1 Esercizio 3

```
--1)
ALTER TABLE COMPORDINE
ADD CONSTRAINT UQ_ArticoloOrdine UNIQUE (CodO, CodA);

--2)
ALTER TABLE LISTINO
ADD CONSTRAINT CK_PrezziQuantita CHECK
  (quantita1>quantita2)
--SE QUANTITA1>QUANTITA2
--ALLORA QUANTITA1/PREZZO1<QUANTITA2/PREZZO2

--COMPORDINE (CODO, CODA, QUANTITA, PREZZO)
--LISTINO (CODA, QUANTITA, PREZZO)

--3)
CREATE ASSERTION vincolo3
CHECK ((SELECT prezzo FROM CompOrdine c WHERE CodA=QualcheCodA)<=(SELECT prezzo FROM Listino l WHERE coda=coda AND c.quantita=l.quantita) )
```

2 Esercizio 5

```

CREATE OR REPLACE FUNCTION es5 RETURNS TRIGGER AS
$$
DECLARE
cursore_ArticoliQuantita CURSOR FOR (SELECT CodA, Quantita
                                                        FROM CompOrdine
                                                        WHERE CodO=NEW.CodO); --cursore per scorrere i codici degli articoli dell'ordine
Quantita_Art_Attuale Magazzino.Quantita%TYPE;
CodA_Attuale Magazzino.CodA%TYPE;
n_articoli_ordine INTEGER:=(SELECT COUNT(*) FROM CompOrdine WHERE CodO=NEW.CodO); --numero di articoli dell'ordine

n_articoli_check INTEGER:=0; --numero di articoli di cui è presente almeno la quantità richiesta
BEGIN
OPEN cursore_ArticoliQuantita;
FOR i IN 1..n_articoli_ordine LOOP
        FETCH cursore_ArticoliQuantita INTO CodA_Attuale, Quantita_Art_Attuale;
        IF EXISTS(SELECT *
                        FROM Magazzino
                        WHERE CodA=CodA_Attuale
                        AND Quantita>=Quantita_Art_Attuale) THEN
                        n_articoli_check=n_articoli_check+1;
        END IF;
END LOOP;

```

```
25 |  IF(n_articoli_ordine<>n_articoli_check)
26 |          UPDATE Ordine SET Completo='N' WHERE CodO=NEW.CodO;
27 |  ELSE IF
28 |  MOVE ABSOLUTE 0 FROM cursore_ArticoliQuantita;
29 |  FOR i IN 1..n_articoli LOOP
30 |          FETCH cursore_ArticoliQuantita INTO CodA_Attuale, Quantita_Art_Attuale;
31 |          UPDATE Magazzino SET Quantita=Quantita-Quantita_Art_Attuale WHERE CodA=CodA_Attuale;
32 |  END LOOP;
33 |  CLOSE cursore_ArticoliQuantita;
34 |  RETURN NEW;
35 |  END;
36 |  $$
37 |  LANGUAGE PLPGSQL;
38 |
39 |  CREATE TRIGGER trig_es5
40 |  BEFORE UPDATE ON b.Ordine
41 |  WHEN (Completo='S')
42 |  EXECUTE FUNCTION es5;
```

3 Esercizio con i Trigger

```
01 |  CREATE TRIGGER cancella_prenotazioni
02 |      AFTER INSERT OR UPDATE
03 |      ON prestito
04 |      FOR EACH ROW
05 |      WHEN (articolo.datapubblicazione IS NOT NULL)
06 |  EXECUTE FUNCTION b.prova;
07 |
08 |  CREATE FUNCTION b.prova() RETURNS TRIGGER AS
09 |  $$
10 |  DECLARE
11 |      id_utente prestito.utente%TYPE=(SELECT UTENTE
12 |                                      FROM PRESTITO
13 |                                      WHERE codprestito = NEW.codprestito);
14 |  BEGIN
15 |      DELETE FROM prenotazione p WHERE p.utente = id_utente;
16 |  end;
17 |  $$
18 |      language plpgsql;
19 |
20 |
21 |
22 |  CREATE TRIGGER richieste_prestito
23 |      AFTER INSERT
24 |      ON prestito
25 |      FOR EACH ROW
```

```
26 |  DECLARE
27 |      v_ISBN libro.ISBN%TYPE = (SELECT ISBN
28 |      FROM prestito p
29 |      JOIN esemplare e
30 |      WHERE p.codicebarre = e.codicebarre
31 |      AND codprestito = NEW.codprestito);
32 |  BEGIN
33 |  DELETE
34 |  from PRENOTAZIONE
35 |  WHERE utente = NEW.utente
36 |    AND ISBN = v_ISBN;
37 |  END;
38 |  $$
```

4 Traccia del 27-03-2015

Si consideri il seguente schema relazionale che descrive un frammento della base di dati per gestire i prestiti in una biblioteca. Di un libro (descrizione del libro) possono esserci pi‘u copie fisiche (esemplari).

L’attributo booleano Prestito indica se l’esemplare pu‘o essere prestato.

L’attributo booleano Consultazione indica se l’esemplare pu‘o essere consultato in loco.

Il prestito riguarda le copie fisiche, la prenotazione riguarda il libro: quando un esemplare del libro è disponibile si può effettuare il prestito.

Ogni utente ha un profilo che regola la durata dei prestiti e il massimo numero di esemplari che può avere in prestito.

Un prestito ha una data di effettuazione, una data di scadenza e una data di restituzione (che pu‘o essere NULL se il libro non ‘e stato ancora restituito), una data di sollecito (che pu‘o essere NULL se il prestito non ‘e ancora scaduto.)

LIBRO(ISBN, Titolo, Editore, Anno)

ESEMPLARE(ISBN, CodiceBarre, Collocazione, Prestito, Consultazione)

UTENTE(CF, CodProfilo, Nome, Cognome, DataN)

PROFILO(CodProfilo, MaxDurata, MaxPrestito)

PRESTITI(CodPrestito, CodiceBarre, Utente, Data, Scadenza, Restituzione, Sollecito)

PRENOTAZIONE(CodPrenotazione, ISBN, Utente, Data)

Esercizio 1 (7 punti)

Si scriva una interrogazione in algebra relazionale che, se valutata, fornisce l’ISBN del libro che ha avuto nell’anno 2014 il maggior numero di prestiti (si intente per numero di prestiti di un libro il numero di prestiti di tutti i suoi esemplari).

Esercizio 2 (8 punti)

Si scriva una interrogazione in SQL che restituisca coppie di utenti che in ogni anno hanno preso in prestito lo stesso numero di libri.

Esercizio 3 (8 punti)

Si implementino nel modo più adeguato i seguenti vincoli:

1. Un utente non può avere più prestiti in corso (esemplari non ancora restituiti) di quanto
2. previsto dal suo profilo.
3. Se è presente un sollecito la restituzione è stata fatta dopo la data di scadenza.
4. Quando viene aggiornato un prestito indicando una data di sollecito, tutte le prenotazioni
5. quell'utente vengo automaticamente cancellate.

Esercizio 4 (8 punti)

Si scriva una funzione che, quando viene eseguita, controlla quali sono i prestiti che hanno raggiunto la scadenza alla data dell'esecuzione. L'effetto della funzione di inserire nel campo Sollecito la data corrente.

Inoltre, la funzione restituisce alla terminazione una stringa contenente, CF, Nome, Cognome, Titolo del libro per tutti i prestiti in ritardo (separati da ;) per cui stato indicato il sollecito.

Esercizio 5 (8 punti)

Si scriva una funzione che riceve in ingresso una stringa contenete delle parole separate tra loro dal simbolo −.

