Ripasso – Manipolazione Dati

- Comandi disponibili
  - **INSERT INTO** tabella [(attributo$_1$, ..., attributo$_N$)]
    VALUES (valore$_{11}$, ..., valore$_{1N}$),
    ...,
    (valore$_{M1}$, ..., valore$_{MN}$);
  - **INSERT INTO** tabella [(attributo$_1$, ..., attributo$_N$)]
    SELECT espressione$_1$, ..., espressione$_N$
    FROM ...
    ... altre clausole ...;
  - **DELETE FROM** tabella [[**AS**] alias]
    WHERE condizione;
  - **UPDATE** tabella [[**AS**] alias]
    SET attributo$_i$ = espressione$_i$, ...
    WHERE condizione;
- La condizione in **DELETE** e **UPDATE** può far uso di query annidate e citare gli attributi della tupla in esame (possibilmente usando alias)
- La keyword **DEFAULT** può essere usata in **INSERT** e **UPDATE** per inserire il valore di default dell'attributo

Ripasso – Creazione Tabelle

- Sintassi

```
CREATE TABLE tabella (
  attributo_i tipo_i [... constraint attributo_i ...], ...
  [... constraint tabella ...]
)
```

- Constraint a livello di attributo:
  - **DEFAULT** value
  - **NOT NULL**
  - [**CONSTRAINT nome] PRIMARY KEY**
  - [**CONSTRAINT** nome] **UNIQUE**
  - [**CONSTRAINT** nome] **REFERENCES** tabella(attr) [**ON DELETE** azione] [**ON UPDATE** azione]
  - [**CONSTRAINT** nome] **CHECK** (condizione)

- Constraint a livello di tabella:
  - [**CONSTRAINT** nome] **PRIMARY KEY** (attributo$_1$, ..., attributo$_N$)
  - [**CONSTRAINT** nome] **UNIQUE** (attributo$_1$, ..., attributo$_N$)
  - [**CONSTRAINT** nome] **FOREIGN KEY** (attributo$_1$, ... attributo$_N$) **REFERENCES** tabella(attr$_1$', ..., attr$_N$') [**ON DELETE** azione] [**ON UPDATE** azione]
  - [**CONSTRAINT** nome] **CHECK** (condizione)

- Azione può essere: **NO ACTION**, **CASCADE**, **SET NULL**, **SET DEFAULT**

Ripasso – Tipi Dato (1)

- Stringhe
  - **`CHAR(n) / CHARACTER(n)`**
    - padding a destra fino ad arrivare ad n
    - padding a destra ignorato nei confronti (non quello a sinistra)
    - n omesso → n = 1
  - **`VARCHAR(n) / CHARACTER VARYING(n)`**
    - lunghezza variabile fino ad n, no padding
  - **`TEXT`** – lunghezza variabile senza limiti
- Numeri
  - **`INT = INTEGER / SHORTINT / BIGINT`** – interi risp. 4, 2, 8 bytes
  - **`FLOAT = REAL / DOUBLE PRECISION`** – virgola mobile risp. 4 e 8 bytes
  - **`NUMERIC(precision) / NUMERIC(precision, scale)`**
    - DECIMAL è alias per NUMERIC
    - precision è num. totale di cifre, scale è num. cifre decimali (0 se omesso)
  - **`SERIAL / BIGSERIAL`** – intero auto-incrementato, risp. 4 o 8 bytes

Ripasso – Tipi Dato (2)

- Booleani
  - **`BOOLEAN`** - assume i valori **`TRUE`**, **`FALSE`** e **`NULL`** = **`UNKNOWN`**
- Tempo
  - **`DATE`** – solo data
  - **`TIME`** [(precision)] [**`WITH TIME ZONE`**]
  - **`TIMESTAMP`** [(precision)] [**`WITH TIME ZONE`**]
    - precision è numero cifre decimali per frazioni di secondo
    - WITH TIME ZONE abilita la memorizzazione della TIME ZONE
- Dati binari
  - **`BIT(n)`** – esattamente n bit, valori codificati come B'0100…'
  - **`BIT VARYING(n)`** – al più n bit
  - **`BYTEA`** – byte array, corrisponde a SQL Blob
- Enumerazioni
  - **`CREATE TYPE`** `enum_type` **`AS ENUM`** `('value1', ..., 'valueN')`
    - una volta definito si può usare come tipi dato regolari
    - specifico di PostgreSQL

