Sommario

[TIPI DI DATO 1](#_heading=h.gjdgxs)

[CREATE TABLE Statement 3](#_heading=h.30j0zll)

[ESEMPIO 4](#_heading=h.1fob9te)

[ALTER TABLE Statement 4](#_heading=h.3znysh7)

[ESEMPI 5](#_heading=h.2et92p0)

[CREATE INDEX Statement 5](#_heading=h.tyjcwt)

[INSERT INTO Statement 5](#_heading=h.3dy6vkm)

[UPDATE Statement 6](#_heading=h.1t3h5sf)

[DELETE FROM Statement 6](#_heading=h.4d34og8)

[SELECT Statement 6](#_heading=h.2s8eyo1)

[OPERATORI E FUNZIONI 7](#_heading=h.17dp8vu)

[FUNZIONI PER IL CONTROLLO DI FLUSSO 8](#_heading=h.3rdcrjn)

[FUNZIONI SULLE STRINGHE 9](#_heading=h.26in1rg)

[FUNZIONI MATEMATICHE 10](#_heading=h.lnxbz9)

[GESTIRE DATE E ORARI 10](#_heading=h.35nkun2)

# TIPI DI DATO

Dati numerici  
**TINYINT[(M)] [UNSIGNED] [ZEROFILL]**

Rappresenta un numero intero composto da 1 byte

**SMALLINT[(M)] [UNSIGNED] [ZEROFILL]**

Rappresenta un numero intero composto da 2 bytes

**MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]**

Rappresenta un numero intero composto da 3 bytes

**INT[(M)] [UNSIGNED] [ZEROFILL]**

Rappresenta un numero intero composto da 4 bytes

**BIGINT[(M)] [UNSIGNED] [ZEROFILL]**

Rappresenta un numero intero composto da 4 bytes

**FLOAT[(M,D)] [UNSIGNED] [ZEROFILL]**

Rappresenta un numero in virgola mobile precisione singola. *M* rappresenta il

numero totale di cifre rappresentate e *D* il numero di cifre decimali.  
**DOUBLE[(M,D)] [UNSIGNED] [ZEROFILL]**

Rappresenta un numero in virgola mobile precisione doppia. *M* rappresenta il

numero totale di cifre rappresentate e *D* il numero di cifre decimali.  
**DECIMAL[(M[,D])] [UNSIGNED] [ZEROFILL]**

Rappresenta un numero decimale esatto, con *M* cifre totali di cui *D* decimali.

**opzioni UNSIGNED e ZEROFILL.**

Con la prima si specifica che il numero è senza segno, per cui non saranno consentiti valori negativi. Con la seconda si indica al server di memorizzare i numeri con degli zeri davanti nel caso in cui la lunghezza sia inferiore a quella massima prevista. Se usate ZEROFILL MySQL aggiungerà automaticamente UNSIGNED.

Date e tempo

**DATE**

Contiene una data nel formato YYYY-MM-GG   
**DATETIME**

contiene una data e un'ora. La visualizzazione è nel formato 'AAAA-MM-GG  
HH:MM:SS'

**TIMESTAMP[(M)]**

In un TIMESTAMP possono essere memorizzati i valori  
corrispondenti al timestamp Unix.Questo tipo di dato è utile per memorizzare automaticamente il momento dell'aggiornamento di una riga di tabella: infatti MySQL può impostare in automatico una colonna TIMESTAMP di una tabella nel momento in cui vieneeffettuata una INSERT o un UPDATE. La visualizzazione del timestampavviene nello stesso formato del DATETIME

**possibile definizione di una colonna TIMESTAMP:**

TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

ON UPDATE CURRENT\_TIMESTAMP

Con questa dichiarazione, la colonna viene automaticamente inizializzata  
e successivamente aggiornata ad ogni modifica della riga con il timestamp  
del momento. Se omettete una delle due dichiarazioni, solo l'altra sarà  
attiva; ovviamente per la clausola DEFAULT potete anche fornire un valore  
costante.

