20. Usare MYSQL Vers. 5.5.31 dalla console di comando in modalità interattiva

DATABASE fornituranew
relativo all'associazione "Fornisce" tra le entità "Fornitore" ed "Articolo" di molteplicità N:N

1) loggarsi al dbms come "root" (dba di default)

```
mysql -u root -p
```

dopo di che verrà richiesta con "Enter password" di digitare la password.
Scrivere "**root**" (verranno visualizzati **4 asterischi**)

2) creiamo un nuovo utente

```
CREATE USER pippo@localhost IDENTIFIED BY "pippo";
N.B. per cancellarlo il dba userà il comando
DROP USER pippo@localhost;
```

3) guardiamo i db creati fino a questo momento

```
SHOW DATABASES;
```

4) creiamo come DBA il database "fornituranew" sul quale opereremo

```
CREATE DATABASE fornituranew;
```

5) selezioniamo il database come DBA

```
USE fornituranew;
```

6) assegniamo tutti i diritti all'utente "pippo" sul db in oggetto (su tutte le sue tabelle)

```
GRANT ALL                       GRANT SELECT, INSERT
ON fornituranew.*               ON fornituranew.fornitore
TO pippo@localhost              TO pippo@localhost
[IDENTIFIED BY "pippo"];        [IDENTIFIED BY "pippo"]
[WITH GRANT OPTION];            [WITH GRANT OPTION];
N.B. per i privilegi normali    N.B. per il privilegio GRANT
REVOKE ALL                      REVOKE GRANT OPTION
ON fornituranew.*               ON fornituranew.*
FROM pippo@localhost;           FROM pippo@localhost;
```

N.B. Per controllare i privilegi assegnati ad un determinato utente possiamo utilizzare

```
SHOW GRANTS FOR 'pippo'@'localhost';
```

oppure, nel caso sia quello con il quale siamo loggati,

```
SHOW GRANTS FOR CURRENT_USER;
```

7) usciamo dalla console come DBA

```
quit;
oppure
exit;
```

8) eseguiamo il login al db come utente "pippo"

```
mysql -u pippo -p
```

dopo di che verrà richiesta con "Enter password" di digitare la password.
Scrivere "**pippo**" (verranno visualizzati **5 asterischi**)

9) selezioniamo il database come utente "pippo"

```
USE fornituranew;
```

N.B. Ora l'utente "pippo" può creare le tabelle appartenenti al db "fornituranew"

10) creiamo la tabella "Fornitore"

```
CREATE TABLE Fornitore
  (
  CodF        VARCHAR(10)      NOT NULL,
  Cognome     VARCHAR(30)      NOT NULL,
  Nome        VARCHAR(30)      NOT NULL,
  DataN       DATE             NOT NULL,
  PRIMARY KEY (CodF),
  CHECK (DataN > "1960-01-01")
  ) Engine InnoDB DEFAULT CHARSET=utf8;
(N.B. Il default MYISAM non supporta l'integrità referenziale e le foreign key
(N.B. Il formato data è AAAA-MM-GG o in alternativa AAAA/MM/GG
(N.B. MYSQL non supporta le clausole CHECK…semplicemente le ignora)
```

11) creiamo la tabella "Articolo"

```
CREATE TABLE Articolo
  (
  CodA           VARCHAR(10)     NOT NULL,
  Descrizione    VARCHAR(20)     NOT NULL,
  Prezzo         DECIMAL (4,2)   NOT NULL,
  PRIMARY KEY (CodA),
  CHECK (Prezzo BETWEEN 1.00 AND 10.00)
  ) Engine InnoDB DEFAULT CHARSET=utf8;
```

N.B. DECIMAL (4,2) significa 4 cifre in totale di cui 2 dopo il punto decimale (analogo a NUMERIC(4,2))

12) creiamo la tabella "Fornisce"

```
(CREATE TABLE Fornisce
  (
  IdF        INT(11)      NOT NULL AUTO_INCREMENT,
  CodF1      VARCHAR(10)  NOT NULL,
  CodA1      VARCHAR(10)  NOT NULL,
  DataF      DATE         NOT NULL,
  Qta        INT (1)      NOT NULL,   // DECIMAL (1,0) oppure NUMERIC (1,0)
  PRIMARY KEY (IdF),
```

```
  FOREIGN KEY (CodF1) REFERENCES Fornitore (CodF)
     ON UPDATE CASCADE
     ON DELETE CASCADE,
  FOREIGN KEY (CodA1)  REFERENCES Articolo (CodA)
     ON UPDATE CASCADE
     ON DELETE CASCADE,
  CHECK (Qta IN (1,2,3))
  ) Engine InnoDB DEFAULT CHARSET=utf8;
```

13) Diamo un'occhiata alle tabelle create finora nel db "forniturane w"

```
SHOW TABLES;
```

N.B. Ora l'utente "pippo" può provvedere a svolgere tutte le query che riterrà opportune

Proviamo a cambiare la struttura della tabella "Fornitore"

14) Aggiungiamo l'attributo non obbligatorio "Email" dichiarato VARCHAR(30)

```
ALTER TABLE Fornitore
  ADD COLUMN Email VARCHAR(30);
```

e guardiamo le modifiche introdotte utilizzando il comando

```
DESCRIBE Fornitore;
```

