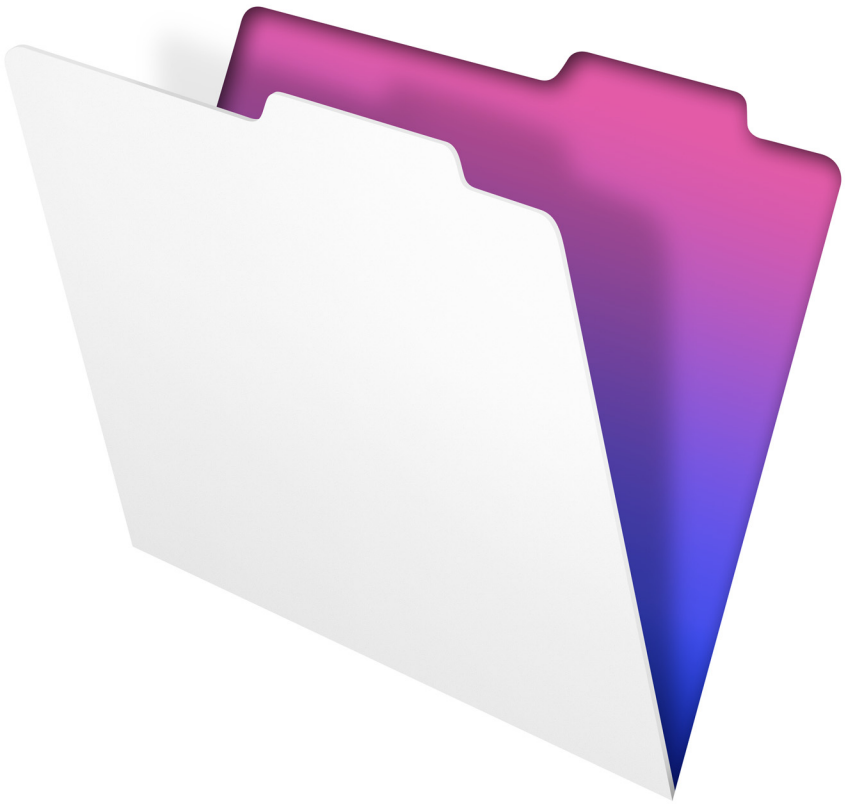
# FileMaker® 13

## Referência SQL



FileMaker, Inc.
5201 Patrick Henry Drive
Santa Clara, Califórnia 95054



Para obter mais informações, visite nosso site em http://www.filemaker.com/br.

Edição: 01

# Conteúdo

# Capítulo 1
# Introdução

Como desenvolvedor do banco de dados, você pode usar o FileMaker Pro para criar soluções de banco de dados sem saber SQL. Mas se tiver algum conhecimento de SQL, pode usar um arquivo de banco de dados do FileMaker como uma fonte de dados ODBC ou JDBC, compartilhando os dados com outros aplicativos via ODBC e JDBC. É possível também usar a função ExecuteSQL do FileMaker Pro para recuperar dados de qualquer ocorrência de tabela em um banco de dados do FileMaker Pro.

Esta referência descreve instruções e padrões SQL permitidos pelo FileMaker. Os drivers cliente ODBC e JDBC do FileMaker oferecem suporte a todas as instruções SQL descritas nesta referência. A função ExecuteSQL do FileMaker Pro oferece suporte somente à instrução SELECT.

## Sobre esta referência

- Para obter informações sobre como usar ODBC e JDBC com as versões anteriores do FileMaker Pro, consulte http://www.filemaker.com/br/support/.
- Esta referência assume que você está familiarizado com as noções básicas para uso de funções do FileMaker Pro, codificação de aplicativos ODBC e JDBC e criação de consultas SQL. Consulte um livro de outro fornecedor para obter mais informações sobre esses tópicos.
- Esta referência usa “FileMaker Pro” para referir-se ao FileMaker Pro e ao FileMaker Pro Advanced, exceto ao descrever recursos específicos do FileMaker Pro Advanced.

### Onde localizar a documentação em PDF

Para acessar PDFs da documentação do FileMaker:

- No FileMaker Pro, escolha o menu **Ajuda** > **Documentação do produto.**
- No FileMaker Server, escolha o menu **Ajuda** > **Documentação do produto.**
- Acesse http://www.filemaker.com/br/support/ para obter documentação adicional Qualquer atualização para este documento também está disponível no site.

## Sobre SQL

SQL, ou Structured Query Language (Linguagem de consulta estruturada), é uma linguagem de programação projetada para consultar dados de um banco de dados relacional. A instrução primária usada para consultar um banco de dados é a instrução SELECT.

Além de ser uma linguagem de consulta de banco de dados, a SQL fornece instruções para manipular dados, o que permite adicionar, atualizar e excluir dados.

A SQL também fornece instruções para realizar definição de dados. Essas instruções permitem que você crie e modifique tabelas e índices.

As instruções e os padrões SQL com suporte ao FileMaker estão descritas no capítulo 2, “Padrões suportados”.

## Uso de um banco de dados do FileMaker como fonte de dados

Ao hospedar um banco de dados do FileMaker como uma fonte de dados ODBC ou JDBC, os dados do FileMaker podem ser compartilhados com aplicativos compatíveis com ODBC e JDBC. Os aplicativos se conectam à fonte de dados do FileMaker usando drivers cliente do FileMaker, criam e executam as consultas SQL usando ODBC ou JDBC e processam os dados recuperados da solução de banco de dados do FileMaker.

