Ripasso – Vincoli

- Vincoli CHECK

```
CREATE TABLE tabella (
  attr_i tipo_i ... CHECK (attribute-level condition)
  ...
  CHECK (tuple-level condition)
)
```

- E' possibile aggiungere o rimuovere vincoli CHECK a livello di tupla successivamente alla definizione della tabella:

```
–ALTER TABLE tabella ADD [CONSTRAINT nome] CHECK (...)
–ALTER TABLE tabella DROP CONSTRAINT nome [CASCADE]
```

- Asserzioni:

```
CREATE ASSERTION nome_asserzione CHECK (condizione)
```

Ripasso – Definizione Viste

- Sintassi

```
CREATE [MATERIALIZED] VIEW vista [(attr1, ..., attrN)] AS
select_query

DROP VIEW vista [CASCADE];
```

- Note

  –la CREATE VIEW definisce una vista che può essere poi interrogata come se fosse una tabella

  –le viste sono di default virtuali; per le viste MATERIALIZED, invece, il DBMS mantiene delle strutture persistenti su disco che facilitano l'accesso alla vista (materializzazione non supportata in PostgreSQL)

  –la rimozione di una vista con DROP VIEW cancella la vista ma non interferisce con le tabelle di partenza su cui era definita la vista

Note sugli Esercizi Proposti

- PostgreSQL non supporta vincoli CHECK (tuple-level, attribute-level) con query annidate e asserzioni in generale
  - PostgreSQL verifica la sintassi del comando (utile), tuttavia non lo esegue
  - limitarsi a testare la query annidata e la query che definisce l'asserzione

- Esercizi 1, 3, 5
  - Fare riferimento allo script create_product_db.sql
- Esercizi 2, 4, 6
  - Fare riferimento allo script create_ship_db.sql

Esercizio 1 – Prodotti

rif. esercizi 7.2.2 (quesiti 1-3, 8) e 7.2.4 (quesiti 4-5)

Sia dato lo schema:

product (<u>model</u>, maker, type)
pc (<u>model</u>, speed, ram, hd, price)
laptop (<u>model</u>, speed, ram, hd, screen, price)
printer (<u>model</u>, color, type, price)

Modificare lo script per creare il DB definendo i seguenti vincoli a livello di attributo:

1. Gli unici tipi ammissibili di prodotto sono 'pc', 'laptop' e 'printer'
2. Un laptop deve essere veloce almeno 1.6 GHz
3. Gli unici tipi ammissibili di stampante sono 'ink-jet' e 'laser'

Introdurre mediante comandi ALTER i seguenti constraint a livello di tupla:

4. I PC con processore più lento di 2.0 GHz devono essere venduti a meno di 1000$
5. I laptop con schermo più piccolo di 15" devono avere un HD di almeno 40 Gb o devono essere venduti a meno di 1000$
6. Le stampanti laser a colori costano almeno 400$
7. Non possono esserci più di 10 prodotti per maker nel DB (*)
8. Ogni model in product deve comparire in pc, laptop o printer (si può scrivere a livello di attributo? come cambia lo script? quale operazione viola il vincolo?) (*)

(*) non supportato da PostgreSQL

Esercizio 1 – Soluzioni (1/2)

1. Gli unici tipi ammissibili di prodotto sono ‘pc’, ‘laptop’ e ‘printer’

```
tab. product, attr. type:  CHECK (type IN ('pc', 'laptop', 'printer'))
```

2. Un laptop deve essere veloce almeno 1.6 GHz

```
tab. laptop, attr. speed:  CHECK (speed >= 1.6)
```

3. Gli unici tipi ammissibili di stampante sono ‘ink-jet’ e ‘laser’

```
tab. printer, attr. type:  CHECK (type IN ('laser', 'ink-jet'))
```

4. I PC con processore più lento di 2.0 GHz devono essere venduti a meno di 1000$

```
ALTER TABLE pc ADD CHECK (speed >= 2.0 OR price < 1000);
```

5. I laptop con schermo più piccolo di 15” devono avere un HD di almeno 40 Gb o devono essere venduti a meno di 1000$

```
ALTER TABLE laptop ADD CHECK(screen >= 15 OR hd >= 40 OR price < 1000)
```

6. Le stampanti laser a colori costano almeno 400$

```
ALTER TABLE printer ADD CHECK (type <> 'laser' OR NOT color
                               OR price >= 400);
```

Esercizio 1 – Soluzioni (2/2)

