21. MySQL: Tipi di dato

Tipi di Dato Numerici in SQL:

Nome del tipo	**Memoria occupata**	**Intervallo di valori consentito**	**Se solo positivi** (UNSIGNED)
TINYINT	1 byte	da -128 a +127	da 0 a +255
SMALLINT	2 byte	da -32768 a +32767	da 0 a +65535
MEDIUMINT	3 byte	da -8388608 a +8388607	da 0 a +16777215
INT	4 byte	da -2147483648 a +2147483647	da 0 a +4294967295
BIGINT	8 byte	da -9223372036854775808 a +9223372036854775807	da 0 a +18446744073709550615
FLOAT(I,D)	4 byte	A seconda dei valori	
DOUBLE(I,D)	8 byte	A seconda dei valori	
DECIMAL(I,D)	Il peso di I + 2 Byte	A seconda dei valori	

I e D rappresentano i numeri Interi e Decimali ammessi.

Un approfondimento lo meritano i tipo **FLOAT, DOUBLE e DECIMAL.**
Ad esempio **FLOAT(5,3)** indica a MySQL di salvare 5 cifre totali di cui 3 per la parte decimale. I numeri quindi saranno salvati fino a 99.999. Numeri con valori diversi, verranno arrotondati. Un numero come 45,7869 diventerà 45,787. **DOUBLE e DECIMAL** funzionano nella stessa maniera ma possono contenere valori maggiori (e quindi occupano più spazio).
Eccezioni dei valori nei dati di tipo numerico
Se all'interno di un campo di tipo numerico si specifica un valore maggiore di quanto ammesso, MySQL salverà nel database il massimo numero ammesso per quel campo. Per spiegare meglio il concetto, ammettiamo di avere un campo **TINYINT (UNSIGNED ovvero privo di segno, quindi positivo: da 0 a 255)**: se provassimo ad inserire un valore pari a 4587, MySQL salverà 255 come valore.
E' importante quindi porre molta attenzione nella scelta dei campi di tipo numerico. Anche perché sono i dati a cui MySQL accede più rapidamente.

I modificatori dei tipi numerici: AUTO_INCREMENT, UNIQUE e ZEROFILL
Il primo modificatore, **AUTO_INCREMENT,** permette di creare un campo numerico che aumenta ogni nuova riga. Se si cancellasse questo ultimo record e se ne aggiungesse un altro, il valore del nuovo campo AUTO_INCREMENT sarà 3. Questo perché, anche se una riga è stata cancellata, MySQL si ricorda il valore massimo e a questo aggiunge ogni volta una unità. Non è possibile modificare il valore che viene sommato alla nuova riga (è e resta +1).
E' consigliabile utilizzare sempre un campo auto incrementale a cui fare riferimento per selezionare, modificare o cancellare i record. Tenetevi quindi sempre da parte un campo (magari di nome "id") di tipo auto incrementale. Sarà comodissimo in molti casi.

Il secondo modificatore per i dati numerici, **UNIQUE**, identifica un campo i cui valori sono uno diverso dall'altro. Se si tenta di aggiungere un record con lo stesso valore, MySQL genera un errore tipo:

1062 – Duplicate entry '*N*' for key *N*

L'ultimo modificatore ammesso da MySQL per i campi numerici è **ZEROFILL**, che consente di inserire tanti 0 quanti sono ammessi dal tipo di campo, prima della cifra che realmente viene salvata. Ad esempio, con un campo di tipo INT(5) e ZEROFILL, per salvare un numero come 78, questo verrà immagazzinato nella forma 00078. Può essere utile per salvare dati bancari che prevedono una lunghezza fissa, come ABI, CAB o numero di conto corrente.

Tipi di Stringhe in MYSQL:
Un tipo di dato stringa è un campo che può contenere qualunque tipo di carattere: numerico, alfanumerico, simboli ecc.
Come per i campi numerici ce ne sono di vari tipi, a seconda della quantità di dati che si vuole salvare.

Nella tabella seguente, maggiori informazioni:

Nome del tipo	**Dimensioni massime**	**Memoria occupata**
CHAR	255 byte	X byte (*)
VARCHAR	255 byte	X+1 byte (*)
TINYTEXT	255 byte	X+1 byte (*)
TINYBLOB	255 byte	X+2 byte (*)
TEXT	65535 byte	X+2 byte (*)
BLOB	65535 byte	X+2 byte (*)
MEDIUMTEXT	1,6 MB	X+3 byte (*)
MEDIUMBLOB	1,6 MB	X+3 byte (*)
LONTEXT	4,2 GB	X+4 byte (*)
LONGBLOB	4,2 GB	X+4 byte (*)

(*) X è lo spazio occupato dal testo all'interno del campo , ovvero i byte occupati in memoria.

