9 Prova del 19/06/2019

Schema base di dati

Si consideri il seguente schema relazionale (chiavi primarie sottolineate) inerente a una possiile organizzazione di istituti di ricerca:

```
SEZIONE(codice, nome, direttore)
RICERCATORE(codiceFiscale, nome, cognome, sezApp) --sezApp è il codice della sezione in
    cui il ricercatore lavora
PARTECIPA(ricercatore, progetto)PROGETTO(\underline{codice}, obiettivo, CodRespProg, dataInizio,
    dataFine)-CodRespProg è il c.f.  del responsabile del progetto
```

Domanda 1 [7 punti]

1) Determinare i vincoli di integrità che si possono desumere usando la notazione '→'.
2) Creare un dominio per rappresentare il codice fiscale
3) Scrivere il codice PostgreSQL che generi TUTTE le tabelle per rappresentare lo schema relazionale specificando tutti i possibili controlli di integrità referenziale e almeno 2 controlli di correttezza dei dati usando il CHECK.

Suggerimento: PostgreSQL ammette vincoli di integrità referenziali circolari. Essi devono essere aggiunti DOPO la creazione delle tabelle con il comando 'ALTER TABLE ADD constraintTable' e si devono dichiarare con le opzioni 'DEFERRABLE INITIALLY DEFERRED' (sono 2 opzioni!). Tutti gli INSERT/UPDATE/DELETE che riguardano le tabelle con vincoli circolari deve essere fatti dentro delle transazioni.'

Soluzione

1) I vincoli di integrità secondo la notazione '→' sono:

SEZIONE.direttore → RICERCATORE.codiceFiscale
RICERCATORE.sezApp → SEZIONE.codice
I due vincoli precedenti sono in dipendenza reciproca (circolarità).
PROGETTO.CodRespProg → RICERCATORE.codiceFiscale
PARTECIPA.ricercatore → RICERCATORE.codiceFiscale
PARTECIPA.progetto → PROGETTO.codice

2) e 3)

```
CREATE DOMAIN codiceFiscaleD CHAR(16) CHECK (VALUE SIMILAR TO
     '[A-Z]{6}[0-9]{2}[A-Z][0-9]{2}[A-Z][0-9]{3}[A-Z]');

CREATE TABLE SEZIONE (
   codice VARCHAR PRIMARY KEY,
   nome VARCHAR NOT NULL CHECK (nome<>''),
   direttore codiceFiscaleD NOT NULL
);

CREATE TABLE RICERCATORE (
   codiceFiscale codiceFiscaleD PRIMARY KEY,
   nome VARCHAR NOT NULL CHECK (nome<>''),
   cognome VARCHAR NOT NULL CHECK (cognome<>''),
   sezApp VARCHAR NOT NULL REFERENCES SEZIONE
);

ALTER TABLE SEZIONE ADD FOREIGN KEY(direttore) REFERENCES RICERCATORE DEFERRABLE
     INITIALLY DEFERRED;

CREATE TABLE PROGETTO (
   codice VARCHAR PRIMARY KEY,
   obiettivo VARCHAR NOT NULL CHECK (obiettivo<>''),
```

```
    CodRespProg codiceFiscaleD NOT NULL REFERENCES RICERCATORE,
    dataInizio DATE NOT NULL,
    dataFine DATE NOT NULL,--si può assumere che ci sia una data fine prevista
    CHECK (dataInizio < dataFine)
);

CREATE TABLE PARTECIPA (
    ricercatore codiceFiscaleD  REFERENCES RICERCATORE,
    progetto VARCHAR REFERENCES SEZIONE,
    PRIMARY KEY (ricercatore, progetto)
);
```

Domanda 2 [7 punti]

Scrivere il codice PostgreSQL che determina le soluzioni alle seguenti due interrogazioni usando il minor numero di JOIN:

(a) Fra i progetti aventi responsabili della stessa sezione, visualizzare quello(i) con durata maggiore, mostrando il codice del progetto, il codice fiscale del responsabile, il nome della sezione e la durata in giorni.

(b) Trovare i progetti che coinvolgano tutte le sezioni. In altre parole, i progetti dove partecipa almeno un ricercatore per sezione. Mostrare il codice del progetto e il numero totale dei ricercatori che partecipano al progetto.