Si scriva una interrogazione in SQL dinamico che recupera i libri in cui almeno una delle parole della stringa compare nel titolo. La funzione restituisce una stringa contenente i titoli dei libri recuperati separati dal simbolo −

```
01 |  drop schema u;
02 |  create schema u;
03 |
04 |  create table u.libro(
05 |      isbn    varchar(10)
06 |          primary key,
07 |      titolo  varchar(20),
08 |      editore varchar(20),
09 |      anno    varchar(4)
10 |  );
11 |
12 |  create table u.esemplare(
13 |      ISBN          varchar(10),
14 |      codicebarre   varchar(20)
15 |          primary key,
16 |      collocazione  varchar(10),
```

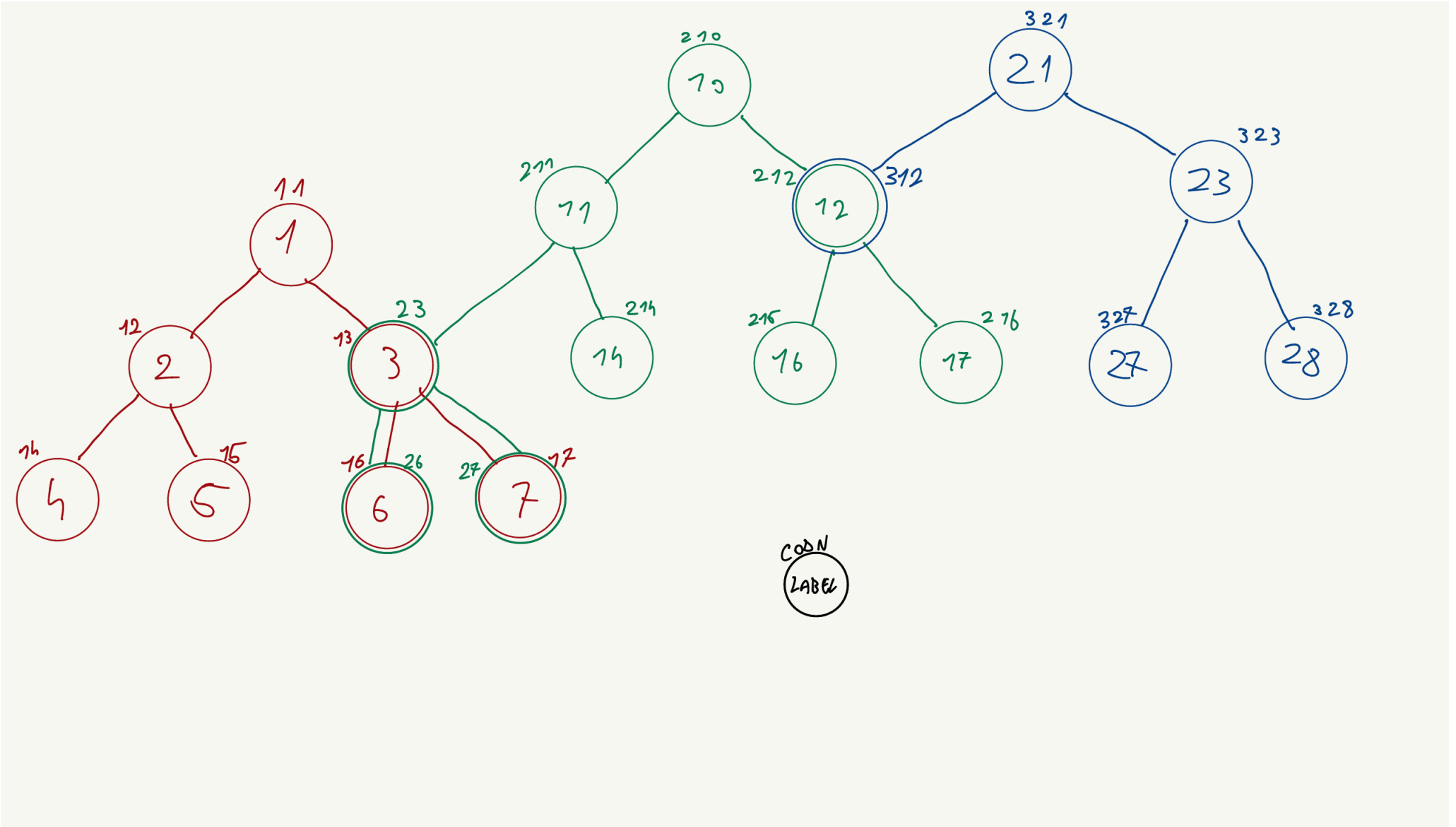
```
17 |     prestito      boolean,
18 |     consultazione boolean,
19 |     constraint fk_libro foreign key (ISBN) references u.libro(isbn)
20 | );
21 |
22 |
23 | create table u.profilo(
24 |     codprofilo  varchar(5)
25 |         primary key,
26 |     maxdurata   int,
27 |     maxprestito int
28 | );
29 |
30 | create table u.utente(
31 |     cf         varchar(10) primary key ,
32 |     codprofilo varchar(5),
33 |     nome       varchar(15),
34 |     cognome    varchar(15),
35 |     data      date,
36 |
37 |     constraint fk_profilo foreign key (codprofilo) references u.profilo(codprofilo)
38 | );
39 | create table u.prestiti(
40 |     codprestito  serial primary key,
41 |     codbarre     varchar(20),
42 |     utente       varchar(10),
43 |     data         date,
44 |     scadenza     date,
45 |     restituzione date,
46 |     sollecito    date,
47 |
48 |     constraint fk_esemplare foreign key (codbarre) references u.esemplare(codicebarre),
49 |     constraint fk_utente foreign key (utente) references u.utente(cf)
50 | );
51 | create table u.prenotazione(
52 |     codprenotazione serial,
53 |     isbn            varchar(10),
54 |     utente          varchar(10),
55 |     data            date,
56 |     constraint fk_libro foreign key (isbn) references u.libro(isbn),
57 |     constraint fk_utente foreign key (utente) references u.utente(cf)
58 | );
59 |
60 | insert into u.libro (isbn, titolo, editore, anno)
61 | VALUES ('978-88-17-00000-0', 'Il Signore degli Anelli', 'Mondadori', '1954');
62 |
63 | insert into u.esemplare(isbn, codicebarre, collocazione, prestito, consultazione)
64 | VALUES ('978-88-17-00000-0', '9788817000000', 'A1', false, false);
65 |
```

```
insert into u.profilo(codprofilo, maxdurata, maxprestito)
VALUES ('A', 30, 3);

insert into u.utente(cf, codprofilo, nome, cognome, data)
VALUES ('RSSMRA80A01F205X', 'A', 'Mario', 'Rossi', '1980-01-01');

insert into u.prestiti(codbarre, utente, data, scadenza, restituzione, sollecito)
VALUES ('9788817000000', 'RSSMRA80A01F205X', '2018-01-01', '2018-01-31', null, false);
```

```
create function u.funtion_1() returns varchar
as
$$
declare
    output_finale        text;
    data_corrente        u.prestiti.sollecito%TYPE;
    risultato_query      record;
    cursore_2 cursor for (select *
                          from prestiti);
    query_utente_nome    varchar(1000);
    query_utente_cognome varchar(1000);
    query_libro_titolo   varchar(1000);
    nome_utente          u.utente.nome%TYPE;
    cognome_utente       u.utente.cognome%TYPE;
    titolo_libro         u.libro.titolo%TYPE;


begin
    open cursore_2;
        loop
            fetch cursore_2 into risultato_query;
            exit when not found;
            if (risultato_query.scadenza > data_corrente) then
                update u.prestiti set sollecito=True where codprestito=risultato_query.codprestito;
                query_utente_nome = 'select ut.nome' ||
                                    'from u.presito p join u.utente ut on ut.cf='
                                        || '''' || risultato_query.utente || '''';
                execute query_utente_nome into nome_utente;
                query_utente_cognome = 'select ut.cognome' ||
                                       'from u.presito p join u.utente ut on u.cf='
                                           || '''' || risultato_query.utente || '''';
                execute query_utente_cognome into cognome_utente;
                query_libro_titolo = 'select l.titolo' ||
                                     'from (u.presito p join u.esemplare e on p.codbarre=' ||
                                     'e.codbarre) join u.libro l on e.isbn = l.isbn' ||
                                     'where p.codprestito=' || '''' || risultato_query.codprestito || '''';
                execute query_libro_titolo into titolo_libro;
                output_finale = risultato_query.utente || nome_utente ||
                                cognome_utente || titolo_libro;
```

```
40 |
41 |             end if;
42 |         end loop;
43 | end
44 | $$
45 |     language plpgsql;
```

5 Alberi/Nodi



5.1 Esercizio 1

ALBERO (CodA, root)

NODO (codA, codN, label)

ARCO (codA, codArco, padre, figlio)

