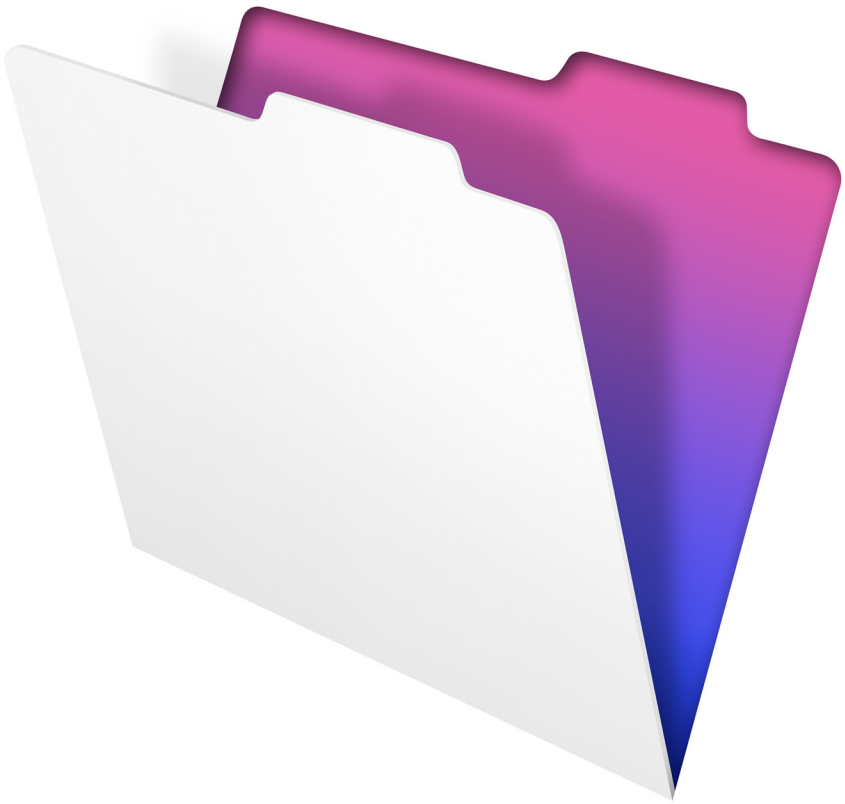
FileMaker® 13

Guida SQL



FileMaker, Inc.
5201 Patrick Henry Drive
Santa Clara, California 95054 Stati Uniti

Per ulteriori informazioni, visitare il nostro sito Web all'indirizzo http://www.filemaker.com/it/.

Edizione: 01

Indice

Capitolo 1
Introduzione

Come sviluppatore di database, è possibile utilizzare FileMaker Pro per creare le soluzioni di database senza specifiche conoscenze di SQL. Ma se si dispone di qualche conoscenza di SQL, è possibile utilizzare un file di database FileMaker come un'origine dati ODBC o JDBC, condividendo i dati con altre applicazioni che fanno uso di ODBC e JDBC. È anche possibile utilizzare la funzione EseguiSQL di FileMaker Pro per recuperare i dati dalle ricorrenze di tabella in un database FileMaker Pro.

Questo riferimento descrive le istruzioni SQL e gli standard supportati da FileMaker. I driver client ODBC e JDBC di FileMaker supportano tutte le istruzioni SQL descritte in questo riferimento. La funzione EseguiSQL di FileMaker Pro supporta solo l'istruzione SELECT.

Informazioni su questa guida

- Per informazioni su come utilizzare ODBC e JDBC con le versioni precedenti di FileMaker Pro, vedere http://www.filemaker.com/it/support/.
- Questa guida presuppone che si abbiano le conoscenze di base sull'uso delle funzioni di FileMaker Pro, sulla codifica delle applicazioni ODBC e JDBC e sulla costruzione di query SQL. Per informazioni su questi argomenti consultare il materiale di riferimento relativo.
- In questa guida il termine "FileMaker Pro" indica sia FileMaker Pro sia FileMaker Pro Advanced, a meno che non vengano descritte caratteristiche specifiche di FileMaker Pro Advanced.

Dove reperire la documentazione PDF

Per accedere alla documentazione PDF di FileMaker:

- In FileMaker Pro, selezionare il menu **Aiuto** > **Documentazione prodotto.**
- In FileMaker Server, selezionare il menu **Aiuto** > **Documentazione prodotto.**
- Per ulteriori informazioni, accedere al sito http://www.filemaker.com/it/support/. Eventuali aggiornamenti a questo documento sono disponibili anche sul sito web.

Informazioni su SQL

SQL o Structured Query Language, è un linguaggio di programmazione progettato per effettuare query sui dati di un database relazionale. L'istruzione primaria utilizzata per una query su a database è SELECT.

Oltre alla lingua per effettuare le query su un database, SQL fornisce istruzioni per la manipolazione dei dati, che consentono di aggiungere, aggiornare ed eliminare i dati.

SQL fornisce anche le istruzioni per eseguire la definizione dei dati. Queste istruzioni consentono di creare e modificare tabelle e indici.

Le istruzioni SQL e gli standard supportati da FileMaker sono descritti in capitolo 2, “Standard supportati.”

Utilizzo di un database FileMaker come origine dati

Quando si ospita un database FileMaker come un'origine dati ODBC o JDBC, i dati FileMaker possono essere condivisi con le applicazioni compatibili con ODBC e JDBC. Le applicazioni collegano all'origine dati FileMaker utilizzando i driver client FileMaker, costruiscono ed eseguono le query SQL utilizzando ODBC o JDBC ed elaborano i dati recuperati dalla soluzione di database FileMaker.

Vedere *Guida ODBC e JDBC di FileMaker* per informazioni estese su come è possibile utilizzare il software FileMaker come origine dati per applicazioni ODBC e JDBC.

I driver client ODBC e JDBC di FileMaker supportano tutte le istruzioni SQL descritte in questo riferimento.

Utilizzo della funzione EseguiSQL

La funzione EseguiSQL di FileMaker Pro permette di recuperare i dati dalle ricorrenze di tabella indicate nel grafico delle relazioni ma indipendenti da eventuali relazioni definite. È possibile recuperare dati da più tabelle senza creare giunzioni tabella o relazioni tra le tabelle. In alcuni casi, è possibile ridurre la complessità del grafico delle relazioni utilizzando la funzione EseguiSQL.

I campi in cui si esegue la query con la funzione EseguiSQL non devono essere su un formato specifico, pertanto è possibile utilizzare la funzione EseguiSQL per recuperare i dati indipendentemente dal contesto del formato. A causa di questa indipendenza dal contesto, l'uso della funzione EseguiSQL negli script può migliorare la portabilità degli script. È possibile utilizzare la funzione EseguiSQL ovunque sia possibile specificare dei calcoli, anche per grafici e per la creazione di resoconti.

La funzione EseguiSQL supporta solo l'istruzione SELECT, descritta nella sezione “Istruzione SELECT” a pagina 7.

Inoltre la funzione EseguiSQL accetta soltanto la sintassi SQL-92 con data e ora in formato ISO, senza parentesi ({}). La funzione EseguiSQL non accetta le costanti di data, ora e indicatore data e ora in formato ODBC/JDBC tra parentesi.

Per informazioni sulla sintassi e l'utilizzo della funzione EseguiSQL, consultare la Guida di FileMaker Pro.

Capitolo 2
Standard supportati

Questo riferimento descrive le istruzioni e i costrutti SQL supportati da FileMaker. I driver client ODBC e JDBC di FileMaker supportano tutte le istruzioni SQL descritte in questo capitolo. La funzione EseguiSQL di FileMaker Pro supporta solo l'istruzione SELECT.

Utilizzare i driver client per accedere a una soluzione di database FileMaker da un'applicazione compatibile con ODBC o JDBC. La soluzione di database FileMaker può essere ospitata da FileMaker Pro o da FileMaker Server.

- Il driver client ODBC supporta ODBClivello 1 3.5 con alcune funzioni di livello 2.
- Il driver client JDBC fornisce supporto parziale per la specifica JDBC 3.0.
- I driver client ODBC e JDBC entrambi supportano la conformità SQL-92 entry-level, con alcune funzioni intermedie SQL-92.

Supporto dei caratteri Unicode

I driver client ODBC e JDBC supportano le API Unicode. Tuttavia, se si sta creando un'applicazione personalizzata che usa i driver client, usare ASCII per i nomi dei campi, i nomi delle tabelle e i nomi dei file (in caso venissero utilizzati strumenti o applicazioni di query diversi da Unicode).

Nota Per inserire e recuperare i dati Unicode, utilizzare `SQL_C_WCHAR`.

Istruzioni SQL

I driver client ODBC e JDBC supportano le seguenti istruzioni SQL:

- SELECT(pagina 7)
- DELETE(pagina 16)
- INSERT(pagina 16)
- UPDATE(pagina 18)
- CREATE TABLE(pagina 19)
- ALTER TABLE(pagina 20)
- CREATE INDEX(pagina 21)
- DROP INDEX(pagina 21)

I driver client supportano anche la mappatura dei dati di tipo FileMaker su dati di tipo ODBC SQL e JDBC SQL. Vedere la *Guida ODBC e JDBC di FileMaker* per conversioni del tipo di dati. Per ulteriori informazioni sulla creazione di query SQL, consultare un manuale di terze parti.

