11-11-2022

TIPI DI DATI

I tipi di dati in MySQL sono i seguenti mostrati in tabella:

Numerici

Tipo	Byte	Min. Value (Signed/Unsigned)	Max. Value (Signed/Unsigned)
TINYINT	1	-128/0	127/255
SMALLINT	2	-32.768/0	32.767/65535
MEDIUMINT	3	-8.388.608/0	8.388.607/16.777.215
INT	4	-2.147.483.648/0	2.147.483.647/4.294.967.295
BIGINT	8	-9.223.372.036.854.775.808/0	9.223.372.036.854.775.807 /18.446.744.073.709.551.615
FLOAT	4	+/-1.175494351E-38	+/-3.402823466E+38
DOUBLE	8	+/-2.225073858507201E-308	+/-1.7976931348623157E+308

Tipo	Byte	Min. Value (Signed/Unsigned)	Max. Value (Signed/Unsigned)
INTEGER	Equivale ad INT		
DOUBLE PRECISION	Equivale a DOUBLE		
REAL	Equivale a DOUBLE		
DECIMAL(M[,D])	M+2	Tutti i numeri di M cifre di cui D decimali.	
NUMERIC(M[,D])	Equivale a DECIMAL		
BIT(M)	M	una sequenza di M bit (MySQL 5.5)	

> Per gli interi, ci sono diverse varianti e ognuna di esse varia rispetto ai byte usati per essere rappresentati. Di default gli interi sono con il segno.

Date e Tempi

Tipo	Byte	Range
DATE	3	dal '01/01/1000' al '31/12/9999'
DATETIME	8	dal '01/01/1000 00:00:00' al '31/12/9999 23:59:59'
TIMESTAMP[(M)]	4	dal '01/01/1970' al '31/12/2037'
TIME	3	da '-838:59:59' a '838:59:59'
YEAR[(M)]	1	per YEAR(4) dal '1901' al '2144'

Testo e Altro

Tipo	Byte	Max Length
CHAR[(M)]/BINARY[(M)]	M	M
VARCHAR(M)/VARBINARY[(M)]	L+1	M
TINYBLOB/ TINYTEXT	L+1	255
BLOB/TEXT	L+2	65.535
MEDIUMBLOB/MEDIUMTEXT	L+3	16.777.215
LONGBLOB/LONGTEXT	L+4	4.294.967.295
ENUM('value1','value2',...)	1 o 2 byte	65535 elementi
SET ('value1','value2',...)	1,2,3,4 o 8 byte	64 elementi
JSON	---	---

L è la lunghezza effettiva del testo memorizzato.

`CHAR` e `BINARY` sono caratteri di M byte.
Inoltre si ha che L lunghezza stringa, M lunghezza della stringa

`ENUM` può assumere solo 1 valore specificato nella lista dei valori permessi
`SET` è come `ENUM` ma il campo può avere più valori, quindi è un *insieme di valori*.
`JSON`: in *javascript* gli oggetti vengono definiti cosi:

```
{
"x":5,
"y":10
}
```

E questo formato è utile per trasferire dati. JSON serve per scambiare documenti di questi tipi.

Client disponibili per MySQL:

- "*PHPMyAdmin*"
- "*MySQL Workbanch*"

ACCOUNT E PRIVILEGI

Come si crea un utente:

`CREATE USER 'name'@'host' IDENTIFIED BY 'PASSWORD'` dove:

- `@localhost` vorrà dire che si può collegare solo dal pc corrente
- `@*` vorrà dire che si potrà collegare da ogni possibile pc

> Di norma si utilizza ***root*** solo come **utente locale** e tutti gli altri utenti da remoto.

Come assegnare privilegi

`GRANT privilege,… ON *| 'db'. *| 'db'. 'table' TO 'username'@'host', …;` dove:

- "privilege" può essere dei seguenti valori:
  - ALL
  - USAGE: serve per usare il sistema, altrimenti può accedere ma non può fare nulla
  - SELECT, INSERT, UPDATE, DELETE
  - CREATE, ALTER, INDEX, DROP, CREATE VIEW, TRIGGER: dove `ALTER` modifica tabelle e `INDEX` crea nuovi *indici*.