Scrivere una funzione somma_nodi che riceve in input un codice di albero ed un codice nodo e restituisce la somma delle label di tutti i nodi che vanno dal nodo input fino alla radice.

```
01 |  DROP SCHEMA IF EXISTS e CASCADE;
02 |  CREATE SCHEMA e;
03 |
04 |  CREATE TABLE e.Nodo
05 |  (
06 |      CodA  INTEGER, --FK
07 |      CodN  INTEGER, --PK
08 |      Label INTEGER,
09 |
10 |      CONSTRAINT Nodo_PK PRIMARY KEY (CodN)
11 |  );
12 |
13 |  CREATE TABLE e.Albero
14 |  (
15 |      CodA INTEGER,
16 |      root INTEGER,
17 |
18 |      CONSTRAINT Albero_PK PRIMARY KEY (CodA),
19 |      CONSTRAINT Nodo_PK FOREIGN KEY (root) REFERENCES e.Nodo (CodN)
20 |  );
21 |
22 |  CREATE TABLE e.Arco
23 |  (
24 |      CodA    INTEGER, --FK
25 |      CodArco INTEGER, --PK
26 |      Padre   INTEGER, --FK
27 |      Figlio  INTEGER, --FK
28 |
29 |      CONSTRAINT Arco_PK PRIMARY KEY (CodArco),
30 |      CONSTRAINT Albero_FK FOREIGN KEY (CodA) REFERENCES e.Albero (CodA),
31 |      CONSTRAINT Padre_FK FOREIGN KEY (Padre) REFERENCES e.Nodo (CodN),
32 |      CONSTRAINT Figlio_FK FOREIGN KEY (Figlio) REFERENCES e.Nodo (CodN)
33 |  );
34 |
35 |  ALTER TABLE e.Nodo
36 |      ADD CONSTRAINT Albero_FK FOREIGN KEY (CodA) REFERENCES e.Albero (CodA);
37 |
38 |  INSERT INTO e.Albero(coda)
```

```
39 |      (values (1),
40 |              (2),
41 |              (3));
42 |
43 |  INSERT INTO e.Nodo(codn, coda, label) --nel disegno i numeri dei nodi corrispondono ai label
44 |      ( --albero 1
45 |          values (11, 1, 1),
46 |                 (12, 1, 2),
47 |                 (13, 1, 3),
48 |                 (14, 1, 4),
49 |                 (15, 1, 5),
50 |                 (16, 1, 6),
51 |                 (17, 1, 7),
52 |                 --albero 2
53 |                 (210, 2, 10),
54 |                 (211, 2, 11),
55 |                 (212, 2, 12),
56 |                 (23, 2, 3),
57 |                 (26, 2, 6),
58 |                 (27, 2, 7),
59 |                 (214, 2, 14),
60 |                 (215, 2, 15),
61 |                 (216, 2, 16),
62 |                 --albero 3
63 |                 (321, 3, 21),
64 |                 (312, 3, 12),
65 |                 (315, 3, 15),
66 |                 (316, 3, 16),
67 |                 (323, 3, 23),
68 |                 (327, 3, 27),
69 |                 (328, 3, 28));
70 |
71 |  UPDATE e.Albero
72 |  SET root=11
73 |  WHERE CodA = 1;
74 |
75 |  UPDATE e.albero
76 |  SET root=210
77 |  WHERE CodA = 2;
78 |
79 |  UPDATE e.albero
80 |  SET root=321
81 |  WHERE CodA = 3;
82 |
83 |  INSERT INTO e.Arco(CodA, codarco, padre, figlio)
84 |      ( --ALBERO 1
85 |          values (1, 112, 11, 12),
86 |                 (1, 113, 11, 13),
87 |                 (1, 124, 12, 14),
```

```
88 |                 (1, 125, 12, 15),
89 |                 (1, 136, 13, 16),
90 |                 (1, 137, 13, 17),
91 |                 --ALBERO 2
92 |                 (2, 21011, 210, 211),
93 |                 (2, 21012, 210, 212),
94 |                 (2, 2113, 211, 23),
95 |                 (2, 21114, 211, 214),
96 |                 (2, 21215, 212, 215),
97 |                 (2, 21216, 212, 216),
98 |                 (2, 236, 23, 26),
99 |                 (2, 237, 23, 27),
100 |                --ALBERO 3
101 |                (3, 32112, 321, 312),
102 |                (3, 32123, 321, 323),
103 |                (3, 31215, 312, 315),
104 |                (3, 31216, 312, 316),
105 |                (3, 32327, 323, 327),
106 |                (3, 32328, 323, 328));
```

5.2 Esercizio 2

ALBERO (CodA, root)

NODO (codA, codN, label)

ARCO (codA, codArco, padre, figlio)

Usando i cursori e la funziona somma_nodi sviluppata nell'esercizio1, scrivere una funzione somma_max che, dato un albero, trovi il valore del cammino di somma massima dalle foglie alle radici.

```
01 |  CREATE OR REPLACE FUNCTION e.somma_nodi(albero e.Albero.CodA%TYPE, nodo_input e.Nodo.CodN%TYPE)
02 |      RETURNS INT
03 |  AS
04 |  $$
05 |  DECLARE
06 |      somma_nodi INTEGER            := 0;
07 |      root_a     e.Albero.root%TYPE;
08 |      padre      e.Arco.padre%TYPE;
09 |      cursore    e.Arco.figlio%TYPE := nodo_input;
10 |      labelv     e.Nodo.label%TYPE  := 0; --variabile label
11 |  BEGIN
12 |      SELECT root, N.label --trovo radice dell'albero e relativo label
13 |      into root_a, labelv --e lo salvo in due variabili distinte
14 |      FROM e.Albero AS A
15 |               NATURAL JOIN e.Nodo AS N
16 |      WHERE albero = A.coda;
```

```
17 |      somma_nodi = somma_nodi + labelv; --sommo il label della radice
18 |
19 |      WHILE cursore != root_a
20 |          LOOP
21 |              --salgo di nodo in nodo fino a raggiungere la radice precedentemente trovata
22 |              SELECT A.padre INTO padre
23 |              FROM e.Nodo AS N
24 |                       NATURAL JOIN e.Arco AS A
25 |              WHERE figlio = cursore;
26 |              SELECT N.label INTO labelv
27 |              FROM e.Nodo AS N
28 |              WHERE cursore = N.codn;
29 |              somma_nodi := somma_nodi + labelv; --sommo i label dei nodi
30 |              cursore = padre;
31 |          END LOOP;
32 |
33 |      RETURN somma_nodi;
34 |  END
35 |  $$
36 |      LANGUAGE plpgsql;
37 |
38 |  SELECT e.somma_nodi(2, 23);
39 |
40 |
41 |  CREATE or REPLACE FUNCTION e.max_cammino(cod_albero e.albero.codA%TYPE)
42 |      RETURNS INT
43 |  AS
44 |  $$
45 |  DECLARE
46 |      foglia e.Nodo.CodN%TYPE; --variabile in cui conserviamo il CodN di una foglia
47 |      max    INTEGER := 0; --max valore del percorso dalla foglia alla radice
48 |      i      INTEGER := 0; --indice per loop
49 |      n      INTEGER := 0; --numero di foglie dell'albero (da usare come max del loop)
50 |      cursore CURSOR FOR --cursore che scorre tutti i nodi senza figli(foglie)
51 |          (SELECT N.CodN --tutti i nodi
52 |           FROM e.Nodo AS N
53 |           WHERE coda = cod_albero
54 |           EXCEPT --meno
55 |           SELECT A.padre --nodi con figli
56 |           FROM e.Arco AS A);
57 |
58 |  BEGIN
59 |      n := (SELECT COUNT(*) --numero di foglie dell'albero
60 |            FROM (SELECT N.CodN
61 |                  FROM e.Nodo AS N
62 |                  WHERE coda = cod_albero
63 |                  EXCEPT
64 |                  SELECT A.padre
65 |                  FROM e.Arco AS A) AS Q);
```

```
66 |
67 |      OPEN cursore;
68 |
69 |      WHILE i < n
70 |          LOOP
71 |              FETCH cursore INTO foglia;
72 |              IF (max < e.somma_nodi(cod_albero, foglia)) THEN
73 |                  max = e.somma_nodi(cod_albero, foglia);
74 |              END IF;
75 |              i := i + 1;
76 |          END LOOP;
77 |
78 |      CLOSE cursore;
79 |      RETURN max;
80 |  END;
81 |
82 |  $$
83 |      LANGUAGE plpgsql;
```

6 Autocheck (08-12-2022)

Si consideri il seguente schema relazionale che descrive un frammento della base di dati per gestire le informazioni relative a viaggi in autostrada.

Nella tabella tariffe sono indicati i costi dal casello di ingresso al casello di uscita per una determinata categoria di automobili. Un viaggio inizia da un casello in ingresso nel giorno dataI al tempo TempoI e finisce nel casello Uscita nel giorno DataF al tempo TempoF.

Per i viaggi iniziati ma non ancora conclusi DataF, TempoF, Uscita e Tariffa sono NULL.

La tabella CHECK contiene le rilevazioni dei tutor fatte al passaggio dell'automobile (identificata dalla targa) nel tragitto. Ogni tutor ha una velocit‘a massima consentita descritta nella tabella PCHECK.