Soluzione

Una possibile soluzione per la parte (a) prevede di usare una vista per semplificare la scrittura della query finale.

```
(a)
CREATE VIEW DURATA_MASSIMA_PROGETTO_PER_SEZIONE AS
  SELECT R.sezApp AS sezione, MAX(P.dataFine-P.dataInizio) AS durata
  FROM RICERCATORE R JOIN PROGETTO P ON R.codiceFiscale=P.CodRespProg
  GROUP BY R.sezApp;

SELECT P.codice, R.codicefiscale, S.nome, V.durata
FROM SEZIONE S JOIN RICERCATORE R ON R.sezApp=S.codice JOIN PROGETTO P ON
     R.codiceFiscale=P.CodRespProg
   JOIN DURATA_MASSIMA_PROGETTO_PER_SEZIONE V ON S.codice=V.codice
WHERE P.dataFine-P.dataInizio = V.durata;

--(b)
--Una soluzione è basata su una vista che calcola il numero di ricercatori e il
     numero di sezioni per ciascun progetto
CREATE VIEW SEZIONI_PER_PROGETTO AS
  SELECT P.codice, COUNT(DISTINCT R.sezApp) AS n_sezioni, COUNT(R.codiceFiscale)
     AS n_ricercatori
  FROM RICERCATORE R JOIN PARTECIPA PA ON R.codiceFiscale=PA.ricercatore JOIN
     PROGETTO P ON PA.progetto=P.codice
  GROUP BY P.codice;

SELECT V.codice, V.n_ricercatori
FROM SEZIONI_PER_PROGETTO V
WHERE V.n_sezioni = (
   SELECT COUNT(*)
   FROM sezione
);

--Un'altra soluzione è basata ponendo una query innestata nella condizione having
SELECT P.progetto, COUNT(8) AS n_ricercatori
```

```sql
FROM Partecipa P JOIN Ricercatore R ON (P.ricercatore=R.codiceFiscale)
GROUP BY P.progetto
HAVING COUNT(DISTINCT R.sezApp) = (
    SELECT COUNT(*)
    FROM sezione
);
```

Domanda 3 [7 punti]

Data una base di dati PostgreSQL che contenga le tabelle della Domanda 1, scrivere un programma Java (solo metodo main) che, leggendo il codice di una sezione da console, visualizzi il risultato della query dell'esercizio 2(a). Se il codice della sezione non esiste, il programma deve richiedere il dato.

Soluzione

```java
public static void main(String[] args) throws ClassNotFoundException, SQLException {
   Class.forName("org.postgresql.Driver");
   try (Scanner keyb = new Scanner(System.in);
         Connection con = DriverManager.getConnection("jdbc:postgresql://localhost:5432/posenato",
     "posenato", "")) {
      String codiceSezione = null;
      ResultSet rs = null;
      while (codiceSezione == null) {
         System.out.print("Inserire il codice sezione: ");
         codiceSezione = keyb.next();
         try (PreparedStatement pst = con.prepareStatement("SELECT count(*) FROM SEZIONE S WHERE
     s.codice = ?")) {
            pst.clearParameters();
            pst.setString(1, codiceSezione);
            rs = pst.executeQuery();
            rs.next();
            if (rs.getInt(1) != 1) {
               System.out.println("Non esiste la sezione con codice " + codiceSezione " + ". Ripetere
     inserimento.");
               codiceSezione = null;
               continue;
            }
         } catch (SQLException e) {
            System.out.println("Problema durante prima query: " + e.getMessage());
            return;
         }
      }
      //la sezione esiste
      try (PreparedStatement pst = con.prepareStatement(<codice query 2a>+ "AND s.codice = ?")) {
            pst.clearParameters();
            pst.setString(1, codiceSezione);
            rs = pst.executeQuery();
            if (!rs.isBeforeFirst()) {//isBeforeFirst restituisce false se non ci sono righe!
               System.out.println("Non esiste nessun progetto associato alla sezione con codice " +
     codiceSezione);
            } else {
               // rs ha i dati cercati
               System.out.println(String.format(
                  "I progetti di durata massima in giorni della sezione data sono:\n" + "%20s %16s %20s
     %20s %d", "Progetto", "Codice Fiscale", "Sezione", "Sezione", "Durata"));
               while (rs.next()) {
                  System.out.println(String.format("%20s %16s %20s %20s %d", rs.getString(1),
     rs.getString(2), rs.getString(3), rs.getString(4), rs.getInt(5)));
               }
            }
      } catch (SQLException e) {
            System.out.println("Problema durante seconda query: " + e.getMessage());
            return;
      }
   } catch (SQLException e) {
      System.out.println("Problema durante la connessione iniziale alla base di dati: " +
     e.getMessage());
      return;
   }
}
```

Domanda 4 [5 punti]

Si considerino le tabelle **`SEZIONE`** e **`RICERCATORE`** della Domanda 1.

