```
/* SOCIAL NETWORK */

DROP SCHEMA IF EXISTS social_network CASCADE;


------------------------- creazione dello schema
CREATE SCHEMA social_network;

SET search_path TO social_network;


------------------------- creazione delle tabelle e dei vincoli intrarelazionali
CREATE TABLE student(
        id integer NOT NULL
,       name varchar(20) NOT NULL
,       age integer NOT NULL
,       CONSTRAINT pk_student PRIMARY KEY (id)
);

CREATE TABLE friend(
        id1 integer
,       id2 integer
,       CONSTRAINT pk_friend PRIMARY KEY (id1, id2)
-- si osservi che id1 ed id2 vengono comunque impostati come NOT NULL (vincolo d'integrit‡
dell'entit‡)
);

COMMENT ON TABLE friend IS 'Symmetric relationship';

CREATE TABLE likes(
        id1 integer
,       id2 integer
,       CONSTRAINT pk_likes PRIMARY KEY (id1, id2)
);

COMMENT ON TABLE friend IS 'Non-symmetric relationship';


------------------------- creazione dei vincoli interrelazionali
ALTER TABLE friend
ADD CONSTRAINT fk_friend1_student FOREIGN KEY (id1)
REFERENCES student (id)
ON DELETE CASCADE ON UPDATE RESTRICT;

ALTER TABLE friend
ADD CONSTRAINT fk_friend2_student FOREIGN KEY (id2)
REFERENCES student (id)
ON DELETE CASCADE ON UPDATE RESTRICT;

ALTER TABLE likes
ADD CONSTRAINT fk_likes1_student FOREIGN KEY (id1)
```

```
REFERENCES student (id)
ON DELETE CASCADE ON UPDATE RESTRICT;

ALTER TABLE likes
ADD CONSTRAINT fk_likes2_student FOREIGN KEY (id2)
REFERENCES student (id)
ON DELETE CASCADE ON UPDATE RESTRICT;



-------------------------- istanziazione
start transaction;
insert into Student(id, name, age) values (1510, 'Jordan', 16);
insert into Student(id, name, age) values (1689, 'Gabriel', 16);
insert into Student(id, name, age) values (1381, 'Tiffany', 16);
insert into Student(id, name, age) values (1709, 'Cassandra', 16);
insert into Student(id, name, age) values (1101, 'Haley', 17);
insert into Student(id, name, age) values (1782, 'Andrew', 17);
insert into Student(id, name, age) values (1468, 'Kris', 17);
insert into Student(id, name, age) values (1641, 'Brittany', 17);
insert into Student(id, name, age) values (1247, 'Alexis', 19);
insert into Student(id, name, age) values (1316, 'Austin', 19);
insert into Student(id, name, age) values (1911, 'Gabriel', 19);
insert into Student(id, name, age) values (1501, 'Jessica', 19);
insert into Student(id, name, age) values (1304, 'Jordan', 20);
insert into Student(id, name, age) values (1025, 'John', 20);
insert into Student(id, name, age) values (1934, 'Kyle', 20);
insert into Student(id, name, age) values (1661, 'Logan', 20);

insert into Friend(id1, id2) values (1510, 1381);
insert into Friend(id1, id2) values (1510, 1689);
insert into Friend(id1, id2) values (1689, 1709);
insert into Friend(id1, id2) values (1381, 1247);
insert into Friend(id1, id2) values (1709, 1247);
insert into Friend(id1, id2) values (1689, 1782);
insert into Friend(id1, id2) values (1782, 1468);
insert into Friend(id1, id2) values (1782, 1316);
insert into Friend(id1, id2) values (1782, 1304);
insert into Friend(id1, id2) values (1468, 1101);
insert into Friend(id1, id2) values (1468, 1641);
insert into Friend(id1, id2) values (1101, 1641);
insert into Friend(id1, id2) values (1247, 1911);
insert into Friend(id1, id2) values (1247, 1501);
insert into Friend(id1, id2) values (1911, 1501);
insert into Friend(id1, id2) values (1501, 1934);
insert into Friend(id1, id2) values (1316, 1934);
insert into Friend(id1, id2) values (1934, 1304);
insert into Friend(id1, id2) values (1304, 1661);
insert into Friend(id1, id2) values (1661, 1025);
insert into Friend(id1, id2) select id2, id1 from Friend;

insert into Likes(id1, id2) values(1689, 1709);
```

```
insert into Likes(id1, id2) values(1709, 1689);
insert into Likes(id1, id2) values(1782, 1709);
insert into Likes(id1, id2) values(1911, 1247);
insert into Likes(id1, id2) values(1247, 1468);
insert into Likes(id1, id2) values(1641, 1468);
insert into Likes(id1, id2) values(1316, 1304);
insert into Likes(id1, id2) values(1501, 1934);
insert into Likes(id1, id2) values(1934, 1501);
insert into Likes(id1, id2) values(1025, 1101);
commit;



-------------------------- Alcuni esempi
-------------------------- Quante persone hanno ricevuto dei like?
