Basi di dati A.A. 2023/24
Esercitazione N. 6
Testo e soluzioni

Luca Andolfi, Maurizio Lenzerini

Dobbiamo organizzare una base di dati per memorizzare le informazioni sui turisti, i viaggi e le città visitate dai turisti nei viaggi. Per questo dobbiamo definire tre tabelle, con nomi turista, viaggio e visita, mostrate qui sotto, per le quali valgono le seguenti osservazioni.

Di ogni turista interessa il codice fiscale (di tipo varchar(20)), che è chiave, il cognome (di tipo varchar(50)) e l'anno di nascita (integer). Di ogni viaggio interessa il codice (integer), che è chiave, e l'anno di svolgimento (integer). Di ogni visita interessa il codice fiscale del turista che ha effettuato la visita, la città visitata (varchar(20)), il viaggio nell'ambito del quale si è svolta la visita e il costo (integer maggiore di 0) della visita stessa. Turista, viaggio e città formano una chiave per visita. Si noti che quando si elimina un viaggio o ne si cambia il codice, occorre propagare questi cambiamenti nella tabella visita.

turista

codicefiscale	*cognome*	*annonascita*
10	Rossi	1980
20	Bianchi	1990
30	Verdi	1985
40	Gialli	1995

viaggio

codice	*anno*
1	2000
2	2005
3	2010
4	2012
5	2017
6	2020

visita

turista	*citta*	*viaggio*	*costo*
10	Roma	1	500
10	Viterbo	1	200
10	Napoli	2	100
20	Roma	1	300
30	Venezia	2	700
30	Roma	3	400
30	Torino	3	500
40	Roma	4	600
40	Verona	4	400
40	Genova	5	100
40	Livorno	5	200
40	Firenze	6	700

Esercizio 1

Definire la base di dati in accordo ai requisiti illustrati sopra.

Soluzione esercizio 1

La soluzione è molto semplice. L'unica caratteristica non completamente banale riguarda la decisione sulla politica di reazione alla violazione del vincolo di foreign key da "visita" a "viaggio": poiché le specifiche richiedono che quando si elimina un viaggio o ne si cambia il codice occorre propagare questi cambiamenti nella tabella "visita", optiamo per la politica "cascade" sia per la delete sia per l'update.

```
create table turista (
  codfis varchar(20) primary key,
  cognome varchar(50),
  annonascita integer
);

create table viaggio (
  codice integer primary key,
  anno integer
);

create table visita (
  codturista varchar(20) references turista,
  citta varchar(20),
  codviaggio integer references viaggio on delete cascade on update cascade,
  costo integer,
  constraint pk_visita primary key (codturista,codviaggio,citta),
  constraint costo_non_negativo check (costo >= 0)
);


INSERT INTO turista VALUES ('10', 'Rossi',1980), ('20', 'Bianchi',1990), ('30', 'Verdi',1985),
('40', 'Gialli',1995);

INSERT INTO viaggio VALUES (1,2000),(2,2005),(3,2010),(4,2012),(5,2017),(6,2020);

INSERT INTO visita VALUES ('10', 'Roma',1,100), ('10', 'Viterbo',1,200), ('10', 'Napoli',2,100),
('20', 'Roma',1,300), ('30', 'Venezia',2,700), ('30', 'Roma',3,400), ('30', 'Torino',2,500),
('40', 'Roma',4,600), ('40', 'Verona',4,400), ('40', 'Genova',5,100), ('40', 'Livorno',5,200),
('40', 'Firenze',6,700);
```

Esercizio 2

Creare una funzione che restituisce le città visitate da un turista il cui codice fiscale è dato come parametro. Mostrare poi un esempio di chiamata di tale funzione.

Soluzione esercizio 2

Definizione della funzione:

```
CREATE OR REPLACE FUNCTION citta_visitate(t VARCHAR)
   RETURNS TABLE (c varchar(40)) AS
$$
BEGIN
  RETURN QUERY SELECT distinct citta from visita where codturista = t;
END;
$$ language plpgsql;
```

Esempio di chiamata:

```
select *
from citta_visitate('30')
```

Esercizio 3

Aggiungere un attributo “quantiviaggi” di tipo integer alla tabella "turista", in modo che per ogni turista l’attributo riporti quanti viaggi ha svolto. Riguardo all'aggiornamento di questo attributo è stato stabilito che il suo valore per le varie tuple deve essere aggiornato globalmente per tutta la tabella ogni volta che viene invocato l'inserimento di una tupla nella tabella “turista”. In altre parole, ogni volta che invochiamo l'inserimento di una qualunque tupla nella tabella “turista”, ricalcoliamo il numero di viaggi di tutti i turisti.