15) Modifichiamo il tipo dell'attributo "Email" in obbligatorio decimale con 3 cifre per la parte intera e due per la parte decimale

```
ALTER TABLE Fornitore
  MODIFY COLUMN Email DECIMAL (5,2) NOT NULL;
```

e guardiamo le modifiche introdotte utilizzando il comando

```
DESCRIBE Fornitore;
```

16) Oppure modifichiamo il tipo dell'attributo "Email" in opzionale intero di 4 cifre

```
ALTER TABLE Fornitore
  MODIFY COLUMN Email INTEGER (4);
```

e guardiamo le modifiche introdotte utilizzando il comando

```
DESCRIBE Fornitore;
```

N.B Il tipo dati INTEGER (4) equivale a DECIMAL (4,0) o NUMERIC (4,0)

17) Eliminiamo ora l'attributo "Email"

```
ALTER TABLE Fornitore
  DROP COLUMN Email;
```

e guardiamo le modifiche introdotte utilizzando il comando

```
DESCRIBE Fornitore;
```

18) inseriamo nella tabella "Fornitore" le seguenti istanze (n-ple)

- "F_01" **(n.b. formato data AAAA/MM/GG o in alternativa AAAA-MM-GG)**

```
INSERT INTO Fornitore (CodF, Cognome, Nome, DataN)
       VALUES ('F_01', 'ROSSI', 'MARIO', '1965-01-01');
```

- "F_02" (non è possibile inserire solo qualche valore di attributo da valorizzare essendo tutti obbligatori)

```
INSERT INTO Fornitore
       VALUES ('F_02', 'GIALLI', 'ANDREA', '1960-08-08');
```

- "F_03" (eventualmente è possibile alterare l'ordine delle colonne dando opportunamente i valori)

```
INSERT INTO Fornitore (Cognome, CodF, DataN, Nome,)
       VALUES ('ROSSI', 'F_03', '1963-07-03', 'ELIA');
```

- "F_04"

```
INSERT INSERT INTO Fornitore (CodF, Cognome, Nome, DataN)
       VALUES ('F_04', 'BIANCHI', 'VINCENZO', '1968-11-24');
```

19) inseriamo nella tabella "Articolo" le seguenti istanze (n-ple)

- "A_01"

```
INSERT INTO Articolo (CodA, Descrizione, Prezzo)
       VALUES ('A_01', 'PLUMCAKE', 2.75);
```

- "A_02" (non è possibile inserire solo qualche valore di attributo da valorizzare essendo tutti obbligatori)

```
INSERT INTO Articolo
       VALUES ('A_02', 'CORNFLAKES', 3.50);
```

- "A_03" (eventualmente è possibile alterare l'ordine delle colonne dando opportunamente i valori)

```
INSERT INTO Articolo (Descrizione, CodA, Prezzo)
       VALUES ('ARANCIATA', 'A_03', 1.75);
```

- "A_04"

```
INSERT INTO Articolo
       VALUES ('A_04', 'CARTA IGIENICA', 2.25);
```

- "A_05"

```
INSERT INTO Articolo
VALUES ('A_05', 'SPAZZOLINO', 1.90);
```

20) inseriamo nella tabella “Fornisce” le seguenti istanze (n-ple)

```
INSERT INTO Fornisce (IdF, CodF1, CodA1, DataF, Qta)
       VALUES (NULL, 'F_01', 'A_01', '2009-10-11', 1);
```

```
INSERT INTO Fornisce
       VALUES (NULL, 'F_01', 'A_02', '2009-11-12', 3);
```

```
INSERT INTO Fornisce
       VALUES (NULL, 'F_02', 'A_01', '2009-12-12', 2);
```

```
INSERT INTO Fornisce
       VALUES (NULL, 'F_02', 'A_03', '2009-12-13', 3);
```

```
INSERT INTO Fornisce
       VALUES (NULL, 'F_03', 'A_04', '2009-07-15', 2);
```

```
INSERT INTO Fornisce
       VALUES (NULL, 'F_03', 'A_03', '2009-07-15', 1);
```

```
INSERT INTO Fornisce
       VALUES (NULL, 'F_03', 'A_02', '2009-08-11', 3);
```

21) aggiorniamo nella tabella “Fornitore” ponendo tutti i cognomi delle istanze ad “ESPOSITO”

```
UPDATE Fornitore
SET Cognome = “ESPOSITO”;
```

N.B. Tutti i record esistenti nella tabella “Fornitore” avranno ora lo stesso cognome (aggiornamento massivo). Solo se la tabella è vuota non ci saranno istanze modificate!

22) aggiorniamo nella tabella “Fornitore” il nome di “ESPOSITO” “ELIA” in “ALEX”

```
UPDATE Fornitore
SET Nome = “ALEX”
WHERE CodF = “F_03”;
```

N.B. Verrà aggiornato un solo record nella tabella “Fornitore” (o nessuno se il codice impostato è inesistente)

23) cancelliamo dalla tabella “Fornitore” l’istanza ora aggiornata

```
DELETE FROM Fornitore
WHERE CodF = “F_03”;
```

24) cancelliamo dalla tabella "Fornitore" tutte le istanze presenti

```
DELETE FROM Fornitore;
```

N.B. Tutti i record esistenti nella tabella "Fornitore" verranno cancellati (cancellazione massiva). Solo se la tabella è vuota non verrà cancellata alcuna istanza!