7. Non possono esserci più di 10 prodotti per maker nel DB

```
ALTER TABLE product ADD CHECK ( 10 >= ( SELECT COUNT(*)
                                        FROM   product p
                                        WHERE  p.maker = maker ) );
```

8. Ogni model in product deve comparire in pc, laptop o printer

```
ALTER TABLE product ADD CHECK (model IN ( ( SELECT model
                                            FROM   pc )
                                          UNION
                                          ( SELECT model
                                            FROM   laptop )
                                          UNION
                                          ( SELECT model
                                            FROM   printer ) ) );
```

Il vincolo si può scrivere a livello di attributo. Modificando lo script si verifica un problema di dipendenza circolare: il vincolo nella tabella product dipende dalle definizioni delle tabelle pc, laptop e printer, ma queste dipendono da product per il vincolo di integrità referenziale – la soluzione è 'rompere' il loop definendo i vincoli di integrità soltanto in un secondo tempo mediante ALTER. Il vincolo così definito (a livello di attributo o di tupla) può essere violato soltanto nel caso in cui sia rimossa una tupla da pc, laptop o printer.

Esercizio 2 – Navi da battaglia

rif. esercizio 7.2.5 (quesiti 3, 6, 7) e 7.3.2 (quesito 4, 7)

Sia dato lo schema:
classes (class, type, country, num_guns, bore, displacement)
ships (name, class, launched)
outcomes (ship, battle, result)
battles (name, date)

Modificare lo script per creare il DB definendo i seguenti vincoli a livello di attributo:
1. Non ci possono essere battaglie 'nel futuro' (usare funzione NOW())
2. Il risultato di una battaglia per una nave può essere soltanto 'ok', 'sunk', 'damaged'
3. Nessuna classe di navi può avere cannoni con calibro massimo maggiore di 18"
4. Nessuna nave può avere più di 15 cannoni
5. La Svizzera (country = 'Switzerland') non può avere navi da guerra.

Introdurre mediante comandi ALTER i seguenti constraint a livello di tupla:

6. Navi con più di 9 cannoni devono avere calibro massimo minore o uguale a 14"
7. Gli incrociatori non possono avere stazza maggiore di 35000 t
8. Nessuna nave può partecipare ad una battaglia prima di essere stata varata (può essere scritto anche come constraint a livello di attributo?) (*)

(*) non supportato da PostgreSQL

Esercizio 2 – Soluzioni (1/2)

1. Non ci possono essere battaglie ‘nel futuro’ (usare funzione NOW())

```
tab. battles, attr. date:     CHECK (CAST(date AS TIMESTAMP) < NOW())
```

2. Il risultato di una battaglia può essere soltanto ‘ok’, ‘sunk’, ‘damaged’

```
tab. outcomes, attr. result:  CHECK(result IN ('ok','damaged','sunk'))
```

3. Nessuna classe di navi può avere cannoni con calibro massimo maggiore di 18”

```
tab. classes, attr. bore:     CHECK (bore <= 16)
```

4. Nessuna nave può avere più di 15 cannoni

```
tab. classes, attr. num_guns: CHECK (num_guns <= 15)
```

5. La Svizzera (country = ‘Switzerland’) non può avere navi da guerra.

```
tab. classes, attr. country:  CHECK (country <> 'Switzerland')
```

6. Navi con più di 9 cannoni devono avere calibro massimo minore o uguale a 14”

```
ALTER TABLE classes ADD CHECK (num_guns <= 9 OR bore <= 14);
```

Esercizio 2 – Soluzioni (2/2)

7. Gli incrociatori non possono avere stazza maggiore di 35000 t

```
ALTER TABLE classes ADD CHECK (type = 'bb' OR displacement <= 35000)
```

8. Nessuna nave può partecipare ad una battaglia prima di essere stata varata (può essere scritto anche come constraint a livello di attributo?) (*)

```
ALTER TABLE outcomes
ADD CHECK (ship IN ( SELECT s.name
                     FROM   ships s
                     WHERE  s.launched <=
                            ( SELECT EXTRACT(YEAR FROM b.date) AS byear
                              FROM   battles b
                              WHERE  b.name = battle ) ) );
```

Il vincolo si può scrivere a livello di attributo. Si noti che il vincolo è verificato soltanto a seguito di inserimento o modifica tupla in outcomes. Pertanto, il vincolo può essere violato nei casi che (1) la data di varo della nave in ships sia modificata o (2) la data della battaglia sia modificata.