Ripasso – Rimozione e Modifica Tabelle e Attributi

- Rimozione tabelle
  - `DROP TABLE tabella [CASCADE]`

- Aggiunta attributo
  - `ALTER TABLE tabella ADD COLUMN attributo tipo [constraint];`
- Rimozione attributo
  - `ALTER TABLE tabella DROP COLUMN attributo [CASCADE]`
- Renaming
  - `ALTER TABLE tabella RENAME TO tabella'`
  - `ALTER TABLE tabella RENAME COLUMN attributo TO attributo'`
- Modifica tipo attributo
  - `ALTER TABLE tabella ALTER COLUMN attributo TYPE tipo`
- Modifica default attributo
  - `ALTER TABLE tabella ALTER COLUMN attributo SET DEFAULT valore`
  - `ALTER TABLE tabella ALTER COLUMN attributo DROP DEFAULT`

Ripasso – Modifica Vincoli

- Aggiunta o rimozione constraint NOT NULL
  - **ALTER TABLE** tabella **ALTER COLUMN** attributo **SET NOT NULL**
  - **ALTER TABLE** tabella **ALTER COLUMN** attributo **DROP NOT NULL**
- Aggiunta altri constraint
  - **ALTER TABLE** tabella **ADD CHECK (...)**
  - **ALTER TABLE** tabella **ADD** [**CONSTRAINT** nome] **CHECK** (...)
  - **ALTER TABLE** tabella **ADD** [**CONSTRAINT** nome] **UNIQUE** (...)
  - **ALTER TABLE** tabella **ADD** [**CONSTRAINT** nome] **FOREIGN KEY** ...
- Rimozione constraint
  - **ALTER TABLE** tabella **DROP CONSTRAINT** nome [**CASCADE**]

Esercizio 1 – Prodotti

rif. esercizio 2.3.1 (quesiti 3 e 4 corrispondenti a (e), (f))

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)

Scrivere i comandi SQL per effettuare le seguenti modifiche:

1. Creare la tabella smartphone(model, screen, bluetooth, wifi, gps, resolution, price), con bluetooth, wifi, gps attributi booleani
2. Ristrutturare il database, aggiungendo price a product e togliendolo dalle tre tabelle pc, laptop e printer
3. Rimuovere l'attributo color di printer
4. Aggiungere l'attributo optical_disk a laptop, valori 'cd', 'dvd', 'none', default 'none ‘
5. Ridenominare la tabella laptop in notebook e il suo attributo screen in monitor
6. Specificare che le stampanti sono di default delle stampanti inkjet
7. Specificare che l'attributo maker di product non può essere null
8. Ristrutturare il database creando una tabella maker(name, address, phone); name è una stringa di max 20 caratteri; popolare la tabella a partire dai maker in product e aggiungere un vincolo di integrità referenziale tra le due tabelle
9. Cancellare la tabella printer e provare a cancellare product - cosa succede?

Esercizio 1 – Soluzioni (1/5)

1. Creare la tabella smartphone(model, screen, bluetooth, wifi, gps, resolution, price), con bluetooth, wifi, gps attributi booleani

```
CREATE TABLE smartphone (
  model      INT PRIMARY KEY
             REFERENCES product(model)
             ON DELETE CASCADE
             ON UPDATE CASCADE,
  screen     DECIMAL(2,1),
  bluetooth  BOOLEAN,
  wifi       BOOLEAN,
  gps        BOOLEAN,
  resolution DECIMAL(3,1),
  price      INTEGER
)
```

Nota: la scelta dei tipi dato per model e price ricalca la scelta effettuata nelle altre tabelle del database. Per screen è usato un DECIMAL(2,1) in quanto si assume che le dimensioni siano inferiori a 10 ” e un decimale è sufficiente. Per resolution – la risoluzione della fotocamera in MP – si usa DECIMAL(3,1) in quanto si ipotizza che risoluzioni superiori a 10MP possano essere usate, mentre un decimale è sufficiente.

