22 MySQL: 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. Vedremo ora i principali, tenendo a mente un paio di regole generali:

- Un'espressione che contiene un valore *NULL* restituisce sempre *NULL* come risultato, salvo poche eccezioni.
- Fra il nome di una funzione e le parentesi che contengono i parametri non devono rimanere spazi. È possibile modificare questo comportamento con l'opzione *–sql-mode=IGNORE_SPACE*, ma in questo caso i nomi di funzione diventano parole riservate.

Quelle che stiamo per elencare, come detto, sono solo alcune delle funzioni disponibili: per una lista e una descrizione completa vi rimandiamo al manuale ufficiale.

(A) **Operatori aritmetici**:

- "+" (addizione)
- "-" (sottrazione)
- "*****" (moltiplicazione)
- "/" (divisione)
- "**%**" (modulo – resto della divisione)

Ricordate che una divisione per zero dà come risultato (e modulo) NULL.

(B) **Operatori di confronto**:

il risultato di un'espressione di confronto può essere "**1**" (vero), "**0**" (falso), o NULL.

Gli operatori sono:

- "=" (uguale)
- "< >" o "**!**=" (diverso)
- "<" (minore)
- ">" (maggiore)
- "<=" (minore o uguale)
- ">=" (maggiore o uguale)
- "<=>" (uguale *null-safe*)

Con quest'ultimo operando otteniamo il valore 1 se entrambi i valori sono null, e 0 se uno solo dei due lo è.

(C) **Operatori logici**:

- NOT
- AND
- OR
- XOR (OR esclusivo)

Come sinonimo di NOT possiamo usare "**!**";
Come sinonimo di AND possiamo usare "**&&**",
Come sinonimo di OR possiamo usare "||".

(E) Operatori **IS NULL / IS NOT NULL - BETWEEN / NOT BETWEEN - IN /NOT IN**

Abbiamo poi

- **IS NULL** e **IS NOT NULL** per verificare se un valore è (o non è) NULL;
- **BETWEEN** per test su valori compresi fra due estremi (inclusi) **NOT BETWEEN** per valori esterni ai due stremi;
- **IN** per verificare l'appartenenza di un valore ad una lista di valori dati, **NOT IN** per verificare la non appartenenza degli stessi.

Vediamo un 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;
```

Questa query estrae le righe di t1 in cui *a* è uguale a *b* ed è minore o uguale a *c*, *d* è uguale a 5 o a 8, *e* è compreso fra 7 e 9, *f* ha uno dei valori espressi fra parentesi e *g* non ha un valore NULL.

N.B. Si consiglia di utilizzare sempre le parentesi, invece di dover imparare a memoria il lungo elenco delle precedenze.

(F) Operatore **LIKE / NOT LIKE**

Molto importante è l'operatore **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'.

(G) Funzione **CAST**

La funzione **CAST** converte un dato in un tipo diverso da quello originale (ad esempio un numero in stringa). Il tipo di dato ottenuto può essere: DATE, DATETIME, TIME, DECIMAL, SIGNED INTEGER, UNSIGNED INTEGER, BINARY, CHAR. Con questi ultimi due può essere specificata anche la lunghezza richiesta.

SELECT CAST(*espressione* AS DATE)
-> converte in formato data
SELECT CAST(*espressione* AS BINARY(5))
-> converte in una stringa binaria di 5 byte

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.

REVERSE e REPLACE inverte una stringa e sostituisce in una stringa un carattere in un altro.

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 SUBSTR(‘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);
```

Funzioni su date e ore

Teniamo presente che le date vengono fornite da MySQL nel formato 'AAAA-MM-GG', mentre le ore sono nel formato 'HH:MM:SS'. Il timestamp è formato dalle due stringhe separate da uno spazio.

CURRENT_DATE e **CURRENT_TIME** restituiscono, rispettivamente, la data e l'ora attuali; **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.