Se non vi sono infrazioni il campo Infrazione ha valore NULL.

AUTO(Targa, CF P, Categoria)

TARIFFE(Ingresso, Uscita, KM, Categoria, Costo)

PCHECK(PuntoCheck, V elocitaMax)

CHECK(PuntoCheck, Targa, V elocita, Data, Tempo, Infrazione)

V IAGGIO(CodV , Targa, DataI, DataF, TempoI, TempoF, Ingresso, Uscita, Tariffa, KM)

Esercizio 01 (Punti 6), 15 minuti

Si scriva il seguente trigger. Quando viene inserito un check per un viaggio si controlla se la velocit‘a rilevata è superiore alla velocit‘a massima. Se è superiore, si pone a TRUE il campo infrazione del CHECK.

Esercizio 02 (Punti 6, 15 minuti)

Si scriva il seguente trigger. Quando viene aggiornato un viaggio esprimendo un valore per il casello di uscita si aggiornano anche gli attributi Km e Tariffaa recuperando i valori dalla tabella TARIFFE (la tariffa dipende da ingresso, uscita e categoria dell'auto).

Creazione DB

```
01 |  CREATE SCHEMA v;
02 |
03 |
04 |  CREATE TABLE v.TARIFFE
05 |  (
06 |      Ingresso  VARCHAR(32) NOT NULL,
07 |      Uscita    VARCHAR(32) NOT NULL,
08 |      KM        INTEGER NOT NULL,
09 |      Categoria INTEGER NOT NULL,
10 |      Costo     DOUBLE PRECISION NOT NULL,
11 |      CONSTRAINT PK_TARIFFE PRIMARY KEY (Ingresso, Uscita, Categoria)
12 |  );
13 |
14 |  CREATE TABLE v.AUTO
15 |  (
16 |      Targa     VARCHAR(10) NOT NULL,
17 |      CODFIS    VARCHAR(16) NOT NULL,
18 |      Categoria INTEGER NOT NULL,
19 |      CONSTRAINT PK_AUTO PRIMARY KEY (Targa)
20 |  );
21 |
22 |
23 |  CREATE TABLE v.PCHECK
24 |  (
25 |      PuntoCheck    VARCHAR(10) NOT NULL,
26 |      VelocitaMax   INTEGER NOT NULL,
27 |
28 |      CONSTRAINT PK_PCHECK PRIMARY KEY (PuntoCheck)
29 |  );
30 |
31 |  CREATE TABLE v.CHECK
32 |  (
33 |      PuntoCheck VARCHAR(10) NOT NULL,
34 |      Targa      VARCHAR(10) NOT NULL,
35 |      Velocita   INTEGER NOT NULL,
36 |      Data      DATE NOT NULL,
37 |      Tempo      TIME NOT NULL,
38 |      Infrazione BOOLEAN,
39 |
40 |      CONSTRAINT PK_CHECK PRIMARY KEY (PuntoCheck, Targa, Data, Tempo),
41 |      CONSTRAINT FK_PCHECK FOREIGN KEY (PuntoCheck) REFERENCES v.PCHECK (PuntoCheck),
42 |      CONSTRAINT FK_AUTO FOREIGN KEY (Targa) REFERENCES v.AUTO (Targa)
```

```
);

CREATE TABLE v.VIAGGIO
(
    CodiceViaggio VARCHAR(10) NOT NULL,
    Targa         VARCHAR(10) NOT NULL,
    DataI         DATE NOT NULL,
    DataF         DATE,
    OraI          TIME NOT NULL,
    OraF          TIME,
    Ingresso      VARCHAR(32) NOT NULL,
    Uscita        VARCHAR(32),
    Tariffa       DOUBLE PRECISION,
    KM            INTEGER,

    CONSTRAINT PK_VIAGGIO PRIMARY KEY (CodiceViaggio),
    CONSTRAINT FK_VIAGGIO_AUTO FOREIGN KEY (Targa) REFERENCES v.AUTO (Targa),
    CONSTRAINT CK_VIAGGIO CHECK (DataF > DataI OR (DataF = DataI AND OraF > OraI)),
    CONSTRAINT CK_VIAGGIO2 CHECK (KM > 0),
    CONSTRAINT CK_VIAGGIO3 CHECK (Tariffa > 0)
);
```

Inserimento

```
INSERT INTO v.AUTO(targa, codfis, categoria) values (1, 1, 1),
                                                    (2, 2, 2);


INSERT INTO v.PCHECK(puntocheck, velocitamax) values ('prova', 10);
INSERT INTO v.PCHECK(puntocheck, velocitamax) values ('prova1', 10);

INSERT INTO v.TARIFFE(ingresso, uscita, km, categoria, costo) values ('ingresso1_cat1','uscita1_cat1', 10, 1, 10),
                                                                     ('ingresso1_cat2','uscita1_cat2', 10, 2, 20),
                                                                     ('ingresso2_cat1','uscita2_cat1', 20, 1, 11),
                                                                     ('ingresso2_cat2','uscita2_cat2', 20, 2, 22);



INSERT INTO v.CHECK(puntocheck, targa, velocita, data, tempo) VALUES ('prova', 1, 9, '2001-01-01', '00:30');
INSERT INTO v.CHECK(puntocheck, targa, velocita, data, tempo) VALUES ('prova1', 1, 11, '2001-01-01', '00:45');

INSERT INTO v.CHECK(puntocheck, targa, velocita, data, tempo)
    VALUES  ('prova', '2', 11,'2007-07-07','00:45'),
            ('prova1', '2', 9,'2007-07-07','00:40'),
            ('prova1', '2', 11,'2007-12-07','01:50');

INSERT INTO v.CHECK(puntocheck, targa, velocita, data, tempo)
```

```
25 |  values ('prova', '1', 9, '2001-01-01', '00:00:00'); -- non infrangono
26 |
27 |  INSERT INTO v.CHECK(puntocheck, targa, velocita, data, tempo)
28 |  values ('prova', '1', 110, '2001-01-01', '00:00:10'); -- infrangono
29 |
30 |  INSERT INTO v.VIAGGIO(CODICEVIAGGIO, TARGA, DATAI, ORAI, INGRESSO)
31 |  (
32 |      values (1, 1, '2001-01-01','01:01', 'ingresso1_cat1'),
33 |             (2, 1, '2001-01-01', '11:11', 'ingresso2_cat1'),
34 |             (3, 2, '2012-12-12', '12:12', 'ingresso1_cat2'),
35 |             (4, 2, '2022-12-22', '22:22', 'ingresso2_cat2')
36 |  );
37 |
38 |  UPDATE v.viaggio
39 |  SET    uscita='uscita1_cat1'
40 |  WHERE codiceviaggio='1';
41 |
42 |  UPDATE v.viaggio
43 |  SET    uscita='uscita2_cat1'
44 |  WHERE  codiceviaggio='2';
45 |
46 |  UPDATE v.viaggio
47 |  SET    uscita='uscita1_cat2'
48 |  WHERE codiceviaggio='3';
49 |
50 |  UPDATE v.viaggio
51 |  SET    uscita='uscita2_cat2'
52 |  WHERE codiceviaggio='4';
53 |
54 |  UPDATE v.viaggio
55 |  SET    uscita='uscita2_cat2'
56 |  WHERE codiceviaggio='5';
57 |
58 |  INSERT INTO V.CHECK(puntocheck, targa, velocita, data, tempo)
59 |  values ('prova', '1', 30, '2022-12-14', '12:05');
```

Trigger (es 1)

```
01 |  SELECT codiceviaggio, targa, V.ingresso, V.uscita, tariffa, V.km
02 |  FROM v.viaggio AS V NATURAL JOIN v.AUTO AS A, v.Tariffe AS T
03 |  WHERE T.ingresso=V.ingresso
04 |
05 |  CREATE OR REPLACE FUNCTION v.setInfraction() RETURNS trigger AS
06 |  $$
07 |      DECLARE
08 |          MAXvelocita v.pcheck.velocitaMAX%TYPE;
09 |          BEGIN
10 |              SELECT velocitaMAX INTO MAXvelocita
```

```
                FROM v.pcheck WHERE puntocheck = NEW.puntocheck;

            IF NEW.velocita > MAXvelocita THEN
                UPDATE v.check
                SET infrazione = TRUE -- l'infrazione esiste
                WHERE puntocheck = NEW.puntocheck
                    AND targa = NEW.targa
                    AND velocita=NEW.velocita
                    AND data=NEW.data
                    AND tempo=NEW.tempo;
            END IF;
            RAISE NOTICE 'Infrazione: %', new.infrazione;
        RETURN NULL;
        END
$$  LANGUAGE plpgsql;

CREATE or REPLACE TRIGGER Infractions AFTER INSERT ON v.check
FOR EACH ROW
EXECUTE PROCEDURE v.setInfraction();

-----------------------------------------------------------------------------------------------------------------------
-- Insert per testare il trigger
INSERT INTO v.CHECK(puntocheck, targa, velocita, data, tempo)
values ('prova', '1', 9, '2001-01-01', '00:00:00'); -- non infrangono

INSERT INTO v.CHECK(puntocheck, targa, velocita, data, tempo)
values ('prova', '1', 110, '2001-01-01', '00:00:10'); -- infrangono
```