Nota I driver client ODBC e JDBC non supportano i portali FileMaker .

Istruzione SELECT

Utilizzare l'istruzione `SELECT` per specificare le colonne richieste. Far seguire l'istruzione `SELECT` dalle espressioni di colonna (simili ai nomi di campo) che si desidera recuperare, ad esempio `cognome`). Le espressioni possono includere operazioni matematiche o manipolazioni di stringhe, ad esempio `STIPENDIO * 1.05`.

L'istruzione `SELECT` può utilizzare varie clausole:

```
SELECT [DISTINCT] {* | espressione_colonna [[AS] alias_colonna],...}
FROM nome_tabella [alias_tabella], ...
[ WHERE espr1 operatore_rel espr2 ]
[ GROUP BY {espressione_colonna, ...} ]
[ HAVING espr1 operatore_rel espr2 ]
[ UNION [ALL] (SELECT..). ]
[ ORDER BY {espressione_ordinamento [DESC| ASC]}, ... ]
[ OFFSET n {ROWS| ROW} ]
[ FETCH FIRST [ n [ PERCENT] ] { ROWS| ROW } {ONLY| WITH TIES } ]
[ FOR UPDATE [di {espressione_colonna, ...}] ]
```

Gli elementi racchiusi tra parentesi sono facoltativi.

`alias_colonna` può essere utilizzato per assegnare alla colonna un nome più descrittivo, o per abbreviare il nome di una colonna più lunga. Ad esempio per assegnare l'alias `settore` alla colonna `sett`:

```
SELECT sett AS settore FROM dip
```

Davanti ai nomi dei campi possono essere aggiunti il nome della tabella o l'alias della tabella. Ad esempio, `DIP.COGNOME` o `D.COGNOME`, dove `D` è l'alias della tabella `DIP`.

L'operatore `DISTINCT` può precedere la prima espressione di colonna. Questo operatore elimina le righe doppie dal risultato di una query. Ad esempio:

```
SELECT DISTINCT sett FROM dip
```

clausole SQL

I driver client ODBC e JDBC supportano le seguenti clausole SQL.

Utilizzare questa clausola SQL	Per
FROM (pagina 8)	Indicare le tabelle utilizzate nell'istruzione `SELECT`.
WHERE (pagina 9)	Specificare le condizioni che i record devono soddisfare per essere recuperati, come nel caso di una richiesta di ricerca di FileMaker Pro.
GROUP BY (pagina 10)	Specificare i nomi di uno o più campi in base a cui raggruppare i valori restituiti. Questa clausola viene utilizzata per restituire una serie di valori aggregati tramite la restituzione di una riga per ciascun gruppo, come nel caso di un riassunto parziale di FileMaker Pro.
HAVING (pagina 10)	Specificare le condizioni per i gruppi di record (ad esempio per visualizzare solo i settori per cui l'importo complessivo degli stipendi è superiore a 200.000 Euro).
UNION (pagina 11)	Combinare i risultati di due o più istruzioni `SELECT`.
ORDER BY (pagina 11)	Specificare l'ordinamento dei record.
OFFSET (pagina 12)	Stabilire il numero di righe da saltare prima di iniziare a recuperare le righe.
FETCH FIRST (pagina 12)	Specificare il numero delle righe che deve essere recuperato. Non viene restituito un numero di righe oltre quello specificato sebbene possano essere restituite meno righe se la query genera un numero di righe inferiore a quello specificato.
FOR UPDATE (pagina 13)	Eseguire gli aggiornamenti o le eliminazioni nella posizione attraverso i cursori SQL

Nota Se si tenta di recuperare i dati da una tabella senza colonne, l'istruzione `SELECT` non restituisce nulla.

Clausola FROM

La clausola `FROM` indica le tabelle che vengono utilizzate nell'istruzione `SELECT`. Il formato è:

```
FROM nome_tabella [alias_tabella] [, nome_tabella [alias_tabella]]
```

`nome_tabella` è il nome di una tabella nel database corrente. Il nome tabella deve iniziare con un carattere alfabetico. Se il nome tabella non inizia con un carattere alfabetico, racchiuderlo nelle virgolette doppie (identificativo quotato).

`alias_tabella` può essere utilizzato per assegnare alla tabella un nome più descrittivo, per abbreviare un nome di tabella lungo o per includere la stessa tabella nella query più di una volta (ad esempio, in auto-collegamenti).

I nomi campo iniziano con un carattere alfabetico. Se il nome del campo non inizia con un carattere alfabetico, racchiuderlo nelle virgolette doppie (identificativo quotato). Ad esempio, l'istruzione `EseguiSQL` per il campo `_COGNOME` è:

```
SELECT"_LASTNAME" FROM dip
```

Davanti ai nomi dei campi possono essere aggiunti il nome della tabella o l'alias della tabella. Ad esempio, nel caso di una tabella `FROM dipendente D`, è possibile indicare il campo `COGNOME` come `D.COGNOME`. Gli alias di tabella devono essere utilizzati se l'istruzione `SELECT` unisce una tabella a se stessa. Ad esempio:

```
SELECT * FROM dipendente D, dipendente F WHERE D.id_manager =
F.id_dipendente
```

Il segno uguale (=) comprende soltanto le righe corrispondenti nei risultati.

Per unire più di una tabella ed eliminare tutte le righe che non hanno righe corrispondenti in entrambe le tabelle di origine, è possibile usare `INNER JOIN`. Ad esempio:

```
SELECT*
   FROM Venditori INNER JOIN Dati_Vendite
   ON Venditori.ID_Venditore = Dati_Vendite.ID_Venditore
```

Per unire due tabelle senza eliminare righe nella prima tabella (quella "a sinistra"), è possibile utilizzare `LEFT OUTER JOIN`.

```
SELECT*
   FROM Venditori LEFT OUTER JOIN Dati_Vendite
   ON Venditori.ID_Venditore = Dati_Vendite.ID_Venditore
```

Ogni riga della tabella "Venditori" viene visualizzata nella tabella unita.

Note

- `RIGHT OUTER JOIN` non è supportato attualmente.
- `FULL OUTER JOIN` non è supportato attualmente.

Clausola WHERE

La clausola `WHERE` specifica le condizioni che i record devono soddisfare per essere recuperati. La clausola `WHERE` contiene le condizioni nella forma:

```
WHERE espr1 operatore_rel espr2
```

`espr1` e `espr2` possono essere nomi di campo, valori costanti o espressioni.

`operatore_rel` è l'operatore relazionale che collega le due espressioni. Ad esempio, la seguente istruzione `SELECT` restituisce i nomi dei dipendenti con stipendio uguale o superiore a 20.000 Euro.

```
SELECT cognome, nome FROM dip WHERE stipendio >= 20000
```

La clausola `WHERE` può utilizzare anche espressioni come:

```
WHERE expr1 IS NULL
WHERE NOT expr2
```

Nota Se si utilizzano nomi completamente qualificati nella lista (proiezione) `SELECT`, è necessario utilizzare anche nomi completamente qualificati nella clausola `WHERE` correlata.

Clausola GROUP BY

La clausola `GROUP BY` specifica i nomi di uno o più campi in base a cui raggruppare i valori restituiti. Questa clausola viene utilizzata per restituire un gruppo di valori aggregati. Ha il seguente formato:

```
GROUP BY colonne
```

`colonne` deve corrispondere all'espressione della colonna utilizzata nella clausola `SELECT`. Un'espressione di colonna può essere costituita da uno o più nomi di campi della tabella di database separati da virgole.

Esempio

L'esempio seguente somma gli stipendi di ogni settore.

```
SELECT id_sett, SUM(stipendio) FROM dip GROUP BY id_sett
```

Questa istruzione restituisce una riga per ogni singolo ID settore. Ogni riga contiene l'ID settore e la somma degli stipendi dei dipendenti del settore.

Clausola HAVING

La clausola `HAVING` permette di specificare le condizioni per i gruppi di record, ad esempio per visualizzare solo i settori per cui l'importo complessivo degli stipendi è superiore a 200.000 Euro. Ha il seguente formato:

```
HAVING espr1 operatore_rel espr2
```

`espr1` e `espr2` possono essere nomi di campo, valori costanti o espressioni. Queste espressioni non devono corrispondere all'espressione di una colonna nella clausola `SELECT`.

`operatore_rel` è l'operatore relazionale che collega le due espressioni.

Esempio

Nell'esempio che segue vengono restituiti solo i settori per cui l'importo complessivo degli stipendi è superiore a 200.000 Euro:

```
SELECT id_sett, SUM (stipendio) FROM dip
   GROUP BY id_sett HAVING SUM (stipendio) >200000
```

Operatore UNION

L'operatore UNION combina i risultati di due o più istruzioni SELECT in un solo risultato. Il singolo risultato ottenuto comprende tutti i record restituiti dalle istruzioni SELECT. Per impostazione predefinita, i record duplicati non vengono restituiti. Per restituire i record duplicati, utilizzare la parola chiave ALL (UNION ALL). Il formato è:

```
SELECT istruzione UNION [ALL] SELECT istruzione
```

Quando si usa l'operatore UNION, le liste di selezione di ogni istruzione SELECT devono avere lo stesso numero di espressioni di colonna, con dati dello stesso tipo, e devono essere specificate nello stesso ordine. Ad esempio:

```
SELECT cognome, stipendio, data_assunzione FROM dip UNION SELECT
nome, pagamento, data_nascita FROM persona
```

In questo esempio vi è lo stesso numero di espressioni di colonna e ogni espressione di colonna, in ordine, ha dati dello stesso tipo.