Esercizio 3 – Prodotti

rif. esercizio 7.4.1 (quesiti 1-4)

Sia dato lo schema:

product (<u>model</u>, maker, type)
pc (<u>model</u>, speed, ram, hd, price)
laptop (<u>model</u>, speed, ram, hd, screen, price)
printer (<u>model</u>, color, type, price)

Si scrivano i comandi SQL per creare le seguenti asserzioni (nota: le asserzioni non sono supportate in postgres – al più provare la select usata per definire l'asserzione):

1. Un produttore di laptop non può produrre anche stampanti
2. Se un laptop ha HD più grande di un certo PC, allora quel laptop deve avere anche un prezzo maggiore di quel PC
3. Ciascun model menzionato nella tabella product deve comparire in pc, laptop o printer
4. Per ciascun laptop prodotto, un produttore deve avere a listino anche un PC con velocità CPU maggiore
5. Il prezzo medio dei laptop deve essere superiore al prezzo medio dei PC

Esercizio 3 – Soluzioni (1/3)

1. Un produttore di laptop non può produrre anche stampanti

```
CREATE ASSERTION cannot_make_laptop_and_printer CHECK
( NOT EXISTS ( SELECT *
               FROM   product p1, product p2
               WHERE  p1.maker = p2.maker AND p1.model <> p2.model AND
                      p1.type = 'laptop' AND p2.type = 'printer' ) );
```

2. Se un laptop ha HD più grande di un certo PC, allora quel laptop deve avere anche un prezzo maggiore di quel PC

```
CREATE ASSERTION better_hd_greater_price CHECK
( NOT EXISTS ( SELECT *
               FROM   laptop l, pc p
               WHERE  l.hd > p.hd AND l.price <= p.price ) );
```

Nota: in entrambi i casi il procedimento seguito è il seguente. Si nega la condizione e si scrive una query che ritorna una tupla per ogni caso in cui la condizione è violata (e.g. produttori di laptop che vendono anche stampanti; coppie laptop/PC dove il primo ha HD maggiore del secondo ma prezzo non superiore). Quindi si scrive l'asserzione imponendo che tale query non produca risultati, usando il costrutto NOT EXISTS (…)

Esercizio 3 – Soluzioni (2/3)

3. Ciascun model menzionato nella tabella product deve comparire in pc, laptop o printer

```
CREATE ASSERTION pc_in_product_and_pc_tables CHECK
( NOT EXISTS ( SELECT *
               FROM   product NATURAL LEFT JOIN pc
               WHERE  type = 'pc' AND pc.model IS NULL ) );

CREATE ASSERTION laptop_in_product_and_laptop_tables CHECK
( NOT EXISTS ( SELECT *
               FROM   product NATURAL LEFT JOIN laptop
               WHERE  type = 'laptop' AND laptop.model IS NULL ) );

CREATE ASSERTION printer_in_product_and_printer_tables CHECK
( NOT EXISTS ( SELECT *
               FROM   product
               WHERE  type = 'printer' AND
                      NOT EXISTS ( SELECT *
                                   FROM   printer
                                   WHERE  printer.model =
                                          product.model ) ) );
```

Esercizio 3 – Soluzioni (3/3)

4. Per ciascun laptop prodotto, un produttore deve avere a listino anche un PC con velocità CPU maggiore

```
CREATE ASSERTION must_sell_faster_pc CHECK
( NOT EXISTS ( SELECT *
               FROM   product p1 NATURAL JOIN laptop
               WHERE  NOT EXISTS ( SELECT *
                                   FROM   product p2 NATURAL JOIN pc
                                   WHERE  p2.maker = p1.maker AND
                                          pc.speed > laptop.speed )));
```

5. Il prezzo medio dei laptop deve essere superiore al prezzo medio dei PC

```
CREATE ASSERTION laptop_more_expensive_than_pc CHECK
( ( SELECT AVG(price)
    FROM   laptop ) >= ( SELECT AVG(price)
                         FROM   pc ) );
```

Nota: un'altro tipo frequente di asserzione consiste nel confronto tra aggregati, come nell'esempio sopra.

