Esercizio 1:

Sia dato il seguente schema relazionale relativo a film e attori:
$FILM(\underline{CodiceFilm}, Titolo, Regista, Anno)$;
$ATTORI(\underline{CodiceAttore}, Nome, Cognome, Sesso, DataNascita, Nazionalità)$;
$INTERPRETAZIONE(\underline{Film, Attore}, Ruolo)$.
Si assuma che ogni film sia identificato univocamente da un codice e sia caratterizzato da titolo, regista e anno di uscita. Per semplicità, si assuma che ogni film sia diretto da un unico regista e ogni regista sia identificato univocamente dal suo cognome. Si ammetta la possibilità che vi siano film diversi con lo stesso titolo (questo è il caso, ad esempio, dei remake). Ogni attore sia identificato univocamente da un codice e sia caratterizzato da nome, cognome, sesso, data di nascita e nazionalità. Si assuma che più attori possano recitare in un dato film e che un attore possa recitare in più film. Per semplicità, si assuma che, in ogni film, un attore possa svolgere un solo ruolo.
Definire preliminarmente le chiavi primarie, le eventuali altre chiavi candidate e, se ve ne sono, le chiavi esterne delle relazioni date. Successivamente, formulare opportune interrogazioni in algebra relazionale che permettano di determinare (senza usare l'operatore di divisione e usando solo se necessario le funzioni aggregate):

(a) gli attori che hanno recitato in almeno due film di Avati;

(b) il film (i film se più di uno) col maggior numero di attori;

(c) gli attori che hanno recitato in almeno un film di Tarantino, ma non in tutti.

a.

$FILM\_AVATI \leftarrow \Pi_{CODICEFILM}(\sigma_{REGISTA = AVATI}(FILM))$

$ATTORI\_VALIDI \leftarrow \Pi_{FILM,ATTORE}(\sigma_{FILM = CODF}(FILM\_AVATI \bowtie INTERPRETAZIONE))$

$ATTORI\_VALIDI\_1(FILM1, ATTORE1) \leftarrow ATTORI\_VALIDI$

$S \leftarrow \Pi_{ATTORE}\left(\sigma_{FILM < FILM1 \text{ AND } ATTORE = ATTORE1}(ATTORI\_VALIDI \times ATTORI\_VALIDI\_1)\right)$

b.

$FILM\_NUM \leftarrow {}_{FILM}\gamma_{COUNT(ATTORE)}(INTERPRETAZIONE)$

$FILM\_NUM\_1(FILM1, COUNT1) \leftarrow FILM\_NUM$

$NO\_GOOD(CODICE\_FILM) \leftarrow \Pi_{FILM}\left(\sigma_{FILM \neq FILM1 \text{ AND } COUNT < COUNT1}(FILM\_NUM \times FILM\_NUM1)\right)$

$S \leftarrow \Pi_{CODICEFILM}(FILM) - NO\_GOOD$

c. almeno 1 - tutti i film

$FILM\_TARANTINO(FILM) \leftarrow \Pi_{CODICEFILM}(\sigma_{REGISTA = TARANTINO}(FILM))$

$INT\_TARANTINO \leftarrow \Pi_{FILM,ATTORE}(INTERPRETAZIONE \bowtie FILM\_TARANTINO)$

$ALMENO\_UNO\_TAR(CODICEATTORE, FILM) \leftarrow \Pi_{ATTORE,FILM}(INT\_TARANTINO)$

$REQUISITI(FILM, ATTORE) \leftarrow \Pi_{FILM}(FILM\_TARANTINO) \times \Pi_{CODICEATTORE}(ATTORI)$

STATO_DI_FATTO ⟵ $\Pi_{FILM, ATTORE}$ (INTERPRETAZIONE)