Trigger (es 2)

```
CREATE OR REPLACE FUNCTION v.update_viaggio() RETURNS trigger AS
    $$
    DECLARE
    prezzo v.tariffe.costo%TYPE;
    chilometraggio v.viaggio.KM%TYPE;

    BEGIN
        IF (NEW.uscita IS NOT NULL) THEN --quando viene inserita l'uscita
            SELECT T.costo, T.KM         --salvo il prezzo e i km relativi al viaggio
            INTO --prezzo, chilometraggio
            NEW.KM, NEW.Tariffa
        FROM V.AUTO AS A NATURAL JOIN V.viaggio AS V,
             v.tariffe AS T
        WHERE (NEW.uscita=T.uscita)
        AND   (OLD.ingresso=T.ingresso)
        AND   (T.categoria=A.categoria);

            --UPDATE v.viaggio
```

```
19 |              --SET km=chilometraggio, tariffa=prezzo
20 |            --  WHERE OLD.codiceviaggio=codiceviaggio;
21 |              --RAISE NOTICE 'Old_Viaggio: %', OLD.codiceviaggio;
22 |          END IF;
23 |          RETURN NEW;
24 |      EXCEPTION
25 |      WHEN OTHERS THEN
26 |          RAISE EXCEPTION 'Errore';
27 |      END
28 |
29 |      $$
30 |  LANGUAGE plpgsql;
31 |
32 |
33 |  CREATE OR REPLACE TRIGGER TRIGGER_2_prof BEFORE UPDATE OF uscita ON v.viaggio
34 |      FOR EACH ROW
35 |      EXECUTE PROCEDURE v.update_viaggio();
```

7 Hashtag Foto (08-12-2022)

Si consideri il seguente schema relazionale che descrive un frammento della base di dati per gestire depositi di immagini condivise tra utenti.

Le foto sono contenute in album.

Un album può contenere foto ed altri album (la struttura degli album è analoga a quella di un filesystem con file e cartelle).

Ciascuna foto e ciscun album ha un proprietario (owner) espresso da un identificativo di utente.

Nella tabella ALBUM l'attributo InAlbum è l'identificativo dell'album contenete (si assuma che un album alla radice della gerarchia di contenimento è contenuto nell'album di sistema di identificativo SYSALBUM).

Gli attributi Aggiunta e Rimossa riportano la data di inserimento della foto e la eventuale data di rimozione (attributo parziale). Foto ed album possono avere dei tag associati. L'elenco dei tag ammissibili si trova nella tabella HASHTAG.

I tag associati alle foto e agli album si trovano nelle tabelle TAGFOTO e TAGALBUM rispettivamente.

La tabella VISIBLE indica invece l'accessibilità dei file ai vari utenti.

Il proprietario CodProp dell'album codA concede il diritto di visualizzazione a CodUt su tutte le foto contenute in codA. Infine, nella tabella LOG vengono registrate tutte le operazioni fatte sulle foto dagli utenti: codF è la foto su cui si fa l'operazione, CodU l'utente che fa l'operazione ed Operation il tipo di operazione (ad es. inserimento, cancellazione, visualizzazione).

HASHTAG(parola)

FOTO(CodF, uri, nome, titolo, owner, CodAlbum, Aggiunta, Rimossa)

ALBUM(CodA, nome, titolo, owner, InAlbum)

UTENTE(CodU, nome, cognome, email)

TAGFOTO(CodF, parola)

TAGALBUM(CodA, parola)

VISIBLE(CodProp, CodUt, CodA)

LOG(CodF, CodU, Time, Operation).

Esercizio 05 (Punti 10)

Si scriva una procedura PLSQL che riceve in ingresso l'identificativo di un album e che restituisce una stringa contenete tutti i tag associati all'album e agli album in esso contenuti (ad ogni livello di profondità) senza ripetizioni.

Si consiglia di avvalersi di una tabella TMP(CodA) (che si suppone già definita) dove memorizzare preventivamente l'albero degli album radicato nell'album passato come parametro.

Esercizio 06 (Punti 7)

Usando SQL DINAMICO si scriva una funzione che riceve in ingresso una lista di tag separati dal carattere @ e che restituisce una stringa degli uri delle foto (separati da @) a cui sono associati tutti i tag passati per parametro.

Creazione DB

```
01 |  xDROP SCHEMA f CASCADE;
02 |  CREATE SCHEMA f;
03 |
04 |  CREATE TABLE f.hashtag (
05 |      Parola VARCHAR(50),
06 |
07 |      CONSTRAINT hashtag_pk PRIMARY KEY (parola)
08 |  );
09 |
10 |  CREATE TABLE f.utente(
11 |      CodU INTEGER,
12 |      Nome VARCHAR(50),
13 |      Cognome VARCHAR(50),
14 |      email VARCHAR(50),
15 |      CONSTRAINT utente_pk PRIMARY KEY (codU)
16 |  );
17 |
18 |  CREATE TABLE f.album(
19 |      CodA INTEGER,
20 |      Nome VARCHAR(50),
21 |      Titolo VARCHAR(50),
22 |      Owner INTEGER,
23 |      InAlbum INTEGER,
24 |
25 |      CONSTRAINT album_pk PRIMARY KEY (codA),
```

```
    CONSTRAINT utente_fk FOREIGN KEY (owner) REFERENCES f.utente(codU)
);

CREATE TABLE f.foto (
    Codf SERIAL,
    Uri VARCHAR(100),
    Titolo VARCHAR(50),
    Owner INTEGER,
    CodAlbum INTEGER,
    Aggiunta TIMESTAMP,
    Rimossa TIMESTAMP,

    CONSTRAINT foto_pk PRIMARY KEY (codf),
    CONSTRAINT album_fk FOREIGN KEY (codalbum) REFERENCES f.album(codA),
    CONSTRAINT utente_fk FOREIGN KEY (owner) REFERENCES f.utente(codU)
);

CREATE TABLE f.tagfoto (
    CodF INTEGER,
    Parola VARCHAR(50),

    CONSTRAINT tagfoto_pk PRIMARY KEY (codf, parola),
    CONSTRAINT foto_fk FOREIGN KEY (codf) REFERENCES f.foto(codf),
    CONSTRAINT hashtag_fk FOREIGN KEY (parola) REFERENCES f.hashtag(parola)
);

CREATE TABLE f.visibile(
    CodProp INTEGER,
    CodUt INTEGER,
    CodA INTEGER,

    CONSTRAINT visibile_pk PRIMARY KEY (codprop, codut, coda),
    CONSTRAINT proprietario_fk FOREIGN KEY (codprop) REFERENCES f.utente(codU),
    CONSTRAINT utente_fk FOREIGN KEY (codut) REFERENCES f.utente(codU),
    CONSTRAINT album_fk FOREIGN KEY (coda) REFERENCES f.album(codA)
);

CREATE TABLE f.log(
    CodU INTEGER,
    CodF INTEGER,
    Time TIMESTAMP,
    Operation VARCHAR(50),

    CONSTRAINT log_pk PRIMARY KEY (codu, codf, time),
    CONSTRAINT utente_fk FOREIGN KEY (codu) REFERENCES f.utente(codU),
    CONSTRAINT foto_fk FOREIGN KEY (codf) REFERENCES f.foto(codf)
);

CREATE TABLE f.tagalbum
```

```
(
    CodA INTEGER,
    Parola VARCHAR(50),

    CONSTRAINT tagalbum_pk PRIMARY KEY (coda, parola),
    CONSTRAINT album_fk FOREIGN KEY (coda) REFERENCES f.album(codA),
    CONSTRAINT hashtag_fk FOREIGN KEY (parola) REFERENCES f.hashtag(parola)
);
```