Esercizio 4 – Navi da battaglia

rif. esercizio 7.4.2 (quesiti 3, 6, 7)

Sia dato lo schema:

classes (<u>class</u>, type, country, num_guns, bore, displacement)
ships (<u>name</u>, class, launched)
outcomes (<u>ship</u>, <u>battle</u>, result)
battles (<u>name</u>, date)

Si scrivano i comandi SQL per creare le seguenti asserzioni (nota: le asserzioni non sono supportate in postgres – al più provare la select usata per definire l'asserzione):

1. Nessuna classe può avere più di 3 navi
2. Per ogni classe, deve esserci una nave di quella classe avente lo stesso nome
3. Nessun paese può avere sia incrociatori (bc) che corazzate (bb)
4. Una nave con almeno 9 cannoni non può essere affondata in una battaglia che coinvolge navi con meno di 9 cannoni
5. All'interno di una classe, la prima nave varata deve portare il nome della classe
6. Una battaglia deve coinvolgere almeno due navi di paesi diversi

Esercizio 4 – Soluzioni (1/3)

1. Nessuna classe può avere più di 3 navi

```
CREATE ASSERTION no_class_with_more_than_three_ships CHECK
( NOT EXISTS ( SELECT   class, COUNT(*)
               FROM     classes NATURAL JOIN ships
               GROUP BY class
               HAVING   COUNT(*) > 3 ) );
```

2. Per ogni classe, deve esserci una nave di quella classe avente lo stesso nome

```
CREATE ASSERTION each_class_with_homonymous_ship CHECK
( NOT EXISTS ( SELECT *
               FROM   classes c
               WHERE  c.class NOT IN ( SELECT s.name
                                       FROM   ships s
                                       WHERE  s.class = c.class ) ) );
```

Esercizio 4 – Soluzioni (2/3)

3. Nessun paese può avere sia incrociatori (bc) che corazzate (bb)

```
CREATE ASSERTION no_country_with_battleships_and_battlecruiser CHECK
( NOT EXISTS ( SELECT *
               FROM   classes c1, classes c2
               WHERE  c1.country = c2.country AND c1.class <> c2.class
                      AND c1.type = 'bb' AND c2.type = 'bc' ) );
```

4. Una nave con almeno 9 cannoni non può essere affondata in una battaglia che coinvolge navi con meno di 9 cannoni

```
CREATE ASSERTION cannot_sink_bigger_ship CHECK
( NOT EXISTS ( SELECT *
               FROM   classes c1 JOIN ships s1 ON c1.class = s1.class
                                 JOIN outcomes o1 ON s1.name =o1.ship,
                      classes c2 JOIN ships s2 ON c2.class = s2.class
                                 JOIN outcomes o2 ON s2.name = o2.ship
               WHERE  o1.battle = o2.battle AND s1.name <> s2.name AND
                      c1.num_guns >- 9 AND c2.num_guns < 9 AND
                      o1.result = 'sunk' ) );
```

Esercizio 4 – Soluzioni (3/3)

5. All'interno di una classe, la prima nave varata deve portare il nome della classe

```
CREATE ASSERTION first_ship_has_name_of_the_class CHECK
( NOT EXISTS ( SELECT *
               FROM   ships s1, ships s2
               WHERE  s1.class = s2.class AND
                      s1.name <> s2.name AND s1.name = s1.class AND
                      s1.launched > s2.launched ) );
```

Nota: l'asserzione come formulata richiede che, se la nave con lo stesso nome della classe è nel DB, allora il suo anno di varo deve essere il più vecchio. L'asserzione ammette l'esistenza di classi per cui si conoscono soltanto navi con nome diverso dalla classe.

6. Una battaglia deve coinvolgere almeno due navi di paesi diversi

```
CREATE ASSERTION battles_among_different_countries CHECK
( NOT EXISTS ( SELECT   o.battle
               FROM     outcomes o JOIN ships s ON o.ship = s.name
                                   JOIN classes c ON s.class = c.class
               GROUP BY o.battle
               HAVING   COUNT(DISTINCT c.country) <= 1 ) );
```

Esercizio 5 – Prodotti

Sia dato lo schema:

product (model, maker, type)
pc (model, speed, ram, hd, price)
laptop (model, speed, ram, hd, screen, price)
printer (model, color, type, price)

Si scrivano i comandi SQL per creare le seguenti view (si assumano view non materializzate):

1.color_printer (model, type, price) – contenente le sole stampanti a colori
2.computer (model, maker, type, speed, ram, hd, price) – contenente tutti i pc e laptop nel DB
3.model_price (model, maker, type, price) – contenente i prezzi di tutti i prodotti
4.maker_stats (maker, num_models, min_price, max_price) – con le statistiche per produttore consistenti in numero modelli venduti, prezzo minimo e massimo; per i prezzi appoggiarsi alla vista model_price definita in (3)
5.combination (pc_model, printer_model, price) – contenente tutte le combinazioni PC / printer dello stesso produttore e con prezzo totale – ridotto dello sconto del 5% - inferiore a 1000$ (appoggiarsi a model_price se conveniente)