L'esempio che segue non è valido perché i tipi di dati delle espressioni di colonna sono diversi (STIPENDIO from DIP ha dati di tipo diverso rispetto a COGNOME from AUMENTI). In questo esempio vi è lo stesso numero di espressioni di colonna in ogni istruzione SELECT, ma le espressioni non sono nello stesso ordine per tipo di dati.

```
SELECT cognome, stipendio FROM dip UNION SELECT stipendio, cognome
FROM aumenti
```

Clausola ORDER BY

La clausola ORDER BY indica il modo in cui i record devono essere ordinati. Il formato è:

```
ORDER BY {espressione_ordinamento [DESC | ASC]}, ...
```

espressione_ordinamento può essere: nomi di campo, espressioni o il numero di posizione dell'espressione di colonna da usare. Per impostazione predefinita l'ordinamento viene effettuato in modo crescente (ASC).

Ad esempio, per ordinare in base al cognome e poi al nome, è possibile usare una delle seguenti istruzioni SELECT:

```
SELECT id_dip, cognome, nome FROM dip ORDER BY cognome, nome
```

oppure

```
SELECT id_dip, cognome, nome FROM dip ORDER BY 2,3
```

Nel secondo esempio, cognome è la seconda espressione di colonna dopo SELECT, pertanto ORDER BY 2 ordina in base al cognome.

Clausole OFFSET e FETCH FIRST

Le clausole OFFSET e FETCH FIRST sono utilizzate per restituire un intervallo specificato di righe da un punto particolare di un set di risultati. La capacità di limitare le righe recuperate da set di risultati di grandi dimensioni permette di scorrere i dati e migliora l'efficienza.

La clausola OFFSET indica il numero di righe da saltare prima di iniziare a restituire i dati. Se la clausola OFFSET non è utilizzato in un'istruzione SELECT, la riga di inizio è 0. La clausolaFETCH FIRST specifica il numero di righe che deve essere restituito, sia come un intero non firmato maggiore o uguale a 1 o come una percentuale, dal punto di partenza indicato nella clausola OFFSET. Se entrambe le clausole OFFSET e FETCH FIRST sono utilizzate in un'istruzione SELECT, la clausola OFFSET deve venire prima.

Le clausole OFFSET e FETCH FIRST non sono supportate nelle subquery.

Formato OFFSET

Il formato OFFSET è:

```
OFFSET n {ROWS| ROW} ]
```

n è un intero non firmato. Se n è maggiore del numero delle righe restituite nel set di risultati, non viene restituito nulla e non viene visualizzato nessun messaggio di errore.

ROWS è uguale a ROW.

Formato FETCH FIRST

Il formato FETCH FIRST è:

```
FETCH FIRST [n [PERCENT]] { ROWS| ROW } {ONLY| WITH TIES } ]
```

n è il numero di righe che deve essere restituito. Il valore predefinito è 1 se n èomesso,n è un intero non firmato maggiore o uguale a 1 a meno che sia seguito da PERCENT. Sen è seguito da PERCENT,il valore può essere positivo frazionario o un intero non firmato.

ROWS è uguale a ROW.

WITH TIES deve essere utilizzato con la clausola ORDER BY.

WITH TIES permette di restituire più righe di quelle specificate nel valore di conteggio FETCH perché vengono restituite anche le righe equivalenti, ossia quelle righe che non sono distinte in base alla clausola ORDER BY.

Esempi

Ad esempio, per restituire le informazioni dalla ventiseiesima riga del set risultati ordinato per cognome poi per nome, utilizzare la seguente istruzione SELECT:

```
SELECT id_dip, cognome, nome FROM dip ORDER BY cognome, nome OFFSET
25 ROWS
```

Per specificare che si desidera restituire solo dieci righe:

```
SELECT id_dip, cognome, nome FROM dip ORDER BY cognome, nome OFFSET
25 ROWS FETCH FIRST 10 ROWS ONLY
```

Per restituire le dieci righe e le loro righe equivalenti (righe che sono non distinte in base alla clausola `ORDER BY`):

```
SELECT id_dip, cognome, nome FROM dip ORDER BY cognome, nome OFFSET
25 ROWS FETCH FIRST 10 ROWS WITH TIES
```

Clausola FOR UPDATE

La clausola `FOR UPDATE` blocca gli aggiornamenti o le eliminazioni nella posizione attraverso i cursori SQL. Il formato è:

```
FOR UPDATE [OF espressioni_colonna]
```

`espressioni_colonna` è una lista di nomi campo nella tabella di database che si desidera aggiornare, separati da una virgola. `espressioni_colonna` è opzionale e viene ignorato.

Esempio

L'esempio che segue restituisce tutti i record nel database dei dipendenti per cui il valore del campo `STIPENDIO` è superiore a 20.000 Euro. I record recuperati vengono bloccati. Se il record viene aggiornato o eliminato, il blocco viene mantenuto finché non si applica la modifica. In caso contrario, il blocco viene rilasciato quando si recupera il record successivo.

```
SELECT * FROM dip WHERE stipendio > 20000
   FOR UPDATE OF cognome, nome, stipendio
```

Altri esempi:

Utilizzo di	Esempio SQL
costante di testo	`SELECT 'CatDog' FROM Venditori`
costante numerica	`SELECT 999 FROM Venditori`
costante di data	`SELECT DATE '05/06/2012' FROM Venditori`
costante di ora	`SELECT TIME '02:49:03' FROM Venditori`
costante Indicatore data e ora	`SELECT TIMESTAMP '05/06/2012 02:49:03' FROM Venditori`
colonna di testo	`SELECT Nome_Azienda FROM Dati_Vendite` `SELECT DISTINCT NomeAzienda FROM Dati _Vendite`
colonna numerica	`SELECT Quantità FROM Dati_Vendite` `SELECT DISTINCT Quantità FROM Dati_Vendite`
colonna data	`SELECT Data_Vendita FROM Dati_Vendite` `SELECT DISTINCT Data_Vendita FROM Dati_Vendite`
colonna ora	`SELECT Ora_Vendita FROM Dati_Vendite` `SELECT DISTINCT Ora_Vendita FROM Dati_Vendite`
colonna Indicatore data e ora	`SELECT IndicatoreDataOra_Vendita FROM Dati_Vendite` `SELECT DISTINCT IndicatoreDataOra_Vendita FROM Dati_Vendite`
Colonna[a] BLOB	`SELECT Brochure_Società FROM Dati _Vendite` `SELECT GETAS(Logo_Società, 'JPEG')FROM Dati _Vendite`
carattere jolly*	`SELECT * FROM Venditori` `SELECT DISTINCT * FROM Venditori`

a. Un BLOB è un campo Contenitore di un file di database FileMaker.

Note sugli esempi

Una `colonna` è un riferimento a un campo nel file di database FileMaker. (il campo può contenere molti valori distinti).

Il carattere jolly asterisco (*) rappresenta “tutto”. Per l'esempio `SELECT * FROM Venditori`, il risultato comprende tutte le colonne nella tabella `Venditori`. Per l'esempio `SELECT DISTINCT * FROM Venditori`, il risultato è costituito da tutte le righe uniche nella tabella `Venditori` (senza duplicati).

- FileMaker non memorizza dati per stringhe vuote, quindi le seguenti query non restituiscono mai record:

```
SELECT * FROM test WHERE c =''
SELECT * FROM test WHERE c <>"
```

- Se si usa `SELECT` con dati binari, è necessario utilizzare la funzione `GetAs()` per specificare il flusso da restituire. Per ulteriori informazioni, vedere la seguente sezione “Recupero dei contenuti di un campo Contenitore: Funzione CAST() e funzione GetAs()”.

Recupero dei contenuti di un campo Contenitore: Funzione CAST() e funzione GetAs()

Da un campo Contenitore si possono recuperare dati binari, informazioni di riferimento al file o dati di un tipo specifico di file.

Se esistono file o dati binari JPEG, l'istruzione `SELECT` con `GetAs(nome di campo, "JPEG")` recupera i dati in forma binaria; in caso contrario, l'istruzione `SELECT` con il nome campo restituisce `NULL`.

Per recuperare le informazioni di riferimento al file da un campo Contenitore, come il percorso di un file, di un'immagine o di un filmato QuickTime, utilizzare la funzione `CAST()` con un'istruzione `SELECT`. Ad esempio:

```
SELECT CAST(Brochure_Società AS VARCHAR(NNN))FROM Dati _Vendite
```

In questo esempio se:

- È stato inserito un file nel campo Contenitore utilizzando FileMaker Pro, ma è stato memorizzato solo un riferimento al file, l'istruzione `SELECT` recupera le informazioni di riferimento al file come `SQL_VARCHAR`.
- Sono stati inseriti i contenuti di un file nel campo Contenitore utilizzando FileMaker Pro, l'istruzione `SELECT` recupera il nome del file.
- È stato importato un file nel campo Contenitore da un'altra applicazione, l'istruzione `SELECT` visualizza '?' (il file viene visualizzato come **Senza nome.dat**in FileMaker Pro).

