

- Dato lo schema:

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

Evento(id, data, idescursione, idguida)

Partecipante(idpartecipante, idevento)

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)

Evento(id, data, idescursione, idguida)

Partecipante(idpartecipante, idevento)

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:

$R1 = \textit{Escursione}$

$R2 = \textit{Escursione}$

$$\pi_{\textit{titolo}, \textit{descrizione}, \textit{difficoltà}}(R1) - \pi_{R1.\textit{titolo}, R1.\textit{descrizione}, R1.\textit{difficoltà}}(\sigma_{R1.\textit{costo} < R2.\textit{costo}}(R1 \times R2))$$

# 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, difficolta
FROM escursione
WHERE 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, 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 tutti gli eventi

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

Partecipante(idpartecipante, idevento)

Persona(id, nome, cognome)

Evento(id, data, idescursione, idguida)

Algebra:

$$R_1 = \rho_{id \rightarrow idevento}(\Pi_{id}(Evento))$$

$$R_2 = Partecipante \div R_1$$

$$R_3 = \pi_{nome, cognome}(R_2 \bowtie_{R_2.idpartecipante=Persona.id} Persona)$$



```
SELECT DISTINCT pa.idpartecipante, Persona.nome, Persona.cognome
FROM Partecipante pa, Persona
WHERE (SELECT COUNT(*) FROM Evento) = (SELECT COUNT(DISTINCT idevento)
                                         FROM Partecipante pa1,
                                         WHERE pa1.idpartecipante=pa.idpartecipante)
AND pa.idpartecipante = Persona.id
```



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

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

Partecipante(idpartecipante, idevento)

Persona(id, nome, cognome)

Evento(id, data, idescursione, id guida)

ALGEBRA

$$R_1 = Escursione$$

$$R_2 = Escursione$$

$$R_3 = \pi_{id}(Escursione) - \pi_{R_1.id} \left( \sigma_{R_1.difficolta < R_2.difficolta} (R_1 \times R_2) \right)$$

$$R_4 = \pi_{idguida} (Evento \bowtie_{Evento.idescursione=R_3.id} R_3)$$

$$R_5 = \pi_{idguida}(Evento) - R_4$$

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

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

Partecipante(idpartecipante, idevento)

Persona(id, nome, cognome)

Evento(id, data, idescursione, id guida)

SQL

```
SELECT DISTINCT idguida
FROM Evento
EXCEPT
SELECT DISTINCT idguida
FROM Escursione es, Evento ev
WHERE es.id = ev.idescursione AND
      es.difficoltà = (SELECT MAX(difficoltà)
                       FROM Escursione)
```

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

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

Partecipante(idpartecipante, idevento)

Persona(id, nome, cognome)

Evento(id, data, idescursione, idguida)

SQL

```
SELECT DISTINCT idguida
FROM Evento ev1
WHERE NOT EXIST (
    SELECT * FROM Evento ev2 JOIN Escursione es ON
        ev2.idescursione = es.id
    WHERE ev2.idguida = ev1.idguida
    AND es.difficolta = (SELECT MAX(difficolta)
        FROM Escursione))
```

- Trovare le coppie di persone che hanno partecipato sempre agli stessi eventi

$R1 = PARTECIPANTE$

$R2 = PARTECIPANTE$

$$R3 = \pi_{R1.idp, R2.idp, R1.ide} \left( R1 \bowtie_{\substack{R1.ide=R2.ide \\ \wedge R1.idp > R2.idp}} R2 \right)$$

$$R5 = \pi_{R1.idp, R2.idp, R1.ide} \left( R1 \bowtie_{\substack{R1.ide \neq R2.ide \\ \wedge R1.idp > R2.idp}} R2 \right) - R3$$

$$R6 = \pi_{R1.idp, R2.idp, R2.ide} \left( R1 \bowtie_{\substack{R1.ide \neq R2.ide \\ \wedge R1.idp > R2.idp}} R2 \right) - R3$$

$$\pi_{R1.idp, R2.idp}(R3) - (\pi_{R1.idp, R2.idp}(R5) \cup \pi_{R1.idp, R2.idp}(R6))$$

- Trovare le coppie di persone che hanno partecipato sempre agli stessi eventi

```
SELECT DISTINCT p1.idp, p2.idp
FROM Partecipante p1, Partecipante p2
WHERE p1.ide = p2.ide AND
      p1.idp > p2.idp AND
      NOT EXISTS (SELECT *
                   FROM Evento e
                   WHERE
                     (EXISTS (SELECT * FROM Partecipante p3
                               WHERE p3.ide= e.id AND p3.idp= p1.idp) AND
                      NOT EXISTS (SELECT * FROM Partecipante p3
                                   WHERE p3.ide= e.id AND p3.idp= p2.idp))
                    OR
                     (EXISTS (SELECT * FROM Partecipante p3
                               WHERE p3.ide=e.id AND p3.idp= p2.idp) AND
                      NOT EXISTS (SELECT * FROM Partecipante p3
                                   WHERE p3.ide= e.id AND p3.idp=p1.idp))
                   )
```