Esercizio 5 – Soluzioni (1/3)

1. color_printer (model, type, price) – contenente le sole stampanti a colori

```
CREATE VIEW color_printer (model, type, price) AS
SELECT model, type, price
FROM   printer
WHERE  color = TRUE;
```

2. computer (model, maker, type, speed, ram, hd, price) – contenente tutti i pc e laptop nel DB

```
CREATE VIEW computer (model, maker, type, speed, ram, hd, price) AS
( SELECT model, maker, type, speed, ram, hd, price
  FROM   product NATURAL JOIN pc )
UNION
( SELECT model, maker, type, speed, ram, hd, price
  FROM   product NATURAL JOIN laptop );
```

Esercizio 5 – Soluzioni (2/3)

3. model_price (model, maker, type, price) – contenente i prezzi di tutti i prodotti

```
CREATE VIEW model_price (model, maker, type, price) AS
( SELECT model, maker, type, price
  FROM   product NATURAL JOIN pc )
UNION
( SELECT model, maker, type, price
  FROM   product NATURAL JOIN laptop )
UNION
( SELECT product.model, maker, product.type, price
  FROM   product JOIN printer
         ON product.model = printer.model );
```

4. maker_stats (maker, num_models, min_price, max_price) – con le statistiche per produttore consistenti in numero modelli venduti, prezzo minimo e massimo; per i prezzi appoggiarsi alla vista model_price definita in (3)

```
CREATE VIEW maker_stats (maker, num_models, min_price, max_price) AS
SELECT   maker, COUNT(*) AS num_models,
         MIN(price) AS min_price, MAX(price) AS max_price
FROM     model_price
GROUP BY maker
```

Esercizio 5 – Soluzioni (3/3)

5. combination (pc_model, printer_model, price) – contenente tutte le combinazioni PC / printer dello stesso produttore e con prezzo totale – ridotto dello sconto del 5% - inferiore a 1000$ (appoggiarsi a model_price se conveniente)

```
CREATE VIEW combination (maker, pc_model, printer_model, price) AS
SELECT p.maker, p.model, s.model,
       ((p.price + s.price) * 0.95) AS total_price
FROM   model_price p, model_price s
WHERE  p.type = 'pc' AND s.type = 'printer' AND
       p.maker = s.maker AND (p.price + s.price) * 0.95 < 1000
```

Esercizio 6 – Navi da battaglia

Sia dato lo schema:

classes (class, type, country, num_guns, bore, displacement)
ships (name, class, launched)
outcomes (ship, battle, result)
battles (name, date)

Si scrivano i comandi SQL per creare le seguenti view (si assumano view non materializzate):

1. battleship (ship, class, country, num_guns, bore, displacement, launched) – contenente tutte le corazzate (bb) presenti nel database
2. navy (country, num_bb, num_bc, displacement) – con I numeri di battleship (bb) e battlecruiser (bc) e la stazza totale (sommata sulle navi) per nazione
3. survivor (ship, num_battles) – contenente le navi non affondate e il numero di battaglie cui hanno partecipato (possibilmente zero)
4. sinking (ship, date) – contenente le date di affondamento delle navi affondate in battaglia

Esercizio 6 – Soluzioni (1/2)

1. battleship (ship, class, country, num_guns, bore, displacement, launched)

```
CREATE VIEW battleship (ship, class, country, num_guns, bore,
                        displacement, launched) AS
SELECT name, class, country, num_guns, bore, displacement, launched
FROM   classes NATURAL JOIN ships
WHERE  type = 'bb';
```

2. navy (country, num_bb, num_bc, displacement)

```
CREATE VIEW navy (country, num_bb, num_bc, displacement) AS
SELECT DISTINCT country,
       ( SELECT COUNT(*)
         FROM   classes NATURAL JOIN ships
         WHERE  classes.country = c.country AND type = 'bb' ) num_bb,
       ( SELECT COUNT(*)
         FROM   classes NATURAL JOIN ships
         WHERE  classes.country = c.country AND type = 'bc' ) num_bc,
       ( SELECT SUM(displacement)
         FROM   classes NATURAL JOIN ships
         WHERE  classes.country = c.country ) displacement
FROM   classes c;
```