Per recuperare dati da un campo Contenitore, utilizzare la funzione `GetAs()`. È possibile utilizzare l'opzione `DEFAULT` o specificare il tipo di file. L'opzione `DEFAULT` recupera il flusso master per il contenitore senza la necessità per definisce esplicitamente il tipo di flusso:

```
SELECT GetAs(Brochure_Società, DEFAULT) FROM  Dati_Vendite
```

Per recuperare un tipo di flusso singolo da un contenitore, utilizzare la funzione `GetAs()` con il tipo di file basato su come sono stati inseriti i dati nel campo Contenitore in FileMaker Pro. Ad esempio:

- Se i dati sono stati inseriti utilizzando il comando **Inserisci** >**File** , specificare `"FILE"` nella funzione `GetAs()`. Ad esempio:

  ```
  SELECT GetAs(Brochure_Società, 'FILE') FROM  Dati_Vendite
  ```

- Se i dati sono stati inseriti utilizzando il comando **Inserisci** >**Suono** (Suono standard — formato grezzo Mac OS X), specificare `'snd'` nella funzione `GetAs()`. Ad esempio:

  ```
  SELECT GetAs(Meeting_Società, 'snd') FROM Newsletter_Società
  ```

- Se i dati sono stati inseriti utilizzando il comando **Inserisci** >**Immagine**, trascinandoli oppure incollandoli dagli appunti, specificare uno dei tipi di file elencati nella seguente tabella. Ad esempio:

```
SELECT GetAs(Logo_Società, 'JPEG ') FROM Icone_Società
```

Tipo di file	Descrizione	Tipo di file	Descrizione
'GIFf'	Graphics Interchange Format	'PNTG'	MacPaint
'JPEG'	Immagini fotografiche	'.SGI'	Formato generico bitmap
'JP2'	JPEG 2000	'TIFF'	Formato raster del file per immagini digitali
'PDF '	Portable Document Format	'TPIC'	Targa
'PNGf'	Formato immagine Bitmap	'8BPS'	Photoshop (PSD)

Istruzione DELETE

Usare l'istruzione `DELETE` per eliminare i record da una tabella di database. Il formato dell'istruzione `DELETE` è:

```
DELETE FROM nome_tabella [ WHERE { condizioni } ]
```

Nota La clausola `WHERE` determina i record da eliminare. Se non si include la parola chiave `WHERE`, tutti i record nella tabella vengono cancellati (ma la tabella rimane invariata).

Esempio

Di seguito è riportato un esempio di istruzione `DELETE` eseguita sulla tabella Dipendente:

```
DELETE FROM dip WHERE id_dip = 'E10001'
```

L'istruzione `DELETE` rimuove tutti i record che soddisfano le condizioni della clausola `WHERE`. In questo caso vengono eliminati tutti i record in cui il codice del dipendente è `E10001`. Poiché nella tabella Dipendenti i codici dei dipendenti sono unici, viene eliminato un solo record.

Istruzione INSERT

Utilizzare l'istruzione `INSERT` per creare record in una tabella di database. È possibile specificare:

- Una lista di valori da inserire come nuovo record
- Un'istruzione `SELECT` che copia i dati di un'altra tabella da inserire come gruppo di nuovi record

Il formato dell'istruzione `INSERT` è:

```
INSERT INTO nome_tabella [(nome_colonna, ..).] VALUES (expr, ..).
```

`nome_colonna` è una lista facoltativa di nomi di colonna che specifica il nome e l'ordine delle colonne di cui sono stati specificati i valori nella clausola `VALUES`. Se si omette `nome_colonna`, le espressioni di valore (`expr`) dovranno specificare i valori di tutte le colonne definite nella tabella e dovranno riflettere l'ordine delle colonne definito per la tabella. `nome_colonna` può anche specificare una ripetizione campo, ad esempio `lastDates[4]`.

expr è la lista di espressioni che forniscono i valori delle colonne del nuovo record. In genere, le espressioni sono valori costanti per le colonne (ma possono anche essere una subquery). È necessario racchiudere i valori delle stringhe di caratteri tra coppie di virgolette singole ('). Per includere una virgoletta singola nel valore di una stringa di caratteri racchiusa tra virgolette singole, usare due virgolette singole insieme (ad esempio, 'L''aquilone').

Le subquery devono essere racchiuse tra parentesi.

L'esempio che segue inserisce una lista di espressioni:

```
INSERT INTO dip (cognome, nome, id_dip, stipendio, data_assunzione)
   VALUES ('Smith', "John", "E22345", 27500, DATA ' 5/6/2013')
```

Ciascuna istruzione INSERT aggiunge un record nella tabella di database. In questo caso è stato aggiunto un record alla tabella di database dei dipendenti denominata DIP. I valori sono stati specificati per cinque colonne. Alle restanti colonne della tabella è stato assegnato un valore vuoto, ossia Null.

Nota Nei campi Contenitore, è possibile inserire (INSERT) solo testo, a meno che si prepari un'istruzione parametrizzata e si effettui lo streaming dei dati dall'applicazione. Per utilizzare dati binari, basta semplicemente assegnare il nome file racchiudendolo tra virgolette semplici o utilizzare la funzione PutAs(). Quando si specifica il nome file, il tipo file viene dedotto dall'estensione del file:

```
INSERT INTO nome_tabella (nome_contenitore) VALUES (? AS
'nomefile.file estensione')
```

I tipi di file non supportati sono inseriti come tipo FILE.

Quando si utilizza la funzione PutAs(), specificare il tipo: PutAs(col, 'tipo'), dove il valore del tipo è un tipo file supportato come descritto in“Recupero dei contenuti di un campo Contenitore: Funzione CAST() e funzione GetAs()” a pagina 15.

L'istruzione SELECT è una query che restituisce valori per ciascun valore nome_colonna specificato nella lista dei nomi di colonna. Se si specifica un'istruzione SELECT anziché una lista di espressioni di valori, sarà possibile selezionare un gruppo di righe da una tabella e inserirlo in un'altra tabella tramite una singola istruzione INSERT.

Di seguito è riportato un esempio di istruzione INSERT che utilizza un'istruzione SELECT:

```
INSERT INTO dip1 (nome, cognome, id_dip, settore, stipendio)
   SELECT nome, cognome, id_dip, settore, stipendio from dip
   WHERE settore =' D050'
```

In questo tipo di istruzione INSERT, il numero di colonne da inserire deve corrispondere al numero di colonne dell'istruzione SELECT. La lista di colonne da inserire deve corrispondere alle colonne nell'istruzione SELECT, analogamente a quanto accade per le espressioni di valore nell'altro tipo di istruzione INSERT. Ad esempio, la prima colonna inserita corrisponde alla prima colonna selezionata; la seconda alla seconda, e così via.

Le dimensioni e il tipo di dati di queste colonne corrispondenti devono essere compatibili. Ciascuna colonna della lista SELECT dovrebbe disporre di un tipo di dati accettato dal driver client ODBC o JDBC per un'istruzione INSERT/UPDATE standard della colonna corrispondente nella lista INSERT. Se la dimensione dei valori nella colonna della lista SELECT supera la dimensione dei valori nella colonna della lista INSERT corrispondente, i valori vengono troncati.

L'istruzione SELECT viene valutata prima dell'inserimento di qualsiasi valore.

Istruzione UPDATE

Utilizzare l'istruzione `UPDATE` per cambiare i record in una tabella di database. Il formato dell'istruzione `UPDATE` è:

```
UPDATE nome_tabella SET nome_colonna = expr, ... [ WHERE { condizioni } ]
```

`nome_colonna` è il nome di una colonna di cui si desidera modificare il valore. È possibile modificare più colonne in una singola istruzione.

`expr` è il nuovo valore della colonna.

In genere, le espressioni sono valori costanti per le colonne (ma possono anche essere una subquery). È necessario racchiudere i valori delle stringhe di caratteri tra coppie di virgolette singole ('). Per includere una virgoletta singola nel valore di una stringa di caratteri racchiusa tra virgolette singole, usare due virgolette singole insieme (ad esempio, `'L''aquilone'`).

Le subquery devono essere racchiuse tra parentesi.

La clausola `WHERE` può essere una qualsiasi clausola valida. Determina i record da aggiornare.

Esempi

Di seguito è riportato un esempio di istruzione `UPDATE` eseguita sulla tabella Dip:

```
UPDATE dip SET stipendio=32000; detrazioni=1 WHERE id_dip = 'E10001'
```

L'istruzione `UPDATE` modifica tutti i record che soddisfano le condizioni della clausola `WHERE`. In questo caso vengono modificati lo stipendio e lo stato delle detrazioni per tutti i dipendenti il cui codice è `E10001`. Poiché nella tabella Dipendenti i codici dei dipendenti sono unici, viene aggiornato un solo record.

Di seguito è riportato un esempio con una subquery:

```
UPDATE dip SET stipendio = (SELECT avg(stipendio) from dip) WHERE
id_dip = 'E10001'
```

In questo caso lo stipendio del dipendente il cui codice è `E10001` viene sostituito con lo stipendio medio della società.

Nota Nei campi Contenitore, è possibile aggiornare (`UPDATE`) solo con testo, a meno che si prepari un'istruzione parametrizzata e si effettui lo streaming dei dati dall'applicazione. Per utilizzare dati binari, basta semplicemente assegnare il nome file racchiudendolo tra virgolette singole o utilizzare la funzione `PutAs()`. Quando si specifica il nome file, il tipo file viene dedotto dall'estensione del file:

```
UPDATE nome_tabella SET (nome_contenitore)=? AS 'nomefile.estensione
file'
```

I tipi di file non supportati sono inseriti come tipo FILE.