Esercizio n° 5

```
CREATE OR REPLACE PROCEDURE f.f2_figli(tagalbum f.album.coda%TYPE) AS $$
    DECLARE
        figlio f.album.coda%TYPE;
    BEGIN
        FOR figlio IN (SELECT codA FROM f.album AS A WHERE A.inalbum=tagalbum) LOOP

            CALL f.f2_figli(figlio);
            RAISE NOTICE 'figlio: %', figlio;
            INSERT INTO f.tmp(codA) values (figlio);
        end loop;
    end
$$ language plpgsql;


CREATE OR REPLACE FUNCTION f.f1_rec(IN Input f.album.coda%type)RETURNS VARCHAR(500) AS $$
    DECLARE
        stringa f.tagalbum.parola%TYPE;
        output VARCHAR(500);
        counter INTEGER = 1;
        BEGIN
            CREATE TABLE f.tmp(codA INTEGER);
            INSERT INTO f.tmp VALUES (Input);
            CALL f.f2_figli(Input);
            FOR stringa IN (SELECT DISTINCT parola FROM f.tmp NATURAL JOIN f.tagalbum) LOOP
                IF(counter>1) THEN
                    output=CONCAT(output, ', ', stringa);
                ELSE
                    output=stringa;
                END IF;
                    counter = counter + 1;
            END LOOP;
            DROP TABLE f.tmp CASCADE;
            RETURN output;
        END
$$ LANGUAGE plpgsql;

SELECT f.f1_rec(1);
```

Esercizio n° 6

```
01 | CREATE OR REPLACE FUNCTION f.f_sqlDYN(IN stringa text) RETURNS text AS $$
02 |     DECLARE
03 |         strQRY text;
04 |         numParoleSTR INTEGER = regexp_count(stringa, '@') + 1;
05 |         parolaSTR f.tagfoto.parola%TYPE;
06 |         count INTEGER = 0;
07 |         selectQRY text = 'SELECT uri FROM f.tagfoto as t NATURAL JOIN f.foto WHERE t.parola = ';
08 |         cursQRY refcursor;
09 |         output text;
10 |         correctURI f.foto.uri%TYPE;
11 |     BEGIN
12 |         FOR i in 1..numParoleSTR LOOP
13 |             parolaSTR = split_part(stringa, '@', i);
14 |             IF count = 0 THEN
15 |                 strQRY = concat('(',selectQRY, '''', parolaSTR, '''', ')');
16 |                 count = 1;
17 |             ELSE
18 |                 strQRY = concat(' ', strQRY, ' INTERSECT ', '(',selectQRY, '''', parolaSTR, '''', ')');
19 |             end if;
20 |         end loop;
21 |         RAISE NOTICE 'Query: {%}', strQRY;
22 |
23 |         OPEN cursQRY FOR EXECUTE strQRY;
24 |         count = 0;
25 |         LOOP
26 |             FETCH cursQRY INTO correctURI;
27 |             EXIT WHEN NOT FOUND;
28 |             IF count = 0 THEN
29 |             output = correctURI;
30 |             count = 1;
31 |             ELSE
32 |             output = concat(output, '@', correctURI);
33 |             end if;
34 |         end loop;
35 |         RAISE NOTICE 'RISULTATO: {%}', output;
36 |         RETURN output;
37 |     end
38 | $$ LANGUAGE plpgsql;
39 |
40 | SELECT f.f_sqlDYN('provatabelle1e3');
```

Inserimento

```
INSERT INTO F.album(coda, titolo, inalbum)
    values (1, 'padre1', null),
           (11, 'padre1-figlio1', 1),
           (12, 'padre1-figlio2', 1),
           (13, 'padre1-figlio3', 1),
           --ALBUM 11
           (111, 'padre11-figlio1', 11),
           (112, 'padre11-figlio2', 11),
           (113, 'padre11-figlio3', 11),
           --ALBUM 113
           (1131, 'padre113-figlio1', 113),
           --ALBUM 13
            (131, 'padre13-figlio1', 13),
            (132, 'padre13-figlio2', 13),
           --ALBUM 2
            (2, 'padre2', null),
           (21, 'padre2-figlio1', 2),
           (22, 'padre2-figlio2', 2);


INSERT INTO f.hashtag(parola)
values  ('caso'),
        ('prova'),
        ('sei'),
        ('un'),
        ('grande'),
        ('bionda'),
        ('luigi');

INSERT INTO f.tagalbum(coda, parola)
values (1,'caso'),
       (1,'grande'),
       (11,'bionda'),
       (12,'bionda'),
       (13,'caso'),
       --ALBERO 2
       (2,'bionda'),
       (21,'grande'),
       (22,'caso'),
       (22,'luigi');

INSERT INTO f.foto(codf, uri)
values (1, 'uri1'),
       (2,'uri2'),
       (3,'uri3'),
       (4,'uri4');

INSERT INTO f.hashtag(parola)
```

```
values ('prova11'),
       ('prova12'),
       ('prova13'),
       ('prova21'),
       ('prova22'),
       ('prova23'),
       ('prova31'),
       ('prova32');

INSERT INTO f.tagfoto(codf, parola)
values (1,'prova11'),
       (1,'prova12'),
       (1,'prova13'),
       --foto 2
       (2,'prova21'),
       (2,'prova22'),
       (2,'prova23'),
       --foto 3
       (3,'prova31'),
       (3,'prova32');

INSERT INTO f.hashtag(parola) values ('provatabelle1e3');

INSERT INTO f.tagfoto(codf, parola) VALUES
                                      (1,'provatabelle1e3'),
                                      (3,'provatabelle1e3');
```

8 Riviste Articoli (08-12-2022)

Si consideri il seguente schema relazionale che descrive un frammento della base di dati per gestire gli accessi ad una biblioteca digitale di periodici. Le riviste sono identificate dal codice ISNN e sono strutturate in fascicoli (identificati da CodF).

Ogni fascicolo contiene articoli identificati dall'attibuto Doi.

L'utente identificato dal codice fiscale CF ha un profilo associato Codprofilo che regola le possibilità di accesso (numero di articoli consultati al giorno e al mese).

In ACCESSO ogni istanza memorizza l'accesso di un utente an un articolo. In PAROLECHIAVE viene memorizzato un insieme di parole chiave significative per una rivista.

Le parole chiave sono usate per descrivere astrattamente gli articoli in base al loro contenuto.

L'associazione tra parole chiave e articoli viene memorizzata nella relazione DESCRIZIONE.

RIV ISTA(ISNN, Titolo, Editore, Perodicita)

FASCICOLO(CodF, Titolo, ISNN, Anno, Numero)

ARTICOLO(Doi, CodF, Titolo, Autore, Sommario, PagI, PagF)

UTENTE(CF, email, CodProfilo, Nome, Cognome, DataN)

ACCESSO(CF, Doi, Data)

PROFILO(CodProfilo, Tipo, MaxGiorno, MaxMese)

PAROLECHIAV E(Parola, ISNN)

DESCRIZIONE(Parola, Doi)

Esercizio 03 (Punti 7, 20 minuti)

Si implementi un trigger azionato quando viene inserito un nuovo articolo.

Il trigger cerca la presenza nel sommario dell'articolo delle parole chiave associate alla rivista dell'articolo.

Se viene trovata la presenza di una parola chiave questa viene memorizzata nella tabella DESCRIZIONE.

Esercizio 04 (Punti 7, 25 minuti)

Si scriva una funzione in SQL DINAMICO che riceve in ingresso una stringa di parole chiave separate dal carattere +.

La funzione restituisce la stringa di doi degli articoli a cui sono associate TUTTE le parole chiave nella stringa.