NO_GOOD(FILM, CODICEATTORE) ⟵ REQUISITI − STATO_DI_FATTO

TUTTI_TARANTINO ⟵ $\Pi_{CODICEATTORE}$ (ATTORI) − $\Pi_{CODICEATTORE}$ (NO_GOOD)

S ⟵ $\Pi_{CODICEATTORE}$ (ALMENO_UNO_TAR) − TUTTI_TARANTINO

Esercizio 2:

Con riferimento all'Esercizio 1, formulare opportune interrogazioni in SQL che permettano di determinare quanto richiesto (senza usare l'operatore CONTAINS e usando solo se e quando necessario le funzioni aggregate).

a.
```
CREATE VIEW FILM_AVATI (FILM) AS
  SELECT CODICEFILM
  FROM FILM
  WHERE REGISTA = AVATI

SELECT CODICEATTORE
FROM ATTORI A1
WHERE EXIST ( SELECT *
              FROM INTERPRETAZIONE I1, I2, FILM_AVATI A1, A2
              WHERE A1.CODICEATTORE = I1.CODICEATTORE AND
                    A1.CODICEATTORE = I2.CODICEATTORE AND
                    I1.FILM < I2.FILM AND
                    I1.FILM = A1.FILM AND
                    I2.FILM = A2.FILM
            )
```

b.
```
SELECT FILM, COUNT(ATTORE) AS NUM
FROM INTERPRETAZIONE
GROUP BY FILM
HAVING NUM ≥ ALL (SELECT COUNT(ATTORE)
                  FROM INTERPRETAZIONE
                  GROUP BY FILM
                 )
```

c.
```
CREATE VIEW AS FILM_TARANTINO (FILM) AS
SELECT CODICE FILM
FROM FILM
WHERE REGISTA = TARANTINO

SELECT CODICEATTORE
FROM ATTORE A
WHERE A.CODICEATTORE IN ( SELECT ATTORE
                          FROM INTERPRETAZIONE, FILM
                          WHERE FILM = CODICEFILM AND
                                REGISTA = TARANTINO
                        )
EXCEPT
```

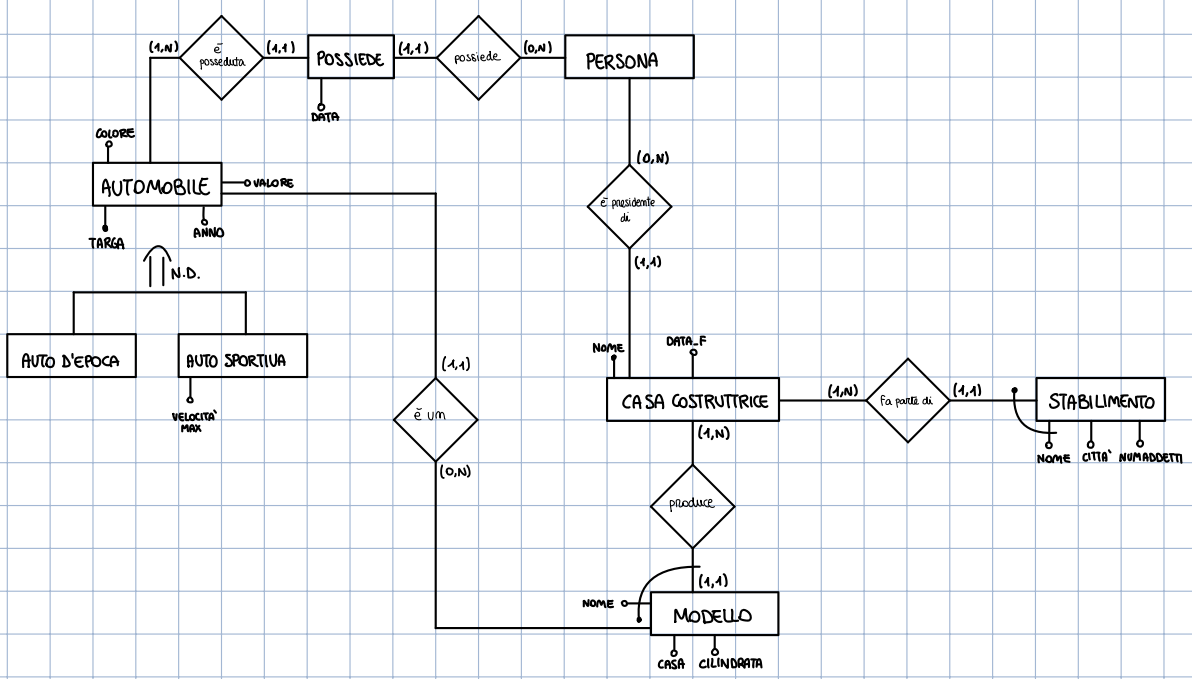
```
SELECT CODICEATTORE
FROM ATTORI A
WHERE NOT EXIST (SELECT *
                 FROM FILM_TARANTINO F
                 WHERE NOT EXIST ( SELECT *
                                   FROM INTERPRETAZIONE
                                   WHERE ATTORE = A.CODICEATTORE
                                       AND FILM = F.FILM
                                   )
                 )
```

