x, y) values ('b', 'a');
insert into T3 values ('a');
```

Si risponda alle seguenti domande:

1. Qual è il risultato della seguente operazione:

   delete from T1 **where** w = 'a';

   e perché? Per rispondere, si tenga presente che

   (a) le clausole **on update** e **on delete** si riferiscono al tipo di operazione che avviene nella tabella riferita dalla chiave esterna: se nella tabella riferita dalla chiave è stata eseguita una cancellazione, si applica la clausola **on delete**; se è stata eseguita un'operazione di aggiornamento, si applica la clausola **on update**;

   (b) porre a nullo un valore è, ovviamente, un'operazione di aggiornamento.

2. Quale vincolo d'integrità è violato dalla seguente operazione?

   update T1 **set** w = 'c' **where** w = 'a';

3. La seguente operazione è portata a termine dal sistema, oppure no?

   delete from T2 **where** x = 'b';

4. Il vincolo di unicità dell'attributo y è necessario? Perché?

3 Esercizio: creazione di una base di dati

Un popolare sito web a tema scacchistico fornisce agli utenti informazioni relative a famose partite di scacchi. Una semplificazione dello schema relazionale della base di dati usata dal sito è il seguente:[3]

GIOCATORE(nome: stringa, nazionalità: stringa)

APERTURA(eco: dom_eco, nome: stringa, variante: stringa)

VNN: {nome}
UNI: {variante}

PARTITA(bianco: stringa, nero: stringa, luogo: stringa, anno: $\mathbb{N}$, round: $\mathbb{N}$, eco: dom_eco, risultato: dom_risultato, num_mosse: $\mathbb{N}$)

CE: bianco $\rightarrow$ GIOCATORE
CE: nero $\rightarrow$ GIOCATORE
CE: eco $\rightarrow$ APERTURA

Si tengano presenti le seguenti osservazioni:

- Per "apertura" s'intende la fase iniziale di una partita. Le aperture sono convenzionalmente identificate attraverso il cosiddetto "codice ECO", che è un codice di tre lettere.
- Il risultato di una partita è espresso mediante una stringa di tre lettere, che può essere 1-0 (vittoria del Bianco), 0-1 (vittoria del Nero), 1/2 (pareggio).

Si implementi in PostgreSQL uno schema SCACCHI che corrisponda allo schema relazionale dato, e vi si inserisca l'informazione seguente:

GIOCATORE	
nome	nazionalità
Karpov	Russia
Kasparov	Russia
Kramnik	Russia
Deep Blue	NULL
Lasker	Germania
Capablanca	Cuba
Alekhine	Russia

APERTURA		
eco	nome	variante
E49	Nimzo-Indiana	1 d4, Cf6; 2 c4, e6; 3 Cc3, Ab4
E86	Est-Indiana	1 d4, Cf6; 2 c4, g6; 3 Cc3, Ag7
A14	Partita Inglese	1 c4, e6; 2 Cf3, d5; 3 g3, Cf6; 4 Ag2, Ae7
B22	Difesa Siciliana	1 e4, c5; 2 c3
D30	Gambetto di donna	1 d4, d5; 2 c4, e6
B17	Caro-Kann	1 e4, c6; 2 d4, d5; 3 Cc3, d:e4; 4 C:e4, Cd7
C01	Difesa francese	1 e4, e6; 2 d4

[3] VNN: X sta per "vincolo di valor non nullo sull'insieme di attributi X"; UNI: X sta per "vincolo d'unicità sull'insieme di attributi Y".

Partita							
bianco	nero	luogo	anno	round	eco	risultato	num_mosse
Deep Blue	Kasparov	Filadelfia	1996	1	B22	1-0	37
Deep Blue	Kasparov	Filadelfia	1996	3	B22	1/2	39
Kasparov	Deep Blue	Filadelfia	1996	6	NULL	NULL	NULL
Kasparov	Karpov	Siviglia	1987	24	A14	NULL	64
Kasparov	Karpov	Siviglia	1986	14	B17	1/2	21
Karpov	Kasparov	Linares	1993	10	E86	0-1	27
Capablanca	Lasker	La Havana	1921	5	NULL	1-0	NULL
Capablanca	Alekhine	Buenos Aires	1927	1	C01	0-1	43