CHAR e VARCHAR
Questi due tipi di campi, nonostante la somiglianza nel nome, si comportano in maniera molto diversa. Il primo ha una lunghezza fissa, mentre il secondo è variabile.
Ciò significa che se creassimo un campo **CHAR(9)** e al suo interno specificassimo "ciao", questo campo occuperebbe comunque 9 byte invece di 4. Con **VARCHAR(9)** invece, scrivendo al suo interno "ciao" il campo occuperebbe 5 byte (guardare la tabella superiore X+1 dove in questo caso è X=4, quindi 4+1=5).

All'interno di una tabella, non è possibile utilizzarli entrambi. Creando ad esempio due campi nella stessa tabella, uno CHAR e l'altro VARCHAR, MySQL renderà entrambi VARCHAR.
TEXT e BLOB
TEXT e BLOB sono i campi di MySQL dedicati a contenere grandi quantità di dati. Fino a 4,2 GB con i LONGTEXT e LONGBLOB.
Il secondo in particolare, il campo di tipo BLOB sta per Bynary Large Object e consente il salvataggio di interi file nel formato binario. Utile per nascondere file dietro username e password, senza così riuscire a rintracciare il percorso fisico del file (che infatti non esiste, essendo incluso direttamente nel database).

I modificatori
I modificatori previsti da questi tipi di campi sono:

- **BINARY**:
  ammesso dai campi CHAR e VARCHAR: i dati salvati saranno trattati come stringhe binarie.
- **DEFAULT**:
  ammesso da tutti i tipi di campi: imposta un valore predefinito nel caso il campo fosse lasciato vuoto.
- **NOT NULL**:
  ammesso da tutti i tipi di campi: impone che il campo non sia lasciato vuoto.
- **NULL**:
  ammesso da tutti i tipi di campi: se il campo non contiene un valore, sarà vuoto.
- **PRIMARY KEY**:
  ammesso da tutti i campi, ma è consigliabile impostarlo su dati di tipo numerico. Serve a impostare un indice i quali dati non devono essere vuoti.

Gli altri modificatori sono **UNIQUE, UNSIGNED e ZEROFILL**.
Tipi di Campo DATA in MySQL:
I tipi di campi data sono tutti quei campi che contengono come valore la data e/o l'ora.
Nella tabella che segue, li mostriamo uno ad uno:

Nome del tipo	**Formato**	**Se vuoto (zero)**
DATETIME	AAAA-MM-GG hh:mm:ss	0000-00-00 00:00:00
DATE	AAAA-MM-GG	0000-00-00
TIME	hh:mm:ss	00:00:00
YEAR	AAAA	0000
TIMESTAMP(2)	AA	00
TIMESTAMP(4)	AAMM	0000
TIMESTAMP(6)	AAMMGG	000000
TIMESTAMP(8)	AAAAMMGG	00000000
TIMESTAMP(10)	AAMMGGhhmm	0000000000
TIMESTAMP(12)	AAMMGGhhmmss	000000000000
TIMESTAMP(14)	AAAAMMGGhhmmss	00000000000000

A=anno, M=mese, G=giorno, h=ora, m=minuti, s=secondi

DATETIME
E' il formato più completo e preciso a nostra disposizione. Varia da 1000-01-01 00:00:00 a 9999-12-31 23:59:59

DATE
Uguale al precedente, ma senza l'ora. Ammette infatti dati a partire da 1000-01-01 al 9999-12-31.

TIME
Salva l'ora. I valori vanno da 00:00:00 a 23:59:59. E' possibile però salvare intervalli di valore tra un evento e un altro e qundi, ammettere ore differenti. In questo caso, i dati vanno da -838:59:59 a 838:59:59.
MySQL legge i valori partendo da destra, quindi, salvando un campo con il contenuto 8:32, nel database verrà interpretato come 00:08:32.

Oltre ai due punti (:) MySQL ammette altri segni di interpunzione senza particolari difficoltà. L'immissione di un valore sbagliato sarà salvato come mezzanotte in punto (00:00:00).

YEAR
Salva l'anno ed è il campo più leggero: 1 solo byte. Ammette valori dal 1901 al 2155. Gli anni possono essere salvati a due o a quattro cifre. MySQL, in caso di anni a due cifre, interpreterà i valori da 70 a 99 come dal 1970 al 1999. Quelli dall'1 al 69, come dal 2001 al 2069.
Per evitare fraintendimenti quindi, è consigliabile impostare l'anno a quattro cifre.