25) creiamo la vista (tabella logica non memorizzata fisicamente nel db) che contiene il codice fornitore assieme alla descrizione degli articoli forniti

```
CREATE VIEW Vista1
AS SELECT CodF, Cognome
   FROM Fornitore;
```

26) aggiorniamo attraverso la vista il cognome del fornitore da "F_01" (tale modifica sarà riportata in tutte le tabelle fisiche memorizzate nel db

```
UPDATE Vista1
SET Cognome = "GENOVESE"
WHERE CodF = "F_01";
oppure
UPDATE Vista1
SET CodF = "X_01"
WHERE CodF = "F_01";
```

27) eliminiamo la vista (tabella logica non memorizzata fisicamente nel db) che contiene il codice

```
DROP VIEW Vista1;
```

28) creiamo la vista (tabella logica non memorizzata fisicamente nel db) che contiene il codice fornitore assieme alla descrizione degli articoli forniti

```
CREATE VIEW Vista2
AS SELECT CodF, Descrizione
   FROM Fornitore, Fornisce, Articolo
   WHERE (CodF = CodF1) AND (CodA1 = CodA);
```

29) aggiorniamo attraverso la vista la descrizione di tutti gli articoli forniti da "F_01" (tale modifica sarà riportata in tutte le tabelle fisiche memorizzate nel db

```
UPDATE Vista2
SET Descrizione = "XXXXXXX"
WHERE CodF = "F_01";
```

30) eliminiamo la vista (tabella logica non memorizzata fisicamente nel db) che contiene il codice

```
DROP VIEW Vista2;
```

31) creiamo un trigger sulla tabella Articolo che controlli la clausola

CHECK (Prezzo BETWEEN 1.00 AND 10.00)

impostata in precedenza nella CREATE TABLE di Articolo, che MySQL non supporta, impostando per esempio a 0 il prezzo di un articolo inserito se esso è inferiore a 1.00 ed a 10.00 se esso è superiore a tale cifra

```
DELIMITER //
CREATE TRIGGER trg1
BEFORE INSERT ON fornituranew.Articolo
FOR EACH ROW
BEGIN
IF NEW.Prezzo < 10.00
   THEN
     SET NEW.Prezzo = 0;
END IF;
IF NEW.Prezzo > 1000.00
   THEN
     SET NEW.Prezzo = 1000;
END IF;
END//
DELIMITER ;
```

N.B. I delimitatori differenti dallo standard ; vengono utilizzati nell'uso di funzioni, stored procedures, e triggers nelle quali occorre definire istruzioni multiple. E' possibile definire un proprio delimitatore alternativo (nel nostro caso //) per segnalare la fine dell'intera procedura, permettendo nel contempo a ciascuno statement di essere correttamente terminato con il ;.

Quindi, quando il codice viene eseguito da un client mysql (come la console), il client è in grado di comprendere quando è giunto alla fine della sua definizione ed è in grado di eseguire l'intero set di istruzioni.

Si osservi che il comando DELIMITER è un commando di un client `mysql` (e di altri tipi di client dbms) e non è uno standard del linguaggio SQL.

N.B. Il qualificatore NEW indica proprio che il nome di colonna che stiamo utilizzando si riferisce al nuovo valore della riga che sta per essere aggiornata. NEW si può utilizzare in caso di INSERT e UPDATE. Analogamente è disponibile il qualificatore OLD che fa riferimento ai valori precedenti la modifica, e si può utilizzare in caso di UPDATE e DELETE. La modifica attraverso l'istruzione SET è possibile solo per i valori NEW e solo nei trigger di tipo BEFORE.

N.B. Attualmente MySQL supporta un solo trigger attivo per ciascuna tabella

32) eliminiamo il trigger precedentemente creato

```
DROP TRIGGER trg1;
```

33) creiamo un trigger sempre sulla tabella Articolo che controlli in modo differente da trg1 la clausola

CHECK (Prezzo BETWEEN 1.00 AND 10.00)

impostata in precedenza nella CREATE TABLE di Articolo, che MySQL non supporta, bloccando l'operazione di inserimento segnalando un opportuno messaggio di testo

```
DELIMITER //
CREATE TRIGGER trg2
BEFORE INSERT ON fornituranew.Articolo
FOR EACH ROW
BEGIN
DECLARE msg VARCHAR(255);
IF NEW.Prezzo < 10.00
   THEN
     SET msg = concat("Prezzo non puo' essere inferiore a 10.00");
     SIGNAL sqlstate '45000' SET message_text = msg;
   ELSE
      IF NEW.Prezzo > 1000.00
         THEN
        SET msg = concat("Prezzo non puo' essere superiore a 1000.00");
        SIGNAL sqlstate '45000' SET message_text = msg;
      END IF;
END IF;
END//
DELIMITER ;
```

34) eliminiamo il trigger precedentemente creato

```
DROP TRIGGER trg2;
```

35) creiamo un trigger sulla tabella Fornitore che controlli la clausola

CHECK (DataN > "1960-01-01")

impostata in precedenza nella CREATE TABLE di Fornitore, che MySQL non supporta, bloccando l'operazione di inserimento segnalando un opportuno messaggio di testo

```
DELIMITER //
CREATE TRIGGER trg3
BEFORE INSERT ON Esempio.Fornitore
FOR EACH ROW
BEGIN
DECLARE msg VARCHAR(255);
IF NEW.DataN <= "1960-01-01"
   THEN
     SET msg = concat("DataN non puo' essere precedente al 1 gennaio 1960");
     SIGNAL sqlstate '45000' SET message_text = msg;
END IF;
END//
DELIMITER ;
```