Consulte o *Guia ODBC e JDBC do FileMaker* para obter mais informações sobre como usar o software FileMaker como uma fonte de dados para aplicativos ODBC e JDBC.

Os drivers cliente ODBC e JDBC do FileMaker oferecem suporte a todas as instruções SQL descritas nesta referência.

## Uso da função ExecuteSQL

A função ExecuteSQL do FileMaker Pro permite recuperar dados de ocorrências de tabela nomeados no gráfico de relacionamentos, mas independente de quaisquer relacionamentos definidos. Você pode recuperar dados de várias tabelas sem criar associações de tabela ou relacionamentos entre as tabelas. Em alguns casos, talvez seja possível reduzir a complexidade do gráfico de relacionamentos usando a função ExecuteSQL.

Os campos de consulta com a função ExecuteSQL não precisam estar em todos os layouts, de modo que você pode usar a função ExecuteSQL para recuperar dados independentes de qualquer contexto de layout. Graças a essa independência de contexto, usar a função ExecuteSQL em scripts pode aprimorar a portabilidade dos scripts. Você pode usar a função ExecuteSQL em qualquer local em que possa especificar cálculos, inclusive para criação de gráficos e relatórios.

A função ExecuteSQL oferece suporte somente à instrução SELECT, descrita na seção “Instrução SELECT” na página 7.

Além disso, a função ExecuteSQL aceita apenas os formatos de data e hora ISO da sintaxe SQL-92 sem chaves ({}). A função ExecuteSQL não aceita o formato ODBC/JDBC para constantes de data, hora e carimbo de data/hora em chaves.

Para obter informações sobre a sintaxe e o uso da função ExecuteSQL, consulte a Ajuda do FileMaker Pro.

# Capítulo 2
# Padrões suportados

Esta referência descreve instruções e criações SQL suportadas pelo FileMaker. Os drivers cliente ODBC e JDBC do FileMaker oferecem suporte a todas as instruções SQL descritas neste capítulo. A função ExecuteSQL do FileMaker Pro oferece suporte somente à instrução SELECT.

Use-os para acessar uma solução de banco de dados do FileMaker em um aplicativo compatível com ODBC ou JDBC. A solução de banco de dados do FileMaker pode ser hospedada pelo FileMaker Pro ou pelo FileMaker Server.

- O driver cliente ODBC oferece suporte a ODBC 3.5 Nível 1 com alguns recursos do Nível 2.
- O driver cliente JDBC oferece suporte parcial à especificação JDBC 3.0.
- Os drivers cliente ODBC e JDBC suportam a conformidade de nível de entrada da SQL-92, com alguns recursos intermediários da SQL-92.

## Suporte a caracteres Unicode

Os drivers cliente ODBC e JDBC suportam a API Unicode. No entanto, se você estiver criando um aplicativo personalizado que usa os drivers cliente, use ASCII para nomes de campo, nomes de tabela e nomes de arquivo (caso uma ferramenta ou um aplicativo de consulta não Unicode seja usado).

Nota Para inserir e recupera dados Unicode, use `SQL_C_WCHAR`.

## instruções SQL

Os drivers cliente ODBC e JDBC oferecem suporte às seguintes instruções SQL:

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

Os drivers cliente também suportam o mapeamento de tipo de dados do FileMaker para tipos de dados SQL ODBC e JDBC. Consulte o *Guia ODBC e JDBC do FileMaker* para conversões de tipo de dados. Para obter mais informações sobre como criar consultas SQL, consulte o manual de outro fornecedor.

Nota Os drivers cliente ODBC e JDBC não suportam os portais doFileMaker.

## Instrução SELECT

Use a instrução `SELECT` para especificar quais colunas estão sendo solicitadas. Siga a instrução `SELECT` com as expressões de coluna (similares aos nomes de campo) que você deseja recuperar (por exemplo, `last_name`). As expressões podem incluir operações matemáticas ou manipulação de cadeias (por exemplo, `SALARY * 1.05`).

A instrução `SELECT` pode usar diversas cláusulas:

```
SELECT [DISTINCT] {* | column_expression [[AS] column_alias],...}
FROM table_name [table_alias], ...
[ WHERE expr1 rel_operator expr2 ]
[ GROUP BY {column_expression, ...} ]
[ HAVING expr1 rel_operator expr2 ]
[ UNION [ALL] (SELECT...) ]
[ ORDER BY {sort_expression [DESC | ASC]}, ... ]
[ OFFSET n {ROWS | ROW} ]
[ FETCH FIRST [ n [ PERCENT ] ] { ROWS | ROW } {ONLY | WITH TIES } ]
[ FOR UPDATE [OF {column_expression, ...}] ]
```

Os itens entre colchetes são opcionais.

`column_alias` pode ser usado para atribuir à coluna um nome mais descritivo ou abreviar um nome de coluna mais longo. Por exemplo, para atribuir o alias `department` à coluna `dept`:

```
SELECT dept AS department FROM emp
```

Os nomes de campo podem ser prefixados com o nome de tabela ou o alias de tabela. Por exemplo, `EMP.LAST_NAME` ou `E.LAST_NAME`, em que `E` é o alias da tabela `EMP`.