Esercizio 1 – Soluzioni (2/5)

2. Ristrutturare il database, aggiungendo price a product e togliendolo dalle tre tabelle pc, laptop e printer

```
ALTER TABLE product ADD COLUMN price INTEGER;

UPDATE product p1
SET price = ( SELECT price
              FROM ( ( SELECT model, price
                       FROM   pc )
                     UNION
                     ( SELECT model, price
                       FROM   laptop )
                     UNION
                     ( SELECT model, price
                       FROM   printer ) ) p
              WHERE p.model = p1.model )

ALTER TABLE pc DROP COLUMN price;
ALTER TABLE laptop DROP COLUMN price;
ALTER TABLE printer DROP COLUMN price;
```

Nota: al posto del comando di UPDATE singolo si possono usare tre comandi separati, uno per ciascuna tabella pc, laptop e printer.

Esercizio 1 – Soluzioni (3/5)

3. Rimuovere l'attributo color di printer

```
ALTER TABLE printer DROP COLUMN color;
```

4. Aggiungere l'attributo optical_disk a laptop, valori ammissibili 'cd', 'dvd', 'none', default ‘none’

```
ALTER TABLE laptop ADD COLUMN optical_disk VARCHAR(10) DEFAULT
'none';
```

oppure con gli enum (soluzione specifica di PostgreSQL):

```
CREATE TYPE optical_disk_type AS ENUM ('cd', 'dvd', 'none');
ALTER TABLE laptop
  ADD COLUMN optical_disk optical_disk_type DEFAULT 'none';
```

5. Ridenominare la tabella laptop in notebook e il suo attributo screen in monitor

```
ALTER TABLE laptop RENAME TO notebook;
ALTER TABLE notebook RENAME COLUMN screen TO monitor;
```

6. Specificare che le stampanti sono di default delle stampanti inkjet

```
ALTER TABLE printer ALTER COLUMN type SET DEFAULT 'inkjet';
```

Esercizio 1 – Soluzioni (4/5)

9. Specificare che l'attributo maker di product non può essere null

```
ALTER TABLE product ALTER COLUMN maker SET NOT NULL;
```

10. Ristrutturare il database creando una tabella maker(name, address, phone); name è una stringa di max 20 caratteri; popolare la tabella a partire dai maker in product e aggiungere un vincolo di integrità referenziale tra le due tabelle

```
ALTER TABLE product ALTER COLUMN maker TYPE VARCHAR(20);

CREATE TABLE maker (
    name VARCHAR(20) PRIMARY KEY,
    address VARCHAR(30),
    phone VARCHAR(20)
)

INSERT INTO maker(name)
SELECT DISTINCT maker
FROM product;

ALTER TABLE product ADD CONSTRAINT product_maker_fkey
      FOREIGN KEY (maker) REFERENCES maker(name)
      ON DELETE CASCADE ON UPDATE CASCADE;
```

Esercizio 1 – Soluzioni (5/5)

11. Cancellare la tabella printer e provare a cancellare product - cosa succede?

```
DROP TABLE printer;

DROP TABLE product CASCADE;
```

Nota: la rimozione di product è rifiutata dal sistema se non si specifica CASCADE, in quanto le tabelle pc, laptop e printer si riferiscono a product tramite i vincoli di chiave esterna sugli attributi model. L'esecuzione del comando con CASCADE causa l'eliminazione della tabella product e dei vincoli di integrità referenziale da pc, laptop e printer verso product. L'eliminazione di product non causa tuttavia l'eliminazione delle tabelle prima dipendenti o la cancellazione delle tuple in esse contenute (cosa che avverrebbe invece se si facesse una 'DELETE FROM product' a causa delle policy ON DELETE CASCADE sui vincoli di integrità referenziale).