Esercizio 6 – Soluzioni (2/2)

3. survivor (ship, num_battles)

```
CREATE VIEW survivor (ship, num_battles) AS
SELECT name,
       ( SELECT COUNT(*)
         FROM   outcomes o
         WHERE  o.ship = s.name ) AS num_battles
FROM   ships s
WHERE  NOT EXISTS ( SELECT *
                    FROM   outcomes o
                    WHERE  o.ship = s.name AND
                           o.result = 'sunk' )
```

4. sinking (ship, date)

```
CREATE VIEW sinking (ship, date) AS
SELECT o.ship, b.date
FROM   outcomes o JOIN battles b ON o.battle = b.name
WHERE  o.result = 'sunk'
```

Esercizi dal libro

7.2.2 e 7.2.4	coperti parzialmente da esercizio 1
7.2.5 e 7.3.2	coperti parzialmente da esercizio 2
7.4.1	coperto totalmente da esercizio 3
7.4.2	coperto totalmente da esercizio 4
7.2.1, 7.2.3	database Movies, riportati in slide seguenti
7.3.1	database Movies, con quesiti relativi anche a vincoli di chiave primaria ed esterna, riportato in slide seguenti
8.1.1, 8.1.2	database Movies, con quesiti su definizione e interrogazione viste, riportati in slide seguenti

Esercizi dal libro – 7.2.1

Exercise 7.2.1: Write the following constraints for attributes of the relation

`Movies(title, year, length, genre, studioName, producerC#)`

a) The length cannot be less than 30 nor more than 500.

b) The year cannot be before 1909.

c) The genre can only be drama, comedy, sciFi, or teen.

Esercizi dal libro – 7.2.3

Exercise 7.2.3: Write the following constraints as tuple-based `CHECK` constraints on one of the relations of our running movies example:

```
Movies(title, year, length, genre, studioName, producerC#)
StarsIn(movieTitle, movieYear, starName)
MovieStar(name, address, gender, birthdate)
MovieExec(name, address, cert#, netWorth)
Studio(name, address, presC#)
```

If the constraint actually involves two relations, then you should put constraints in both relations so that whichever relation changes, the constraint will be checked on insertions and updates. Assume no deletions; it is not always possible to maintain tuple-based constraints in the face of deletions.

a) A star may not appear in a movie made before they were born.

! b) A studio name that appears in `Studio` must also appear in at least one `Movies` tuple.

! c) No two movie executives may have the same address.

! d) A name that appears in `MovieExec` must not also appear in `MovieStar`.

!! e) If a producer of a movie is also the president of a studio, then they must be the president of the studio that made the movie.

Esercizi dal libro – 7.3.1

Exercise 7.3.1: Show how to alter your relation schemas for the movie example:

```
Movie(title, year, length, genre, studioName, producerC#)
StarsIn(movieTitle, movieYear, starName)
MovieStar(name, address, gender, birthdate)
MovieExec(name, address, cert#, netWorth)
Studio(name, address, presC#)
```

in the following ways.

a) Make `movieTitle`, `movieYear`, and `starName` the key for `StarsIn`.

b) Require the referential integrity constraint that the president of every studio appear in `MovieExec`.

c) Require that no movie length be less than 30 nor greater than 500.

! d) Require that no name appear as both a movie star and movie executive (this constraint need not be maintained in the face of deletions).

! e) Require that no two movie executives have the same address.

Esercizi dal libro – 8.1.1

Exercise 8.1.1: From the following base tables of our running example

```
MovieStar(name, address, gender, birthdate)
MovieExec(name, address, cert#, netWorth)
Studio(name, address, presC#)
```

Construct the following views:

a) A view `StudioPres` giving the name, address, and certificate number of all executives who are studio presidents.

b) A view `ExecutiveStar` giving the name, address, gender, birth date, certificate number, and net worth of all individuals who are both executives and stars.

c) A view `RichExec` giving the name, address, certificate number and net worth of all executives with a net worth of at least $5,000,000.

Esercizi dal libro – 8.1.2

Exercise 8.1.2: Write each of the queries below, using one or more of the views from Exercise 8.1.1 and no base tables.

a) Find the names of those executives who are both studio presidents and worth at least $5,000,000.

b) Find the names of females who are both stars and executives.

! c) Find the names of studio presidents who are also stars and are worth at least $10,000,000.