Come rimuovere un utente e i privilegi

```
DROP USER 'name'@'host',…;
REVOKE privilege,… ON *| 'db'. *| 'db'. 'table' FROM 'username'@'host', …;
```

Come creare un database

```
CREATE DATABASE [IF NOT EXISTS] nome;
```

Con questa istruzione il database viene creato se non esiste già, altrimenti dà un errore.

Cancellare un database

```
DROP DATABASE [IF EXISTS] nome;
```

Accesso ad un database

```
USE nome; (nome del db)
```

Gestione Database e Tabelle

```
CREATE TABLE [IF NOT EXISTS] nome (
        campo1 TIPO1 ALTRI PARAMETRI,
        campo2 TIPO2 ALTRI PARAMETRI,
        …
        campoN TIPON ALTRI PARAMETRI,
        PRIMARY KEY(campo1, campo2)
) ENGINE=InnoDB;
```

> `ENGINE`, ogni tabella può avere diversi *engine*. di **default** è `innoDB` che risulta il migliore.

Tutte le istruzioni devono terminare con ";".

`AUTO_INCREMENT`: quel campo **assumerà un valore crescente ogni volta che inserisco un record**. Se cancello un record quel numero viene perso. Solitamente usati per rappresentare degli *ID*. Tale record con questa caratteristica **deve essere anche chiave primaria**.

Esempio:

```sql
-- 1. Crea un database
CREATE DATABASE IF NOT EXISTS prova;

-- 2. Seleziona il database di prova
USE prova;

-- 3. Crea una tabella di esempio
CREATE TABLE country (
  country_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
  country VARCHAR(50) NOT NULL,
  last_update TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY  (country_id)
)ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE city (
  city_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
  city VARCHAR(50) NOT NULL,
  country_id SMALLINT UNSIGNED NOT NULL,
  last_update TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY  (city_id),
  KEY idx_fk_country_id (country_id),
  CONSTRAINT `fk_city_country` FOREIGN KEY (country_id) REFERENCES country (country_id) ON DELETE RESTRICT ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

1. Creo il database "prova" se non esiste;
2. seleziono prova come database "in uso";
3. ogni volta che inserisco un *contry* allora il campo aumenta.
   - `ON UPDATE CURRENT_TIMESTAMP` specifica che ogni volta che si fa update su record, il valore di `LAST_UPDATE` verrà aggiornato. `DEFAULT CHARSET = utf8` indica come i caratteri sono codificati. utf8 è migliore per simboli di alfabeti non inglesi.
   - `KEY idx_fk_country_id(country_id)` → se faccio ricerche su `country_id`, le ricerche

saranno più veloci perchè il database si crea un indice su quell'attributo.

```
[CONSTRAINT [symbol]] FOREIGN KEY
    [index_name] (col_name, ...)
    REFERENCES tbl_name (col_name,...)
    [ON DELETE reference_option]
    [ON UPDATE reference_option]

reference_option:
    RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT
```

`DESCRIBE nome`: mostra **informazioni** sui **campi contenuti** in una tabella, quindi mostra la definizione e come essa è composta (*se quindi definita da terzi*)

Insert

```
INSERT INTO table (field1, …, fieldN) SELECT … [ ON DUPLICATE KEY
UPDATE
        field1=value1,
        …
        fieldN=valueN ]
```

Select

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
    select_expr [, select_expr] ...
    [into_option]
    [FROM table_references
      [PARTITION partition_list]]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}, ... [WITH ROLLUP]]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [into_option]
    [FOR {UPDATE | SHARE}
        [OF tbl_name [, tbl_name] ...]
        [NOWAIT | SKIP LOCKED]
      | LOCK IN SHARE MODE]
    [into_option]

into_option: {
    INTO OUTFILE 'file_name'
        [CHARACTER SET charset_name]
        export_options
  | INTO DUMPFILE 'file_name'
  | INTO var_name [, var_name] ...
}
```

`PARTITION`: **partiziona le tabelle sulla base di alcuni attributi**.