Soluzione esercizio 3

Aggiungiamo l'attributo, contando sul fatto che il valore di default sia null (appena aggiunto l'attributo, il suo valore per tutte le tuple sarà null). Successivamente definiamo un trigger per rispondere all'esigenza di aggiornare il valore di questo attributo ogni volta che venga invocata l'operazione di inserimento nella tabella "turista". Si noti che questa esigenza viene rispettata dichiarando il trigger AFTER INSERT (in modo che non perdiamo l'informazione sul nuovo turista) di tipo FOR EACH STATEMENT.

```
ALTER TABLE turista ADD COLUMN quantiviaggi integer;
```

```
CREATE OR REPLACE FUNCTION aggiorna_quantiviaggi() RETURNS trigger AS
$$
BEGIN
    update turista set quantiviaggi = (select count(distinct codviaggio)
                                       from visita
                                       where codturista = codfis);
RETURN NULL;
END;
$$ language plpgsql;

create trigger trigger_quantiviaggi after insert on turista
  for each statement execute procedure aggiorna_quantiviaggi();
```

Esercizio 4

Aggiungere un attributo “incasso” di tipo integer alla tabella viaggio. Per ogni viaggio l’attributo deve dire qual è la somma incassata dal viaggio, ottenuta sommando i costi sostenuti dai turisti per le visite di quel viaggio. L’attributo deve essere costantemente aggiornato, fin dal momento della sua creazione.

Affinché l'attributo sia costantemente aggiornato, occorre fare in modo che

- al momento dell'aggiunta dell'attributo alla tabella, il valore iniziale dell’attributo per le varie tuple venga calcolato da una chiamata di funzione opportunamente definita,
- successivamente l’attributo sia aggiornato ad ogni operazione che ne cambia potenzialmente il valore, ossia ogni volta che si aggiunge, cancella o aggiorna una tupla della tabella "visita".

Soluzione esercizio 4

Aggiungiamo l'attributo, definiamo la funzione per l'inizializzazione del nuovo attributo ed invochiamo tale funzione.

```
ALTER TABLE viaggio ADD COLUMN incasso integer;

CREATE OR REPLACE FUNCTION inizializza_incassi() RETURNS void AS
$$
BEGIN
    update viaggio set incasso = (select sum(costo)
                                  from visita
                                  where codviaggio = codice)
END;
$$ language plpgsql;
```

select inizializza_incassi()
Occupiamoci ora dell'esigenza di tenere aggiornati i valori del nuovo attributo per le varie tuple della tabella "visita". Per fare questo dobbiamo definire per la tabella "visita" un trigger per l'inserimento, un trigger per la cancellazione ed un trigger per l'aggiornamento. A questo scopo presentiamo due possibili strategie.

Strategia A

La prima soluzione che presentiamo si basa sull'idea di definire una funzione per ciascun trigger. La funzione associata al trigger di inserimento dovrà aggiungere il costo del viaggio, perché questo va a contribuire all'incasso prodotto dal viaggio stesso. La funzione associata al trigger di cancellazione dovrà detrarre il costo del viaggio, perché questo va a diminuire l'incasso prodotto dal viaggio stesso. Infine, la funzione associata al trigger di aggiornamento dovrà detrarre il costo prima dell'aggiornamento ed aggiungere il costo indicato nell'aggiornamento stesso. Per non perdere informazioni relative alla visita (o alle visite) oggetto dell'operazione, definiamo tali trigger coma AFTER e FOR EACH ROW.

```
CREATE OR REPLACE FUNCTION aggiorna_incassi_dopo_insert() RETURNS trigger AS
$$
BEGIN
  update viaggio set incasso = incasso + NEW.costo where codice = NEW.codviaggio;
  RETURN NULL;
END;
$$ language plpgsql;


create trigger trigger_insert_visita after insert on visita
  for each row execute procedure aggiorna_incassi_dopo_insert();




CREATE OR REPLACE FUNCTION aggiorna_incassi_dopo_delete() RETURNS trigger AS
$$
BEGIN
  update viaggio set incasso = incasso - OLD.costo where codice = NEW.codviaggio;
  RETURN NULL;
END;
$$ language plpgsql;


create trigger trigger_update_visita after delete on visita
  for each row execute procedure aggiorna_incassi_dopo_delete();
```

```
CREATE OR REPLACE FUNCTION aggiorna_incassi_dopo_update() RETURNS trigger AS
$$
BEGIN
  update viaggio set incasso = incasso - OLD.costo where codice = NEW.codviaggio;
  update viaggio set incasso = incasso + NEW.costo where codice = NEW.codviaggio;
  RETURN NULL;
END;
$$ language plpgsql;

create trigger trigger_update_visita after update on visita
  for each row execute procedure aggiorna_incassi_dopo_update();
```