**TIME**

contiene un valore di tempo (ore,minuti e secondi)  
**YEAR[(2|4)]**

rappresenta, su quattro cifre, unanno compreso fra 1901 e 2155, oppure 0000. Su due cifre invece i valori vanno da 70 (1970) a 69 (2069).

Stringhe

**CHAR(M)**

stringa di lunghezza fissa (*M*)  
riempita con spazi a destra al momento della memorizzazione, che vengono  
eliminati in fase di lettura

**VARCHAR(M)**

stringa a lunghezza variabile  
**BINARY(M)  
VARBINARY(M)**

BINARY e VARBINARY corrispondono a CHAR  
e VARCHAR, ma memorizzano stringhe di byte invece che di caratteri. Non  
hanno quindi character set.

**TINYBLOB  
TINYTEXT  
BLOB[(M)]  
TEXT[(M)]  
MEDIUMBLOB  
MEDIUMTEXT  
LONGBLOB  
LONGTEXT**

I formati di tipo BLOB e TEXT sono utilizzati rispettivamente per valori  
binari e di testo.  
La lunghezza massima è 255 caratteri per TINYBLOB e TINYTEXT, 65535 per BLOB e  
TEXT, 16.777.215 per MEDIUMBLOB e MEDIUMTEXT, 4 gigabyte per LONGBLOB e  
LONGTEXT.

**ENUM('valore1','valore2',...)**

può contenere uno dei valori elencati nella definizione, oppure NULL o una stringa vuota, che viene assegnata quando si cerca di inserire un valore non valido. I valori possibili possono essere fino a 65535.

**SET('valore1','valore2',...)**

Una colonna SET, come la ENUM, prevede un insieme di valori possibili (fino a 64), ma in questo caso la colonna può assumere anche più di un valore, oppure nessuno.