Esercizio 2 – Prodotti

rif. esercizio 6.5.1 (quesiti 1-7)

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 assuma che tra gli attributi model di pc, laptop e printer e l'attributo model di product sussista un vincolo di integrità referenziale con clausola ON DELETE CASCADE.

Scrivere i comandi SQL per effettuare le seguenti modifiche:

1. Cancellare tutti i PC con hd più piccolo di 200 GB
2. Inserire nel DB il PC modello 1500, prodotto da A e con velocità 3.1 GHz, RAM 1024 MB, HD 300 GB, prezzo 2499
3. Cancellare tutti I laptop venduti da produttori che non vendono PC
4. Il produttore B acquisisce il produttore C – modificare il DB di conseguenza
5. Per ogni PC, raddoppiare dimensione HD e aggiungere 1024 MB di RAM
6. Aggiungere 1" di schermo e sottrarre 200 al prezzo di ogni laptop prodotto da D
7. Modificare il DB aggiungendo per ogni laptop un PC con stesso maker, modello = modello laptop - 1100, stessa RAM, velocità e HD e prezzo = prezzo laptop – 500
8. Rimuovere i vincoli di integrità referenziale e rispondere di nuovo ai quesiti (1) e (3)
9. Cambiare ON DELETE CASCADE con ON DELETE NO ACTION e rispondere a (1)

Esercizio 2 – Soluzioni (1/5)

1. Cancellare tutti i PC con hd più piccolo di 200 GB

```
DELETE FROM product
WHERE  model IN ( SELECT model
                  FROM   pc
                  WHERE  hd < 200 );
```

Nota: in virtù del vincolo di integrità con policy ON DELETE CASCADE, è sufficiente cancellare la tupla da product affinchè la tupla dipendente in pc sia automaticamente cancellata.

2. Inserire nel DB il PC modello 1500, prodotto da A e con velocità 3.1 GHz, RAM 1024 MB, HD 300 GB, prezzo 2499

```
INSERT INTO product(model, maker, type)
VALUES (1500, 'A', 'pc');

INSERT INTO pc(model, speed, ram, hd, price)
VALUES (1500, 3.1, 1024, 300, 2499);
```

Nota: l'ordine delle due INSERT è condizionato dall'esistenza di un vincolo di integrità referenziale da pc(model) a product(model). Se si inserisse per prima la tupla in pc, il DBMS rifiuterebbe l'operazione poichè non esisterebbe nessuna tupla in product con model = 1500.

Esercizio 2 – Soluzioni (2/5)

3. Cancellare tutti I laptop venduti da produttori che non vendono PC

```
DELETE FROM product
WHERE  type = 'laptop' AND
       maker NOT IN ( SELECT DISTINCT maker
                      FROM   product
                      WHERE  type = 'pc' );
```

Nota: a causa del vincolo di integrità referenziale da laptop(model) a product(model), basta cancellare la tupla in product per cancellare in automatico la tupla dipendente in laptop.

4. Il produttore B acquisisce il produttore C – modificare il DB di conseguenza

```
UPDATE product
SET    maker = 'B'
WHERE  maker = 'C'
```

5. Per ogni PC, raddoppiare dimensione HD e aggiungere 1024 MB di RAM

```
UPDATE pc
SET    hd = hd * 2,
       ram = ram + 1024;
```

Esercizio 2 – Soluzioni (3/5)

6. Aggiungere 1” di schermo e sottrarre 200 al prezzo di ogni laptop prodotto da D

```
UPDATE laptop
SET    screen = screen + 1,
       price = price - 200
WHERE  model IN ( SELECT model
                  FROM   product
                  WHERE  maker = 'A' );
```