Quando si utilizza la funzione `PutAs()`, specificare il tipo: `PutAs(col, 'tipo')`, dove il valore del tipo è un tipo file supportato come descritto in“Recupero dei contenuti di un campo Contenitore: Funzione CAST() e funzione GetAs()” a pagina 15.

Istruzione CREATE TABLE

Utilizzare l'istruzione `CREATE TABLE` per creare una tabella in un file di database. Il formato dell'istruzione `CREATE TABLE` è:

```
CREATE TABLE nome_tabella ( lista_elementi_tabella [,
lista_elementi_tabella...] )
```

All'interno dell'istruzione, si specifica il nome e il tipo di dati di ogni colonna.

- `nome_tabella` è il nome della tabella. `nome_tabella` ha un limite di 100 caratteri. Non è possibile definire una tabella con lo stesso nome. Il nome tabella deve iniziare con un carattere alfabetico. Se il nome tabella non inizia con un carattere alfabetico, racchiuderlo nelle virgolette doppie (identificativo quotato).
- Il formato per `lista_elementi_tabella` è:

  ```
  nome_campo tipo_campo [DEFAULT expr] [UNIQUE| NOT NULL| CHIAVE
  PRIMARIA| GLOBALE]
  [EXTERNAL stringa_percorso_relativo [SECURE| OPEN
  stringa_percorso_calc]]
  ```

  - `nome_campo` è il nome del campo. Due campi nella stessa tabella non possono avere lo stesso nome. È possibile specificare una ripetizione di campo inserendo un numero tra parentesi quadre. Ad esempio: `lastDates[4]`. I nomi campo iniziano con un carattere alfabetico. Se il nome del campo non inizia con un carattere alfabetico, racchiuderlo nelle virgolette doppie (identificativo quotato). Ad esempio, l'istruzione `CREATE TABLE` per il campo `_COGNOME` è:

    ```
    CREATE TABLE"_DIPENDENTE" (ID INTERO CHIAVE PRIMARIA,"_NOME"
    VARCHAR(20),"_COGNOME" VARCHAR(20))
    ```

  - `tipo_campo` può essere uno dei seguenti valori: `NUMERIC`, `DECIMAL`, `INT`, `DATE`, `TIME`, `TIMESTAMP`, `VARCHAR`, `CHARACTER VARYING`, `BLOB`, `VARBINARY`, `LONGVARBINARY`, or `BINARY VARYING`. Per `NUMERIC` e `DECIMAL`, è possibile specificare la precisione e la scala. Ad esempio: `DECIMAL(10,0)`. Per `TIME` e `TIMESTAMP`, è possibile specificare la precisione. Ad esempio: `TIMESTAMP(6)`. Per `VARCHAR` e `CHARACTER VARYING`, è possibile specificare la lunghezza della stringa. Ad esempio: `VARCHAR(255)`.
  - La parola chiave `DEFAULT` permette di impostare un valore predefinito per una colonna. Per `expr`, è possibile utilizzare un valore costante o un'espressione. Le espressioni consentite sono `USER`, `USERNAME`, `CURRENT_USER`, `CURRENT_DATE`, `CURDATE`, `CURRENT_TIME`, `CURTIME`, `CURRENT_TIMESTAMP`, `CURTIMESTAMP`, e `NULL`.
  - Se una colonna viene definita `UNIQUE` si seleziona automaticamente l'opzione di verifica **unique** per il campo corrispondente nel file di database FileMaker.
  - Se una colonna viene definita `NOT NULL` si seleziona automaticamente l'opzione di verifica **Not Empty** per il campo corrispondente nel file di database FileMaker. Il campo viene contrassegnato come **Required Value** nella scheda **Campi** della finestra di dialogo Gestisci database in FileMaker Pro.

- Per definire una colonna come un campo Contenitore, inserire `BLOB`, `VARBINARY` o `BINARY VARYING` per `tipo_campo`.
- Per definire una colonna come un campo Contenitore che memorizza i dati esternamente, utilizzare la parola chiave `EXTERNAL`. `stringa_percorso_relativo` definisce la cartella in cui i dati sono memorizzati esternamente rispetto alla posizione del database FileMaker. Questo percorso deve essere indicato come directory di base nella finestra di dialogo Gestisci contenitori di FileMaker Pro. Specificare `SECURE` per un'archiviazione protetta o `OPEN` per un'archiviazione di tipo open storage. Se si usa la memoria aperta, `stringa_percorso_calc` è la cartella all'interno della cartella `stringa_percorso_relativo` dove gli oggetti del contenitore devono essere memorizzati. Il percorso deve utilizzare le barre (/) nella cartella nome.

Esempi

Utilizzo di	Esempio SQL
colonna di testo	`CREATE TABLE T1 ( C1 VARCHAR, C2 VARCHAR (50), C3 VARCHAR (1001), C4 VARCHAR (500276))`
colonna di testo, `NOT NULL`	`CREATE TABLE T1NN ( C1 VARCHAR NOT NULL, C2 VARCHAR (50) NOT NULL, C3 VARCHAR (1001) NOT NULL, C4 VARCHAR (500276) NOT NULL)`
colonna numerica	`CREATE TABLE T2 (C1 DECIMAL, C2 DECIMAL (10,0), C3 DECIMAL (7539,2), C4 DECIMAL (497925,301))`
colonna data	`CREATE TABLE T3 (C1 DATE, C2 DATE, C3 DATE, C4 DATE)`
colonna ora	`CREATE TABLE T4 (C1 TIME, C2 TIME, C3 TIME, C4 TIME)`
colonna Indicatore data e ora	`CREATE TABLE T5 (C1 TIMESTAMP, C2 TIMESTAMP, C3 TIMESTAMP, C4 TIMESTAMP)`
colonna per campo Contenitore	`CREATE TABLE T6 (C1 BLOB, C2 BLOB, C3 BLOB, C4 BLOB)`
colonna per campo Contenitore memoria esterna	`CREATE TABLE T7 (C1 BLOB EXTERNAL 'File/MioDatabase/' SECURE)` `CREATE TABLE T8 (C1 BLOB EXTERNAL 'File/MioDatabase/' OPEN 'Oggetti')`

Istruzione ALTER TABLE

Utilizzare l'istruzione `ALTER TABLE` per cambiare la struttura di una tabella esistente in un file di database. È possibile modificare una sola colonna in ogni istruzione. I formati dell'istruzione `ALTER TABLE` sono:

```
ALTER TABLE nome_tabella ADD [COLUMN] definizione_colonna

ALTER TABLE nome_tabella DROP [COLUMN] nome_colonna_non_qualificato

ALTER TABLE nome_tabella ALTER [COLUMN] definizione_colonna SET
DEFAULT expr

ALTER TABLE nome_tabella ALTER [COLUMN] definizione_colonna DROP
DEFAULT
```

È necessario conoscere la struttura della tabella e sapere come modificarla prima di usare l'istruzione `ALTER TABLE`.

Esempi

Per	Esempio SQL
aggiungere colonne	`ALTER TABLE Venditori ADD (C1 VARCHAR)`
rimuovere colonne	`ALTER TABLE Venditori DROP C1`
impostare il valore predefinito per una colonna	`ALTER TABLE Venditori ALTER Società SET DEFAULT ' FileMaker '`
rimuovere il valore predefinito per una colonna	`ALTER TABLE Venditori ALTER Società DROP DEFAULT`

Nota `SET DEFAULT` e `DROP DEFAULT` non influiscono sulle righe esistenti nella tabella, ma cambiano il valore predefinito per le righe aggiunte successivamente alla tabella.

Istruzione CREATE INDEX

Utilizzare l'istruzione `CREATE INDEX` per velocizzare le ricerche nel file di database. Il formato dell'istruzione `CREATE INDEX` è:

```
CREATE INDEX ON nome_tabella.nome_colonna
CREATE INDEX ON nome_tabella (nome_colonna)
```

`CREATE INDEX` è supportato per una sola colonna (gli indici a più colonne non sono supportati). Gli indici non sono consentiti sulle colonne che corrispondono a campi di tipo Contenitore, campi Riassunto, campi per cui è prevista l'opzione di memorizzazione globale o campi Calcolo non memorizzati in un file di database FileMaker.

Con la creazione di un indice per una colonna di testo si imposta automaticamente l'opzione di memorizzazione **Indicizzazione** su **Minimo** per il campo corrispondente nel file di database FileMaker. Con la creazione di un indice per una colonna non di testo (o per una colonna formattata come testo giapponese) si imposta automaticamente l'opzione di memorizzazione **Indicizzazione** su **Tutti** per il campo corrispondente nel file di database FileMaker.

Con la creazione di un indice per tutte le colonne si imposta automaticamente l'opzione di memorizzazione **Indicizzazione** su **Crea automaticamente gli indici quando necessario** per il campo corrispondente nel file di database FileMaker.

FileMaker crea automaticamente gli indici secondo necessità. Utilizzando `CREATE INDEX` l'indice da creare viene generato subito anziché su richiesta.