O operador `DISTINCT` pode preceder a primeira expressão de coluna. Esse operador elimina as linhas duplicadas do resultado de uma consulta. Por exemplo:

```
SELECT DISTINCT dept FROM emp
```

## Cláusulas SQL

Os drivers cliente ODBC e JDBC oferecem suporte às seguintes cláusulas SQL.

| Use esta cláusula SQL | Para |
|---|---|
| FROM (página 8) | Indicar quais tabelas são usadas na instrução `SELECT`. |
| WHERE (página 9) | Especificar as condições que os registros devem atender para serem recuperados (como uma solicitação de busca do FileMaker Pro). |
| GROUP BY (página 10) | Especificar os nomes de um ou mais campos que servirão como base para o agrupamento dos valores retornados. Essa cláusula é usada para retornar um conjunto de valores agregados retornando uma linha de cada grupo (como um sub-resumo do FileMaker Pro). |
| HAVING (página 10) | Especificar condições de grupos de registros (por exemplo, exibir somente os departamentos com salários que totalizam mais de US$ 200.000). |
| UNION (página 11) | Combinar os resultados de duas ou mais instruções `SELECT` em um único resultado. |
| ORDER BY (página 11) | Indicar como os registros são classificados. |
| OFFSET (página 12) | Informar o número de linhas a serem ignoradas antes de começar a recuperar linhas. |
| FETCH FIRST (página 12) | Especificar o número de linhas a serem recuperadas. Não mais do que o número especificado de linhas é recuperado, embora menos linhas possam ser retornadas se o resultado da consulta for menor que o número de linhas especificado. |
| FOR UPDATE (página 13) | Executar atualizações ou exclusões posicionadas por meio de cursores SQL. |

Nota Se você tentar recuperar dados de uma tabela sem colunas, a instrução `SELECT` não retornará resultados.

## Cláusula FROM

A cláusula `FROM` indica quais tabelas são usadas na instrução `SELECT`. O formato é:

```
FROM table_name [table_alias] [, table_name [table_alias]]
```

`table_name` é o nome de uma tabela no banco de dados atual. O nome da tabela deve começar com um caractere alfabético. Se o nome da tabela começar com um caractere diferente de alfabético, coloque-o entre aspas duplas (identificador entre aspas).

`table_alias` pode ser usado para atribuir à tabela um nome mais descritivo, para abreviar um nome de tabela mais longo ou para incluir a mesma tabela na consulta mais de uma vez (por exemplo, em associações automáticas).

Nomes de campo começam com um caractere alfabético. Se o nome do campo começar com um caractere diferente de alfabético, coloque-o entre aspas duplas (identificador entre aspas). Por exemplo, a instrução `ExecuteSQL` para o campo nomeado `_LASTNAME` é:

```
SELECT "_LASTNAME" from emp
```

Os nomes de campo podem ser prefixados com o nome de tabela ou o alias de tabela. Por exemplo, dada a especificação de tabela `FROM employee E`, você pode fazer referência ao campo `LAST_NAME` como `E.LAST_NAME`. Os alias de tabela deverão ser usados se a instrução `SELECT` associar uma tabela a si mesmo. Por exemplo:

```
SELECT * FROM employee E, employee F WHERE E.manager_id =
F.employee_id
```

O sinal de igualdade (=) inclui somente as linhas correspondentes nos resultados.

Se você estiver associando mais de uma tabela e quiser descartar todas as linhas que não têm linhas correspondentes em ambas as tabelas de origem, use `INNER JOIN`. Por exemplo:

```
SELECT *
   FROM Salespeople INNER JOIN Sales_Data
   ON Salespeople.Salesperson_ID = Sales_Data.Salesperson_ID
```

Se você estiver associando duas tabelas, mas não quiser descartar as linhas da primeira tabela (a tabela “à esquerda”), use `LEFT OUTER JOIN`.

```
SELECT *
   FROM Salespeople LEFT OUTER JOIN Sales_Data
   ON Salespeople.Salesperson_ID = Sales_Data.Salesperson_ID
```

Cada linha da tabela “Salespeople” aparecerá na tabela associada.

**Notas**

- `RIGHT OUTER JOIN` não é suportada.
- `FULL OUTER JOIN` não é suportada.

## Cláusula WHERE

A cláusula `WHERE` especifica as condições que os registros devem atender para serem recuperados. A cláusula `WHERE` possui condições no formato:

```
WHERE expr1 rel_operator expr2
```

`expr1` e `expr2` podem ser nomes de campo, valores de constante ou expressões.

`rel_operator` é o operador relacional que vincula as duas expressões. Por exemplo, a instrução `SELECT` a seguir recupera os nomes dos funcionários que ganham US$ 20.000 ou mais.

```
SELECT last_name,first_name FROM emp WHERE salary >= 20000
```

A cláusula `WHERE` também pode usar expressões como:

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

Nota Se você usar nomes totalmente qualificados na lista `SELECT` (projeção), também deverá usar nomes totalmente qualificados na cláusula `WHERE` relacionada.

## Cláusula GROUP BY

A cláusula `GROUP BY` especifica os nomes de um ou mais campos que servirão como base para o agrupamento dos valores retornados. Essa cláusula é usada para retornar um conjunto de valores agregados. Ela tem o seguinte formato:

```
GROUP BY columns
```

`columns` deve corresponder à expressão de coluna usada na cláusula `SELECT`. Uma expressão de coluna pode ser um ou mais nomes de campo da tabela de banco de dados separada por vírgulas.

### Exemplo

O exemplo a seguir soma os salários de cada departamento.

```
SELECT dept_id, SUM (salary) FROM emp GROUP BY dept_id
```

Esta instrução retorna uma linha para cada ID de departamento diferente. Cada linha contém a ID do departamento e a soma dos salários dos funcionários no departamento.