SELECT COUNT(DISTINCT id2) FROM Likes; -- 8


-------------------------- Quali inserimenti vanno a buon fine?
INSERT INTO likes (id1, id2) VALUES (1025,1689); -- OK -- Nuovo like

INSERT INTO likes (id1, id2) VALUES (1247,1934); -- OK -- Nuovo like

INSERT INTO likes (id1, id2) VALUES (1501,1934); -- DUPL. PK -- Like gi‡ presente

INSERT INTO likes (id1, id2) VALUES (1267,1468); -- VIOL. FK -- 1267 Non esiste

INSERT INTO likes (id1, id2) VALUES (1101,1404); -- VIOL. FK -- 1404 Non esiste


-------------------------- Quali aggiornamenti vanno a buon fine?
update Friend set id1 = 1501 where id2 = 1934; -- DUPL. PK -- 1501 gi‡ presente

update Likes set id2 = 1711 where id2 = 1709; -- FK VIOLATION -- 1711 Non esiste

update Student set age = 19 where name = 'John'; -- OK -- Aggiornamento


-------------------------- Esempio di cancellazioni di dati
DELETE FROM student WHERE name = 'Kris'; -- OK
```

```
/* BIBLIOTECA */



/* CREAZIONE DELLO SCHEMA*/

CREATE SCHEMA biblioteca;

SET search_path TO biblioteca;

CREATE TABLE biblioteca.socio
(
  ci character varying(10) NOT NULL,
  nome character varying(50) NOT NULL,
  sesso character(1),
  CONSTRAINT socio_pkey PRIMARY KEY (ci),
  CONSTRAINT socio_sesso_check CHECK (sesso IN ('M','F'))
);

CREATE TABLE biblioteca.genere
(
  nome character varying(50) NOT NULL,
  sala character(1) NOT NULL,
  CONSTRAINT genere_pkey PRIMARY KEY (nome)
);

CREATE TABLE biblioteca.libro
(
  isbn character varying(13) NOT NULL,
  titolo character varying(250),
  autore character varying(50),
  genere character varying(50),
  CONSTRAINT libro_pkey PRIMARY KEY (isbn),
  CONSTRAINT fk_libro_genere FOREIGN KEY (genere)
      REFERENCES biblioteca.genere (nome)
      ON UPDATE CASCADE ON DELETE RESTRICT
);

CREATE TABLE biblioteca.ha_letto
(
  ci character varying(10) NOT NULL,
  isbn character varying(13) NOT NULL,
  CONSTRAINT pk_ha_letto PRIMARY KEY (ci, isbn),
  CONSTRAINT fk_ha_letto_libro FOREIGN KEY (isbn)
      REFERENCES biblioteca.libro (isbn)
      ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT fk_ha_letto_socio FOREIGN KEY (ci)
      REFERENCES biblioteca.socio (ci)
      ON UPDATE CASCADE ON DELETE CASCADE
);
```

```
insert into socio values
('AA1111111','Ulderico Cavalli','M'),
('BB2222222','Clotilde Bianchi','F'),
('CC3333333','Ellade Pedone','M'),
('DD4444444','Ignazio Torre','M'),
('EE5555555','Regina Neri','F'),
('FF6666666','Germana Alfieri','F');

insert into genere values
('giallo','A'),
('orrore','A'),
('poesia','B');

insert into libro values
('88-11-11111-1','Il cane dei Baskerville','A. C. Doyle','giallo'),
('88-22-22222-2','I delitti della Rue Morgue','E. A. Poe','giallo'),
('88-33-33333-3','La bottiglia di Amontillado','E. A. Poe','orrore'),
('88-44-44444-4','Il gatto nero','E. A. Poe','orrore'),
('88-55-55555-5','Ossi di seppia','E. Montale','poesia'),
('88-66-66666-6','A ciascuno il suo','L. Sciascia','giallo'),
('88-77-77777-7','Canti','G. Leopardi','poesia'),
('88-88-88888-8','Finzioni','L. Borges',null);

insert into ha_letto values
('CC3333333','88-11-11111-1'),
('CC3333333','88-33-33333-3'),
('FF6666666','88-33-33333-3'),
('BB2222222','88-77-77777-7'),
('BB2222222','88-55-55555-5'),
('AA1111111','88-88-88888-8');
```

```
set search_path to biblioteca;

-- i nomi dei soci di sesso femminile che hanno letto qualche libro
-- inner join esplicito
select distinct nome
from socio
        join ha_letto on socio.ci = ha_letto.ci