# CREATE TABLE Statement

**CREATE TABLE *nome\_tabella***

**(*nome\_colonna1* data\_type [NOT NULL | NULL] [DEFAULT {literal | (expr)} ]**

**[AUTO\_INCREMENT] [UNIQUE [KEY]] [[PRIMARY] KEY]**

**[CHECK (expr)],**

**…………………..**

**…………………..**

***nome\_colonnan* data\_type [NOT NULL | NULL] [DEFAULT {literal | (expr)} ]**

**[AUTO\_INCREMENT] [UNIQUE [KEY]] [[PRIMARY] KEY]**

**[CHECK (expr)],**

**PRIMARY KEY (*nome\_colonna1,..,nome\_colonnam)***

**FOREIGN KEY (*nome\_colonna1,..nome\_colonnam*) REFERENCES *nome\_tabella(nome\_colonna1,..come\_colonnam)***

**[ON DELETE RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT]**

**[ON UPDATE RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT];**

# ESEMPIO

**CREATE TABLE Anagrafiche (**

**ID\_anagrafica int NOT NULL AUTO\_INCREMENT PRIMARY KEY,**

**Nome VARCHAR(100) NOT NULL,**

**Cognome VARCHAR(100) NOT NULL,**

**Pensione ENUM('pubblico', 'privato'),**

**Anni INT CHECK (Anni >= 60),**

**Regione int(5),**

**FOREIGN KEY (Regione) references Regioni (Cod\_Regione)**

**ON DELETE SET NULL**

**ON UPDATE CASCADE);**

# ALTER TABLE Statement

**ALTER TABLE *nome\_tabella***

**| ADD [COLUMN] (*nome\_colonna1* definizione\_colonna1,...*nome\_colonnan* definizione\_colonnan)**

**| ADD [CONSTRAINT [*symbol*]] PRIMARY KEY(*nome\_colonna1,..,nome\_colonnam*)**

**| ADD CONSTRAINT *simbolo\_fk* FOREIGN KEY (*nome\_colonna1,..nome\_colonnam*)**

**REFERENCES *nome\_tabella(nome\_colonna1,..come\_colonnam)***

**[ON DELETE RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT]**

**[ON UPDATE RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT]**

**| MODIFY [COLUMN] *nome\_colonna* definizione\_colonna**

**| DROP [COLUMN] *nome\_colonna***

**| DROP PRIMARY KEY**

**| DROP {INDEX|KEY} *nome\_indice***

**| DROP FOREIGN KEY *simbolo\_fk***

**| RENAME [TO] *nuovo\_nome\_tabella***

**definizione\_colonna:**

**tipo\_dato [NOT NULL | NULL] [DEFAULT valore\_default]**

**[AUTO\_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]**

# ESEMPI

[ALTER](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/8.0/en/alter-table.html)[TABLE](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/8.0/en/alter-table.html)**anagrafiche ADD (Tessera\_Sanitaria varchar(40),Codice\_Fiscale char(16));**

[ALTER](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/8.0/en/alter-table.html)[TABLE](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/8.0/en/alter-table.html)**anagrafiche ADD CONSTRAINT FK\_1 FOREIGN KEY (Regione)**

**REFERENCES regioni(Cod\_Regione)**

**ON**[DELETE](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/8.0/en/delete.html)**RESTRICT ON**[UPDATE](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/8.0/en/update.html)**RESTRICT;**

# CREATE INDEX Statement

**CREATE [UNIQUE} INDEX *nome\_indice* ON *nome\_tabella***

**(*nome\_colonna1, .., , nome\_colonnam*);**

# INSERT INTO Statement

**INSERT INTO *nome\_tabella* [ *nome\_colonna1, …. ,nome\_colonnam*) ]**

**VALUES ( *elenco\_valori* )**

# UPDATE Statement

**UPDATE *nome\_tabella*  
SET *nome\_colonna1=espressione* [,  
*nome\_colonna2=espressione2* ...]  
[WHERE *condizioni*]  
[ORDER BY ...]  
[LIMIT *numero\_righe*]**

**l'uso di ORDER BY ha senso se accoppiato con LIMIT.**

# DELETE FROM Statement

**DELETE FROM *nome\_tabella*  
[WHERE *condizioni*]  
[ORDER BY ...]  
[LIMIT *numero\_righe*]**

**l'uso di ORDER BY ha senso se accoppiato con LIMIT.**

# SELECT Statement

**SELECT**

**[ALL | DISTINCT]**

**\*|Nome\_colonna1[[as] nome\_colonna1],…..nome\_colonnan[[as] nome\_colonnan**

**[FROM tabelle|FROM tabella1 LEFT JOIN tabella2 ON tabella1.column\_name = tabella2.column\_name| FROM tabella1 RIGHT JOIN tabella2 ON tabella1.column\_name = tabella2.column\_name**

**[WHERE condizioni]**

**[GROUP BY {nome\_colonna | espressione | posizione}**

**[ASC | DESC]**

**[HAVING condizioni]**

**[ORDER BY {nome\_colonna | espressione | posizione}**

**[ASC | DESC] , ...]**

**[LIMIT [offset,] numero\_righe]**

**]**

**SELECT**, seguita da una o più colonne della tabella Risultato, oppure delle espressioni ottenute con funzioni applicate alle colonne, o con espressioni matematiche, o con funzioni che restituiscono valori indipendenti (come nel caso di NOW()).

**AS** attribuisce un alias alle colonne , utile per rendere i nomi dei campi più semplici , o per attribuire un nome alle espressioni

**DISTINCT** , permette di escludere dal risultato le righe duplicate, ovvero quelle identiche ad altre righe. Se ci sono due o più righe di risultato uguali, con DISTINCT ne vedremo una sola. ALL è l’opposto di DISTINCT, cioè estrae tutto, ed è il valore applicato per default

**FROM**, seguita dai nomi di una o più tabelle dalle quali devono essere estratti i Dati, Le query più semplici estraggono dati da una sola tabella, ma è molto frequente aver bisogno di combinare più tabelle. In questo caso si effettua una JOIN.