## Cláusula HAVING

A cláusula `HAVING` permite que você especifique condições de grupos de registros (por exemplo, exibir somente os departamentos com salários que totalizam mais de US$ 200.000). Ela tem o seguinte formato:

```
HAVING expr1 rel_operator expr2
```

`expr1` e `expr2` podem ser nomes de campo, valores de constante ou expressões. Essas expressões não precisam corresponder a uma expressão de coluna na cláusula `SELECT`.

`rel_operator` é o operador relacional que vincula as duas expressões.

### Exemplo

O exemplo a seguir retorna somente os departamentos cujas somas salariais são superiores a US$ 200.000:

```
SELECT dept_id, SUM (salary) FROM emp
   GROUP BY dept_id HAVING SUM (salary) > 200000
```

## Operador UNION

O operador UNION combina os resultados de duas ou mais instruções SELECT em um único resultado. Esse resultado é todos os registros retornados pelas instruções SELECT. Por padrão, os registros duplicados não são retornados. Para retornar registros duplicados, use a palavra-chave ALL (UNION ALL). O formato é:

```
SELECT statement UNION [ALL] SELECT statement
```

Ao usar o operador UNION, as listas de seleção de cada instrução SELECT devem ter o mesmo número de expressões de coluna, com os mesmos tipos de dados, e devem ser especificadas na mesma ordem. Por exemplo:

```
SELECT last_name, salary, hire_date FROM emp UNION SELECT name, pay,
birth_date FROM person
```

Este exemplo tem o mesmo número de expressões de coluna, e cada expressão de coluna, na ordem, tem o mesmo tipo de dados.

O exemplo a seguir não é válido porque os tipos de dados das expressões de coluna são diferentes (SALARY em EMP tem um tipo de dados diferente de LAST_NAME em RAISES). Esse exemplo tem o mesmo número de expressões de coluna em cada instrução SELECT, mas as expressões não estão na mesma ordem por tipo de dados.

```
SELECT last_name, salary FROM emp UNION SELECT salary, last_name FROM
raises
```

## Cláusula ORDER BY

A cláusula ORDER BY indica como os registros serão classificados. O formato é:

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

sort_expression pode ser nomes de campo, expressões ou o número posicional da expressão de coluna a ser usada. O padrão é realizar a classificação em ordem crescente (ASC).

Por exemplo, para realizar a classificação por last_name e depois por first_name, use uma das instruções SELECT a seguir:

```
SELECT emp_id, last_name, first_name FROM emp ORDER BY last_name,
first_name
```

ou

```
SELECT emp_id, last_name, first_name FROM emp ORDER BY 2,3
```

No segundo exemplo, last_name é a segunda expressão de coluna após SELECT; portanto, ORDER BY 2 realiza a classificação por last_name.

## Cláusulas OFFSET e FETCH FIRST

As cláusulas `OFFSET` e `FETCH FIRST` são usadas para retornar um intervalo especificado de linhas que começam com um ponto de início em particular em um conjunto de resultados. A capacidade de limitar as linhas recuperadas dos grandes conjuntos de resultados permite "percorrer" entre os dados e aprimorar a eficiência.

A cláusula `OFFSET` indica o número de linhas a ignorar antes de começar a retornar dados. Se a cláusula `OFFSET` não for usada em uma instrução `SELECT`, a linha inicial será 0. A cláusula `FETCH FIRST` especifica o número de linhas a serem retornadas, assim como um número inteiro sem sinal maior que ou igual a 1 ou como uma porcentagem, do ponto inicial indicado na cláusula `OFFSET` . Se `OFFSET` e `FETCH FIRST` forem ambas usadas em uma instrução `SELECT`, a cláusula `OFFSET` deve vir primeiro.

As cláusulas `OFFSET` e `FETCH FIRST` não são suportadas em subconsultas.

### formato OFFSET

O formato `OFFSET` é:

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

`n` é um número inteiro sem sinal. Se `n` é maior que o número de linhas retornado em um conjunto de resultados, então nada é retornado e nenhuma mensagem de erro é exibida.

`ROWS` é o mesmo que `ROW`.

### formato FETCH FIRST

O formato `FETCH FIRST` é:

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

`n` é o número de linhas a serem retornadas. O valor padrão é 1 se `n` for omitido, `n` é um número inteiro sem sinal maior que ou igual a 1, a menos que seja seguido por `PERCENT`. Se `n` for seguido por `PERCENT`, o valor pode ser uma fração positiva ou um número inteiro sem sinal.

`ROWS` é o mesmo que `ROW`.

`WITH TIES` deve ser usado com a cláusula `ORDER BY`.

A cláusula `WITH TIES` permite que mais linhas sejam retornadas do que o especificado no valor de conta `FETCH` porque as linhas pares, as que não são diferenciadas com base na cláusula `ORDER BY`, também são retornadas.

### Exemplos

Por exemplo, para retornar informações de vinte e seis linhas do conjunto de resultados classificado por `last_name` em seguida por `first_name`, use a seguinte instrução `SELECT`:

```
SELECT emp_id, last_name, first_name FROM emp ORDER BY last_name,
first_name OFFSET 25 ROWS
```

Para especificar que você deseja retornar apenas dez linhas:

```
SELECT emp_id, last_name, first_name FROM emp ORDER BY last_name,
first_name OFFSET 25 ROWS FETCH FIRST 10 ROWS ONLY
```

Para retornar as dez linhas e suas linhas pares (linhas que não são diferenciadas com base na cláusula `ORDER BY`):

```
SELECT emp_id, last_name, first_name FROM emp ORDER BY last_name,
first_name OFFSET 25 ROWS FETCH FIRST 10 ROWS WITH TIES
```

## Cláusula FOR UPDATE

A cláusula `FOR UPDATE` bloqueia os registros das atualizações ou exclusões posicionadas através dos cursores SQL. O formato é:

```
FOR UPDATE [OF column_expressions]
```

`column_expressions` é uma lista de nomes de campo na tabela de banco de dados que você pretende atualizar, separados por vírgula. `column_expressions` é opcional e, portanto, ignorado.

