- 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)

- SQL:

```
SELECT titolo, descrizione, durata, 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, 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)
DataEscursione(id, data, idescursione, id guida)

```
SELECT nome, cognome
FROM Persona p
WHERE NOT EXIST(
    SELECT *
    FROM Escursione e
    WHERE NOT EXIST(
        SELECT *
        FROM Partecipante part, DataEscursione de
        WHERE part.idescursione = de.id
        AND part.idpartecipante = p.id
        AND de.idescursione = e.id
    )
)
```

- 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)

```
SELECT nome, cognome
FROM Persona p, Partecipante pa, DataEscursione de
WHERE p.id = pa.idpartecipante
AND pa.idescursione = de.id
GROUP BY p.nome, p.cognome
HAVING COUNT(DISTINCT de.idescursione) = (SELECT COUNT(*)
                                          FROM Escursione)
```

- 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)

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 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)

```
SELECT idguida
FROM DataEscursione de
JOIN Escursione e ON e.id = de.idescursione
GROUP BY idguida
HAVING MAX(e.difficolta) < (SELECT MAX(difficolta) FROM
Escursione)
```

- 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)

```
SELECT DISTINCT idguida
FROM DataEscursione de1
WHERE NOT EXIST(
   SELECT *
   FROM Escursione e
   JOIN DataEscursione de2 ON de2.idescursione = e.id
   WHERE de2.idguida = de1.idguida
   AND e.difficolta = (SELECT MAX(difficolta) FROM escursione)
)
```

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

# SQL

```
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 dataEscursione e
                  WHERE  e.id <> p1.ide AND (
                 (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))))
```

- Dire ogni accompagnatore quante escursioni ha guidato;

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

- 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(*) >=
                    (SELECT max(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));
```

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

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

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

# Query Ricorsive sintassi

WITH R AS <Definizione di R> <Query che usa R>

In SQL possiamo usare delle relazioni temporanee usate in modo simile ai predicati intensionali Datalog.

Stratificazione: in una query ricorsiva l'uso della negazione ddve essere monotona. Evitare la negazione nella definizione con mutua ricorsione

- Scrivere una query per trovare un cammino che va dal nodo X al nodo Y

```
WITH
    RECURSIVE Cammino(da,a) AS
        (SELECT da,a FROM Arco)
        UNION
        (SELECT R1.da, R2.a
        FROM Arco AS R1, Cammino AS R2
        WHERE R1.a = R2.da);
  SELECT * FROM Cammino
  WHERE da = X and a = Y;
```

- Volo(compagnia, da, a)
- Trovare le città raggiungibili con un volo AZ ma non con un volo V7

- SequelDI(film,sequel)

1. Trovare le coppie (x,y) tali che y è sequel di x o il sequel del sequel ecc. Definiamo la relazione FollowON

2. Trovare i film x che hanno almeno due FollowON (sequel e sequel del sequel)

3. Trovare le coppie (x,y) tali che y è il FollowON e y ha al piu' un solo FollowON

# 1

```
WITH RECURSIVE FollowON AS (
   SELECT film AS x, sequel AS y
   FROM SequelDI
   UNION
   SELECT f.x, s.sequel
   FROM FollowON f JOIN
   SequelDI s ON f.y = s.film)
SELECT * FROM FollowON;
```

- 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 re lazioni:
  - 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 indi cazione di cognome e nome del professore e denominazione dei due corsi.