**LEFT JOIN**. la query ottiene gli stessi risultati della join tra tabelle con in più le righe senza corrispondente che si trovano nella tabella di sinistra. In queste righe della tabella risultato, i campi dell’altra tabella saranno valorizzati a NULL

**RIGHT JOIN** ,Le right outer join restituiscono invece le righe della seconda tabella che non hanno corrispondente nella prima

**WHERE**, che specifica le condizioni in base alle quali ogni riga sarà estratta

oppure no dalle tabelle;

**GROUP BY**, che specifica le colonne sui cui valori devono essere raggruppate le righe nel risultato: tutte le righe con valori uguali verranno ridotte a una;

**HAVING**, che specifica ulteriori condizioni da applicare alle righe dopo il

raggruppamento effettuato da GROUP BY; La clausola HAVING svolge una funzione di selezione delle righe, esattamente come WHERE. La differenza è che WHERE viene applicata sulle righe delle tabelle originali, mentre HAVING viene applicata sulle righe della tabella risultato dopo i raggruppamenti richiesti dalle GROUP BY. In pratica, è utile per effettuare test sui valori restituiti dalle funzioni di colonna.

**ORDER BY**, che specifica in quale ordine figureranno le righe del resultset;

**LIMIT**, che stabilisce il massimo numero di righe da estrarre

# OPERATORI E FUNZIONI

Operatori e funzioni vengono utilizzati in diversi punti delle istruzioni SQL. Ad esempio per determinare i valori da selezionare, per determinare le condizioni in una WHERE, o nelle clausole ORDER BY, GROUP BY, HAVING

**operatori aritmetici:**

“+” (addizione)

“–” (sottrazione)

“\*” (moltiplicazione)

“/” (divisione)

“%” (modulo – resto della divisione)

**operatori di confronto:**

“=” (uguale)

“<>” o “!=” (diverso)

“<” (minore)

“>” (maggiore)

“<=” (minore o uguale)

“>=” (maggiore o uguale)

**operatori logici:**

NOT

AND

OR

**IS NULL e IS NOT NULL**

per verificare se un valore è (o non è) NULL;

**BETWEEN** **valore1 AND valore2**

per test su valori compresi fra due estremi (inclusi);

**IN** **(lista\_valori separati da virgole)**

per verificare l’appartenenza di un valore ad una lista di valori dati.

**esempio:**

SELECT a,b,c,d,e,f,g FROM t1

WHERE a=b AND a<=c

AND (d=5 OR d=8)

AND e BETWEEN 7 and 9

AND f IN(‘a’,’b’,’c’)

AND g IS NOT NULL;

**LIKE**

utilizzabile per trovare corrispondenze parziali sulle stringhe. Possiamo usare due caratteri jolly nella stringa da trovare: “%” che rappresenta “qualsiasi numero di caratteri o nessun carattere”, e “\_” che invece corrisponde esattamente ad un carattere.

Quindi, ad esempio:

SELECT \* FROM tab1 WHERE colonna LIKE ‘pao%’;

SELECT \* FROM tab1 WHERE colonna LIKE ‘\_oro’;

SELECT \* FROM tab1 WHERE colonna LIKE ‘%oro’;

La prima query troverà ‘paolo’, ‘paola’ e ‘paolino’; la seconda troverà ‘moro’ ma non

‘tesoro’ perchè si aspetta esattamente un carattere in testa alla stringa; l’ultima invece

troverà ‘moro’, ‘tesoro’ e anche ‘oro’.

# FUNZIONI PER IL CONTROLLO DI FLUSSO

Sono utili quando vogliamo eseguire dei test sui valori contenuti in una tabella e

decidere cosa estrarre in base al risultato. Le indichiamo con la loro sintassi:

– **CASE** valore **WHEN** [valore1] **THEN** risultato1 [**WHEN** [valore2] **THEN** risultato2 ….]

[**ELSE** risultatoN] **END**

– **CASE WHEN** [condizione1] **THEN** risultato1 [**WHEN** [condizione2] **THEN** risultato2 …]

[**ELSE** risultatoN] **END**

– **IF**(espressione1,espressione2,espressione3)

– I**FNULL**(espressione1,espressione2)

– **NULLIF**(espressione1,espressione2)

Le prime due (CASE) sono quasi uguali: nel primo caso viene specificato un valore che

sarà confrontato con quelli espressi dopo la WHEN; il primo che risulta uguale

determinerà il risultato corrispondente (quello espresso con THEN).