where sesso = 'F';

-- natural join
select distinct nome
from socio natural join ha_letto
where sesso = 'F';

--- inner join esplicito sul where
select distinct nome
from socio, ha_letto
where sesso = 'F' and socio.ci = ha_letto.ci;

-- caso con exists
select s.nome
from socio as s
where sesso = 'F' and exists (select * from ha_letto as h where s.ci = h.ci);

-- caso con in
select s.nome
from socio as s
where sesso = 'F' and s.ci in (select h.ci from ha_letto as h);

-- i titoli dei libri nella sala 'A'
select titolo
from libro natural join genere
where sala = 'A';

-- gli autori ed il genre dei libri letti da soci maschi
select distinct autore, genere
from socio, libro, ha_letto
where ha_letto.isbn = libro.isbn and ha_letto.ci = socio.ci and sesso = 'M'

-- i titoli dei libri gialli letti da Ellade Pedone
select titolo
from socio natural join ha_letto natural join libro
where genere = 'giallo' and nome = 'Ellade Pedone'

-- i titoli dei libri e la sala in cui sono collocati
select titolo, sala
from libro join genere on genere = nome

-- i titoli dei libri e la sala in cui sono collocati, inclusi i libri di cui
-- non è possibile ricavare la collezione (genere)
select titolo, sala
from libro left outer join genere on libro.genere = genere.nome
```

```
-- i numeri di CI delle coppie di soci che hanno letto uno stesso libro
select distinct h1.ci, h2.ci
from ha_letto as h1, ha_letto as h2
where h1.ci < h2.ci and h1.isbn = h2.isbn

-- i numeri di CI di chi ha letto dei libri senza genere
select distinct ci
from ha_letto natural join libro
where genere is null

-- i numeri di CI dei soci che hanno letto almento un libro
-- situato nella sala A
select distinct ci
from ha_letto, libro, genere
where libro.isbn = ha_letto.isbn and genere = nome and sala = 'A'

select socio.ci
from socio
where exists (select *
                        from ha_letto as h, libro as l, genere as c
                       where h.ci = socio.ci and h.isbn=l.isbn
                        and l.genere=c.nome and c.sala = 'A')
```

```
DROP SCHEMA IF EXISTS movies CASCADE;

CREATE SCHEMA movies;

SET search_path TO movies;


------------------------- Definizione delle tabelle e dei vincoli intrarelazionali
CREATE TABLE movie(
        mid integer PRIMARY KEY -- diversa sintassi, imposto il vincolo sulla colonna
,       title varchar(50) NOT NULL
,       year integer
,       director varchar(50)
);

COMMENT ON COLUMN movie.year IS 'Talvolta l''anno non è noto';
COMMENT ON COLUMN movie.director IS 'Talvolta il regista non è noto';

CREATE TABLE reviewer(
        rid integer PRIMARY KEY
,       name varchar(50) NOT NULL
);

CREATE TABLE rating(
        rid integer NOT NULL
,       mid integer NOT NULL
,       stars integer NOT NULL
,       ratingDate date NOT NULL
,       CONSTRAINT pk_rating PRIMARY KEY (rid, mid, ratingDate)
);


-------------------------  Aggiungo i vincoli interrelazionali
ALTER TABLE rating
ADD CONSTRAINT fk_rating_reviewer FOREIGN KEY (rid)
REFERENCES reviewer (rid)
ON DELETE CASCADE ON UPDATE RESTRICT;

ALTER TABLE rating
ADD CONSTRAINT fk_rating_movie FOREIGN KEY (mid)
REFERENCES movie (mid)
ON DELETE CASCADE ON UPDATE RESTRICT;
```

```
start transaction;
insert into Movie values(101, 'Gone with the Wind', 1939, 'Victor Fleming');
insert into Movie values(102, 'Star Wars', 1977, 'George Lucas');
insert into Movie values(103, 'The Sound of Music', 1965, 'Robert Wise');
