Esercitazione

Prof. Alfredo Pulvirenti

- Dato lo schema:

Escursione(id, titolo, descrizione, durata, difficoltà, costo)

DataEscursione(id, data, idescursione, id guida)

Partecipante(idpartecipante, idescursione)

Persona(id, nome, cognome)

- Indicare le chiavi primarie ed esterne dello schema e le relazioni esistenti tra le tabelle .

Escursione(id, titolo, descrizione, durata, difficoltà, costo)

DataEscursione(id, data, idescursione, idguida)

Partecipante(idpartecipante, idescursione)

Persona(id, nome, cognome)

Rispondere alle seguenti query in algebra relazionale ed SQL:

- Trovare le escursioni (indicando titolo, descrizione e difficoltà) che hanno un costo massimo

Escursione(id, titolo, descrizione, durata, difficoltà, costo)

- Algebra: $\pi_{titolo,descrizione,durata,difficolta}(Escursione)$

$-$

$$\pi_{titolo,descrizione,durata,difficolta}\left(\sigma_{costo1>costo}\left(\delta_{\substack{id1\leftarrow id,\\ titolo1\leftarrow titolo,\\ descrizione1\leftarrow titolo,\\ durata1\leftarrow durata,\\ difficolta1\leftarrow difficolta,\\ costo1\leftarrow costo}}(Escursione)\times Escursione\right)\right)$$

Rispondere alle seguenti query in algebra relazionale ed SQL:

- Trovare le escursioni (indicando titolo, descrizione e difficoltà) che hanno un costo massimo

Escursione(id, titolo, descrizione, durata, difficoltà, costo)

- SQL:

```
SELECT titolo, descrizione, durata, difficolta
FROM escursione
WHERE costo = (SELECT max(costo)
                FROM escursione)
```

Espressione equivalente usa la HAVING

```
SELECT titolo, descrizione, durata, difficolta
FROM escursione
HAVING costo = (SELECT max(costo) FROM
                escursione)
```

Rispondere alle seguenti query in algebra relazionale ed SQL:

- Trovare le escursioni (indicando titolo, descrizione e difficoltà) che hanno un costo massimo

  Escursione(id, titolo, descrizione, durata, difficoltà, costo)

- SQL:

```
SELECT titolo, descrizione, durata, difficolta
FROM escursione e
WHERE NOT EXISTS (SELECT *
                  FROM escursione
                  WHERE costo > e.costo)
```

- Trovare i partecipanti (dando nome e cognome in output) che hanno partecipato a tutte le escursioni

Escursione(id, titolo, descrizione, durata, difficoltà, costo)

Partecipante(idpartecipante, idescursione)

Persona(id, nome, cognome)

Algebra

$$P2 := \pi_{idpartecipante,ide}(Partecipante \triangleright\!\triangleleft_{idescursione=id} \delta_{ide \leftarrow idescursione} DataEscursione)$$

$$Persona \triangleright\!\triangleleft_{id=idpartecipante} (P2 \div \delta_{id \rightarrow ide}(\pi_{id}(Escursione)))$$

- Trovare i partecipanti (dando nome e cognome in output) che hanno partecipato a tutte le escursioni

Escursione(id, titolo, descrizione, durata, difficoltà, costo)
Partecipante(idpartecipante, idescursione)
Persona(id, nome, cognome)
DataEscursione(id, data, idescursione, id guida)

SQL

```
SELECT nome, cognome
FROM persona p
WHERE NOT EXISTS (SELECT *
                  FROM escursione e
                  WHERE NOT EXISTS (SELECT *
                                    FROM partecipante, dataEscursione de
                                    WHERE idpartecipante=p.ip AND
                                          idescrusione =de.id AND
                                          de.idescursione= e.id ))
```

- Trovare le guide che non hanno mai partecipato ad escursioni di difficoltà massima
-

  Escursione(id, titolo, descrizione, durata, difficoltà, costo)

  Partecipante(idpartecipante, idescursione)

  Persona(id, nome, cognome)

  DataEscursione(id, data, idescursione, id guida)