7. Modificare il DB aggiungendo per ogni laptop un PC con stesso maker, modello = modello laptop - 1100, stessa RAM, velocità e HD e prezzo = prezzo laptop – 500

```
INSERT INTO product (model, maker, type)
SELECT (model - 1100) AS model, maker, 'pc' AS type
FROM   product
WHERE  type = 'laptop';

INSERT INTO pc (model, speed, ram, hd, price)
SELECT (model - 1100) AS model, speed, ram, hd, (price - 500) AS
price
FROM   laptop;
```

Esercizio 2 – Soluzioni (4/5)

8. Rimuovere i vincoli di integrità referenziale e rispondere ai quesiti (1) e (3)

```
(1) DELETE FROM product
    WHERE  model IN ( SELECT model
                      FROM   pc
                      WHERE  hd < 200 );
    DELETE FROM pc
    WHERE  hd < 200;

(3) DELETE FROM laptop
    WHERE  model IN ( SELECT model
                      FROM   product
                      WHERE  type = 'laptop' AND
                             maker NOT IN ( SELECT DISTINCT maker
                                            FROM   product
                                            WHERE  type = 'pc' ) );
    DELETE FROM product
    WHERE  type = 'laptop' AND
           maker NOT IN ( SELECT DISTINCT maker
                          FROM   product
                          WHERE  type = 'pc' );
```

Esercizio 2 – Soluzioni (5/5)

9. Cambiare ON DELETE CASCADE con ON DELETE NO ACTION e rispondere a (1)

Nella definizione dello schema, nell'istruzione CREATE TABLE per la tabella pc:

```
CREATE TABLE pc (
  model INT PRIMARY KEY
            REFERENCES product(model)
            ON DELETE NO ACTION
            ON UPDATE NO ACTION
            DEFERRABLE INITIALLY DEFERRED,
  ...
```

Comandi SQL:

```
BEGIN TRANSACTION;

DELETE FROM product
WHERE  model IN ( SELECT model
                  FROM   pc
                  WHERE  hd < 200 );

DELETE FROM pc
WHERE  hd < 200;

COMMIT;
```

Esercizio 3 – Navi da battaglia

rif. esercizio 2.3.2 (quesiti 3, 4 corrispondenti a (e), (f))

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 assuma l'assenza di qualunque vincolo di integrità referenziale.

Scrivere i comandi SQL per effettuare le seguenti modifiche:

1. Si vuole tenere traccia dell'equipaggio di ciascuna nave:
   - aggiungere la tabella sailor (code, name, surname, rank) con name e surname obbligatori e opportuna scelta dei tipi dato;
   - aggiungere la tabella 'ponte' crew (ship_name, sailor_code, from_date, to_date) con le date opzionali e opportuna scelta dei tipi dato;
   - introdurre vincoli di integrità referenziale ove possibile
2. Ridenominare l'attributo name di ships in ship
3. Cancellare l'attributo bore da classes
4. Aggiungere a ships l'attributo yard per il cantiere di costruzione della nave
5. Si modifichi il DB, creando e popolando nuove tabelle se necessario, affinché esso contenga l'informazione necessaria per associare una nave – direttamente o indirettamente – all'alleanza tra stati (e.g. Asse, Alleati) cui appartiene.

Esercizio 3 – Soluzioni (1/3)

1. Si vuole tenere traccia dell'equipaggio di ciascuna nave:

   – aggiungere la tabella sailor (code, name, surname, rank) …

```
CREATE TABLE sailor (
  code    INTEGER PRIMARY KEY,
  name    VARCHAR(30) NOT NULL,
  surname VARCHAR(30) NOT NULL,
  rank    VARCHAR(20)
);
```