Esempio

```
CREATE INDEX ON Venditori.ID_Venditore
```

Istruzione DROP INDEX

Utilizzare l'istruzione `DROP INDEX` per rimuovere un indice da un file di database. Il formato dell'istruzione `DROP INDEX` è:

```
DROP INDEX ON nome_tabella.nome_colonna
DROP INDEX ON nome_tabella (nome_colonna)
```

Se il file di database è troppo grande o non si usa spesso un campo nelle query, rimuovere l'indice.

Se le query non sono soddisfacenti e si sta lavorando con un file di database FileMaker particolarmente grande, con molti campi di testo indicizzati, è possibile eliminare gli indici da alcuni campi. È anche possibile eliminare gli indici dai campi che si usano raramente nell'istruzione `SELECT`.

Eliminando un indice da qualsiasi colonna, si seleziona automaticamente l'opzione di memorizzazione **Nessuno** e si elimina **Crea automaticamente gli indici quando necessario** in **Indicizzazione** per il campo corrispondente nel file di database FileMaker.

L'attributo `PREVENT INDEX CREATION` non è supportato.

Esempio

```
DROP INDEX ON Venditori.ID_Venditore
```

Espressioni SQL

Utilizzare le espressioni nelle clausole `WHERE`, `HAVING`, e `ORDER BY` delle istruzioni `SELECT` per generare query dettagliate e sofisticate. Elementi di espressione validi sono:

- Nomi campo
- Costanti
- Notazione esponenziale/scientifica
- Operatori numerici
- Operatori alfabetici
- Operatori data
- Operatori relazionali
- Operatori logici
- Funzioni

Nomi campo

L'espressione più comune è un semplice nome di campo, come ad esempio `calc` o `Dati_vendita.Fattura_ID`.

Costanti

Le costanti sono valori che non cambiano. Ad esempio, nell'espressione `PREZZO * 1.05`, il valore 1.05 è una costante. In alternativa è possibile assegnare il valore 30 alla costante `Numero_Di_Giorni_A_Giugno`.

È necessario racchiudere le costanti di caratteri tra coppie di virgolette singole ('). Per includere una virgoletta singola in una costante di caratteri racchiusa tra virgolette singole, usare due virgolette singole insieme (ad esempio, `'L''aquilone'`).

Per applicazioni ODBC e JDBC, FileMaker accetta le costanti di data, ora e indicatore data e ora in formato ODBC/JDBC in parentesi graffe ({}), ad esempio:

- `{D '06/05/2012'}`
- `{T '14:35:10'}`
- `{TS '06/05/2012 14:35:10'}`

FileMaker accetta un indicatore di tipo (`D`, `T`, `TS`) sia maiuscolo che minuscolo. È possibile inserire qualsiasi numero di spazi dopo l'indicatore di tipo, o anche omettere lo spazio.

FileMaker accetta anche la sintassi SQL-92 con data e ora in formato ISO, senza parentesi:

- `DATA 'AAAA-MM-GG'`
- `ORA 'HH:MM:SS'`
- `INDICATORE DATA E ORA 'AAAA-MM-GG HH:MM:SS'`

La funzione EseguiSQL FileMaker Pro accetta solo la sintassi SQL-92 con data e ora in formato ISO senza parentesi.

Costante	Sintassi accettabile (esempi)
Testo	`'Parigi'`
Numero	`1.05`

Data	`DATE '05/06/2012'`
	`{ D '2012-06-05' }`
	`{06/05/2012}`
	`{06/05/2012}`
	Nota La sintassi dell'anno a 2 cifre non è supportata per il formato ODBC/JDBC o per il formato SQL-92.

Ora	`TIME '14:35:10'`
	`{ T '14:35:10' }`
	`{14:35:10}`

Indicatore data e ora	`TIMESTAMP '05/06/2012 14:35:10'`
	`{ TS '2012-06-05 14:35:10'}`
	`{06/05/2012 14:35:10}`
	`{06/05/2012 14:35:10}`
	Assicurarsi che **Tipo di dati restrittivo: Data a 4 cifre** non sia selezionato come opzione di verifica nel file di database FileMaker per un campo che utilizza questa sintassi dell'anno a 2 cifre.
	Nota La sintassi dell'anno a 2 cifre non è supportata per il formato ODBC/JDBC o per il formato SQL-92.

Quando si inseriscono valori di data e ora utilizzare il formato locale del file di database. Ad esempio, se il database è stato creato in un sistema di lingua italiana, utilizzare i formati data e ora italiani.

Notazione esponenziale/scientifica

I numeri possono essere espressi utilizzando la notazione scientifica.

Esempio

```
SELECT colonna1 / 3.4E+7 FROM tabella1 WHERE calc < 3.4E-6 * colonna2
```

Operatori numerici

È possibile comprendere i seguenti operatori nelle espressioni numeriche: +, -, *, /, and ^ o ** (esponenziazione).

È possibile anteporre alle espressioni numeriche un più (+) oppure un meno (-) unario.

Operatori alfabetici

È possibile concatenare i caratteri.

Esempi

Negli esempi che seguono, `cognome` è `BIANCHI ` e nome è `'FABIO '`:

Operatore	Concatenazione	Esempio	Risultato
+	Mantiene gli spazi finali	nome + cognome	'FABIO BIANCHI'
-	Sposta gli spazi finali in fondo	nome - cognome	'FABIOBIANCHI'

Operatori data

È possibile modificare le date.

Esempi

Negli esempi seguenti, `data_assunzione` è `DATE '30/01/2013'`.

Operatore	Effetto sulla data	Esempio	Risultato
+	Aggiunge un numero di giorni ad una data	`data_assunzione`+5	DATE '04/02/2013'
-	Trova il numero di giorni tra due date	`data_assunzione - DATE '01/01/2013'`	`29`
	Sottrae un numero di giorni da una data	`data_assunzione - 10`	DATE '20/01/2013'

Altri esempi:

```
SELECT Data_Vendita, Data_Vendita + 30 AS agg FROM Dati_Vendite
SELECT Data_Vendita, Data_Vendita - 30 AS agg FROM Dati_Vendite
```

Operatori relazionali

Operatore	Significato
=	Uguale
<>	Diverso da
>	Maggiore di
>=	Maggiore o uguale a
<	Minore di
<=	Minore o uguale a
LIKE	Corrisponde ad una struttura
NOT LIKE	Non corrisponde ad una struttura
IS NULL	Uguale a NULL
IS NOT NULL	Diverso da NULL
BETWEEN	Intervallo di valori tra un limite inferiore e un limite superiore
IN	Un membro di un gruppo di valori specificati o un membro di una subquery
NOT IN	Non un membro di un gruppo di valori specificati né un membro di una subquery

Operatore	Significato
EXISTS	'Vero' se una subquery ha restituito almeno un record
ANY	Confronta un valore con ogni valore restituito da una subquery (l'operatore deve essere preceduto da =, <>, >, >=, <o <=); =Any equivale a In
ALL	Confronta un valore con ogni valore restituito da una subquery (l'operatore deve essere preceduto da =, <>, >, >=, <o <=)

Esempi

```
SELECT Dati_Vendite.ID_Fattura FROM Dati _Vendite
   WHERE Dati_Vendite.ID_Venditore = 'SP-1'

SELECT Dati_Vendite.Importo FROM Dati_Vendite WHERE
Dati_Vendite.ID_Fattura <> 125

SELECT Dati_Vendite.Importo FROM Dati_Vendite WHERE
Dati_Vendite.Quantità >3000

SELECT Dati_Vendite.Ora_Vendita FROM Dati_Vendite
   WHERE Dati_Vendite.Ora_Vendita < '12:00:00'

SELECT Dati_Vendite.Nome_Società FROM Dati _Vendite
   WHERE Dati_Vendite.Nome_Società LIKE '%University'

SELECT Dati_Vendite.Nome_Società FROM Dati _Vendite
   WHERE Dati_Vendite.Nome_Società NOT LIKE '%University'

SELECT Dati_Vendite.Quantità FROM Dati _Vendite WHERE
Dati_Vendite.Quantità IS NULL

SELECT Dati_Vendite.Quantità FROM Dati _Vendite WHERE
Dati_Vendite.Quantità IS NOT NULL

SELECT Dati_Vendite.ID_Fattura FROM Dati _Vendite
   WHERE Dati_vendite.ID_Fattura BETWEEN 1 AND 10

SELECT COUNT(Dati_Vendite.ID_Fattura) AS agg
   FROM Dati_Vendite WHERE Dati_Vendite.INVOICE_ID IN (50,250,100)

SELECT COUNT(Dati_Vendite.ID_Fattura) AS agg
   FROM Dati_Vendite WHERE Dati_Vendite.INVOICE_ID NOT IN (50,250,100)

SELECT COUNT(Dati_Vendite.ID_Fattura) AS agg FROM Dati_Vendite
   WHERE Dati_vendite.INVOICE_ID NOT IN (SELECT Dati_vendite.ID_Fattura
   FROM Dati_Vendite WHERE Dati_Vendite.ID_Venditore = 'SP-4')

SELECT*
   FROM Dati_Vendite WHERE EXISTS (SELECT Dati_Vendite.Importo
   FROM Dati_Vendite WHERE Dati_Vendite.ID_Venditore IS NOT NULL)
```

```
SELECT*
   FROM Dati_Vendite WHERE Dati_Vendite.Importo = ANY (SELECT
Dati_Vendite.Quantità
   FROM Dati_Vendite WHERE Dati_Vendite.ID_Venditore = 'SP-1')

SELECT*
   FROM Dati_Vendite WHERE Dati_Vendite.Importo = ALL (SELECT
Dati_Vendite.Quantità
   FROM Dati_Vendite WHERE Dati_Vendite.ID_Venditore IS NULL)
```

Operatori logici

È possibile combinare due o più condizioni. Le condizioni devono essere correlate con `AND` o `OR`, come ad esempio:

```
stipendio = 40000 AND detrazioni = 1
```

L'operatore logico NOT è utilizzato per invertire il significato, come ad esempio:

```
NOT (stipendio = 40000 AND detrazioni = 1)
```

Esempi

```
SELECT* FROM Dati _Vendite WHERE Dati_Vendite.Nome_Società
   NOT LIKE '%Università' AND Dati_Vendite.Importo > 3000

SELECT* FROM Dati _Vendite WHERE (Dati_Vendite.Nome_Società
   LIKE '%Università' OR Dati_Vendite.Importo > 3000)
   AND Dati_Vendite.ID_Venditore = 'SP-1'
```

Precedenza operatori

Più le espressioni sono complesse, più l'ordine con cui le espressioni vengono valutate è importante. Questa tabella mostra l'ordine in cui sono valutati gli operatori. Gli operatori nella prima linea sono valutati per primi, e così via. Gli operatori sulla stessa riga vengono valutati da sinistra a destra nell'espressione.