Un vincolo con opzione 'INITIALLY DEFERRED' viene valutato solo all'esecuzione di un commit.

Si scriva quindi uno script SQL che popoli la tabella SEZIONE e la tabella RICERCATORE (che sono vuote) con una nuova sezione e due nuovi ricercatori della sezione, di cui uno è anche il direttore.

Si scriva poi uno script che cambia la sezione al direttore della sezione 'x' e assegni un nuovo direttore alla sezione 'x'. Come si deve scrivere per garantire la coerenza?

Soluzione

```
BEGIN;
   INSERT INTO RICERCATORE VALUES ('AAABBB01A01Z112E', 'AAA', 'BBB', 1),
      ('BBBCCC01A01Z112C', 'BBB', 'CCC', 1);
   INSERT INTO SEZIONE VALUES (1, 'sezione', 'BBBCCC01A01Z112C');
COMMIT;
```

Il secondo script deve essere all'interno di una transazione.

```
BEGIN;
   UPDATE PERSONA SET sezApp = 2 --assumo che sia 2 la nuova sezione
   WHERE codiceFiscale IN (SELECT direttore FROM sezione WHERE codice = x);

   UPDATE sezione SET direttore = 'AAABBB01A01Z112E'
   WHERE codice = x;
COMMIT;
```

Domanda 5 [6 punti]

Si consideri il seguente risultato del comando ANALYZE su una query inerenti a 2 tabelle:

```
GroupAggregate  (cost=7712.31..7919.04 ROWS=7709...)
GROUP KEY: i.nomeins
-> Sort  (cost=7712.31..7755.52 ROWS=17285...)
   Sort KEY: i.nomeins
   -> Hash JOIN  (cost=287.80..6495.68
     ROWS=17285...)
      Hash Cond: (ie.id_insegn = i.id)
      ->  Seq Scan ON inserogato ie
     (cost=0.00..5970.21 ROWS=17285...)
           Filter: ((annoaccademico)::TEXT >
     '2010/2011'::TEXT)
      ->  Hash  (cost=185.69..185.69 ROWS=8169
     width=43)
           ->  Seq Scan ON insegn i
     (cost=0.00..185.69 ROWS=8169 width=43)
```

```
GroupAggregate  (cost=7484.53..7691.26 ROWS=7709...)
GROUP KEY: i.nomeins
-> Sort  (cost=7484.53..7527.75 ROWS=17285...)
   Sort KEY: i.nomeins
   -> Hash JOIN  (cost=694.18..6267.91 ROWS=17285...)
      Hash Cond: (ie.id_insegn = i.id)
      -> Bitmap Heap Scan ON inserogato ie
     (cost=406.38..5742.44 ROWS=17285)
         Recheck Cond: ((annoaccademico)::TEXT >
     '2010/2011'::TEXT)
         -> Bitmap INDEX Scan ON
     inserogato_annoaccademico_idx1  (cost=0.00..402.06
     ROWS=17285 width=0)
              INDEX Cond: ((annoaccademico)::TEXT >
     '2010/2011'::TEXT)
         -> Hash  (cost=185.69..185.69 ROWS=8169 width=43)
            -> Seq Scan ON insegn i  (cost=0.00..185.69
     ROWS=8169 width=43)
```

A sinistra la query originale, a destra la query dopo la creazione di uno o più indici.

Desumere il testo della query e scrivere il codice del/degli indici creati.

Soluzione

È un hash join, quindi c'è un JOIN con condizione ie.id_insegn = i.id Poi c'è un Sep Scan con condizione (annoaccademico)::text > '2010/2011'. Infine, c'è un group key. Quindi c'è un group by i.nomeins.

```
SELECT i.nomeins, COUNT(DISTINCT ie.annoaccademico) AS naa
FROM insegn i JOIN inserogato ie ON ie.id_insegn=i.id
WHERE ie.annoaccademico> '2010/2011'
GROUP BY(i.nomeins);
```

Lo studente non era tenuto a scrivere `COUNT(DISTINCT ie.annoaccademico) AS naa` perché non desumibile dalla query, mentre `i.nomeins` era da inserire perché molto probabile visto il `GROUP BY`. Sicuramente NON si poteva inserire `*`.

L'unico indice creato è quello relativo all'anno accademico. Infatti nel secondo explain, compare Index Cond: ((annoaccademico)::text > '2010/2011'::text)

```
CREATE INDEX ON inserogato (annoaccademico);
```