- Trovare le coppie di persone che hanno partecipato sempre gli stessi eventi

```

SELECT DISTINCT p1.idp, p2.idp
FROM Partecipante p1, Partecipante p2
WHERE p1.ide = p2.ide AND
      p1.idp > p2.idp AND
      NOT EXISTS
        (SELECT *
         FROM Evento e
         WHERE
           NOT EXISTS
             (SELECT * FROM Partecipante p3, Partecipante p4
              WHERE p3.ide=e.id AND p4.ide=e.id
                 AND p3.idp=p2.idp AND p4.idp=p1.idp ))

```

- Dire ogni accompagnatore quanti eventi ha guidato;

```
SELECT idguida, COUNT (*)  
FROM Evento  
GROUP BY idguida
```

- 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).

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

DataEscursione(id, data, idescursione, idguida)

```
CREATE TRIGGER esperienza
AFTER INSERT ON Evento
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 e, Evento de
        WHERE de.idescursione = e.id AND
              de.idguida = N.idGuida AND
              difficolta = Y-1;
        IF X < 5 THEN
            DELETE FROM Evento 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).

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

DataEscursione(id, data, idescursione, idguida)

```
CREATE TRIGGER AutorizzaGuida
BEFORE INSERT ON Evento
FOR EACH ROW
DECLARE Livello, Conteggio NUMBER
BEGIN
    SELECT difficolta INTO Livello FROM Escursione
    WHERE id = NEW.idescursione;
    IF Livello > 1 THEN
        SELECT COUNT(*) INTO Conteggio
        FROM Evento de, Escursione e
        WHERE e.id = de.idescursione AND de.idguida=NEW.idguida
        AND e.difficolta = (Livello - 1);

        IF Conteggio < 5 THEN
            SIGNAL SQLSTATE '0000001' "Escursione non adeguata"
        END IF
    END IF
END
```

- Si consideri un database SQL per memorizzare un grafo direzionato.
- Il database contiene un'unica tabella:  $\text{ARCO}(n1, n2)$ .
- 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(*) >= (SELECT MAX(A2.outdegree)
                     FROM (SELECT n1, COUNT(*) outdegree
                           FROM Arco
                           GROUP BY n1) A2)
```

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
                      (SELECT COUNT (*)
                       FROM Arco WHERE n1 = A2.n1) >
                      (SELECT COUNT (*)
                       FROM Arco WHERE n1 = A1.n1))
```

- 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(\*) outDegree**  
**FROM arco GROUP BY n1) R;**

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

- Si consideri lo schema di base di dati sulle relazioni:
  - MATERIE (Codice, Facoltà, Denominazione, Professore)
  - STUDENTI (Matricola, Cognome, Nome, Facoltà)
  - PROFESSORI (Matricola, Cognome, Nome)
  - ESAMI (Studente, Materia, Voto, Data)
  - PIANIDISTUDIO (Studente, Materia, Anno)
  
- Formulare in algebra relazionale ed in SQL le seguenti query:
  1. gli studenti che hanno riportato in almeno un esame una votazione pari a 30, mostrando , per ciascuno di essi, nome e cognome e data della prima di tali occasioni;
  2. per ogni insegnamento della facoltà di ingegneria, gli studenti che hanno superato l' esame nell'ultima seduta svolta;
  3. gli studenti che hanno superato tutti gli esami previsti dal rispettivo piano di studio;
  4. per ogni insegnamento della facoltà di lettere, lo studente (o gli studenti) che hanno superato l'esame con il voto più alto;
  5. gli studenti che hanno in piano di studio solo gli insegnamenti della propria facoltà;
  6. nome e cognome degli studenti che hanno sostenuto almeno un esame con un professore che ha il loro stesso nome proprio.

- Si consideri lo schema relazionale composto dalle seguenti relazioni:
  - PROFESSORI (Codice, Cognome, Nome)
  - CORSI ( Codice, Denominazione, Professore)
  - STUDENTI (Matricola, Cognome, Nome)
  - ESAMI (Studente, Corso, Data, Voto)
- Formulare le espressioni dell'algebra che producano:
  1. Gli esami superati dallo studente Pico della Mirandola (supposto unico), con indicazione, per ciascuno, della denominazione del corso, del voto e del cognome del professore;
  2. i professori che tengono due corsi (e non più di due), con indicazione di cognome e nome del professore e denominazione dei due corsi.