`GROUP_BY .... WITH ROLLUP` serve aggiungere una riga fittizia che è la riga dei totali.

Per esempio:
Data una tabella con attributi *(Imp, mese, anno, salario)*

- In SQL dovrei fare

```
SELECT mese,anno, SUM(salario)
FROM ...
GROUP BY mese,anno with rollup -- restituisce tutti i valori del ragruppamento e
poi, per Anno
```

Restituisce **tutti i totali**, mettendo `NULL` sul mese, quindi:

ANNO	MESE	TOTALE_MESE
2020	gennaio	tot_gennaio
2020	febbraio	tot_febbraio
..	..	..
2020	NULL	TOT_2020
..	..	..
2021	NULL	TOT_2021

```
[LIMIT {[offset,] row_count | row_count OFFSET offset}]
[into_option]
```

`LIMIT`: limita il numero di record da restituire. Retituisci tot record a partire dal record iniziale
Esempio:
`LIMIT 10 offset 20`: dal risultato 20, restituisce 10 risultati, quindi fino al *29* della tabella.

`FOR {UPDATE | SHARE}`: gestisce i lock dei database e la concorrenza.

I valori NULL

- Il valore **NULL** per un campo assume il significato di «mancante, sconosciuto» ed esso è trattato diversamente dagli altri valori
- Per testare il valore di **NULL non si possono usare i consueti operatori di confronto** `=,<, o <>`
  - Esistono due operatori di confronto appositi per valori NULL: **IS NULL** e **IS NOT NULL**
- Quando si usa `ORDER BY` i valori **NULL** sono inseriti **all'inizio con ASC** ed **alla fine con DESC**.

Update

```
UPDATE [LOW_PRIORITY] [IGNORE] table_reference
    SET col_name1={expr1|DEFAULT} [, col_name2={expr2|DEFAULT}] ...
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]

Multiple-table syntax:

UPDATE [LOW_PRIORITY] [IGNORE] table_references
    SET col_name1={expr1|DEFAULT} [, col_name2={expr2|DEFAULT}] ...
    [WHERE where_condition]
```

- `SET`: specifichiamo quali colonne modificare e quali valori assegnare;
- `WHERE`: le condizioni che determinano quali righe saranno modificate;
- `ORDER BY`: per decidere in che ordine effettuare gli aggiornamenti;
- `LIMIT`: per indicare il numero massimo di righe da modificare.

Delete

```
Single-table syntax:

DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tbl_name
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]

Multiple-table syntax:

DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
    tbl_name[.*] [, tbl_name[.*]] ...
    FROM table_references
    [WHERE where_condition]

Or:

DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
    FROM tbl_name[.*] [, tbl_name[.*]] ...
    USING table_references
    [WHERE where_condition]
```

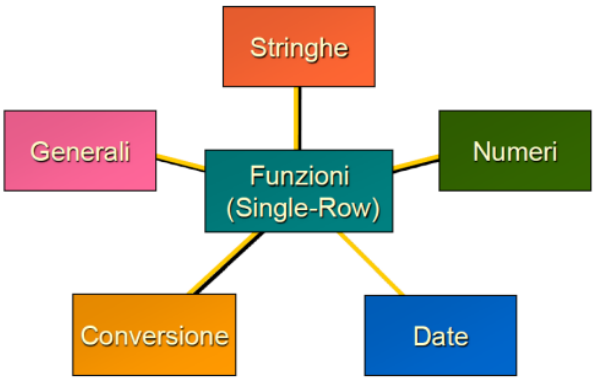
- `ORDER BY` e `LIMIT` funzionano come in `UPDATE`;
- `WHERE`: stabilisce le condizioni in base alle quali le righe verranno eliminate


```
Single-table syntax:

DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tbl_name
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]

Multiple-table syntax:

DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
    tbl_name[.*] [, tbl_name[.*]] ...
    FROM table_references
    [WHERE where_condition]

Or:

DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
    FROM tbl_name[.*] [, tbl_name[.*]] ...
    USING table_references
    [WHERE where_condition]
```