insert into Movie values(104, 'E.T.', 1982, 'Steven Spielberg');
insert into Movie values(105, 'Titanic', 1997, 'James Cameron');
insert into Movie values(106, 'Snow White', 1937, null);
insert into Movie values(107, 'Avatar', 2009, 'James Cameron');
insert into Movie values(108, 'Raiders of the Lost Ark', 1981, 'Steven Spielberg');

insert into Reviewer values(201, 'Sarah Martinez');
insert into Reviewer values(202, 'Daniel Lewis');
insert into Reviewer values(203, 'Brittany Harris');
insert into Reviewer values(204, 'Mike Anderson');
insert into Reviewer values(205, 'Chris Jackson');
insert into Reviewer values(206, 'Elizabeth Thomas');
insert into Reviewer values(207, 'James Cameron');
insert into Reviewer values(208, 'Ashley White');

insert into Rating values(201, 101, 2, '2011-01-22');
insert into Rating values(201, 101, 4, '2011-01-27');
insert into Rating values(202, 106, 4, '2011-01-29');
insert into Rating values(203, 103, 2, '2011-01-20');
insert into Rating values(203, 108, 4, '2011-01-12');
insert into Rating values(203, 108, 2, '2011-01-30');
insert into Rating values(204, 101, 3, '2011-01-09');
insert into Rating values(205, 103, 3, '2011-01-27');
insert into Rating values(205, 104, 2, '2011-01-22');
insert into Rating values(205, 108, 4, '2011-01-27');
insert into Rating values(206, 107, 3, '2011-01-15');
insert into Rating values(206, 106, 5, '2011-01-19');
insert into Rating values(207, 107, 5, '2011-01-20');
insert into Rating values(208, 104, 3, '2011-01-02');
commit;
```

```
SET search_path TO movies;


/*
Ottieni i nomi dei critici, i titoli dei film, la corrispondente valutazione e
la data della valutazione, ordinati come segue:
 - nome del critico (in ordine alfabetico);
 - titolo del film (in ordine alfabetico)
 - valutazione (dalla più alta alla più bassa).
Qual è la data nella quinta riga del risultato?
*/

select reviewer.name, movie.title, rating.stars, rating.ratingdate
from movie join rating on movie.mid = rating.mid
        join reviewer on reviewer.rid = rating.rid
order by reviewer.name asc, movie.title asc, rating.stars desc

 -- Quanti film nella base di dati sono stati prodotti tra il 1977
 -- e il 1985 inclusi?
 select count(title)
 from movie
 where year between  1977 and 1985

-- Qual è la valutazione media dei film di James Cameron?
select avg(stars)
from movie, rating
where movie.mid = rating.mid and director = 'James Cameron'



-- Qual è la data della critica più recente a un film di Victor Fleming
select max(ratingDate)
from rating natural join movie
where director = 'Victor Fleming';

-- Versione senza max(ratingDate).
-- Idea: tra tutti i film con review di Victor Fleming non deve esistere un altro film con review di Victor
Fleming con data review successiva a quella della tupla che sto considerando
-- Serve il distinct in quanto potrei avere due review fatte nella data più recente
select distinct ratingDate
from rating natural join movie
where director = 'Victor Fleming' and not exists (select *
                                                  from rating r2 natural join movie m2
                                                  where m2.director = 'Victor Fleming'
                                                  and rating.ratingDate <
r2.ratingDate);
```

```
-- Ottieni il titolo e la differenza tra voto massimo e voto minimo per ogni film che abbia
-- ricevuto almeno due valutazioni. Ordina il risultato alfabeticamente rispetto al tiolo.