Esercizio 3:

Si vuole realizzare una base di dati per la gestione di informazioni circa un insieme di automobili caratterizzato dal seguente insieme di requisiti.

- Ogni automobile sia identificata univocamente dalla sua targa e sia caratterizzata da un modello, un anno di fabbricazione, un colore, un valore di mercato e uno o più proprietari. Fra le automobili, vogliamo tener traccia del sottoinsieme delle automobili storiche (un'auto si dice storica se sono trascorsi 25 o più anni dall'anno di fabbricazione), del sottoinsieme delle automobili sportive, caratterizzate dalla velocità massima, e del sottoinsieme delle auto storiche sportive.
- Ogni modello sia caratterizzato da un nome, una casa costruttrice e una cilindrata. Il nome identifichi univocamente il modello all'interno dei modelli proposti dalla casa costruttrice (non si esclude la possibilità che case costruttrici diverse propongano modelli, ovviamente diversi, con lo stesso nome).
- Ogni casa costruttrice sia identificata univocamente dal proprio nome e sia caratterizzata dall'anno di fondazione e da un insieme di stabilimenti. Una stessa persona possa essere presidente di più case costruttrici. Ogni stabilimento sia caratterizzato un nome, che lo identifica univocamente nell'ambito della casa costruttrice, una città ove ha sede e un numero di addetti.

Si definisca uno schema Entità-Relazioni che descriva il contenuto informativo del sistema, illustrando con chiarezza le eventuali assunzioni fatte. Lo schema dovrà essere completato con attributi ragionevoli per ciascuna entità (identificando le possibili chiavi) e relazione. Vanno specificati accuratamente i vincoli di cardinalità e partecipazione di ciascuna relazione. Si definiscano anche eventuali regole di gestione (regole di derivazione e vincoli di integrità) necessarie per codificare alcuni dei requisiti attesi del sistema.



Esercizio 4:

Si considerino i seguenti schedule:

$s_1 : r_1(x), r_2(x), w_1(x), r_3(x), w_3(x), w_2(y)w_1(y);$

$s_2 : r_2(x), r_1(x), w_1(x), r_3(x), w_1(y), w_3(x), w_2(y);$

$s_3 : r_2(x), w_2(y), r_1(x), w_1(x), r_3(x), w_1(y), w_3(x).$

(a) Per ogni coppia di schedule s_i, s_j, con $1 \leq i, j \leq 3$ e $i \neq j$, stabilire se s_i e s_j sono o meno equivalenti rispetto alle viste e se sono o meno equivalenti rispetto ai conflitti.

