

[Home Page](#) [Novità](#) [Aiuto](#)

Il linguaggio SQL

La clausola WHERE

http://www.vbsimple.net/database/db_08_02.htm

Difficoltà: 3 / 5

Sebbene si tratti di una clausola anziché di un'istruzione ed essendo utilizzata da quasi tutte le altre istruzioni si è preferito trattarla a parte. La sua unica finalità è quella di limitare le selezioni ed includere solo le righe che corrispondono ai criteri specificati. Il suo utilizzo generale è:

```
WHERE <campo> <predicato> <valori>
```

Campo naturalmente si riferisce al nome di un campo presente in una delle origini, predicato si riferisce ai simboli che stabiliscono una regola ai valori specificati di seguito. Il predicato si riferisce al contenuto del campo e può essere uno dei seguenti:

=	è uguale al valore
<	è minore del valore
>	è maggiore del valore
<=	è minore oppure uguale al valore
>=	è maggiore oppure uguale al valore
<>	è differente dal valore
LIKE	è simile al valore
NOT LIKE	non è simile al valore
BETWEEN	è compreso tra i due valori specificato a seguito
NOT BETWEEN	non è compreso tra i due valori
IS NULL	è il valore NULL
IS NOT NULL	non è il valore NULL

Alcuni database consentono l'uso di ulteriori predicati:

!=	non è uguale (cioè è differente) al valore
!<	non è minore (cioè è maggiore o uguale) del valore
!>	non è maggiore (cioè è minore o uguale) del valore
IN	corrisponde ad uno dei valori specificati a seguito
NOT IN	non corrisponde ad alcuno dei valori specificati a seguito

Prima di vedere il suo utilizzo è necessario chiarire tre semplici concetti:

1. I campi numerici vanno testati con valori numerici come 1, 2, 3 o 40000.
2. I campi di testo vanno verificati con valori di testo, quindi stringhe racchiuse dal carattere di apostrofo come 'pippo', 'ciccio' o 'casa mia'. La regola comune stabilisce che le ricerche su campi di testo sono effettuate ignorando le differenze tra maiuscolo e minuscolo (*case insensitive*).
3. Gli altri tipi di campo (data, caratteri unicode, o altro genere) vanno verificati con i simboli decisi dal singolo driver del database.

L'utilizzo dell'operatore WHERE è relativamente semplice:

```
SELECT nome, cognome FROM dipendenti WHERE id > 2
```

Restituirà due campi di quelle righe in cui il campo ID contiene un valore superiore a 2. Come si nota, il campo specificato nell'operatore WHERE non deve necessariamente essere presente nell'elenco dei campi restituiti.

```
SELECT nome, cognome FROM dipendenti WHERE id BETWEEN 2 AND 4
```

Restituirà i due campi di quelle righe in cui il campo ID contiene un valore compreso tra 2 e 4.

```
SELECT nome, cognome FROM dipendenti WHERE ruolo LIKE 'direttore%'
```

Restituirà quelle righe in cui il valore del campo RUOLO inizia per DIRETTORE. Il carattere di percentuale % corrisponde al [carattere jolly](#) standard che indica qualsiasi corrispondenza al suo posto. Il carattere % può essere utilizzato soltanto come prefisso o come suffisso o entrambe le cose; non va invece utilizzato al centro di una frase:

1. `SELECT nome, cognome FROM dipendenti WHERE ruolo LIKE 'direttore%'`
2. `SELECT nome, cognome FROM dipendenti WHERE ruolo LIKE '%direttore'`
3. `SELECT nome, cognome FROM dipendenti WHERE ruolo LIKE '%direttore%'`

Le tre istruzioni restituiscono rispettivamente le righe in cui il campo RUOLO:

1. inizia per DIRETTORE, e cioè è seguito da qualsiasi altra combinazione di caratteri;
2. termina per DIRETTORE, e cioè è preceduto da qualsiasi combinazione di caratteri;
3. include il termine DIRETTORE, e cioè è preceduto o seguito da qualsiasi altro carattere.