select m.title, max(ra.stars) - min(ra.stars)
from movie m, rating ra
where m.mid = ra.mid
group by m.title, m.mid
having count(*) >=2
order by m.title;

-- versione senza having count(*) >=2
-- Intuititvamente: per tenere un film con una sua review, deve esistere una review per lo stesso film
con data differente della prima (data è parte della PK)
select m.title, max(r1.stars) - min(r1.stars)
from movie m, rating r1
where m.mid = r1.mid and exists (
            select * from rating r2
            where r1.mid = r2.mid and r1.ratingDate != r2.ratingDate)
group by m.title, m.mid
order by m.title;



-- Tra i film che sono stati recensiti almeno una volta, trova (in ordine alfabetico)
-- i titoli di quelli che hanno ricevuto solo valutazioni >= 4
select distinct title
from rating, movie
where rating.mid = movie.mid and movie.mid not in (select r2.mid
                                                   from rating r2
                                                   where r2.stars <4)
order by title;

--Intuitivamente: dato un film recensito (quindi con una sua review), non deve esistere una review per
lo stesso film con voto <4
select distinct m.title
from movie m natural join rating
where not exists (select * from rating r where r.mid = m.mid and r.stars <4)
order by title;


-- Quanti registi hanno diretto film più vecchi del film con mid=107
select count(distinct director)
from movie
where movie.year < (select year from movie m2 where m2.mid=107);

-- Proposta da un vostro collega (e abbastanza simile all'idea di un altro)
-- E' ok, ma non avendo specificato la condizione di JOIN, faccio un prodotto cartesiano
-- IN GENERALE è da stare attenti alla dimensionalità di output (quadratica)
select count(distinct director)
from movie, (select year from movie m2 where m2.mid=107) as y
where movie.year < y.year;
```

```
-- Ottieni il titolo e la valutazione massima per ogni film che è stato valutato
-- almeno una volta, in ordine decrescente rispetto al voto
select title, max(stars) as max_voto
from movie join rating on movie.mid = rating.mid
group by title, movie.mid
order by max_voto desc;

-- Intuitivamente: dato un film con una sua review, non deve esistere per lo stesso film una review
con voto inferiore a quella che sto considerando
-- Il distinct serve perché potrei avere due review con lo stesso voto massimo
select distinct title, stars
from movie join rating on movie.mid = rating.mid
where not exists (select *
                  from rating rin
                  where rin.mid = rating.mid and rating.stars < rin.stars )
order by stars desc;



-- Ottieni, in ordine alfabetico, i nomi dei registi dei film che hanno ricevuto una valutazione media
-- di tutti i loro film superiore a 3 (presta attenzione al fatto che il nome di un regista può non essere
noto)
select director, avg(stars)
from movie join rating on movie.mid = rating.mid
where director is not null
group by director
having avg(stars) > 3
order by director;



-- Ottieni, per ciascun critico, il nome del critico e il numero di film da costui recensiti,
-- ordinando il risultato rispetto al nome.
select name, count(distinct mid)
from reviewer natural join rating
group by rid, name
order by name



-- Ripeti l'interrogazione precedente, ma includi nel risultato (in ordine alfabetico)
-- soltanto i critici che hanno recensito uno e un solo un film.
select name, count(distinct mid)
from reviewer natural join rating
group by rid, name
having count(distinct mid) = 1
order by name;

-- Intuitivamente: tengo una coppia reviewer-review, se non esiste review fatta dallo stesso reviewer
per un film diverso da quello coinvolto dalla review considerata
```

```
select name, count(distinct mid)
from reviewer natural join rating
where not exists (select *
                  from rating rin
                  where reviewer.rid = rin.rid and rating.mid != rin.mid)
group by rid, name
order by name;



-- Qual è il critico che ha recensito tutti i film di Spielberg?
-- Intuitivamente: dato un reviewer, non deve esistere un film di Spielberg t.c. non esista una review
del reviewer considerato
select reviewer.name
from reviewer
where not exists (select *
                  from movie
                  where director = 'Steven Spielberg' and
                  not exists(select *
                             from rating
                             where rating.mid = movie.mid and rating.rid = reviewer.rid ));
                 ?
```