36) eliminiamo il trigger precedentemente creato

```
DROP TRIGGER trg3;
```

37) creiamo un altro trigger sulla tabella Fornisce che controlli la clausola

CHECK (Qta IN (1,2,3))

impostata in precedenza nella CREATE TABLE di Fornisce, che MySQL non supporta, bloccando l'operazione di inserimento segnalando un opportuno messaggio di testo

```
DELIMITER //
CREATE TRIGGER trg4
BEFORE INSERT ON fornituranew.Fornisce
FOR EACH ROW
BEGIN
DECLARE msg VARCHAR(255);
IF NEW.Qta < 1
   THEN
     SET msg = concat("Quantita' non puo' essere inferiore a 1");
     SIGNAL sqlstate '45000' SET message_text = msg;
   ELSE
      IF NEW.Qta > 3
         THEN
           SET msg = concat("Quantita' non puo' essere superiore a 3");
           SIGNAL sqlstate '45000' SET message_text = msg;
      END IF;
END IF;
END//
DELIMITER ;
```

38) eliminiamo il trigger precedentemente creato

```
DROP TRIGGER trg4;
```

OPERATORI E FUNZIONI SQL

1) Creazione di una tabella con una chiave numerica ad autoincremento

```
CREATE TABLE tabella
(
id INT NOT NULL AUTO_INCREMENT ,
nome VARCHAR( 30 ) NOT NULL ,
prezzo DECIMAL( 6, 2 ) NOT NULL ,
note VARCHAR (100),
PRIMARY KEY ( id )
) ENGINE = InnoDB;
```

2) Inserimento di una istanza in una tabella con chiave numerica ad autoincremento

```
INSERT INTO tabella (id ,nome ,prezzo, note)
VALUES (NULL , "pippo", 44.25, NULL);
```

oppure nei seguenti modi alternativi

```
INSERT INTO tabella (id ,nome ,prezzo)
VALUES (NULL , "pippo", 44.25);
```

```
INSERT INTO tabella (nome ,prezzo)
VALUES ("pippo", 44.25);
```

3) Inserire le seguenti altre due istanze nella tabella

```
INSERT INTO tabella (id ,nome ,prezzo, note)
VALUES (NULL , "pluto", 144.75, "praticamente un pazzo");
```

```
INSERT INTO tabella (id ,nome ,prezzo, note)
VALUES (NULL , "paperino", 200.00, NULL);
```

4) Uso degli operatori **IS NULL** ed **IS NOT NULL**

```
SELECT *
FROM tabella
WHERE note IS NULL;
```

Risultato:

```
+----+----------+--------+------+
| id | nome     | prezzo | note |
+----+----------+--------+------+
|  1 | pippo    |  44.25 | NULL |
|  3 | paperino | 200.00 | NULL |
+----+----------+--------+------+
```

N.B. se la query viene terminata con **\G** invece che con il punto e virgola **;** i record saranno mostrati verticalmente invece che orizzontalmente

```
SELECT *
FROM tabella
WHERE note IS NOT NULL;
```

Risultato:

```
+----+-------+--------+----------------------+
| id | nome  | prezzo | note                 |
+----+-------+--------+----------------------+
|  2 | pluto | 144.75 | praticamente un pazzo |
+----+-------+--------+----------------------+
```

5) Uso degli operatori **BETWEEN** ed **NOT BETWEEN**

```
SELECT *
FROM tabella
WHERE prezzo BETWEEN 50.00 AND 200.00;
```

Risultato:

```
+----+---------+--------+----------------------+
| id | nome    | prezzo | note                 |
+----+---------+--------+----------------------+
|  2 | pluto   | 144.75 | praticamente un pazzo |
|  3 | paperino | 200.00 | NULL                 |
+----+---------+--------+----------------------+
```

```
SELECT *
FROM tabella
WHERE prezzo NOT BETWEEN 50.00 AND 200.00;
```

Risultato:

```
+----+-------+--------+------+
| id | nome  | prezzo | note |
+----+-------+--------+------+
|  1 | pippo |  44.25 | NULL |
+----+-------+--------+------+
```

6) Uso degli operatori **IN** ed **NOT IN**

```
SELECT nome
FROM tabella
WHERE prezzo IN (44.25, 200,00);
```

Risultato:

```
+----------+
| nome     |
+----------+
| pippo    |
| paperino |
+----------+
```

```
SELECT nome
FROM tabella
WHERE prezzo NOT IN (44.25, 200,00);
```

Risultato:

```
+-------+
| nome  |
+-------+
| pluto |
+-------+
```

7) Uso degli operatori **LIKE** ed **NOT LIKE**
(L'operatore **LIKE** si usa per trovare corrispondenze parziali sulle stringhe attraverso l'impiego dei caratteri jolly "**%**" che rappresenta "qualsiasi numero di caratteri o nessun carattere", e "_" che invece corrisponde esattamente ad un carattere)

```
SELECT *
FROM tabella
WHERE nome LIKE "pa%";
```