   – aggiungere la tabella crew (ship_name, sailor_code, from_date, to_date) …

```
CREATE TABLE crew (
  ship_name   VARCHAR(30) REFERENCES ships(name)
                          ON DELETE CASCADE ON UPDATE CASCADE,
  sailor_code INTEGER REFERENCES sailor(code)
                      ON DELETE CASCADE ON UPDATE CASCADE,
  from_date   DATE,
  to_date     DATE,

  PRIMARY KEY (ship_name, sailor_code)
);
```

Esercizio 3 – Soluzioni (2/3)

2. Ridenominare l'attributo name di ships in ship

```
ALTER TABLE ships RENAME COLUMN name TO ship;
```

3. Cancellare l'attributo bore da classes

```
ALTER TABLE classes DROP COLUMN bore;
```

4. Aggiungere a ships l'attributo yard per il cantiere di costruzione della nave

```
ALTER TABLE ships ADD COLUMN yard VARCHAR(30);
```

Esercizio 3 – Soluzioni (3/3)

3. Si modifichi il DB, creando e popolando nuove tabelle se necessario, affinché esso contenga l'informazione necessaria per associare una nave – direttamente o indirettamente – all'alleanza tra stati (e.g. Asse, Alleati) cui appartiene.

```
CREATE TABLE country (
  name     VARCHAR(30) PRIMARY KEY,
  alliance VARCHAR(30)
)

INSERT INTO country (name)
SELECT DISTINCT country
FROM   classes;

ALTER TABLE classes ADD FOREIGN KEY (country) REFERENCES country(name)
                        ON DELETE SET NULL ON UPDATE CASCADE;
```

Nota: l'associazione è indiretta, in quanto è il paese ad appartenere ad una alleanza e non la nave direttamente. Nella soluzione, si 'espande' l'attributo country di classes in modo da memorizzare anche l'informazione sull'alleanza cui il paese fa parte. L'attributo country in classes è usato per popolare i nomi nella nuova tabella e diventa quindi chiave esterna. Non si crea una tabella per l'alleanza perchè si suppone basti memorizzarne il nome.

Esercizio 4 – Navi da battaglia

rif. esercizio 6.5.2

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 assuma l'assenza di qualunque vincolo di integrità referenziale.

Scrivere i comandi SQL per effettuare le seguenti modifiche:

1. Inserire i due incrociatori (bc) britannici ("Gt. Britain") Nelson e Rodney varati nel 1927 ed appartenenti alla classe Nelson con 9 cannoni da 16" e stazza 34000 t
2. Inserire le corazzate italiane (bc) Vittorio Veneto e Italia, varate nel 1940, e Roma, varata nel 1942, tutte e tre appartenenti alla classe Vittorio Veneto con 9 cannoni da 15" e stazza 41000 t
3. Cancellare dal DB tutte le classi con meno di tre navi
4. Cancellare dal DB tutte le navi affondate in battaglia
5. Convertire il calibro dei cannoni da pollici a centimetri (1" = 2.54 cm) e la stazza in tonnellate metriche (moltiplicare per 1.1)

Esercizio 4 – Soluzioni (1/3)

1. Inserire i due incrociatori (bc) britannici (“Gt. Britain”) Nelson e Rodney varati nel 1927 ed appartenenti alla classe Nelson con 9 cannoni da 16” e stazza 34000 t

```
INSERT INTO classes(class, type, country, num_guns, bore,
displacement)
VALUES ('Nelson', 'bb', 'Gt. Britain', 9, 16, 34000);

INSERT INTO ships (name, class, launched)
VALUES ('Nelson', 'Nelson', 1927),
       ('Rodney', 'Nelson', 1927);
```

2. Inserire le corazzate italiane (bc) Vittorio Veneto e Italia, varate nel 1940, e Roma, varata nel 1942, tutte e tre appartenenti alla classe Vittorio Veneto con 9 cannoni da 15” e stazza 41000 t

```
INSERT INTO classes(class, type, country, num_guns, bore,
displacement)
VALUES ('Vittorio Veneto', 'bb', 'Italy', 9, 15, 41000);

INSERT INTO ships (name, class, launched)
VALUES ('Vittorio Veneto', 'Vittorio Veneto', 1940),
       ('Italia', 'Vittorio Veneto', 1940),
       ('Roma', 'Vittorio Veneto', 1942);
```