Precedenza	Operatore
1	`'-'` unario, `'+'` unario
2	^, **
3	*, /
4	+, -
5	=, <>, <, <=, >, >=, Like, Not Like, Is Null, Is Not Null, Between, In, Exists, Any, All
6	Not
7	E
8	GE

Nell'esempio che segue viene mostrata l'importanza della precedenza:

```
WHERE stipendio > 40000 OR data_assunzione > (DATE '30/01/2008') AND
settore = 'D101'
```

Poiché AND viene valutato per primo, questa query recupera i dipendenti nel reparto D101 assunti dopo il mercoledì 30 gennaio 2008, e tutti i dipendenti con uno stipendio superiore a 40.000 Euro, indipendentemente dal settore e dalla data di assunzione.

Per forzare la clausola in modo che venga valutata in un ordine diverso, usare le parentesi e racchiudere le condizioni da valutare per prime. Ad esempio:

```
WHERE (stipendio > 40000 OR data_assunzione > DATE '30/01/2008') AND
settore = 'D101'
```

Recupera i dipendenti nel settore D101 con uno stipendio superiore a 40.000 Euro o assunti dopo il mercoledì 30 gennaio 2008.

Funzioni SQL

FileMaker SQL supporta molte funzioni utilizzabili nelle espressioni. Alcune funzioni restituiscono stringhe di caratteri, alcune numeri, alcune date e altre valori che dipendono dalle condizioni incontrate dagli argomenti delle funzioni.

Funzioni aggregate

Le funzioni aggregate restituiscono un solo valore da un gruppo di record. È possibile usare una funzione aggregata come parte dell'istruzione `SELECT`, con il nome di un campo (ad esempio, `AVG(STIPENDIO)`) o in combinazione con un'espressione di colonna (ad esempio, `AVG(STIPENDIO * 1.07)`).

È possibile far precedere all'espressione di colonna l'operatore `DISTINCT` per eliminare i valori duplicati. Ad esempio:

```
COUNT (DISTINCT cognome)
```

In questo esempio vengono contati solo i valori univoci di cognome.

Funzione aggregata	Restituisce
`SUM`	Il totale dei valori di un'espressione di un campo numerico. Ad esempio, `SUM(STIPENDIO)` restituisce la somma di tutti i valori del campo stipendio.
`AVG`	La media dei valori di un'espressione di un campo numerico. Ad esempio, `AVG(STIPENDIO)` restituisce la media di tutti i valori del campo stipendio.
`COUNT`	Il numero di valori in qualsiasi espressione campo. Ad esempio, `COUNT(NOME)` restituisce il numero di valori dei nomi. Quando si usa `COUNT` con il nome di un campo, `COUNT` restituisce il numero di valori di campi non nulli. Un esempio particolare è `COUNT(*)`, che restituisce il numero di record del gruppo, compresi i record con valori nulli.
`MAX`	Il valore massimo in qualsiasi espressione campo. Ad esempio, `MAX(STIPENDIO)` restituisce il valore massimo del campo stipendio.
`MIN`	Il valore minimo in qualsiasi espressione campo. Ad esempio, `MIN(STIPENDIO)` restituisce il valore minimo del campo stipendio.

Esempi

```
SELECT SUM (Dati_Vendite.Quantità) AS agg FROM Dati_Vendite

SELECT AVG (Dati_Vendite.Quantità) AS agg FROM Dati_Vendite

SELECT COUNT (Dati_Vendite.Quantità) AS agg FROM Dati_Vendite

SELECT MAX (Dati_Vendite.Quantità) AS agg FROM Dati_Vendite
   WHERE Dati_vendite.Importo < 3000

SELECT MIN (Dati_Vendite.Quantità) AS agg FROM Dati_Vendite
   WHERE Dati_Vendite.Importo > 3000
```

Funzioni che restituiscono stringhe di caratteri

Funzioni che restituiscono stringhe di caratteri	Descrizione	Esempio
CHR	Converte un codice ASCII in una stringa da un carattere	CHR(67) restituisce **C**
CURRENT_USER	Restituisce l'ID di accesso specificato al momento della connessione	
DAYNAME	Restituisce il nome del giorno che corrisponde a una data specificata	
RTRIM	Rimuove gli spazi finali da una stringa	RTRIM(' ABC ') restituisce ' ABC'
TRIM	Rimuove gli spazi iniziali e finali da una stringa	TRIM(' ABC ') restituisce 'ABC'
LTRIM	Rimuove gli spazi iniziali da una stringa	LTRIM(' ABC') restituisce 'ABC'
MAIUSCOLO	Trasforma tutte le lettere di una stringa in maiuscole	UPPER('Allen') restituisce 'ALLEN'
MINUSCOLO	Trasforma tutte le lettere di una stringa in minuscole	LOWER('Allen') restituisce 'allen'
LEFT	Restituisce i caratteri più a sinistra di una stringa	LEFT('Mattson',3) restituisce 'Mat'
MONTHNAME	Restituisce i nomi dei mesi di calendario	
RIGHT	Restituisce i caratteri più a destra di una stringa	RIGHT('Mattson',4) restituisce 'tson'
SUBSTR SUBSTRING	Restituisce una sottostringa di una stringa, con i parametri della stringa, il primo carattere da estrarre e il numero di caratteri da estrarre (opzionale)	SUBSTR('Conrad',2,3) restituisce 'onr' SUBSTR('Conrad',2) restituisce 'onrad'
SPACE	Genera una stringa di spazi vuoti	SPACE(5) restituisce ' '
STRVAL	Converte un valore di qualsiasi tipo in una stringa di caratteri	STRVAL('Woltman') restituisce 'Woltman' STRVAL(5 * 3) restituisce '15' STRVAL(4 = 5) restituisce 'Falso' STRVAL(DATE '25/12/2008') restituisce "25/12/2008"
TIME TIMEVAL	Restituisce l'ora del giorno sotto forma di stringa	Alle 21:49 , TIME() restituisce **21:49:00**
USERNAME USER	Restituisce l'ID di accesso specificato al momento della connessione	

Nota La funzione TIME() non è più in uso. Al suo posto, utilizzare la funzione standard SQL CURRENT_TIME.

Esempi

```
SELECT CHR(67) + SPACE(1) + CHR(70) FROM Venditori

SELECT RTRIM(' ' + Venditori.ID_Venditore) AS agg FROM Venditori

SELECT TRIM(SPACE(1) + Venditori.ID_Venditore) AS agg FROM Venditori

SELECT LTRIM(' ' + Venditori.ID_Venditore) AS agg FROM Venditori

SELECT UPPER(Venditori.Venditore) AS agg FROM Venditori

SELECT LOWER(Venditori.Venditore) AS agg FROM Venditori

SELECT LEFT(Venditori.Venditore, 5) AS agg FROM Venditori

SELECT RIGHT(Venditori.Venditore, 7) AS agg FROM Venditori

SELECT SUBSTR(Venditori.ID_Venditore, 2, 2) +
SUBSTR(Venditori.ID_Venditore, 4, 2) AS agg FROM Venditori

SELECT SUBSTR(Venditori.Salesperson_ID, 2) +
SUBSTR(Venditori.ID_Venditore, 4) AS agg FROM Venditori

SELECT SPACE(2) + Venditori.ID_Venditore AS ID_Venditore FROM Venditori

SELECT STRVAL('60506') AS agg FROM Dati_Vendite WHERE
Dati_Vendite.Fattura = 1
```

Funzioni che restituiscono numeri

Funzioni che restituiscono numeri	Descrizione	Esempio
ABS	Restituisce il valore assoluto dell'espressione numerica	
ATAN	Restituisce l'arcotangente dell'argomento come angolo espresso in radianti	
ATAN2	Restituisce l'arcotangente delle coordinate x e y come angolo espresso in radianti	
CEIL CEILING	Restituisce il valore intero più piccolo, maggiore o uguale all'argomento	
DEG DEGREES	Restituisce il numero di gradi dell'argomento che è un angolo espresso in radianti	
DAY	Restituisce il giorno di una data	`DAY(DATE "30/01/2012")` restituisce **30**
DAYOFWEEK	Restituisce il giorno della settimana (1-7) di un'espressione di data	`DAYOFWEEK(DATE ' 01/05/2004')` restituisce **7**
MOD	Divide due numeri e restituisce il resto ottenuto dalla divisione	`MOD (10,3)` restituisce **1**