Esiste un altro carattere jolly che indica invece un singolo carattere, è rappresentato dal simbolo `_` e può essere utilizzato sia come prefisso, come suffisso che inframezzato al testo:

1. `SELECT nome, cognome FROM dipendenti WHERE nome LIKE 'francesc_'`
2. `SELECT nome, cognome FROM dipendenti WHERE nome LIKE '_ino'`
3. `SELECT nome, cognome FROM dipendenti WHERE nome LIKE 'gianpier_ ca_ill%'`
4. `SELECT nome, cognome FROM dipendenti WHERE nome LIKE '_____'`

Restituiranno rispettivamente le righe in cui il campo NOME:

1. inizia per FRANCESC, quindi include sia Francesco che Francesca, ma non Franceschino;
2. termina per INO, quindi include Gino, Pino e Lino, ma non Paperino;
3. inizia per **GIAMPIER**, seguito da un carattere, da uno **spazio**, da **CA**, un altro carattere qualsiasi, da **ILL** e qualsiasi altra combinazione; include quindi Gianpiero Camilli, Gianpiera Carillo o Gianpiero Cavillosi.
4. contiene esattamente 5 caratteri qualsiasi, non quattro né sei né otto.

È fondamentale capire la differenza tra i due caratteri jolly:

- il simbolo `_` indica un solo carattere qualsiasi;
- il simbolo `%` indica qualsiasi combinazione di qualsiasi carattere.



Alcune versioni del driver per il **database Access** utilizzano i caratteri jolly standard del DOS, cioè ? al posto di _ per indicare un carattere qualunque e * al posto di % per indicare una combinazione composta da qualsiasi carattere.

Abbiamo sinora visto solo selezioni formate da un'unica condizione di vincolo ma naturalmente è possibile combinare più condizioni utilizzando gli appositi operatori booleani:

```
AND      Specifica che il criterio precedente e quello successivo debbano
restituire risultato Vero
OR       Specifica che almeno uno dei due criteri debba restituire risultato
Vero
```

Altresì ogni condizione può essere invertita utilizzando l'operatore NOT. Vediamone il funzionamento pratico:

```
SELECT * FROM dipendenti WHERE nome='francesco' AND ruolo='direttore'
```

Restituirà tutti i campi della tabella dipendenti in cui il nome dell'impiegato è Francesco ed il suo ruolo è direttore. Sono esclusi tutti gli altri Francesco, tutti gli altri direttori e tutti quei dipendenti che presentano nome o ruolo differenti.

```
SELECT * FROM dipendenti WHERE id=0 OR id > 10
```

Restituirà tutti i campi della tabella dipendenti in cui il campo ID contiene valore 0 oppure un valore superiore a 10. Sono quindi esclusi tutti quelli che vanno da 1 a 9 (supposto naturalmente che ID sia un campo intero). Complicando un attimo le cose otteniamo:

```
SELECT * FROM dipendenti WHERE id=0 OR id=1 OR id > 10
```

Restituirà tutte le righe in cui il campo ID sia uguale a 0 oppure a 1 oppure sia maggiore di 10.

Nel caso in cui appaiano i due operatori AND e OR nella stessa condizione, l'operatore AND avrà la precedenza su OR:

```
SELECT * FROM dipendenti WHERE id=0 OR id=5 AND id BETWEEN 4 AND 10
```

Restituirà quelle righe in cui il valore ID è 0 oppure è 5 e compreso tra 4 e 10. Se nella tabella dipendenti avessimo 20 righe, con ID da 0 a 19, la selezione sopra **riporterebbe** due sole righe, **quelle con ID 0 e ID 5**. Questo perché l'interprete SQL legge il testo così:

```
SELECT * FROM dipendenti WHERE
id=0 OR
(id=5 AND id BETWEEN 4 AND 10)
```

che viene quindi calcolata in due parti: è estratto il valore 5, è posto a paragone con l'ultima condizione che estrae le righe con ID compresi tra 4 e 10. Queste due condizioni separate restituiscono soltanto una riga, cioè la 5. I valori 4, 6, 7, 8, 9 e 10 sono scartati perché esclusi dall'operatore AND. Alla fine dell'espressione sono quindi restituite le righe

con ID 0 e 5. Volendo forzare invece l'altro ragionamento è possibile riscrivere il codice come segue:

```
SELECT * FROM dipendenti WHERE  
(id=0 OR id=5) AND  
id BETWEEN 4 AND 10
```

Così facendo **sarà estratta soltanto la riga con ID 5** perché la prima valutazione estrarrà le due righe con ID 0 e 5 e queste saranno poste a confronto con la condizione per cui ID sia un numero compreso tra 4 e 10. Dei due numeri estratti in precedenza soltanto l'ultimo ricade in questa ipotesi.