Esercizio 4 – Soluzioni (2/3)

3. Cancellare dal DB tutte le classi con meno di tre navi

```
DELETE FROM classes
WHERE  class IN ( SELECT   class
                  FROM     ships
                  GROUP BY class
                  HAVING   COUNT(*) < 3 )

DELETE FROM outcomes
WHERE  ship IN ( SELECT name
                 FROM   ships
                 WHERE  class IN ( SELECT   class
                                   FROM     ships
                                   GROUP BY class
                                   HAVING   COUNT(*) < 3 ) )
DELETE FROM ships
WHERE  class IN ( SELECT   class
                  FROM     ships
                  GROUP BY class
                  HAVING   COUNT(*) < 3 )
```

Esercizio 4 – Soluzioni (3/3)

4. Cancellare dal DB tutte le navi affondate in battaglia

```
DELETE FROM ships
WHERE  name IN ( SELECT ship
                 FROM   outcomes
                 WHERE  result = 'sunk' );

DELETE FROM outcomes
WHERE  result = 'sunk';
```

5. Convertire il calibro dei cannoni da pollici a centimetri (1” = 2.54 cm) e la stazza in tonnellate metriche (moltiplicare per 1.1)

```
UPDATE classes
SET    bore = bore * 2.54,
       displacement = 1.1 * displacement;
```

Esercizi dal libro

2.3.1, 2.3.2	coperti parzialmente da esercizi 1 e 3, fare riferimento a script creazione database prod e ship per codice SQL definizione tabelle
6.5.1 e 6.5.2	coperti totalmente da esercizi 2 e 4
7.1.1, 7.1.4	database Movies, riportati in slide seguenti
7.1.2	fare riferimento allo script SQL usato a laboratorio per soluzione
7.1.3, 7.1.5	database Ships, riportati in slide seguenti (da fare su carta, i dati del DB non rispettano vincoli integrità)

Esercizi dal libro – 7.1.3, 7.1.5

Exercise 7.1.3: Suggest suitable keys for the relations of the battleships database

```
Classes(class, type, country, numGuns, bore, displacement)
Ships(name, class, launched)
Battles(name, date)
Outcomes(ship, battle, result)
```

of Exercise 2.4.3. Modify your SQL schema from Exercise 2.3.2 to include declarations of these keys.

Exercise 7.1.5: Write the following referential integrity constraints for the battleships database as in Exercise 7.1.3. Use your assumptions about keys from that exercise, and handle all violations by setting the referencing attribute value to `NULL`.

a) Every battle mentioned in `Outcomes` must be mentioned in `Battles`.

b) Every ship mentioned in `Outcomes` must be mentioned in `Ships`.

c) Every class mentioned in `Ships` must be mentioned in `Classes`.

Esercizi dal libro – 7.1.1

Exercise 7.1.1: Our running example movie database of Section 2.2.8 has keys defined for all its relations.

```
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#)
```

Declare the following referential integrity constraints for the movie database as in Exercise 7.1.1.

a) The producer of a movie must be someone mentioned in `MovieExec`. Modifications to `MovieExec` that violate this constraint are rejected.

b) Repeat (a), but violations result in the deletion or update of the

c) Repeat (a), but violations result in the `producerC#` in `Movie` being set to `NULL`. offending `Movie` tuple.

d) A star appearing in `StarsIn` must also appear in `MovieStar`. Handle violations by deleting violating tuples.

e) A movie that appears in `StarsIn` must also appear in `Movie`. Handle violations by rejecting the modification.

Esercizi dal libro – 7.1.4

! **Exercise 7.1.4:** We would like to declare the constraint that every movie in the relation `Movie` must appear with at least one star in `StarsIn`. Can we do so with a foreign-key constraint? Why or why not?