Risultato:

```
+----+----------+--------+------+
| id | nome     | prezzo | note |
+----+----------+--------+------+
|  3 | paperino | 200.00 | NULL |
+----+----------+--------+------+
```

```
SELECT *
FROM tabella
WHERE nome NOT LIKE "pa%";
```

Risultato:

```
+----+-------+--------+-----------------------+
| id | nome  | prezzo | note                  |
+----+-------+--------+-----------------------+
|  1 | pippo |  44.25 | NULL                  |
|  2 | pluto | 144.75 | praticamente un pazzo |
+----+-------+--------+-----------------------+
```

```
SELECT *
FROM tabella
WHERE nome LIKE "p%pp%";
```

Risultato:

```
+----+-------+--------+------+
| id | nome  | prezzo | note |
+----+-------+--------+------+
|  1 | pippo |  44.25 | NULL |
+----+-------+--------+------+
```

```
SELECT *
FROM tabella
WHERE nome LIKE "p__pp%";
```

Risultato:

```
Empty set (0.00 sec)
```

```
SELECT *
FROM tabella
WHERE nome LIKE "p%pp_";
```

Risultato:

```
+----+-------+--------+------+
| id | nome  | prezzo | note |
+----+-------+--------+------+
|  1 | pippo |  44.25 | NULL |
+----+-------+--------+------+
```

```
SELECT *
FROM tabella
WHERE nome LIKE "_____";
```

Risultato:

```
+----+-------+--------+-----------------------+
| id | nome  | prezzo | note                  |
+----+-------+--------+-----------------------+
|  1 | pippo |  44.25 | NULL                  |
|  2 | pluto | 144.75 | praticamente un pazzo |
+----+-------+--------+-----------------------+
```

8) Funzioni sulle stringhe **CONCAT** e **CONCAT_WS**
(si utilizzano per concatenare due o più stringhe, nel secondo caso aggiungendo un separatore)

```
SELECT CONCAT_WS ('+',id, nome, note) AS Concatenazione
FROM tabella;
```

Risultato(N.B. la funzione CONCAT_WS concatena anche le n-ple che hanno valore NULL) :

```
+------------------------------+
| Concatenazione               |
+------------------------------+
| 1+pippo                      |
| 2+pluto+praticamente un pazzo |
| 3+paperino                   |
+------------------------------+
```

```
SELECT CONCAT (id,'+', nome,'+', note) AS Concatenazione
FROM tabella;
```

Risultato (N.B. la funzione CONCAT da NULL se uno dei campi considerati è NULL)

```
+------------------------------+
| Concatenazione               |
+------------------------------+
| NULL                         |
| 2+pluto+praticamente un pazzo |
| NULL                         |
+------------------------------+
```

9) Funzioni sulle stringhe **LOWER** ed **UPPER**
(consentono di trasformare una stringa, rispettivamente, in tutta minuscola o tutta maiuscola)

```
SELECT LOWER (note) AS Tutto_Minuscolo
FROM tabella;
```

Risultato:

```
+----------------------+
| Tutto_Minuscolo      |
+----------------------+
| NULL                 |
| praticamente un pazzo |
| NULL                 |
+----------------------+
```

```
SELECT UPPER (note) AS 'Tutto Maiuscolo'
FROM tabella;
```

Risultato:

```
+----------------------+
| Tutto_Maiuscolo      |
+----------------------+
| NULL                 |
| PRATICAMENTE UN PAZZO |
| NULL                 |
+----------------------+
```

10) Funzioni sulle stringhe **LEFT** e **RIGHT**
(estraggono n caratteri a sinistra o a destra della stringa)

```
SELECT LEFT (nome, 2) AS Sinistra
FROM tabella;
```

Risultato:

```
+----------+
| Sinistra |
+----------+
| pi       |
| pl       |
| pa       |
+----------+
```

```
SELECT RIGHT (nome, 3) AS Destra
FROM tabella;
```

Risultato:

```
+--------+
| Destra |
+--------+
| ppo    |
| uto    |
| ino    |
+--------+
```

11) Funzioni sulle stringhe **LENGTH** e **CHAR_LENGTH**
(restituiscono la lunghezza di una stringa, con la differenza che la prima misura la lunghezza in byte, mentre la seconda restituisce il numero di caratteri; evidentemente i valori saranno diversi per le stringhe che contengono caratteri multi-byte)

```
SELECT nome, LENGTH (nome) AS Lunghezza
FROM tabella;
```

Risultato:

```
+----------+-----------+
| nome     | Lunghezza |
+----------+-----------+
| pippo    |         5 |
| pluto    |         5 |
| paperino |         8 |
+----------+-----------+
```

Risultato medesimo in questo caso (nessun carattere multibyte):

```
+----------+-----------+
| nome     | Lunghezza |
+----------+-----------+
| pippo    |         5 |
| pluto    |         5 |
| paperino |         8 |
+----------+-----------+
```

12) Funzioni sulle stringhe **LPAD** e **RPAD**
(aggiungono, a sinistra (LPAD) o a destra, i caratteri necessari a portare la stringa alla lunghezza specificata, eventualmente accorciandola se più lunga)

```
SELECT LPAD (nome, 7, '_')
FROM tabella;
```