ALGEBRA

$$R1 := \pi_{id}(Escursione)$$

$$-$$

$$\pi_{id}\left(\sigma_{difficolta1>difficolta}\left(\delta_{\substack{id1\leftarrow id,\\ titolo1\leftarrow titolo,\\ descrizione1\leftarrow titolo,\\ durata1\leftarrow durata,\\ difficolta1\leftarrow difficolta,\\ costo1\leftarrow costo}}(Escursione)\times Escursione\right)\right)$$

$$\pi_{idguida}(DataEscursione)-\pi_{idguida}(DataEscursione \triangleright\triangleleft_{idescrusione=id} R1)$$

- Trovare le guide che non hanno mai partecipato ad escursioni di difficoltà massima **[3 punti]**;

  Escursione(id, titolo, descrizione, durata, difficoltà, costo)

  Partecipante(idpartecipante, idescursione)

  Persona(id, nome, cognome)

  DataEscursione(id, data, idescursione, id guida)

SQL

```
SELECT DISTINCT idguida
FROM DataEscursione
EXCEPT
SELECT DISTINCT idguida
FROM escursione,DataEscursione
WHERE escursione.id = DataEscursione.idescrusione AND difficoltà = (SELECT
max(difficolta) FROM escrusione)
```

- Trovare le coppie di persone che hanno partecipato sempre alle stesse escursioni

ALGEBRA

Trovare le coppie di persone che hanno partecipato sempre alle stesse escursioni

```
SELECT p1.idpartecipante,p2.idpartecipante
FROM partecipante p1, partecipante p2
WHERE p1.idescursione=p2.idescursione AND
   p1.idpartecipante< p2.idpartecipante
   AND (p1.idpartecipante,p2.idpartecipante) NOT IN
    (SELECT p1.idpartecipante,p2.idpartecipante
     FROM partecipante p1, partecipante p2
     WHERE p1.idescursione<>p2.idescursione AND
        p1.idpartecipante< p2.idpartecipante AND
           (NOT EXISTS
              (SELECT *
               FROM partecipante e1
               WHERE e1.idpartecipante = p1.idpartecipante AND
                     e1.idescursione = p2.idescursione)
           OR NOT EXISTS
              (SELECT *
               FROM partecipante e2
               WHERE e2.idpartecipante = p2.idpartecipante AND
                     e2.idescursione = p1.idescursione)))
```

- Dire ogni accompagnatore quante escursioni ha guidato;

```
SELECT idguida, count(*)
FROM DataEscursione
GROUP BY idguida
```

```
CREATE TRIGGER esperienza
after insert on DataEscursione
for each row
Declare X number;
Declare Y number;
referencing new as N

Begin
SELECT difficolta into Y
FROM escursione
WHERE id= N.idescursione;

if Y > 1 then
    SELECT count(*) into X
    FROM escursione,dataEscursione
    WHERE   dataEscursione.idescursione = escursione.id AND
            N.idGuida = dataEscrusione.idguida AND
            difficolta = Y-1
    if X < 5 then
        DELETE FROM DataEscursione WHERE id=N.id
    end if;
end if;
End;
```

Definiti 5 livelli di difficoltà per le escursioni. Creare un vincolo di integrità che garantisca che ogni accompagnatore prima di guidare una escursione di livello x abbia guidato almeno 5 escursioni di livello (x-1)

SINTASSI ORACLE

```
CREATE TRIGGER esperienza
BEFORE insert ON DataEscursione
for each row

WHEN  (
(SELECT difficolta
FROM escursione
WHERE id= new.idescursione) > 1)
AND
 (SELECT count(*) into X
     FROM escursione,dataEscursione
     WHERE   dataEscursione.idescursione = escursione.id AND
             new.idGuida = dataEscrusione.idguida AND
             difficolta = Y-1) <5))
BEGIN
   SIGNAL SQLSTATE '00001' ('VIOLATO VINCOLO DI INTEGRITA')
END;
```

Definiti 5 livelli di difficoltà per le escursioni. Creare un vincolo di integrità che garantisca che ogni accompagnatore prima di guidare una escursione di livello x abbia guidato almeno 5 escursioni di livello (x-1)

SINTASSI DB2

- Si consideri un database SQL per memorizzare un grafo direzionato.
- Il database contiene un'unica tabella:

  ARCO(n1,n2).
- La tupla (X,Y) in questa tabella codifica il fatto che c'è un arco diretto dal nodo con l'identificatore X a quello con l'identificatore Y.
- Non ci sono duplicati
- Si supponga che ogni nodo nel grafo fa parte di almeno un arco.