Funzioni e operatori

Le funzioni possono essere di 2 tipi:

- **single row**: agiscono solo su una riga
- **multiple raw** agiscono allo stesso tempo su più righe e danno risultati su piu righe e sono diverse da quelle di aggregazione che lavorano sulla group by



Gli operatori sono funzioni a tutti gli effetti.

Operatori

- Aritmetici:
  - "+" (**addizione**);
  - "-" (**sottrazione**);
  - "*" (**moltiplicazione**);
  - "/" (**divisione**);
  - "%" (**modulo**);

- Matematici:
  - **ABS**(X)
  - **FLOOR**(X)
  - **CEILING**(X)
  - **SIN**(X)
  - **COS**(X)
  - **LN**(X)
  - **LOG**(X)
  - **LOG**(B,X)

- Logici:
  - **NOT**(!);
  - **AND**(&&);
  - **OR**(||)
  - **XOR**;

- Confronto (Risultato 1 o 0):
  - =, <=> (NULL-safe);
  - <> , !=;
  - <= , < , >= ;
  - >;
  - **IS NULL**;
  - **IS NOT NULL**.

La funzione `<=> (NULL -safe);` serve per il confronto in cui il valore **NULL** viene considerato un valore a tutti gli effetti. Se ho 2 valori **NULL entrambi sono considerati come uguali**.

Operatori

- Per controllare se un numero è all'interno di un range di valori si può usare una delle seguenti espressioni:
  - *expr* **BETWEEN** *min* **AND** *max*
  - *expr* **NOT BETWEEN** *min* **AND** *max*
- Per confrontare rispetto ad una lista fissata di valori:
  - *expr* **IN** (*value, ...*)
  - *expr* **NOT IN** (*value, ...*)

- **COALESCE**(*val, ...*): restituisce il primo elemento non-NULL di una lista;
- **INTERVAL**(*N,N1,N2,N3,...*): Ritorna:
  - 0 se N < N1;
  - 1 se N < N2;
  - ecc...
  - -1 se N è NULL.

Controllo del flusso

`COALESCE`: ho un valore, se questo è NULL, allora voglio **restituire il primo argomento non nullo**.

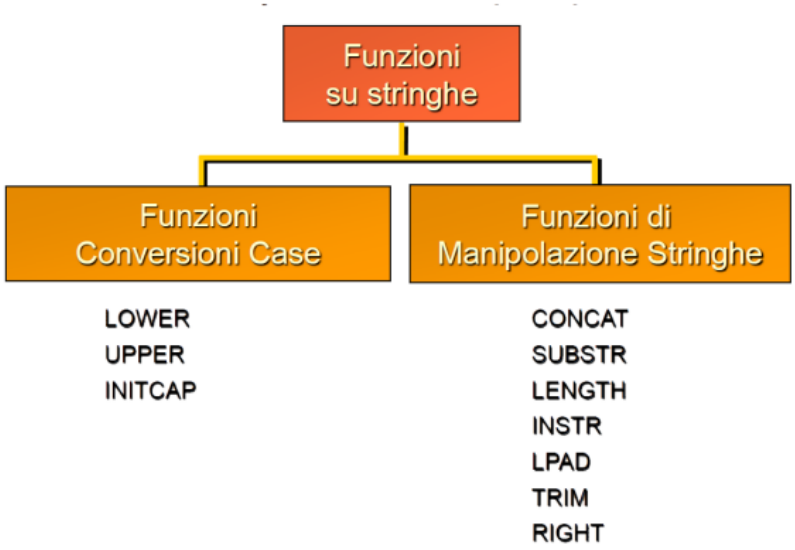
```
CASE value
    WHEN compare_value THEN result
   [WHEN compare_value THEN result ...]
   [ELSE result]
END

CASE
    WHEN condition THEN result
   [WHEN condition THEN result ...]
   [ELSE result]
END


IF(expr1,expr2,expr3)

IFNULL(expr1,expr2)


NULLIF(expr1,expr2)
```