### Exemplo

O exemplo a seguir retorna todos os registros do banco de dados de funcionários que têm um valor de campo `SALARY` superior a US$ 20.000. Quando cada registro é retornado, ele é bloqueado. Se o registro for atualizado ou excluído, o bloqueio será mantido até que você confirme a alteração. Do contrário, o bloqueio será liberado quando você pesquisar o próximo registro.

```
SELECT * FROM emp WHERE salary > 20000
   FOR UPDATE OF last_name, first_name, salary
```

### Exemplos adicionais:

| Uso | SQL de amostra |
|---|---|
| constante de texto | `SELECT 'CatDog' FROM Salespeople` |
| constante numérica | `SELECT 999 FROM Salespeople` |
| constante de data | `SELECT DATE '2012-06-05' FROM Salespeople` |
| constante de hora | `SELECT TIME '02:49:03' FROM Salespeople` |
| constante de carimbo de data/hora | `SELECT TIMESTAMP '2012-06-05 02:49:03' FROM Salespeople` |
| coluna de texto | `SELECT Company_Name FROM Sales_Data`<br>`SELECT DISTINCT Company_Name FROM Sales_Data` |
| coluna numérica | `SELECT Amount FROM Sales_Data`<br>`SELECT DISTINCT Amount FROM Sales_Data` |
| coluna de data | `SELECT Date_Sold FROM Sales_Data`<br>`SELECT DISTINCT Date_Sold FROM Sales_Data` |
| coluna de hora | `SELECT Time_Sold FROM Sales_Data`<br>`SELECT DISTINCT Time_Sold FROM Sales_Data` |
| coluna de carimbo de data/hora | `SELECT Timestamp_Sold FROM Sales_Data`<br>`SELECT DISTINCT Timestamp_Sold FROM Sales_Data` |
| coluna BLOB[a] | `SELECT Company_Brochures FROM Sales_Data`<br>`SELECT GETAS(Company_Logo, 'JPEG') FROM Sales_Data` |
| Curinga * | `SELECT * FROM Salespeople`<br>`SELECT DISTINCT * FROM Salespeople` |

a. Um BLOB é um campo de contêiner de arquivo de banco de dados do FileMaker.

### Notas dos exemplos

Uma `column` é uma referência para um campo no arquivo de banco de dados do FileMaker. (O campo pode conter muitos valores distintos.)

O caractere curinga de asterisco (*) é a forma abreviada para “tudo”. No exemplo `SELECT * FROM Salespeople`, o resultado é todas as colunas da tabela `Salespeople`. No exemplo `SELECT DISTINCT * FROM Salespeople`, o resultado é todas as linhas exclusivas da tabela `Salespeople` (sem duplicatas).

- O FileMaker não armazena dados de cadeias vazias; portanto, as consultas nunca retornarão registros:

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

- Se você usar `SELECT` com dados binários, deve usar a função `GetAs()` para especificar o fluxo a ser retornado. Consulte a seção “Recuperação do conteúdo de um campo de contêiner: CAST() function and GetAs() function”, a seguir para obter mais informações.

### Recuperação do conteúdo de um campo de contêiner: CAST() function and GetAs() function

Você pode recuperar dados binários, informações de referência de arquivo ou dados de um tipo de arquivo específico em um campo de contêiner.

Se os dados do arquivo ou dados binários JPEG existem, a instrução `SELECT` com `GetAS(field name, 'JPEG')` recupera os dados no formato binário; caso contrário, a instrução `SELECT` com nome de campo retorna `NULL`.

Para recuperar informações de referência de arquivo de um campo de container, como o caminho para um arquivo, uma imagem ou um filme do QuickTime, use a função `CAST()` com uma instrução `SELECT`. Por exemplo:

```
SELECT CAST(Company_Brochures AS VARCHAR(NNN)) FROM Sales_Data
```

Neste exemplo, se você

- Tiver inserido um arquivo em um campo de container, usando o FileMaker Pro, mas tiver armazenado somente uma referência ao arquivo, a instrução `SELECT` recuperará as informações de referência de arquivo como tipo `SQL_VARCHAR`.
- Tiver inserido o conteúdo de um arquivo no campo de container usando o FileMaker Pro, a instrução `SELECT` recuperará o nome do arquivo.
- Tiver importado um arquivo para o campo de contêiner de outro aplicativo, a instrução `SELECT` exibirá '?' (o arquivo é exibido como **Sem nome.dat** no FileMaker Pro).