Risultato:

```
+---------------------+
| LPAD (nome, 7, '_') |
+---------------------+
| __pippo             |
| __pluto             |
| paperin             |
+---------------------+
```

```
SELECT RPAD (nome, 7, '*')
FROM tabella;
```

Risultato:

```
+---------------------+
| RPAD (nome, 7, '*') |
+---------------------+
| pippo**             |
| pluto**             |
| paperin             |
+---------------------+
```

13) Funzioni sulle stringhe **LTRIM** e **RTRIM**
(eliminano gli spazi a sinistra (LTRIM) o a destra (RTRIM) di una stringa purchè diversa da NULL)

```
SELECT LTRIM ( RIGHT (note, 6))
FROM tabella;
```

Risultato:

```
+-------------------------+
| LTRIM ( RIGHT (note, 6)) |
+-------------------------+
| NULL                    |
| pazzo                   |
| NULL                    |
+-------------------------+
```

N.B. Si noti l'effetto di LTRIM guardando l'output della sola istruzione RIGHT (note, 6) in cui c'è lo spazio iniziale:

```
SELECT RIGHT (note, 6)
FROM tabella;
```

Risultato:

```
+-----------------+
| RIGHT (note, 6) |
+-----------------+
| NULL            |
|  pazzo          |
| NULL            |
+-----------------+
```

14) Funzioni sulle stringhe **SUBSTRING**
(restituisce una parte della stringa, a partire dal carattere specificato fino alla fine della stringa o, se indicato, per un certo numero di caratteri)

```
SELECT SUBSTRING ('accidenti', 5);
```

Risultato:

```
+-------------------------+
| substring('accidenti',5) |
+-------------------------+
| denti                   |
+-------------------------+
```

```
SELECT SUBSTRING ('accidenti', 5, 3);
```

Risultato:

```
+---------------------------+
| substring('accidenti',5,3) |
+---------------------------+
| den                       |
+---------------------------+
```

15) Funzioni sulle stringhe **REVERSE e REPLACE**
(la prima inverte una stringa mentre la seconda sostituisce in una stringa una sequenza di caratteri in un'altra)

```
SELECT REVERSE ('accidenti');
```

Risultato:

```
+----------------------+
| REVERSE ('accidenti') |
+----------------------+
| itnedicca            |
+----------------------+
```

```
SELECT REPLACE ('uccello','cc','g');
```

Risultato:

```
+-----------------------------+
| REPLACE ('uccello','cc','g') |
+-----------------------------+
| ugello                      |
+-----------------------------+
```

16) Funzione matematica **ABS**
(restituisce il valore assoluto (non segnato) di un numero)

```
SELECT ABS (-34.65) AS 'Valore assoluto';
```

Risultato:

```
+-----------------+
| Valore assoluto |
+-----------------+
|           34.65 |
+-----------------+
```

17) Funzione matematica **POWER**
(effettua l'elevamento a potenza (richiede base ed esponente))

```
SELECT POWER (2,8) AS Potenza;
```

Risultato:

```
+---------+
| Potenza |
+---------+
|     256 |
+---------+
```

18) Funzione matematica **RAND**
(genera un valore casuale compreso tra 0 e 1)

```
SELECT id, RAND () AS Random
FROM tabella;
```

Risultato:

```
+----+--------------------+
| id | Random             |
+----+--------------------+
|  1 | 0.4945261743800027 |
|  2 | 0.2950904753944748 |
|  3 | 0.9918738845660108 |
+----+--------------------+
```

```
SELECT RAND () AS Random;
```

Risultato:

```
+--------------------+
| Random             |
+--------------------+
| 0.07409802738027463 |
+--------------------+
```

19) Funzioni matematiche di arrotondamento **FLOOR, CEILING, ROUND e TRUNCATE**

- **FLOOR** (arrotonda all'intero inferiore)
- **CEILING** (all'intero superiore)
- **ROUND** (arrotonda all'intero superiore da .5 in su, altrimenti all'inferiore)
- **TRUNCATE** che tronca il numero (non arrotonda) alla quantità specificata di decimali

```
SELECT FLOOR (prezzo) AS 'Uso Floor'
FROM tabella;
```

Risultato:

```
+-----------+
| Uso Floor |
+-----------+
|        44 |
|       144 |
|       200 |
+-----------+
```

```
SELECT CEILING (prezzo) AS 'Uso Ceiling'
FROM tabella;
```

Risultato:

```
+-------------+
| Uso Ceiling |
+-------------+
|          45 |
|         145 |
|         200 |
+-------------+
```

```
SELECT ROUND (prezzo) AS 'Uso Round'
FROM tabella;
```

Risultato:

```
+-----------+
| Uso Round |
+-----------+
|        44 |
|       145 |
|       200 |
+-----------+
```

```
SELECT TRUNCATE (prezzo, 1) AS 'Uso Truncate'
FROM tabella;
```

Risultato:

```
+--------------+
| Uso Truncate |
+--------------+
|         44.2 |
|        144.7 |
|        200.0 |
+--------------+
```

N.B. Se abbiamo bisogno di generare un intero compreso fra x e y, possiamo usare questa formula: FLOOR(x + RAND() * (y – x + 1)). Supponiamo di voler generare un numero casuale tra 1 e 100

```
SELECT FLOOR (1 + RAND() * 100) AS 'Numero casuale';
```