Creazione DB

```
01 |  DROP SCHEMA IF EXISTS r CASCADE ;
02 |  CREATE SCHEMA IF NOT EXISTS r;
03 |
04 |  CREATE TABLE r.rivista (
05 |      isnn VARCHAR(32),
06 |      titolo VARCHAR(32),
07 |      editore VARCHAR(32),
08 |      periodicita VARCHAR(32),
09 |      CONSTRAINT pk_rivista PRIMARY KEY (isnn)
10 |  );
11 |  CREATE TABLE r.fascicolo(
12 |      CodF VARCHAR(32),
13 |      isnn VARCHAR(32),
14 |      numero int,
15 |      anno int,
16 |      CONSTRAINT pk_fascicolo PRIMARY KEY (CodF),
17 |      CONSTRAINT fk_fascicolo_rivista FOREIGN KEY (isnn) REFERENCES r.rivista(isnn)
18 |  );
19 |  CREATE TABLE r.articolo (
20 |      doi      VARCHAR(32),
21 |      CodF     VARCHAR(32),
22 |      titolo   VARCHAR(32),
23 |      autore   VARCHAR(32),
24 |      sommario VARCHAR(32),
```

```
    PagI     INTEGER,
    PagF     INTEGER,
    CONSTRAINT pk_articolo         PRIMARY KEY (doi),
    CONSTRAINT FK_ARTICOLO_RIVISTA FOREIGN KEY (CodF) REFERENCES r.fascicolo(CodF)
);
CREATE TABLE r.Profilo
(
    CodProfilo VARCHAR(32),
    Tipo       VARCHAR(32),
    MaxGiorno  INTEGER,
    MaxMese    INTEGER,
    CONSTRAINT PK_Profilo PRIMARY KEY (CodProfilo)
);
CREATE TABLE r.Utente
(
    CF          VARCHAR(32),
    email       VARCHAR(32),
    CodProfilo  VARCHAR(32),
    Nome        VARCHAR(32),
    Cognome     VARCHAR(32),
    DataNascita DATE,
    CONSTRAINT PK_Utente         PRIMARY KEY (CF),
    CONSTRAINT FK_Utente_Profilo FOREIGN KEY (CodProfilo) REFERENCES r.Profilo (CodProfilo)
);

CREATE TABLE r.accesso(
    CF   VARCHAR(32),
    doi  VARCHAR(32),
    data DATE,
    CONSTRAINT pk_accesso          PRIMARY KEY (CF,doi,data),
    CONSTRAINT fk_accesso_articolo FOREIGN KEY (doi) REFERENCES r.articolo(doi),
    CONSTRAINT fk_accesso_utente   FOREIGN KEY (CF)  REFERENCES r.utente(CF)
);
CREATE TABLE r.Descrizione
(
    Parola VARCHAR(32),
    Doi    VARCHAR(32),
    CONSTRAINT FK_Descrizione_Articolodoc FOREIGN KEY (Doi) REFERENCES r.Articolo (Doi)
);
CREATE TABLE r.ParoleChiave
(
    Parola VARCHAR(32),
    ISNN   VARCHAR(32),
    CONSTRAINT PK_ParoleChiave PRIMARY KEY (Parola, ISNN),
    CONSTRAINT FK_ParoleChiave_ FOREIGN KEY (ISNN) REFERENCES r.rivista(ISNN)
);
```

Esercizio 3

```
CREATE OR REPLACE FUNCTION r.funz_1() RETURNS TRIGGER AS $$
    DECLARE
        word r.parolechiave.parola%TYPE;
        cursore CURSOR FOR SELECT parola FROM r.fascicolo NATURAL JOIN r.parolechiave
                           WHERE r.fascicolo.codf=NEW.codf;
        i int:=0;
        n int:=(SELECT COUNT(*) FROM r.fascicolo NATURAL JOIN r.parolechiave);
    BEGIN
        OPEN cursore;
        WHILE (i<n) LOOP
            FETCH cursore into word;
            IF(NEW.sommario LIKE '%' || word || '%') THEN
                INSERT INTO r.descrizione(parola, doi)
                values (word, new.doi);
            END IF;
            i=i+1;
            END LOOP;
        CLOSE cursore;
        RETURN NEW;
    END ;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER trigg_1 AFTER INSERT ON r.articolo
FOR EACH ROW
EXECUTE PROCEDURE r.funz_1();
```

Esercizio 4

```

CREATE OR REPLACE FUNCTION r.pro(string VARCHAR(500)) RETURNS VARCHAR(500) AS $$
    DECLARE
        numParoleSTR INTEGER;
        parolaSTR r.descrizione.parola%TYPE;
        numDOI INTEGER;
        cursDOI CURSOR FOR SELECT doi FROM r.articolo;
        currentDOI r.descrizione.doi%TYPE;
        match INTEGER;
        output VARCHAR(500);
    BEGIN
        string=replace(string, '+', '@');
        numParoleSTR = regexp_count(string, '@') + 1;
        numDOI = (SELECT Count(doi) FROM r.articolo);
        OPEN cursDOI;

        FOR i in 1..numDOI LOOP
            FETCH cursDOI INTO currentDOI;
            match=0;
```

```
20 |
21 |             FOR j in 1..numParoleSTR LOOP
22 |                 parolaSTR = split_part(string, '@', j);
23 |
24 |                 IF EXISTS (SELECT * FROM r.descrizione as d
25 |                           WHERE d.doi=currentDOI AND d.parola=parolaSTR) THEN
26 |                 match = match +1;
27 |                 end if;
28 |                 IF match = numParoleSTR THEN
29 |                     output=CONCAT(output, currentDOI);
30 |                 end if;
31 |             end loop;
32 |         end loop;
33 |         RETURN output;
34 |     END;
35 | $$ LANGUAGE plpgsql;
36 |
37 | SELECT r.pro('prova2');
```

Inserimento

```
01 | --RIVISTA
02 | INSERT INTO r.rivista(ISNN)
03 | values ('rivista1'),
04 |        ('rivista2');
05 |
06 | --FASCICOLO
07 | INSERT INTO r.fascicolo(isnn, codf)
08 | values ('rivista1','fascicolo1'),
09 |        ('rivista2','fascicolo2');
10 |
11 | --PAROLECHIAVE
12 | INSERT INTO r.parolechiave(isnn, parola)
13 | values ('rivista1', 'silvio1'),
14 |        ('rivista1', 'forzanapoli1'),
15 |        ('rivista2', 'barra2'),
16 |        ('rivista2', 'prova2');
17 |
18 | --ARTICOLO
19 | INSERT INTO r.articolo(doi, codf, sommario)
20 | values ('doi1', 'fascicolo1', 'silvio1 barra2 forzanapoli1'),
21 |        ('doi2', 'fascicolo2', 'silvio1 barra2 prova2');
```

9 Log Richieste (13-12-2022)

Si consideri il seguente schema relazionale che descrive i dati di gestione in un sistema per l'accesso parallelo.

L'intuizione è che una transazione (sequenza atomica di operazioni) prima di poter operare su una risorsa deve chiedere l'autorizzazione all'accesso (in lettura o scrittura) e solo quando la risorsa le viene assegnata può operare per leggere o modificare il valore della risorsa.

Le operazioni possono essere CREA (crea una nuova risorsa), CANCELLA (cancella una risorsa) MODIFICA (modifica il valore della risorsa) COMMIT (chiusura della transazione) ABORT (annulla una transazione).

Le risorse condivise son identificate dall'attributo CodRisorsa e lo stato indica: se sono libere UNLOCK, in uso in lettura R − LOCK o in uso in scrittura W − LOCK.

Le transazioni identificate da CodTransazione inoltrano richieste di operazioni che sono registrate nella tabella RICHIESTE.

Ogni richiesta ha un tempo di inoltro, un tipo di accesso (o R-LOCK o W-LOCK) e la risorsa richiesta.

Nella tabella ASSEGNAZIONE sono memorizzate le assegnazioni correnti delle risorse alle transazioni.

Nella tabella LOG vengono invece registrate le operazioni svolte dalle transazioni sulle risorse a loro assegnate.

In particolare, per ogni operazione MODIFICA si registra il valore della risorsa prima e dopo l'operazione; per operazione CANCELLA si esprime solo V alorePrima; per CREA si esprime solo il valore V aloreDopo; e per COMMIT ed ABORT e CHECK non si esprime nessun valore (NULL).

In aggiunta per l'operazione CHECK non si esprime neppure il codice della transazione (il record di CHECK rappresenta un punto di con- trollo del sistema). Per tutte le operazioni è riportato un timestamp, cioè una marca temporale univoca nel sistema che tiene traccia della sequenza delle operazioni.

LOG(Cod, Operazione, CodRisorsa, V alorePrima, V aloreDopo, CodTransazione, Timestamp)

RISORSA(CodRisorsa, Locazione, V alore, Stato)

RICHIESTE(CodTransazione, Tempo, tipoAccesso, CodRisorsa)

ASSEGNAZIONE(CodTransazione, Tempo, CodRisorsa, tipoAccesso)

Esercizio 01 (Punti 8)

Si implementi il seguente trigger.

Quando una transazione registra una operazione di ABORT sul log, tutte le scritture fatte dalla transazione e riportate sul LOG devono essere annullate in ordine inverso a quelle in cui sono state fatte.

Per annullare le scritture si deve consultare il log e si deve assegnare ad ogni risorsa scritta dalla transizione il valore ValorePrima riportato nel LOG.

Inoltre, le risorse assegnate alla transazione devono tornare libere: si rimuovono le assegnazioni alla transazione e lo stato della risorsa assume valore UNLOCK.