TIMESTAMP
Questo campo salva (nel formato scelto dal numero tra le parentesi, si veda la tabella superiore) il momento esatto in cui la tabella viene modificata. Quindi può essere utile per visualizzare (senza doverlo calcolare ogni volta) il momento dell'ultima modifica del record in cui il campo TIMESTAMP appartiene.
Ammette anni compresi tra il 1970 e il 2037.
Tutti i tipi di TIMESTAMP occupano lo stesso spazio: 4 byte. Perché questo? Nonostante i vari formati? Perché MySQL salva comunque tutti i dati e poi ne visualizza solo quelli richiesti. Ad esempio, con TIMESTAMP(2) il database visualizza solo due cifre dell'anno, ma in memoria ha tutti gli altri dati (anno a 4 cifre, mese, giorno, ora, minuti e secondi). Quando infatti modifichiamo il tipo di TIMESTAMP, ad esempio con TIMESTAMP(8) lui ha tutti i dati in memoria.
La stessa cosa avviene quando abbassiamo il valore di TIMESTAMP, visualizzando quindi meno dati. MySQL non cancellerà i vari valori, semplicemente li nasconderà.
Altri Tipi di Dato in MySQL:
I tipi di campi che ancora non abbiamo trattato sono i campi a scelta. Quando l'utente dovrà selezionare per forza una delle voci previste (ad esempio da un menù a tendina:) è bene porre la propria attenzione su questi campi, perché MySQL ci accede più rapidamente di quelli testuali.

ENUM
Indica a MySQL le varie possibilità previste. Ad esempio, con:

CREATE TABLE nome_tabella (nome_campo ENUM('mare','montagna','lago') ….);

Si impone l'utente la scelta di uno di queste tre possibilità. Altri valori, saranno trattati come valori vuoti (NULL), a meno che non sia definito un valore di **default(_campo ENUM('mare','montagna','lago') DEFAULT 'mare'…) o che venga scelto il valore 0 che equivale alla stringa vuota "".**
Si possono inserire fino a 65.535 voci.

Tornando all'esempio precedente: ENUM('mare','montagna','lago') a questo tipo di campo (che chiameremo "scelta_vacanze") è possibile selezionare una voce come se ci si trovasse di fronte a un array (che parte da 1 e non da 0 come in JAVA).

Ad esempio, con:
SELECT scelta_vacanze FROM nomeTabella WHERE scelta_vacanze = 2
Avremo come risultato il valore "montagna".

SET
Questo tipo di dato è uguale a ENUM, con la differenza di poter effettuare una scelta multipla. Il campo ENUM infatti, consente di scegliere un solo valore alla volta.
SET è utilizzato invece in quei casi in cui i valori si possono presentare contemporaneamente (l'esempio classico è dato dai permessi su un file). A causa della modalità in cui vengono salvati i

valori inseriti, essi non possono contenere virgole. Il campo può contenere fino ad un massimo di 64 specificazioni.

DEFAULT

Può essere utile una struttura che assegna automaticamente ad un attributo un valore particolare da noi prescelto, senza doverlo riempire ” a mano ” ogni volta.Ad esempio pensiamo ad un sistema di ordini di prodotti on line e assumiamo di essere dei clienti: quando scegliamo un articolo è ovvio che la quantità deve essere maggiore o uguale ad 1.Una buona applicazione lato-server deve considerare questi piccoli – ma importanti -particolari.
Esempio.
ProdottoScelto smallint default 1
Inoltre possiamo definire anche un attributo che ci informa dell’utente che sta interrogando la base di dati:
Cliente char(15) default user
In questo modo il processo sul server potrà sapere quale utente ha avuto accesso alla base di dati.
Ultima possibilità è quella di annullare una precedente impostazione di un attributo, dando il valore null.
Dato smallint default null

Può essere utile una struttura che assegna automaticamente ad un attributo un valore particolare da noi prescelto, senza doverlo riempire ” a mano ” ogni volta.Ad esempio pensiamo ad un sistema di ordini di prodotti on line e assumiamo di essere dei clienti: quando scegliamo un articolo è ovvio che la quantità deve essere maggiore o uguale ad 1.Una buona applicazione lato-server deve considerare questi piccoli – ma importanti -particolari.Esempio.ProdottoScelto 1Inoltre possiamo definire anche un attributo che ci informa dell’utente che sta interrogando la base di dati:Cliente In questo modo il processo sul server potrà sapere quale utente ha avuto accesso alla base di dati.Ultima possibilità è quella di annullare una precedente impostazione di un attributo, dando il valore null.Dato