Funzioni che restituiscono numeri	Descrizione	Esempio
EXP	Restituisce un valore che è la base del logaritmo naturale (e) elevato alla potenza specificata dall'argomento	
FLOOR	Restituisce il valore intero più elevato, inferiore o uguale all'argomento	
HOUR	Restituisce la parte dell'ora di un valore	
INT	Restituisce la parte intera di un numero	`INT(6.4321)` restituisce **6**
LENGTH	Restituisce la lunghezza di una stringa	`LENGTH('ABC')` restituisce **3**
MONTH	Restituisce il mese di una data	`MONTH(DATE "30/01/2012")` restituisce **1**
LN	Restituisce il logaritmo naturale dell'argomento	
LOG	Restituisce il logaritmo comune dell'argomento	
MAX	Restituisce il maggiore di due numeri	`MAX(66,89)` restituisce **89**
MIN	Restituisce il minore di due numeri	`MIN(66,89)` restituisce **66**
MINUTE	Restituisce la parte dei minuti di un valore	
NUMVAL	Converte una stringa di caratteri in un numero. Se la stringa di caratteri non è un numero valido, la funzione fallisce.	`NUMVAL('123')` restituisce **123**
PI	Restituisce il valore della costante matematica pi	
RADIANS	Restituisce il numero di radianti di un argomento espresso in gradi	
ROUND	Arrotonda un numero	`ROUND (123.456,0)` restituisce **123** `ROUND (123.456,2)` restituisce **123,46** `ROUND (123.456, -2)` restituisce **100**
SECOND	Restituisce la parte dei secondi di un valore	
SIGN	Un indicatore del segno dell'argomento: -1 per negativo, 0 per 0 e 1 per positivo	
SIN	Restituisce il seno dell'argomento	
SQRT	Restituisce la radice quadrata dell'argomento	
TAN	Restituisce la tangente dell'argomento	
YEAR	Restituisce l'anno di una data	`ANNO(DATA "30/01/2013")` restituisce **2013**

Funzioni che restituiscono date

Funzioni che restituiscono date	Descrizione	Esempio
CURDATE CURRENT_DATE	Restituisce la data di oggi	
CURTIME CURRENT_TIME	Restituisce l'ora corrente	
CURTIMESTAMP CURRENT_TIMESTAMP	Restituisce il valore corrente dell'indicatore data e ora	
TIMESTAMPVAL	Converte una stringa di caratteri in un indicatore data e ora	TIMESTAMPVAL("30/01/2013 14:00:00") restituisce il suo valore indicatore data e ora.
DATE TODAY	Restituisce la data di oggi	Se la data odierna è 21/11/2013, DATE() restituisce **21/11/2013**
DATEVAL	Converte una stringa di caratteri in una data	DATEVAL("30/01/2013") restituisce **30/01/2013**

Nota La funzione DATE() non è più in uso. Al suo posto, utilizzare la funzione standard SQL CURRENT_DATE.

Funzioni condizionali

Funzioni condizionali	Descrizione	Esempio
CASE WHEN	**Formato CASE semplice** Per determinare il risultato, confrontare il valore di *input_exp* con i valori degli argomenti *value_exp*. CASE *input_exp* {WHEN *value_exp* THEN *risultato*...} [ELSE *risultato*] END	SELECT ID_Fattura, CASE Nome_Azienda WHEN 'REGNO UNITO Esportazioni' THEN ' REGNO UNITO Esportazioni Trovato' WHEN 'Fornitori arredamento casa' THEN ' Fornitori arredamento casa Trovato' ELSE 'Né REGNO UNITO Esportazioni né Fornitori arredamento casa' END, ID_Venditore FROM Dati _Vendite
	Formato CASE cercato Restituisce un risultato basato sul fatto che la condizione specificata dall'espressione WHEN sia vera. CASE {WHEN *esp_booleana* THEN *risultato...*} [ELSE *risultato*] END	SELECT ID_Fattura, Importo, CASE WHEN Importo > 3000 THEN ' sopra 3000' WHEN Importo < 1000 THEN ' sotto 3000' ELSE' tra 1000 e 3000' END, ID_Venditore FROM Dati _Vendite

Funzioni condizionali	Descrizione	Esempio
COALESCE	Restituisce il primo valore non NULLO	SELECT ID_Venditore, COALESCE(Direttore_Vendite, Venditore) FROM Venditori
NULLIF	Confronta due valori e restituisce NULL se i due valori sono uguali; in caso contrario restituisce il primo valore.	SELECT ID_Fattura, NULLIF (Importo, -1), ID_Venditore FROM Dati _Vendite

Parole chiave SQL riservate

La seguente tabella elenca le parole chiave riservate che non devono essere utilizzate come nomi di colonne, tabelle, alias o altri oggetti definiti dall'utente. Se vengono segnalati errori di sintassi, è possibile che sia stata utilzzata una di queste parole riservate. Se si vuole utilizzare una di queste parole chiave, è necessario utilizzare le virgolette per evitare che queste siano considerate come parole chiave.

Ad esempio, la seguente istruzione `CREATE TABLE` illustra come utilizzare la parola chiave `DEC` come un nome elemento dati.

```
create table t ("dec" numerico)
```

ABSOLUTE	CHAR	CURTIME
ACTION	CHARACTER	CURTIMESTAMP
ADD	CHARACTER_LENGTH	DATA
TUTTO	CHAR_LENGTH	DATEVAL
ALLOCATE	CHECK	DAY
ALTER	CHR	DAYNAME
E	CLOSE	DAYOFWEEK
ANY	COALESCE	DEALLOCATE
ARE	COLLATE	DEC
AS	COLLATION	DECIMAL
ASC	COLUMN	DECLARE
ASSERTION	COMMIT	DEFAULT
AT	CONNECT	DEFERRABLE
AUTHORIZATION	CONNECTION	DEFERRED
AVG	CONSTRAINT	DELETE
BEGIN	CONSTRAINTS	DESC
BETWEEN	CONTINUE	DESCRIBE
BINARY	CONVERT	DESCRIPTOR
BIT	CORRESPONDING	DIAGNOSTICS
BIT_LENGTH	COUNT	DISCONNECT
BLOB	CREATE	DISTINCT
BOOLEAN	CROSS	DOMAIN
BOTH	CURDATE	DOUBLE
BY	CURRENT	DROP
CASCADE	CURRENT_DATE	ELSE
CASCADED	CURRENT_TIME	END
CASE	CURRENT_TIMESTAMP	END_EXEC
CAST	CURRENT_USER	ESCAPE
CATALOG	CURSOR	EVERY

EXCEPT
EXCEPTION
EXEC
EXECUTE
EXISTS
EXTERNAL
EXTRACT
FALSO
FETCH
FIRST
FLOAT
FOR
FOREIGN
FOUND
FROM
FULL
GET
GLOBAL
GO
GOTO
GRANT
GROUP
HAVING
HOUR
IDENTITY
IMMEDIATE
IN
INDEX
INDICATOR
INITIALLY
INNER
INPUT
INSENSITIVE
NSERT
INT
INTEGER
INTERSECT
INTERVAL
INTO
IS
ISOLATION
JOIN
KEY
LANGUAGE
LAST
LEADING
LEFT
LENGTH
LEVEL
LIKE
LOCAL
LONGVARBINARY
MINUSCOLO
LTRIM
MATCH
MAX
MIN
MINUTE
MODULE
MONTH
MONTHNAME
NAMES
NATIONAL
NATURAL
NCHAR
NEXT
NO
NOT
NULL
NULLIF
NUMERIC
NUMVAL
OCTET_LENGTH
OF
OFFSET
ON
ONLY
OPEN
OPTION
GE
ORDER
OUTER
OUTPUT
OVERLAPS
PAD
PART
PARTIAL
PERCENT
POSITION
PRECISION
PREPARE
PRESERVE
PRIMARY
PRIOR
PRIVILEGES
PROCEDURE
PUBLIC
READ
REAL
REFERENCES
RELATIVE
RESTRICT
REVOKE
RIGHT
ROLLBACK
ROUND
ROW
ROWID
ROWS
RTRIM
SCHEMA
SCROLL
SECOND
SECTION
SELECT
SESSION
SESSION_USER

SET
SIZE
SMALLINT
SOME
SPACE
SQL
SQLCODE
SQLERROR
SQLSTATE
STRVAL
SUBSTRING
SUM
SYSTEM_USER
TABLE
TEMPORARY
THEN
TIES
TIME
TIMESTAMP
TIMESTAMPVAL
TIMEVAL
TIMEZONE_HOUR
TIMEZONE_MINUTE
TO
TODAY
TRAILING
TRANSACTION
TRANSLATE
TRANSLATION
TRIM
VERO
UNION
UNIQUE
UNKNOWN
UPDATE
UPPER
USAGE
USER
USERNAME
USING
VALUE
VALUES
VARBINARY
VARCHAR
VARYING
VIEW
WHEN
WHENEVER
WHERE
WITH
WORK
WRITE
YEAR
ZONE

Indice

G

H

I

L

N

O

P

R

S

T

U

V

W