Esercizio 02 (Punti 8)

Si scriva una funzione con parametro intero Tout che per tutte le transazioni T1 che sono in attesa per una risorsa per un tempo superiore a Tout (differenza tra il tempo di registrazione della richiesta e il tempo corrente) controlli se ci sia un deadlock (cioè se esiste un'altra transazione T2 che occupa la risorsa richiesta e la transazione T2 richiede una risorsa asegnata alla transazione T1).

La funzione restituisce una stringa coi codici delle transazioni T1 in deadlock cosi trovate.

Creazione DB

```
01 |  DROP SCHEMA IF EXISTS l CASCADE ;
02 |  CREATE SCHEMA l;
03 |
04 |  CREATE TABLE l.risorsa(
05 |      CodRisorsa INTEGER  ,
06 |      Locazione VARCHAR(50)   ,
07 |      Valore VARCHAR(50)   ,
08 |      STATO VARCHAR(50)   ,
09 |
10 |      CONSTRAINT PK_risorsa PRIMARY KEY (CodRisorsa)
11 |  );
12 |
13 |  CREATE TABLE l.richieste(
14 |      CodTransazione INTEGER   ,
15 |      Tempo INTEGER   ,
16 |      TipoAccesso VARCHAR(50)   ,
17 |      CodRisorsa INTEGER   ,
18 |
19 |      CONSTRAINT PK_richieste PRIMARY KEY (CodTransazione),
20 |      CONSTRAINT FK_richieste_risorsa FOREIGN KEY (CodRisorsa) REFERENCES l.risorsa(CodRisorsa)
21 |  );
22 |
23 |  CREATE TABLE l.assegnazione(
24 |      CodTransazione INTEGER,
25 |      Tempo INTEGER,
26 |      TipoAccesso VARCHAR(50),
27 |      CodRisorsa INTEGER,
28 |
29 |      CONSTRAINT FK_assegnazione_risorsa FOREIGN KEY (CodRisorsa) REFERENCES l.risorsa(CodRisorsa)
30 |  );
31 |
32 |  CREATE TABLE l.log(
33 |      Cod INTEGER  ,
34 |      Operazione VARCHAR(50)  ,
35 |      CodRisorsa INTEGER  ,
36 |      ValorePrima VARCHAR(50)  ,
37 |      ValoreDopo VARCHAR(50)  ,
38 |      CodTransazione INTEGER  ,
39 |      TimeStamp INTEGER  ,
40 |
41 |      CONSTRAINT log_pk PRIMARY KEY (Cod),
```

```
    CONSTRAINT risorse_fk FOREIGN KEY (CodRisorsa) REFERENCES l.risorsa(CodRisorsa)
);
```

Esercizio 1

```
DELETE FROM l.log WHERE cod=99;

CREATE OR REPLACE FUNCTION l.funzione1() RETURNS TRIGGER
AS
    $$
    DECLARE
        controllo integer:=0;
    count INTEGER:=(SELECT COUNT(*)
                    FROM l.log
                    WHERE codtransazione=NEW.codtransazione);
    operazioniprecedenti CURSOR FOR (
        SELECT codrisorsa
        FROM l.log
        WHERE codtransazione=NEW.codtransazione
        ORDER BY log.timestamp DESC
    );
    BEGIN
        OPEN operazioniprecedenti;
        FOR i IN 1..count LOOP
        FETCH operazioniprecedenti INTO controllo;
        RAISE NOTICE 'controllo{%}', controllo;
        UPDATE l.risorsa SET valore=valoreprima, stato='UNLOCK' FROM l.log WHERE risorsa.codrisorsa=controllo;
        END LOOP;
        RETURN NEW;
    end;
    $$
LANGUAGE plpgsql;


CREATE OR REPLACE TRIGGER trigger1 BEFORE INSERT OR UPDATE ON l.log
    FOR EACH ROW
    WHEN (NEW.operazione='ABORT')
        EXECUTE FUNCTION l.funzione1();
UPDATE l.risorsa SET valore=valoreprima FROM l.log WHERE risorsa.codrisorsa=1;

INSERT INTO l.log(cod, operazione, codrisorsa, valoreprima, valoredopo, codtransazione, timestamp)
values (99, 'ABORT', 4, NULL, NULL, 51, 8);
```

Esercizio 2

```
CREATE OR REPLACE PROCEDURE l.funzione2(IN Tout INTEGER) AS
$$
```

```
03 |      DECLARE
04 |          tattuale INTEGER=5; --sono le 5
05 |          T1_n_transazioni INTEGER:=(SELECT count(codtransazione) FROM l.richieste WHERE (tattuale-tempo)>Tout);  --le richieste sono state fatte alle 2
06 |          T1_c_transazioni_risorse CURSOR FOR                                                                    --
07 |              SELECT codtransazione, ris.codrisorsa
08 |              FROM l.richieste ric JOIN l.risorsa ris ON ris.codrisorsa = ric.codrisorsa
09 |              WHERE (tattuale-ric.tempo)>Tout AND ris.stato<>'UNLOCK'; --T1
10 |          T1_transazione l.richieste.codtransazione%TYPE;
11 |          T1_risorsa l.richieste.codrisorsa%TYPE;
12 |
13 |          T2_n_transazioni INTEGER:=(SELECT COUNT(*) FROM l.assegnazione);
14 |          T2_c_transazioni_risorse CURSOR FOR SELECT codtransazione, codrisorsa FROM l.assegnazione;
15 |          T2_transazione l.richieste.codtransazione%TYPE;
16 |          T2_risorsa l.richieste.codrisorsa%TYPE;
17 |
18 |          --temporegistrazione INTEGER;
19 |          --tempo_corrente INTEGER:=5;
20 |              --tempoattesa INTEGER:=temporegistrazione-tempo_corrente;
21 |          output VARCHAR(500):='';
22 |      BEGIN
23 |          OPEN T1_c_transazioni_risorse;                      --T1 APERTO
24 |          for i IN 1..T1_n_transazioni LOOP
25 |              RAISE NOTICE 'i(%)', i;
26 |              FETCH T1_c_transazioni_risorse INTO T1_transazione, T1_risorsa;
27 |              OPEN T2_c_transazioni_risorse;                                    --T2 APERTO
28 |              FOR j IN 1..T2_n_transazioni LOOP
29 |                  RAISE NOTICE 'j(%)', j;
30 |                  FETCH T2_c_transazioni_risorse INTO T2_transazione, T2_risorsa;
31 |                  raise notice 'T1:(%), T2:(%)', T1_risorsa, T2_risorsa;
32 |                  if(T1_risorsa=T2_risorsa) THEN
33 |                      RAISE NOTICE 'siamo uguali';
34 |                  output=output || ' ' || T1_transazione;
35 |                  end if;
36 |                  end loop;
37 |
38 |              CLOSE T2_c_transazioni_risorse;                                   --T2 CHIUSO
39 |              end loop;
40 |          CLOSE T1_c_transazioni_risorse;                        --T1 chiuso
41 |  RAISE NOTICE 'output:{%}', output;
42 |      end
43 |  $$
44 |  LANGUAGE plpgsql;
45 |
46 |  CALL l.funzione2(2); --le richieste aspettano per massimo TOUT ore
```

Inserimento

```
01 |  INSERT INTO l.risorsa(codrisorsa, valore, stato)
```

```
02 |  values (1, '1risorsa52', 'w-lock'),
03 |         (2, '2risorsa', 'w-lock'),
04 |         (3, '3risorsa', 'w-lock'),
05 |         (4, '4risorsa', 'w-lock'),
06 |         (5, '5risorsa', 'w-lock');
07 |
08 |  INSERT INTO l.assegnazione(codtransazione, tempo, codrisorsa)
09 |  VALUES (52, 1, 1),
10 |         (52, 1, 2),
11 |         (53, 1, 3),
12 |         (53, 1, 4),
13 |         (51, 1, 5);
14 |
15 |  INSERT INTO l.log(cod, operazione, codrisorsa, valoreprima, valoredopo, codtransazione, timestamp)
16 |  VALUES (21, 'MODIFICA', 1, null, 'valoredopo1', 51, 1),
17 |         (22, 'MODIFICA', 2, null, 'valoredopo2', 51, 2),
18 |         (231, 'CANCELLA', 1, 'valoredopo1', null, 52, 2),
19 |         (222, 'MODIFICA', 2, 'valoredopo2', 'valoredopodopo2', 52, 2),
20 |         (23, 'MODIFICA', 3, 'valore3', 'valoredopo3', 53, 3);
21 |
22 |  INSERT INTO l.richieste(codtransazione, tempo, tipoaccesso, codrisorsa)
23 |  VALUES (51, 2, 'accesso1',1),
24 |         (52, 4, 'accesso2',5),
25 |         (53, 2, 'accesso3',2);
```