Para recuperar dados de um campo de container, use a função `GetAs()`. Você pode usar a opção `DEFAULT` ou especificar o tipo de arquivo. A opção `DEFAULT` recupera o fluxo mestre do container sem precisar definir explicitamente o tipo de fluxo:

```
SELECT GetAs(Company_Brochures, DEFAULT) FROM Sales_Data
```

Para recuperar um tipo de fluxo individual de um container, use a função `GetAs()` com o tipo de arquivo baseado em como os dados foram inseridos no campo de container do FileMaker Pro. Por exemplo:

- Se os dados tiverem sido inseridos por meio do comando **Inserir** > **Arquivo**, especifique `'FILE'` na função `GetAs()`. Por exemplo:

  ```
  SELECT GetAs(Company_Brochures, 'FILE') FROM Sales_Data
  ```

- Se os dados tiverem sido inseridos por meio do comando **Inserir** > **Som** (Som padrão — no formato raw do MAC OS X), especifique `'snd'` na função `GetAs()`. Por exemplo:

  ```
  SELECT GetAs(Company_Meeting, 'snd ') FROM Company_Newsletter
  ```

- Se os dados tiverem sido inseridos por meio do comando **Inserir** > **Imagem**, arraste e solte ou cole da Prancheta, especifique um dos tipos de arquivo listados na tabela a seguir. Por exemplo:

```
SELECT GetAs(Company_Logo, 'JPEG') FROM Company_Icons
```

| Tipo de arquivo | Descrição | Tipo de arquivo | Descrição |
|---|---|---|---|
| 'GIFf' | Graphics Interchange Format | 'PNTG' | MacPaint |
| 'JPEG' | Imagens fotográficas | '.SGI' | Formato de bitmap genérico |
| 'JP2 ' | JPEG 2000 | 'TIFF' | Formato de arquivo raster para imagens digitais |
| 'PDF ' | Portable Document Format | 'TPIC' | Targa |
| 'PNGf' | Formato de imagem de bitmap | '8BPS' | PhotoShop (PSD) |

## Instrução DELETE

Use a instrução `DELETE` para excluir registros de uma tabela de banco de dados. O formato da instrução `DELETE` é:

```
DELETE FROM table_name [ WHERE { conditions } ]
```

Nota A cláusula `WHERE` determina quais registros serão excluídos. Se você não incluir a palavra-chave `WHERE`, todos os registros da tabela serão excluídos (mas a tabela será mantida intacta).

### Exemplo

Um exemplo de uma instrução `DELETE` na tabela Employee é:

```
DELETE FROM emp WHERE emp_id = 'E10001'
```

Cada instrução `DELETE` remove todos os registros que atendem às condições na cláusula `WHERE`. Nesse caso, cada registro com a ID `E10001` será excluído. Como as IDs de funcionário são exclusivas na tabela Employee, somente um registro será excluído.

## Instrução INSERT

Use a instrução `INSERT` para criar registros em uma tabela de banco de dados. Você pode especificar:

- Uma lista de valores a serem inseridos como um novo registro
- Uma instrução `SELECT` que copia dados de outra tabela a ser inserida como um conjunto de novos registros

O formato da instrução `INSERT` é:

```
INSERT INTO table_name [(column_name, ...)] VALUES (expr, ...)
```

column_name é uma lista opcional de nomes de coluna que fornece o nome e a ordem das colunas cujos valores são especificados na cláusula VALUES. Se você omitir column_name, as expressões de valor (expr) fornecerão valores para todas as colunas definidas na tabela e estarão na mesma ordem em que as colunas são definidas para a tabela. column_name também pode especificar uma repetição de campo; por exemplo, lastDates[4].

expr é a lista de expressões que fornece os valores das colunas do novo registro. Geralmente, as expressões são valores de constante das colunas (mas elas também podem ser uma subconsulta). Você deve colocar os valores de cadeia de caracteres entre aspas simples ('). Para incluir uma aspa simples em um valor de cadeia de caracteres que já está entre aspas simples, use duas aspas simples juntas (por exemplo, 'Don''t').

As subconsultas devem ser colocadas entre parênteses.

O exemplo a seguir insere uma lista de expressões:

```
INSERT INTO emp (last_name, first_name, emp_id, salary, hire_date)
   VALUES ('Smith', 'John', 'E22345', 27500, DATE '05-06-2013')
```

Cada instrução INSERT adiciona um registro à tabela de banco de dados. Nesse caso, um registro foi adicionado à tabela de banco de dados de funcionário, EMP. São especificados valores para cinco colunas. As colunas restantes da tabela recebem um valor em branco, que significa nulo.

Nota Nos campos de contêiner, você pode inserir somente texto usando a instruçãoINSERT, a menos que prepare uma instrução parametrizada e obtenha os dados no aplicativo. Para usar dados binários, basta atribuir o nome de arquivo colocando-o entre aspas simples ou usar a função PutAs(). Ao especificar o nome de arquivo, o tipo de arquivo é deduzido da extensão do arquivo:

```
INSERT INTO table_name (container_name) VALUES(? AS 'filename.file
extension')
```

Tipos de arquivo não suportados serão inseridos como tipo FILE.

Ao usar a função PutAs(), especifique o tipo: PutAs(col, 'type'), em que o valor de tipo é um tipo de arquivo suportado conforme descrito em “Recuperação do conteúdo de um campo de contêiner: CAST() function and GetAs() function” na página 15.

A instrução SELECT é uma consulta que retorna valores para cada valor column_name especificado na lista de nomes de coluna. O uso de uma instrução SELECT em vez de uma lista de expressões de valor permite que você selecione um conjunto de linhas em uma tabela e insira-o em outra tabela usando uma instrução INSERT.