Strategia B

In realtà, possiamo fornire una seconda soluzione, più compatta e sintetica, seguendo una diversa strategia, che chiamiamo B. Questa strategia si basa sull'osservazione che quando scatta un trigger definito sulla insert, OLD assume il valore null, e, analogamente, quando scatta un trigger definito sulla delete, NEW assume il valore null. A questo punto possiamo riunificare le operazioni da svolgere in un'unica funzione che verrà invocata da tutti e tre i trigger e che si occupa sia dell'eventuale incremento e sia dell'eventuale decremento dell'incasso.

```
CREATE OR REPLACE FUNCTION aggiorna_incassi() RETURNS trigger AS
$$
BEGIN
  IF NEW is not null
  THEN update viaggio set incasso = incasso + NEW.costo where codice = NEW.codviaggio;
  END IF;
  IF OLD is not null
  THEN update viaggio set incasso = incasso - OLD.costo where codice = OLD.codviaggio;
  END IF;
  RETURN NULL;
END;
$$ language plpgsql;

create trigger trigger_insert_visita after insert on visita
  for each row execute procedure aggiorna_incassi();
```

```
create trigger trigger_delete_visita after delete on visita
  for each row execute procedure aggiorna_incassi();

create trigger trigger_update_visita after update on visita
  for each row execute procedure aggiorna_incassi();
```

Esercizio 5

Scrivere le query SQL corrispondenti ai seguenti requisiti, ponendo attenzione a non restituire mai duplicati.

1) Per ogni turista calcolare il codice fiscale e le città che ha visitato dopo l'anno 2010.
2) Per ogni turista e per ogni viaggio al quale ha partecipato (un turista partecipa ad un viaggio se partecipa ad almeno una visita di quel viaggio), calcolare il codice fiscale del turista, il codice del viaggio e l'età che il turista aveva al momento del viaggio.
3) Per ogni città calcolare il nome della città ed il numero di turisti minorenni che l'hanno visitata, ma solo se tale numero è maggiore di 1.
4) Per ogni turista e per ogni viaggio calcolare il costo massimo in cui il turista è incorso nelle visite effettuate in quel viaggio, tenendo presente che se il turista non ha partecipato a quel viaggio, il costo dovrà risultare 0.
5) Diciamo che un turista ama una città se l'ha visitata in tutti i viaggi che ha fatto. Per ogni turista che ha fatto almeno un viaggio calcolare il codice fiscale e le città che il turista ama.
6) Per ogni coppia <v, c> dove v è un viaggio e c è una città, calcolare quanti turisti hanno visitato la città c nell'ambito del viaggio v.
7) Calcolare il viaggio per il quale (o i viaggi per i quali) l'incasso derivante da visite di turisti maggiorenni è minimo.
8) Chiamiamo "analoghi" due viaggi in cui sono state visitate le stesse città da parte degli stessi turisti (con costi che possono essere diversi). Calcolare le coppie di viaggi analoghi, evitando nel risultato le coppie ridondanti.
9) Per ogni anno in cui stato effettuato almeno un viaggio, mostrare l'anno e quante città sono state visitate almeno due volte in viaggi effettuati in quell'anno.