Nel secondo caso non c’è un valore di riferimento, ma vengono valutate le varie

condizioni come espressioni booleane: la prima che risulta vera determina il risultato

corrispondente. In entrambi i casi, se è presente il valore ELSE finale viene usato nel

caso in cui nessuna delle condizioni precedenti sia soddisfatta (in mancanza di ELSE

verrebbe restituito NULL).

Con la IF viene valutata la prima espressione: se vera viene restituita la seconda,

altrimenti la terza. IFNULL restituisce la prima espressione se diversa da NULL,

altrimenti la seconda. NULLIF restituisce NULL se le due espressioni sono uguali; in

caso contrario restituisce la prima.

ESEMPI

SELECT OrderID, Quantity,  
CASE  
    WHEN Quantity > 30 THEN "The quantity is greater than 30"  
    WHEN Quantity = 30 THEN "The quantity is 30"  
    ELSE "The quantity is under 30"  
END  
FROM OrderDetails;

SELECT OrderID, Quantity, IF(Quantity>10, "MORE", "LESS")  
FROM OrderDetails;

# FUNZIONI SULLE STRINGHE

**CONCAT e CONCAT\_WS** si utilizzano per concatenare due o più stringhe, nel secondo

caso aggiungendo un separatore.

**LOWER e UPPER** consentono di trasformare una stringa, rispettivamente, in tutta

minuscola o tutta maiuscola.

**LEFT e RIGHT** estraggono n caratteri a sinistra o a destra della stringa.

**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.

**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).

**LTRIM e RTRIM** eliminano gli spazi a sinistra (LTRIM) o a destra.

**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.

Alcuni esempi, seguiti dai rispettivi risultati:

SELECT CONCAT\_WS(‘;’,’Primo’,’Secondo’,’Terzo’);

-> Primo;Secondo;Terzo

SELECT LOWER(‘Primo’);

-> primo

SELECT RIGHT(‘Primo’,2);

-> mo

SELECT LENGTH(‘Primo’);

-> 5

SELECT LPAD(‘Primo’,7,’\_’);

-> \_\_Primo

SELECT LTRIM(‘ Primo’);

-> Primo

SELECT SUBSTRING(‘Primo’,2);

-> rimo

SELECT SUBSTRING(‘Primo’,2,3);

-> rim

# FUNZIONI MATEMATICHE

**ABS** restituisce il valore assoluto (non segnato) di un numero; **POWER** effettua

l’elevamento a potenza (richiede base ed esponente); **RAND** genera un valore casuale

compreso tra 0 e 1.

Abbiamo poi le funzioni di arrotondamento, che sono:

**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

Ecco gli esempi:

SELECT ABS(-7.9);

-> 7.9

SELECT POWER(3,4);

-> 81

SELECT RAND();

-> 0.51551992494196 (valore casuale)

SELECT CEILING(6.15);

-> 7

SELECT ROUND(5.5);

-> 6

SELECT TRUNCATE(6.15,1);

-> 6.1

Se abbiamo bisogno di generare un intero compreso fra x e y, possiamo usare questa

formula: “FLOOR(x + RAND() \* (y – x + 1))”. Ad esempio, per avere un numero

compreso fra 1 e 100:

SELECT FLOOR(1 + RAND() \* 100);

# GESTIRE DATE E ORARI

MySQL esprime il formato della data come AAAA-MM-GG dove

AAAA rappresenta l’anno scritto a quattro cifre, mentre MM e GG simboleggiano,

rispettivamente, il mese ed il giorno. Il tempo viene trattato nel formato OO:MM:SS in

cui i due punti separano, nell’ordine, le ore dai minuti e questi dai secondi.