Ci sono poi le funzioni che estraggono da una data, da un orario o da un timestamp le singole informazioni: **DAY** restituisce il giorno del mese, **DAYOFWEEK** il giorno della settimana (1 è la domenica e 7 il sabato), **DAYOFYEAR** il giorno dell'anno (da 1 a 366), **MONTH** il mese, **YEAR** l'anno; **HOUR**, **MINUTE** e **SECOND**, rispettivamente, ore, minuti e secondi.

In questi esempi ipotizziamo che in questo momento siano le 17.35.21 del 28 dicembre 2005:

SELECT NOW();
-> 2005-12-28 17:35:21
SELECT CURRENT_TIME();
-> 17:35:21
SELECT DATE(NOW()); (*equivale a CURRENT_DATE()*)
-> 2005-12-28
SELECT MONTH(NOW());
-> 12
SELECT DAYOFWEEK(NOW());
-> 4
SELECT MINUTE(NOW());
-> 35

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.

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('2006-01-09 08:59:15′,'%d/%m/%y %H.%i');
-> 09/01/06 08.59
SELECT DATE_FORMAT('2006-01-09 08:59:15′,'%W %e %b %Y – %k:%i');
-> Monday 9 Jan 2006 – 8:59

Funzioni di aggregazione

Le funzioni di aggregazione non lavorano, come quelle viste finora, su un solo dato, ma su insiemi di dati. Ciò significa che ci restituiranno un unico valore come "sintesi" di *n* valori. Normalmente i valori NULL vengono ignorati.

Ecco le principali di queste funzioni:

- **COUNT** conta i valori trovati
- **AVG** ne calcola la media
- **MIN** e **MAX** trovano il minore e il maggiore
- **SUM** effettua la somma

Esempi:

SELECT COUNT(*) FROM *tab1*
-> conta il numero di righe della tabella
SELECT COUNT(*colonna*) FROM *tab1*
-> conta i valori non NULL di colonna
SELECT COUNT(DISTINCT *colonna*) FROM *tab1*
-> conta i diversi valori (duplicati valgono per 1)
SELECT AVG(*colonna*) FROM *tab1*
-> calcola la media dei valori
SELECT AVG(DISTINCT *colonna*) FROM *tab1*
-> calcola la media conteggiando una sola volta i duplicati
SELECT MAX(*colonna*) FROM *tab1*
-> trova il valore massimo
SELECT SUM(*colonna*) FROM *tab1*
-> calcola la somma

Le funzioni di aggregazione vengono usate spesso in combinazione con la clausola GROUP BY (vedere lez.14)).
In questi casi le aggregazioni non vengono svolte su tutte le righe estratte dalla WHERE, ma singolarmente su ogni gruppo di righe formato dalla GROUP BY. In questo caso è possibile aggiungere alla query la clausola **WITH ROLLUP** che produce subtotali ad ogni cambiamento di valori dei campi raggruppati.

SELECT anno, paese, prodotto, SUM(profitti)
FROM vendite GROUP BY anno, paese, prodotto WITH ROLLUP

Questa query genererà totali suddivisi per prodotto, per paese e per anno, ma fornirà anche i totali per paese, per anno e infine un totale generale. Nella tabella risultato le colonne su cui vengono calcolati i totali generali sono mostrate con valore NULL (ad esempio nella riga finale con il totale generale saranno a NULL i valori di anno, paese e prodotto).

Lo standard SQL vuole che in una SELECT che contiene funzioni di aggregazione tutte le colonne su cui tali funzioni non lavorano siano comprese nella GROUP BY. Ad esempio questa query non sarebbe valida, perchè manca la GROUP BY sul campo nome:

```
SELECT ordini.idCliente, clienti.nome, MAX(pagamenti)
FROM ordini,clienti
WHERE ordini.idCliente = clienti.id
GROUP BY ordini.idCliente
```

La query ha tuttavia un senso logico, perchè il valore di nome, dipendendo dalla join, sarà sempre lo stesso in ogni gruppo di righe con lo stesso 'idCliente': di conseguenza è superfluo inserirlo nella GROUP BY. MySQL quindi ci consente di usare questa sintassi: bisogna però fare attenzione a non omettere la GROUP BY su un campo che non ha un valore unico, in quanto ne otterremmo un risultato non prevedibile.