Soluzione 5

1) Per ogni turista calcolare il codice fiscale e le città che ha visitato dopo l'anno 2010.

```
select distinct codturista, citta
from visita v join viaggio g on v.codviaggio = g.codice
where g.anno > 2010
```

2) Per ogni turista e per ogni viaggio al quale ha partecipato (un turista partecipa ad un viaggio se partecipa ad almeno una visita di quel viaggio), calcolare il codice fiscale del turista, il codice del viaggio e l'età che il turista aveva al momento del viaggio.

```
select v.codturista, v.codviaggio, g.anno - t.annonascita
from turista t join visita v on t.codicefiscale = v.codturista join viaggio g
      on v.codviaggio = g.codice
```

3) Per ogni città calcolare il nome della città ed il numero di turisti minorenni (età maggiore di 18) che l'hanno visitata, ma solo se tale numero è maggiore di 1.

```
select v.citta, count(distinct codicefiscale)
from turista t join visita v on t.codicefiscale = v.codturista join viaggio g
      on v.codviaggio = g.codice
where v.anno – t.annonascita >= 18
having count(*) > 1
```

4) Per ogni turista e per ogni viaggio calcolare il costo massimo in cui il turista è incorso nelle visite effettuate in quel viaggio (se il turista non ha partecipato ad alcuna visita di quel viaggio, il costo dovrà risultare 0)

```
select codfis, v2.codviaggio,
     CASE WHEN codfis not in (select codturista from visita v where v.codviaggio =
v2.codviaggio) THEN 0
         ELSE max(costo)
            END as massimo
from turista, visita v2
group by codfis, codviaggio
```

oppure

```
select codfis, v1.codviaggio, 0
from turista, visita v1
where codfis not in (select codturista from visita v2 where v2.codviaggio = v1.codviaggio)
union
select codturista, codviaggio, max(costo)
from visita
group by codturista, codviaggio
```

5) Diciamo che un turista ama una città se l'ha visitata in tutti i viaggi che ha fatto. Per ogni turista che ha fatto almeno un viaggio calcolare il codice fiscale e le città che il turista ama.

```
select distinct v.codturista, v.citta
from visita v
where not exists (select *
                  from visita v1
                  where v.codturista = v1.codturista
                        and v.citta not in (select v2.citta
                                            from visita v2
                                            where v2.codturista = v.codturista
                                                  and v2.codviaggio = v1.codviaggio))
```

6) Calcolare i viaggi che hanno avuto l'incasso minimo tra quelli in cui sono state effettuate almeno 2 visite

```
select codice
from viaggio
where incasso = (select min(incasso)
                 from viaggio
                 where codice in (select codviaggio
                                  from visita
                                  group by codviaggio
                                  having count(*) > 2))
```

7) Per ogni coppia <v, c> dove v è un viaggio e c è una città, calcolare quanti turisti hanno visitato la città c nell'ambito del viaggio v.

```
select codviaggio, citta, count(codturista)
from visita
group by codviaggio, citta
union
select codice, citta, 0
from viaggio, visita
where citta not in (select citta from visita where codviaggio = codice)
```

8) Chiamiamo "analoghi" due viaggi in cui sono state visitate le stesse città da parte degli stessi turisti (con costi che possono essere diversi). Calcolare le coppie di viaggi analoghi, evitando nel risultato le coppie ridondanti.

```
select v1.codice, v2.codice
from viaggio v1, viaggio v2
where v1.codice not in (select codviaggio from visita) and
        v2.codice not in (select codviaggio from visita) and
        v1.codice < v2.codice
union
select v1.codice, v2.codice
from viaggio v1, viaggio v2
where v1.codice < v2.codice and
        not exists (select v3.citta,v3.codturista
                    from visita v3
                    where v3.codviaggio = v1.codice
                    except
                    select v4.citta,v4.codturista
                    from visita v4
                   where v4.codviaggio = v2.codice)
         and
         not exists (select v3.citta,v3.codturista
                     from visita v3
                     where v3.codviaggio = v2.codice
                     except
                     select v4.citta,v4.codturista
                     from visita v4
                     where v4.codviaggio = v1.codice))
```

9) Per ogni anno in cui stato effettuato almeno un viaggio, mostrare l'anno e quante città sono state visitate almeno due volte in viaggi effettuati in quell'anno.

```
select anno, 0
from viaggio
where anno not in (select t1.anno
                                    from (select anno from viaggio) t1, (select citta from visita) t2
             where 2 <= (select count(*)
                    from visita v join viaggio g on v.codviaggio = g.codice
                    where g.anno = t1.anno and v.citta = t2.citta))
union
select t.an, count(distinct t.ci)
from (select t1.anno an, t2.citta ci
    from (select anno from viaggio) t1, (select citta from visita) t2
    where 2 <= (select count(*)
            from visita v join viaggio g on v.codviaggio = g.codice
            where g.anno = t1.anno and v.citta = t2.citta)
    ) t
group by t.an
```