Per impostare le informazioni temporali attuali, si possono usare le

seguenti funzioni:

NOW() restituisce data e ora attuali. Ammette i sinonimi CURRENT\_TIMESTAMP() e

CURRENT\_TIMESTAMP

CURDATE() restituisce data attuale. Ammette i sinonimi CURRENT\_DATE() e CURRENT\_DATE

CURTIME() restituisce orario attuale. Ammette i sinonimi CURRENT\_TIME() e CURRENT\_TIME

**Recuperare le informazioni**

Le informazioni data/ora possono essere lette, in tutto o in parte, prelevandone solo alcuni elementi. Per queste operazioni, esistono apposite funzioni:

**YEAR(), MONTH() e DAY()** che, rispettivamente, restituiscono anno, mese e giorno;

**ore, minuti e secondi:**

**HOUR(), MINUTE() e SECOND()** per estrapolare ore, minuti e secondi da informazioni orarie;

**DAYOFWEEK** per sapere a che giorno della settimana corrisponde una certa data**.** Il risultato sarà un

numero da 1 a 7, dove 1 corrisponderà alla domenica, 2 a lunedì e così via fino al

7, che corrisponde a sabato**.**

**DAYOFYEAR** per ottenere un numero compreso tra 1 e 366 che indicherà il giorno dell’anno corrispondente alla data presa in considerazione**.**

**Formattare date e orari**

Per mostrare l’output agli utenti è necessario formattare opportunamente questi dati

**DATE\_FORMAT()**  permette di esprimere il formato di una data usando unastringa costituita da appositi metacaratteri:

| **Metacarattere** | **Descrizione** |
| --- | --- |
| %d | giorno del mese |
| %m | mese espresso in numero. La variante %M esprime il mese in parole |
| %Y | l’anno su quattro cifre. La variante %y esprime l’anno su due cifre |

Ecco alcuni esempi che mostrano vari modi per esprimere il 1° marzo 2015

> SELECT DATE\_FORMAT('2015-03-01','%d/%m/%Y');

01/03/2015

> SELECT DATE\_FORMAT('2015-03-01','%d/%m/%y');

01/03/15

> SELECT DATE\_FORMAT('2015-03-01','%d %M %Y');

01 March 2015

**TIME\_FORMAT()** svolge lo stesso compito di DATE\_FORMAT(), ma è riferita agli

orari. I metacaratteri più comunemente usati sono: %H per indicare le ore (da 0 a 24, in

alternativa %h le mostra da 0 a 12), %i per i minuti e %S o %s per i secondi.

Alcuni esempi:

> SELECT TIME\_FORMAT('17:25:34','%H-%i');

17-25

> SELECT TIME\_FORMAT('17:25:34','%h:%i %p');

05:25 PM

> SELECT TIME\_FORMAT('17:25:34','sono le %H e %i minuti');

sono le 17 e 25 minuti

**Calcoli con date e orari**

**ADDDATE** calcola la data derivante dalla somma tra il primo argomento, ed un intervallo di tempo espresso come INTERVAL espressione unità. Ad esempio:

> SELECT ADDDATE('2015-03-01',INTERVAL 5 DAY);

2015-03-06

> SELECT ADDDATE('2015-03-01', 5);

2015-03-06

Come si vede, per sommare alla data cinque giorni si possono usare due espressioni

diverse, INTERVAL 5 DAY o semplicemente il numero 5

**ADDTIME** calcola un nuovo orario

Ecco direttamente qualche esempio:

> SELECT ADDTIME('17:25','05:05');

22:30:00

> SELECT ADDTIME('17:25','00:05:05');

17:30:05

**DATE\_SUB** il cui funzionamento è speculare a DATE\_ADD:

> SELECT DATE\_SUB('2015-03-01',INTERVAL 5 DAY);

2015-02-24

**DATEDIFF** per calcolare il periodo che intercorre tra due date. Questa

funzione accetta due argomenti ed il risultato sarà un numero positivo se la prima data

è successiva alla seconda, negativo altrimenti:

> SELECT DATEDIFF('2015-03-01','2015-02-10');

19

> SELECT DATEDIFF('2015-01-01','2015-02-10');

-40