Risultato:

```
+----------------+
| Numero casuale |
+----------------+
|             53 |
+----------------+
```

20) Funzioni sulle date

- **CURRENT_DATE** restituisce la data attuale (formato YYYY-MM-DD)
- **CURRENT_TIME** restituisce l'ora attuale (formato HH-MM-SS);
- **CURRENT_TIMESTAMP** e **NOW**, che sono sinonimi, restituiscono data e ora.
- **DATE** estrae la parte data da un timestamp;
- **TIME** fa lo stesso con la parte ora

```
SELECT CURRENT_DATE ();
```

Risultato:

```
+----------------+
| CURRENT_DATE () |
+----------------+
| 2015-03-26     |
+----------------+
```

```
SELECT FROM_UNIXTIME (UNIX_TIMESTAMP( ), '%d-%m-%Y ');
```

Risultato:

```
+-------------------------------------------+
| FROM_UNIXTIME (UNIX_TIMESTAMP( ), '%d-%m-%Y') |
+-------------------------------------------+
| 26-03-2015                                |
+-------------------------------------------+
```

```
SELECT FROM_UNIXTIME (UNIX_TIMESTAMP(), '%d-%m-%Y %H.%i.%s');
```

Risultato:

```
+-----------------------------------------------------+
| FROM_UNIXTIME (UNIX_TIMESTAMP(), '%d-%m-%Y %H.%i.%s') |
+-----------------------------------------------------+
| 26-03-2015 11.50.55                                 |
+-----------------------------------------------------+
```

```
SELECT CURRENT_TIME ();
```

Risultato:

```
+-----------------+
| CURRENT_TIME ( ) |
+-----------------+
| 11:54:09        |
+-----------------+
```

```
SELECT NOW();
```

Risultato:

```
+---------------------+
| NOW()               |
+---------------------+
| 2015-03-26 11:56:56 |
+---------------------+
```

Ci sono poi, come anticipato, le funzioni che estraggono da una data, da un orario o da un timestamp le singole informazioni:

- **DAY** restituisce il giorno del mese,
- **DAYOFWEEK** restituisce il giorno della settimana (1 è la domenica e 7 il sabato),
- **DAYOFYEAR** restituisce il giorno dell'anno (da 1 a 366),
- **MONTH** restituisce il mese,
- **YEAR** restituisce l'anno;
- **HOUR** restituisce l'ora,
- **MINUTE** restituisce i minuti,
- **SECOND** restituisce i secondi

```
SELECT DATE (NOW());          N.B. (equivale a SELECT CURRENT_DATE())
```

Risultato:

```
+--------------+
| DATE (NOW()) |
+--------------+
| 2015-03-26   |
+--------------+
```

```
SELECT TIME (NOW());          N.B. (equivale a SELECT CURRENT_TIME())
```

Risultato:

```
+--------------+
| TIME (NOW()) |
+--------------+
| 15:30:19     |
+--------------+
```

```
SELECT DAY(NOW());
```

Risultato:

```
+------------+
| DAY(NOW()) |
+------------+
|         26 |
+------------+
```

```
SELECT DAYOFWEEK(NOW());       // dom = 0, lun = 1, mar = 2, merc = 3, etc, etc
```

Risultato:

```
+------------------+
| DAYOFWEEK(NOW()) |
+------------------+
|                5 |
+------------------+
```

```
SELECT DAYOFYEAR(NOW());       // 1 gennaio = giorno 1, etc. etc.
```

Risultato:

```
+------------------+
| DAYOFYEAR(NOW()) |
+------------------+
|               85 |
+------------------+
```

```
SELECT MONTH(NOW());           // genn = 1, febb = 2, mar = 3, etc. etc.
```

Risultato:

```
+--------------+
| MONTH(NOW()) |
+--------------+
|            3 |
+--------------+
```

```
SELECT YEAR(NOW());
```

Risultato:

```
+-------------+
| YEAR(NOW()) |
+-------------+
|        2015 |
+-------------+
```

```
SELECT HOUR(NOW()) AS ORA, MINUTE(NOW()) AS MINUTI, SECOND(NOW()) AS SECONDI;
```

Risultato:

```
+------+--------+---------+
| ORA  | MINUTI | SECONDI |
+------+--------+---------+
|   15 |     44 |      49 |
+------+--------+---------+
```

Ci sono infine due funzioni (**DATE_FORMAT** e **TIME_FORMAT**) che permettono di formattare i valori di date e ore. Esse richiedono in input una data o un timestamp (DATE_FORMAT) o un orario (TIME_FORMAT), nonchè una stringa contenente i simboli per la formattazione voluta

```
SELECT DATE_FORMAT (NOW(), '%d/%m/%Y');
```

Risultato:

```
+---------------------------------+
| DATE_FORMAT (NOW(), '%d/%m/%Y') |
+---------------------------------+
| 26/03/2015                      |
+---------------------------------+
```

```
SELECT TIME_FORMAT (NOW(), '%H:%i:%s');
```

Risultato:

```
+---------------------------------+
| TIME_FORMAT (NOW(), '%H:%i:%s') |
+---------------------------------+
| 15:53:19                        |
+---------------------------------+
```

Elenchiamo alcuni di questi simboli seguiti da qualche esempio:

- Anno: '**%Y**' (4 cifre), '**%y**' (2 cifre)
  - Mese: '**%m**' (2 cifre), '**%c**' (1 o 2 cifre), '**%M**' (nome intero in inglese), '**%b**' (nome abbreviato)
- Giorno del mese: '**%d**' (due cifre), '**%e**' (1 o 2 cifre)
  - Giorno della settimana: '**%W**' (nome intero in inglese), '**%a**' (nome abbreviato), **'%w'** (numerico con 0=domenica e 6=sabato)
- Ore: '**%H**' (2 cifre), '**%k**' (1 o 2 cifre), '**%h**' (2 cifre da 01 a 12), '**%l**' (1 o 2 cifre da 1 a 12)
- Minuti: '**%i**' (2 cifre)
- Secondi: '**%s**' (2 cifre)

```
SELECT DATE_FORMAT (NOW(), '%d/%M/%y');
```

Risultato:

```
+---------------------------------+
| DATE_FORMAT (NOW(), '%d/%M/%y') |
+---------------------------------+
| 26/March/15                     |
+---------------------------------+
```

```
SELECT TIME_FORMAT (NOW(), '%l:%i:%s');
```

Risultato:

```
+---------------------------------+
| TIME_FORMAT (NOW(), '%l:%i:%s') |
+---------------------------------+
| 3:54:59                         |
+---------------------------------+
```

21) Funzioni condizionali in MySQL: **CASE**, **IF**, **IFNULL** e **NULLIF**

CASE
(Mediante il costrutto CASE ... WHEN è possibile creare delle istruzioni condizionali piuttosto complesse prevedendo non solo un'unica condizione, ma una serie di condizioni alternative)

```
SELECT CASE id
        WHEN 1 THEN 'primo'
        WHEN 2 THEN 'secondo'
        ELSE 'altro'
        END
FROM tabella;
```

Risultato:

```
+-------------------------------------------------------------------------+
| primo                                                                   |
| secondo                                                                 |
| altro                                                                   |
+-------------------------------------------------------------------------+
```

IF
(La funzione IF è più semplice di CASE e consente di verificare un'unica condizione. La funzione prevede tre argomenti: il primo è l'espressione da verificare, il secondo è il risultato restituito in caso in cui l'espressione sia vera, il terzo il risultato da restituire nel caso l'espressione sia falsa)

```
SELECT id, nome, prezzo,
 IF ((prezzo > 100.00 AND prezzo < 200.00),'Medio Alto','Scarto') AS valutazione
FROM tabella;
```

Risultato:

```
+----+----------+--------+-------------+
| id | nome     | prezzo | valutazione |
+----+----------+--------+-------------+
|  1 | pippo    |  44.25 | Scarto      |
|  2 | pluto    | 144.75 | Medio Alto  |
|  3 | paperino | 200.00 | Scarto      |
+----+----------+--------+-------------+
```

IFNULL
(La funzione IFNULL intercetta eventuali errori del MySQL (che restituisce NULL) e li gestisce in modo predeterminato. La funzione ISNULL accetta due parametri: il primo è il dato da verificare, il secondo è il risultato da restituire qualora il primo argomento sia NULL)

```
SELECT id, nome, note, IFNULL(note,'Questo campo è a NULL') AS 'Esito IF'
FROM tabella;
```

Risultato:

```
+----+----------+----------------------+--------------------------------------+
| id | nome     | note                 | Esito IF                             |
+----+----------+----------------------+--------------------------------------+
|  1 | pippo    | NULL                 | Questo campo è a NULL                |
|  2 | pluto    | praticamente un pazzo | praticamente un pazzo               |
|  3 | paperino | NULL                 | Questo campo è a NULL                |
+----+----------+----------------------+--------------------------------------+
```

NULLIF
(Questa funzione accetta due argomenti che vengono confrontati tra loro; se sono uguali restituisce NULL in caso contrario restituisce il primo argomento)

```
SELECT *
INTO newtab
FROM tabella AS t1, tabella AS t2;
```

Risultato:

Query parametriche (da linea di comando)

Innanzitutto in modalità MySql console occorre definire il parametro ed assegnargli un valore

```
set @NomeX = "MARIO";
```

per poi eseguire la query in oggetto

```
SELECT *
INTO Fornitore
FROM Nome = @NomeX;
```

che restituirà

```
+-----------+------------------------------+------------------------------+------------+
| CodF      | Cognome                      | Nome                         | DataN      |
+-----------+------------------------------+------------------------------+------------+
| F_01      | ROSSI                        | MARIO                        | 1965-01-01 |
+-----------+------------------------------+------------------------------+------------+
```

Query da file

Creiamo, prima di ogni cosa, il file "insert_articolo.sql" contenente le istruzioni da eseguire sul db (ad esempio tre insert sulla tabella Articolo) all'interno della directory che ospita l'eseguibile di mysql (directory ***usr\local\mysql\bin***)

```
INSERT INTO Articolo
VALUES ("X_01", "LATTE", 11.25);
INSERT INTO Articolo
VALUES ("X_02", "POMATA", 17.25);
INSERT INTO Articolo
VALUES ("X_03", "VINO", 10.75);
```

per poi eseguire la query in oggetto da linea di comando tramite il comando "source"

```
source insert_articolo.sql;
```

Verranno aggiunte le tre nuove n-ple nella tabella in oggetto.