(b) Stabilire se gli schedule dati appartengono o meno a VSR, CSR, 2PL e 2PL stretto.

a) legge $S_1 = \{(r_3(x), w_1(x))\}$ legge $S_2 = \{(r_3(x), w_1(x))\}$ legge $S_3 = \{(r_3(x), w_1(x))\}$

ultima scrittura $S_1 = \{w_3(x), w_1(y)\}$ ultima scrittura $= \{w_3(x), w_2(y)\}$ ultima scrittura $= \{w_3(x), w_1(y)\}$

VISTE → S_1 e S_3 : equivalenti rispetto alle viste

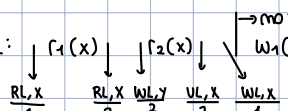
CONFLITTI → S_1 e S_3 :

S_1	S_3
$r_1(x), w_3(x)$	$r_2(x), w_1(x)$
$r_2(x), w_1(x)$	$r_2(x), w_3(x)$
$r_2(x), w_3(x)$	$w_2(y), w_1(y)$
$w_1(x), r_3(x)$	$r_1(x), w_3(x)$
$w_1(x), w_3(x)$	$w_1(x), r_3(x)$
$w_2(y), w_1(y)$	$w_1(x), w_3(x)$

=> sono equivalenti rispetto ai conflitti

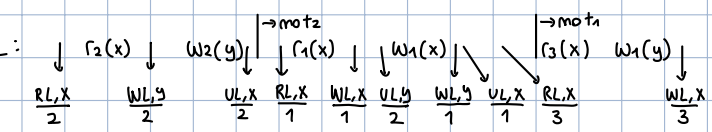
b) $S_1 \in$ VSR, CSR

2PL: $r_1(x)$ $r_2(x)$ $w_1(x)$ $r_3(x)$ $w_3(x)$ $w_2(y)$ $w_1(y)$ → $\notin$ 2PL



$S_3 \in$ VSR, CSR

2PL: $r_2(x)$ $w_2(y)$ $r_1(x)$ $w_1(x)$ $r_3(x)$ $w_1(y)$ $w_3(x)$ → $\in$ 2PL no 2stretto



Esercizio 5:

1. Quali sono i due principali meccanismi forniti dal sistema di basi di dati PostgreSQL per la formulazione di vincoli complessi, non codificabili attraverso chiavi primarie, chiavi esterne o dichiarazioni di univocità e not-null? e
2. Si consideri il seguente schema relazionale:
   STUDENTE(<u>matricola</u>, nome, cognome)
   ESAME(<u>codice</u>, descrizione)
   HA_SOSTENUTO(<u>matricola_studente</u>, <u>codice_esame</u>, valutazione, lode)

   Tenendo presente che:

   - si vogliono conoscere tutti i dettagli riguardanti studenti ed esami;
   - la valutazione è espressa da un numero intero compreso fra 0 e 30 e la lode può essere assegnata solamente a chi ottiene una valutazione pari a 30,

   si presenti del codice SQL che implementa le tre tabelle, unitamente alle chiavi primarie, alle chiavi esterne e a qualsiasi eventuale altro vincolo si ritenga necessario imporre.

1. TRIGGER con UDF (es. CHECK) e TRANSAZIONI
   ↳ in DDL ↳ in DML

2.
```
CREATE TABLE STUDENTE (
  MATRICOLA  INTEGER PRIMARY KEY,
  NOME VARCHAR(20),
  COGNOME VARCHAR(20)
);

CREATE TABLE ESAME (
 CODICE INTEGER PRIMARY KEY,
 DESCRIZIONE VARCHAR(50)
);

CREATE TABLE HA_SOSTENUTO(
  MATRICOLA_STUDENTE INTEGER REFERENCES STUDENTE,
  CODICE_ESAME INTEGER REFERENCES TABLE,
  VALUTAZIONE INTEGER CHECK (VALUTAZIONE BETWEEN 0 AND 30),
  LODE BOOLEAN CHECK (NOT LODE OR VALUTAZIONE = 30)
);
```