1. Scrivere una query SQL per trovare il nodo con il più alto out-degree.

```
SELECT n1
FROM Arco
GROUP BY n1
HAVING count(*) >= ALL
                    (SELECT count(*)
                     FROM Arco
                     GROUP BY n1);
```

2. Se (non) hai usato per il punto 1 la `Group By` scrivi la stessa query senza (con) la `Group By`.

```
SELECT DISTINCT n1
FROM Arco A1
WHERE  NOT EXISTS (SELECT *
                   FROM Arco A2
                   WHERE      A2.n1 > A1.n1 AND
                              (SELECT count(*) FROM Arco WHERE n1= A2.n1) >
                               (SELECT count(*) FROM Arco WHERE n1 = A1.n1));
```

1. Modificare la soluzione per 1 e 2 e trovare gli identificatori con il più alto “in-degree”.

- Scrivere una query per trovare un cammino che va dal nodo X al nodo Y.
- Assunzioni
  - X <> Y.
  - Esiste un solo cammino da X a Y.
  - Il diametro del grafo è al più 5.
  - Il grafo non ha cicli.
- Nella query cosa accade se
  - X=Y (X e Y sono lo stesso nodo).
  - Se non esiste un cammino da X a Y?
  - Se ci sono diversi cammini da X a Y?

```
CREATE VIEW cammino2archi AS
        SELECT R1.n1 AS da, R2.n2 AS a
        FROM Arco AS R1, arco AS R2
        WHERE R1.n2 = R2.n1;

CREATE VIEW cammino3archi AS
SELECT R1.n1 as da, R2.a
                FROM Arco AS R1, Cammino2archi AS R2
                WHERE R1.n2 = R2.da;

CREATE VIEW cammino4archi AS
SELECT R1.n1 as da, R2.a
                FROM Arco AS R1, Cammino3archi AS R2
                WHERE R1.n2 = R2.da
```

```
(SELECT n1,n2 FROM Arco where n1= 3 and n2= 2)
UNION
(SELECT da, a
FROM  cammino2archi
WHERE da=3 and a=2)
UNION
(SELECT da, a
FROM  cammino3archi
WHERE da=3 and a=2)
UNION
(SELECT da, a
FROM  cammino4archi
WHERE da=3 and a=2)
```

- La vista **Cammino(Da,a)** è la chiusura transitiva Relazione **Arco(Da,a)** e si calcola in SQL-99 attraverso la seguente interrogazione ricorsiva

```
WITH  RECURSIVE Cammino(da,a) AS
            (SELECT n1 as da ,n2 as a FROM Arco)
            UNION  ALL
            (SELECT R1.da, R2.a
            FROM Arco AS R1, Cammino AS R2
            WHERE R1.a = R2.da);
    SELECT * FROM Cammino;
```

- Si può modificare la soluzione per trovare il cammino minimo da X a Y?

```
WITH  RECURSIVE CamminoMinimo(da,a,w) AS
                (SELECT n1 as da ,n2 as a, 1 as w FROM Arco)
                UNION ALL
                (SELECT R1.da, R2.a, w+1
                FROM Arco AS R1, CamminoMinimo AS R2
                WHERE R1.a = R2.da AND
                NOT EXISTS (SELECT *
                                FROM CamminoMinimo
                                WHERE da=R1.da and a = R2.a));
     SELECT * FROM Cammino;
```

- Scrivere una query SQL per trovare l'out-degree medio dei nodi nel grafo.

```
SELECT
(sum(R.outDegree) + 0.0) / ( SELECT count(*)
                             FROM ( SELECT n1
                                    FROM arco
                                    UNION
                                    SELECT n2
                                    FROM arco )
FROM (SELECT n1, count(*) AS outDegree
    FROM arco
 GROUP BY n1 ) R;
```