Riprendendo il discorso [fatto in precedenza](#) riguardo i valori [Null](#), è possibile estrarre le righe in cui un determinato campo sia Null usando l'operatore IS NULL come mostrato:

```
SELECT * FROM dipendenti WHERE id IS NULL
```

Restituirà soltanto quelle righe in cui il campo ID sia Null. Viceversa:

```
SELECT * FROM dipendenti WHERE id IS NOT NULL
```

Riporterà tutte quelle righe in cui ID non sia Null.

```
SELECT * FROM dipendenti WHERE id=0 OR id<>0
```

Per assurdo, l'ultima interrogazione produce lo stesso risultato di quella basata su IS NOT NULL, perchè lo ricordiamo, tutti i test matematici su un valore Null restituiscono valore Null, che non corrisponde quindi né a 0 né ad un valore differente da 0. In maniera analoga tutte le altre operazioni (addizione, moltiplicazione, etc..) su valori Null continuano a restituire il valore di origine.

Volendo fare un paragone con un concetto a noi già noto, si tratta di un valore che corrisponde ad un infinito senza segno, verso il quale qualsiasi operazione restituisce infinito.

Abbiamo già detto che alcuni database supportano anche il **predicato IN** ed il suo contrario **NOT IN**; questi possono essere usati in due maniere:

```
SELECT * FROM dipendenti WHERE id IN (1, 3, 5, 7, 9)
```

Interrogazione restituirà tutte quelle righe in cui il campo ID contiene i valori 1 o 3 o 5 o 7 o 9. Corrisponde esattamente a:

```
SELECT * FROM dipendenti WHERE id=1 OR id=3 OR id=5 OR id=7 OR id=9
```

La sua reale utilità si vede quando è impiegato in analisi composte da più di un'interrogazione:

```
SELECT nome, cognome FROM dipendenti WHERE id IN (SELECT id FROM saldi WHERE
totale > 10000)
```

Restituirà nome e cognome dalla tabella dipendenti di quelle righe il cui ID sia incluso nella seconda interrogazione, che estrarrà gli ID della tabella saldi in cui il campo totale è maggiore di 10000. Questa doppia interrogazione produrrà quindi il nome ed il cognome di quelle persone con un saldo superiore a 10000.

Si tratta di un'istruzione molto comoda che a volte può semplificare l'uso due tabelle contemporaneamente, ponendo infatti due interrogazioni separate, anziché una sola contenente due tabelle, e quindi più complessa. Purtroppo non è supportato da tutti i database.

Abbiamo parlato di più tabelle in una singola interrogazione? Abbiamo già visto come funziona la cosa e gli effetti che produce, cioè la ripetizione di tutti i valori di ogni singola tabella con qualsiasi altro valore dell'altra tabella. Vediamo quindi come semplificare e rendere utile queste relazioni tra più tabelle.

Immaginiamo di avere la solita tabella dipendenti composta dai campi ID, Nome, Cognome e Ruolo; immaginiamo anche la presenza di una seconda tabella di nome saldi con i campi ID e Totale, che abbiamo già visto nell'esempio precedente. Riproduciamo l'esempio che estraeva nome e cognome delle persone con un saldo totale maggiore di 10000, utilizzando un'unica interrogazione basata su due tabelle, anziché l'operatore IN che non tutti i database supportano. In questa interrogazione aggiungeremo anche il campo con il valore totale della tabella saldi, operazione impossibile con l'esempio precedente.

```
SELECT dip.id, dip.nome, dip.cognome, s.totale
FROM dipendenti dip, saldi s
WHERE an.id=s.id
AND s.totale > 10000
```

Dovendo mettere più tabelle in relazione, ed avendo più di una tabella il campo di nome ID abbiamo preferito includere anche i prefissi delle tabelle, rinominandoli per comodità **dip** per dipendenti e **s** per saldi. Abbiamo già visto questa tecnica nell'articolo precedente.

L'unione delle due tabelle produrrà la duplicazione di tutti i valori della prima tabella con quelli della seconda. Per escludere i risultati che non sono reali basterà unire i due campi chiave id in un vincolo: **an.id=s.id**, cioè tutti i valori del campo ID devono essere uguali ai corrispondenti nella tabella saldi. La seconda delle condizioni filtrerà semplicemente le righe ottenute in cui il totale non sia maggiore di 10000.

Nel gergo tecnico dei database il campo ID della tabella dipendenti costituisce la chiave primaria (Primary Key o semplicemente PK), mentre il campo ID della tabella saldi costituisce la chiave esterna (Foreign Key oppure FK). La relazione formata è una semplice relazione uno a uno. Abbiamo già trattato questi argomenti nell'[introduzione ai database](#).

La stessa operazione di relazione di due tabelle può essere effettuata utilizzando l'[operatore JOIN](#), nelle sue numerose forme, in grado di superare alcune limitazioni che questa

implementazione possiede.

[Eibia FBI](#)

14 Marzo 2004



[Torna all'indice della sezione Database](#)