Ricerche full-text

Le ricerche full-text sono ricerche basate sul "linguaggio naturale", effettuabile su campi di tipo CHAR, VARCHAR e TEXT. Si tratta del tipo di interrogazione che normalmente facciamo quando utilizziamo un motore di ricerca. Per poterla effettuare è necessario che le colonne interessate facciano parte di un indice FULLTEXT (vedi lez.12) e che la tabella sia di tipo MyISAM.

Le ricerche vengono effettuate con la funzione **MATCH**:

```
SELECT * FROM articoli
WHERE MATCH(titolo,testo) AGAINST('database')
```

Questa query effettua la ricerca del termine 'database' sulle colonne titolo e testo della tabella, ed estrae quelle che hanno una qualche rilevanza. Le due colonne devono far parte dello stesso indice FULLTEXT.

```
SELECT id, titolo, MATCH(titolo,testo) AGAINST('database') as rilevanza
FROM articoli
WHERE MATCH(titolo,testo) AGAINST('database')
```

In questo caso, oltre all'id e al titolo ci verrà fornito, nella colonna 'rilevanza', anche il risultato della funzione MATCH, che è un numero compreso fra 0 e 1.

Segnaliamo anche la funzione **FOUND_ROWS**, da utilizzare per ottenere il numero di righe che sarebbero state trovate da una SELECT in cui abbiamo usato la LIMIT, se tale clausola non fosse stata presente:

```
SELECT SQL_CALC_FOUND_ROWS * FROM tabella LIMIT 10;
SELECT FOUND_ROWS();
```

Nella prima istruzione selezioniamo il contenuto di una tabella limitandolo alle prime 10 righe; tuttavia l'uso della clausola **SQL_CALC_FOUND_ROWS** indica a MySQL che vogliamo sapere quante sarebbero le righe estratte senza la LIMIT; otteniamo poi tale valore con la seconda istruzione, che **deve** essere immediatamente successiva.

Funzioni condizionali in MySQL: CASE, IF, IFNULL e NULLIF

Il DBMS MYSQL supporta, a dire il vero, diverse funzioni per gestire le istruzioni condizionali che prendono il nome di *Control Flow Functions* (letteralmente: "funzioni per il controllo del flusso"). Queste funzioni sono:

- CASE
- IF()
- IFNULL()
- NULLIF()

Vediamole nel dettaglio.

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. Ad esempio:

```
SELECT CASE mese
        WHEN 1 THEN 'Gennaio'
        WHEN 2 THEN 'Febbraio'
        WHEN 3 THEN 'Marzo'
        WHEN 4 THEN 'Aprile'
        WHEN 5 THEN 'Maggio'
        WHEN 6 THEN 'Giugno'
        WHEN 7 THEN 'Luglio'
        WHEN 8 THEN 'Agosto'
        WHEN 9 THEN 'Settembre'
        WHEN 10 THEN 'Ottobre'
        WHEN 11 THEN 'Novembre'
        WHEN 12 THEN 'Dicembre'
        END
FROM calendario
ORDER BY mese ASC;
```

Oppure:

```
SELECT CASE nazione
        WHEN 'IT' THEN 'Italia'
        ELSE 'Estero'
        END
FROM utenti;
```

La funzione 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.

```
IF (Espressione,Vero,Falso)
```

Vediamo un esempio:

```
SELECT nome, cognome, IF (nazione = 'IT','Italia','Estero') AS paese
FROM utenti;
```

La funzione 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.

Facciamo un esempio:

```
SELECT IFNULL(10/0,'Non posso dividere per zero!');
-> Non posso dividere per zero!
```

La funzione NULLIF()

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

Vediamo un esempio:

```
SELECT NULLIF(1,1);
-> NULL
SELECT NULLIF(1,2);
-> 1
```