Se `value` in `CASE` non matcha nessuno dei casi in WHEN allora viene eseguito l'istruzione `ELSE`.

`IF(expr1,expr2,expr3)` : se expr1 è vera, esegue expr2. se expr1 falsa, esegue expr3

```
SELECT film_id, title,
    CASE rating
        WHEN 'G' THEN 'General Audiences'
        WHEN 'PG' THEN 'Parental Guidance Suggested'
        WHEN 'PG-13' THEN 'Parents Strongly Cautioned'
        WHEN 'R' THEN 'Restricted'
        WHEN 'NC-17' THEN 'No one 17 and under admitted'
        ELSE 'There is nothing else'
    END AS ExplainedRating,
    IF(STRCMP(rating, 'NC-17')=0, 'YES', 'no') AS RequireID FROM film;
```

Funzioni su stringhe



- Funzione conversioni case:
  - **LOWER**(*str*) , **LCASE**(*str*)
  - **UPPER**(*str*) , **UCASE**(*str*)
- Funzioni manipolazione stringhe:
  - **ASCII**(*str*): ritorna il valore numerico del carattere più a sinistra di str;
  - **BIN**(*N*): Ritorna una stringa che rappresenta il valore binario di N;
  - **CONCAT_WS**(*separator,str1,str2,...*): il primo argomento è il separatore il resto gli argomenti;
  - **CONV**(*N,from_base,to_base*): converte i numeri tra differenti basi;
- **BIT_LENGHT**(*str*): Ritorna la lunghezza della stringa str in bit;
- **CHAR**(*N,...*): interpreta ogni argomento N come intero e ritorna una stringa consistente dei caratteri dati dal codice numerico degli interi;
- **CHAR_LENGTH**(*str*): ritorna la lunghezza della stringa misurata in caratteri;
- **CONCAT**(*str1,str2,...*): ritorna la stringa che si ottiene concatenando gli argomenti. Ritorna NULL se un argomento è NULL.

`BIT_LENGHT` lunghezza della stringa in bit. (essa dipende dalla rappresentazione dei caratteri su una specifica macchina. Quindi su pc diversi possono esserci valori diversi)
`CHAR_LENGTH` è la lunghezza stringa in caratteri.

Ricerca Full-Text:

- **MATCH** (*col1,col2,...*) **AGAINST** (*expr* [**IN BOOLEAN MODE** | **WITH QUERY EXPANSION**]):
  - **MATCH ... AGAINST** è utilizzata per ricerche full text:
    - ritorna la rilevanza tra il testo che si trova nelle colonne (col1,col2,...) e la query expr.
    - La similarità è un valore positivo in virgola mobile.
- La funzione **match** esegue una ricerca in linguaggio naturale per una stringa contro un testo che è rappresentato da una o più colonne incluse in un indice FULL TEXT.
- La ricerca di default è case-insensitive.

- Voglio cercare all'interno dei testi. Abbiamo visto la `LIKE` ma è pesante a livello computazionale.
- Vengono introdotti gli indici che permettono di fare ricerche nel testo
- `match` prende il testo della `against` e vede se è all'interno delle colonne specificate in modo approssimato.
- non è case insensitive, non vede le differenze fra maiuscole e minuscole.
- Se si usano `VARCHAR` è inutile e non si guadagna in termini di tempo ed è come la LIKE.

> In caso di text fino a 4gb conviene