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.

Per cominciare, i classici **operatori aritmetici**:

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

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

Passiamo agli **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.

Abbiamo quindi i classici **operatori logici**:

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

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

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); **IN**
per verificare l'appartenenza di un valore ad una lista di valori dati.

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.

Come avete visto abbiamo usato le parentesi per indicare che l'espressione "d=5 or d=8" deve essere valutata prima delle altre. Il consiglio è di utilizzarle sempre in questi casi, invece di imparare a memoria il lungo elenco delle precedenze.

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'.

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

È anche possibile utilizzare l'operatore **BINARY** come sintassi veloce per considerare la stringa seguente come binaria; questo fa sì che eventuali confronti vengano fatti byte per byte e non carattere per carattere, rendendo sempre significativa la differenza fra maiuscole e minuscole, così come gli spazi in fondo alle stringhe.

SELECT 'a' = 'A'
SELECT BINARY 'a' = 'A'

Il primo di questi due test sarà vero, il secondo sarà falso. "BINARY 'a'" equivale a "CAST('a' AS BINARY)".

Esiste poi una funzione **CONVERT (... USING ...)** utile per convertire una stringa fra diversi character set (vedere lez.10). Ad esempio:

SELECT CONVERT('abc' USING utf8)

Restituisce la stringa 'abc' nel set di caratteri utf8. Attenzione: esiste un'altra sintassi della funzione CONVERT, che però è un sinonimo di CAST: ad esempio CONVERT(*espressione*,DATE) corrisponde al primo esempio visto in precedenza su CAST.

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*)

- IFNULL(*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.

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.

FIND_IN_SET, infine, è una funzione particolarmente
utile con i campi di tipo SET, per verificare se un dato valore è attivo.

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
```

SELECT * FROM *tabella* WHERE FIND_IN_SET('Primo',*col1*)
> 0
-> (restituisce le righe in cui il valore 'Primo' è attivo nella colonna col1, ipotizzando che si tratti di una colonna di tipo SET)

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);