Este é um exemplo de instrução INSERT que usa uma instrução SELECT:

```
INSERT INTO emp1 (first_name, last_name, emp_id, dept, salary)
   SELECT first_name, last_name, emp_id, dept, salary from emp
   WHERE dept = 'D050'
```

Nesse tipo de instrução INSERT, o número de colunas a serem inseridas deve corresponder ao número de colunas na instrução SELECT. A lista de colunas a serem inseridas deve corresponder às colunas da instrução SELECT exatamente como seria em uma lista de expressões de valor no outro tipo de instrução INSERT. Por exemplo, a primeira coluna inserida corresponde à primeira coluna selecionada; a segunda coluna inserida corresponde à segunda coluna selecionada e assim sucessivamente.

O tamanho e o tipo de dados dessas colunas correspondentes devem ser compatíveis. Cada coluna da lista `SELECT` deve ter um tipo de dados que o driver cliente ODBC ou JDBC aceite em uma instrução `INSERT`/`UPDATE` da coluna correspondente da lista `INSERT`. Os valores são truncados quando o tamanho do valor na coluna de lista `SELECT` é maior que o tamanho da coluna de lista `INSERT` correspondente.

A instrução `SELECT` é avaliada antes que qualquer valor seja inserido.

## Instrução UPDATE

Use a instrução `UPDATE` para alterar registros em uma tabela de banco de dados. O formato da instrução `UPDATE` é:

```
UPDATE table_name SET column_name = expr, ... [ WHERE { conditions } ]
```

`column_name` é o nome de uma coluna cujo valor será alterado. Várias colunas podem ser alteradas em uma única instrução.

`expr` é o novo valor da coluna.

Geralmente, as expressões são valores de constante das colunas (mas elas também podem ser uma subconsulta). Você deve colocar os valores de cadeia de caracteres entre aspas simples ('). Para incluir uma aspa simples em um valor de cadeia de caracteres que já está entre aspas simples, use duas aspas simples juntas (por exemplo, `'Don''t'`).

As subconsultas devem ser colocadas entre parênteses.

A cláusula `WHERE` é qualquer cláusula válida. Ela determina quais registros são atualizados.

### Exemplos

Um exemplo de uma instrução `UPDATE` na tabela Employee é:

```
UPDATE emp SET salary=32000, exempt=1 WHERE emp_id = 'E10001'
```

Cada instrução `UPDATE` altera todos os registros que atendem às condições na cláusula `WHERE`. Nesse caso, o status do salário e da isenção são alterados para todos os funcionários que têm a ID de funcionário `E10001`. Como as IDs de funcionário são exclusivas na tabela Employee, somente um registro será atualizado.

Este é um exemplo com uma subconsulta:

```
UPDATE emp SET salary = (SELECT avg(salary) from emp) WHERE emp_id =
'E10001'
```

Nesse caso, o salário é alterado para a média salarial da empresa para o funcionário que tem a ID de funcionário `E10001`.

Nota Nos campos de contêiner, você pode atualizar somente texto usando a instrução `UPDATE`, a menos que prepare uma instrução parametrizada e obtenha os dados no aplicativo. Para usar dados binários, basta atribuir o nome de arquivo colocando-o entre aspas simples ou usar a função `PutAs()`. Ao especificar o nome de arquivo, o tipo de arquivo é deduzido da extensão do arquivo:

```
UPDATE table_name SET (container_name) = ? AS 'filename.file
extension'
```

Tipos de arquivo não suportados serão inseridos como tipo FILE.

Ao usar a função `PutAs()`, especifique o tipo: `PutAs(col, 'type')`, em que o valor de tipo é um tipo de arquivo suportado conforme descrito em “Recuperação do conteúdo de um campo de contêiner: CAST() function and GetAs() function” na página 15.

## Instrução CREATE TABLE

Use a instrução `CREATE TABLE` para criar uma tabela em um arquivo de banco de dados. O formato da instrução `CREATE TABLE` é:

```
CREATE TABLE table_name ( table_element_list [,
table_element_list...] )
```

Na instrução, você especifica o nome e o tipo de dados de cada coluna.

- `table_name` é o nome da tabela. `table_name` tem um limite de 100 caracteres. Uma tabela com o mesmo nome ainda não deve estar definida. O nome da tabela deve começar com um caractere alfabético. Se o nome da tabela começar com um caractere diferente de alfabético, coloque-o entre aspas duplas (identificador entre aspas).
- O formato de `table_element_list` é:

  ```
  field_name field_type [DEFAULT expr]
  [UNIQUE | NOT NULL | PRIMARY KEY | GLOBAL]
  [EXTERNAL relative_path_string [SECURE | OPEN calc_path_string]]
  ```

  - `field_name` é o nome do campo. Nenhum campo na mesma tabela pode ter o mesmo nome. Especifique uma repetição de campo usando um número entre colchetes. Por exemplo: `lastDates[4]`. Nomes de campo começam com um caractere alfabético. Se o nome do campo começar com um caractere diferente de alfabético, coloque-o entre aspas duplas (identificador entre aspas). Por exemplo, a instrução `CREATE TABLE` para o campo nomeado `_LASTNAME` é:

    ```
    CREATE TABLE "_EMPLOYEE" (ID INT PRIMARY KEY, "_FIRSTNAME"
    VARCHAR(20), "_LASTNAME" VARCHAR(20))
    ```

  - `field_type` pode ser qualquer um destes itens: `NUMERIC`, `DECIMAL`, `INT`, `DATE`, `TIME`, `TIMESTAMP`, `VARCHAR`, `CHARACTER VARYING`, `BLOB`, `VARBINARY`, `LONGVARBINARY` ou `BINARY VARYING`. Para `NUMERIC` e `DECIMAL`, você pode especificar a precisão e a escala. Por exemplo: `DECIMAL(10,0)`. Para `TIME` e `TIMESTAMP`, você pode especificar a precisão. Por exemplo: `TIMESTAMP(6)`. Para `VARCHAR` e `CHARACTER VARYING`, você pode especificar o tamanho da cadeia. Por exemplo: `VARCHAR(255)`.
  - A palavra-chave `DEFAULT` permite que você defina um valor padrão para uma coluna. Para `expr`, você pode usar um valor de constante ou uma expressão. As expressões permitidas são `USER`, `USERNAME`, `CURRENT_USER`, `CURRENT_DATE`, `CURDATE`, `CURRENT_TIME`, `CURTIME`, `CURRENT_TIMESTAMP`, `CURTIMESTAMP` e `NULL`.
  - Definir uma coluna como `UNIQUE` seleciona automaticamente a opção de validação **Exclusivo** para o campo correspondente no arquivo de banco de dados do FileMaker.
  - Definir uma coluna como `NOT NULL` seleciona automaticamente a opção de validação **Não vazio** para o campo correspondente no arquivo de banco de dados do FileMaker. O campo é sinalizado como um **Valor necessário** na guia **Campos** da caixa de diálogo Gerenciar banco de dados no FileMaker Pro.

- Para definir uma coluna como um campo de contêiner, use `BLOB`, `VARBINARY` ou `BINARY VARYING` em `field_type`.
- Para definir uma coluna como um campo de contêiner que armazena dados externamente, use a palavra-chave `EXTERNAL`. `relative_path_string` define a pasta em que os dados são armazenados externamente, relativa ao local do banco de dados do FileMaker. Esse caminho deve ser especificado como diretório base na caixa de diálogo Gerenciar containers do FileMaker Pro. Você deve especificar `SECURE` para armazenamento seguro ou `OPEN` para armazenamento aberto. Se você estiver usando o armazenamento aberto, `calc_path_string` será a subpasta da pasta `relative_path_string` em que os objetos container serão armazenados. O caminho deve usar barras (/) no nome da pasta.

### Exemplos

| Uso | SQL de amostra |
|---|---|
| coluna de texto | `CREATE TABLE T1 (C1 VARCHAR, C2 VARCHAR (50), C3 VARCHAR (1001), C4 VARCHAR (500276))` |
| coluna de texto, `NOT NULL` | `CREATE TABLE T1NN (C1 VARCHAR NOT NULL, C2 VARCHAR (50) NOT NULL, C3 VARCHAR (1001) NOT NULL, C4 VARCHAR (500276) NOT NULL)` |
| coluna numérica | `CREATE TABLE T2 (C1 DECIMAL, C2 DECIMAL (10,0), C3 DECIMAL (7539,2), C4 DECIMAL (497925,301))` |
| coluna de data | `CREATE TABLE T3 (C1 DATE, C2 DATE, C3 DATE, C4 DATE)` |
| coluna de hora | `CREATE TABLE T4 (C1 TIME, C2 TIME, C3 TIME, C4 TIME)` |
| coluna de carimbo de data/hora | `CREATE TABLE T5 (C1 TIMESTAMP, C2 TIMESTAMP, C3 TIMESTAMP, C4 TIMESTAMP)` |
| coluna para campo de contêiner | `CREATE TABLE T6 (C1 BLOB, C2 BLOB, C3 BLOB, C4 BLOB)` |
| coluna para campo de contêiner de armazenamento externo | `CREATE TABLE T7 (C1 BLOB EXTERNAL 'Files/MyDatabase/' SECURE)`<br>`CREATE TABLE T8 (C1 BLOB EXTERNAL 'Files/MyDatabase/' OPEN 'Objects')` |

## Instrução ALTER TABLE

Use a instrução `ALTER TABLE` para alterar a estrutura de uma tabela existente em um arquivo de banco de dados. Você pode modificar somente uma coluna em cada instrução. Os formatos da instrução `ALTER TABLE` são:

```
ALTER TABLE table_name ADD [COLUMN] column_definition

ALTER TABLE table_name DROP [COLUMN] unqualified_column_name

ALTER TABLE table_name ALTER [COLUMN] column_definition SET DEFAULT
expr

ALTER TABLE table_name ALTER [COLUMN] column_definition DROP DEFAULT
```

Você deve conhecer a estrutura da tabela e como deseja modificá-la antes de usar a instrução `ALTER TABLE`.

### Exemplos

| Para | SQL de amostra |
|---|---|
| adicionar colunas | `ALTER TABLE Salespeople ADD C1 VARCHAR` |
| remover colunas | `ALTER TABLE Salespeople DROP C1` |
| definir o valor padrão de uma coluna | `ALTER TABLE Salespeople ALTER Company SET DEFAULT 'FileMaker'` |
| remover o valor padrão de uma coluna | `ALTER TABLE Salespeople ALTER Company DROP DEFAULT` |

Nota `SET DEFAULT` e `DROP DEFAULT` não afetam linhas existentes na tabela, mas alteram o valor padrão das linhas adicionadas subsequentemente à tabela.

## Instrução CREATE INDEX

Use a instrução `CREATE INDEX` para agilizar as pesquisas no arquivo de banco de dados. O formato da instrução `CREATE INDEX` é:

```
CREATE INDEX ON table_name.column_name
CREATE INDEX ON table_name (column_name)
```

A instrução `CREATE INDEX` é suportada em uma única coluna (índices de várias colunas não são suportados). Os índices não são permitidos em colunas que correspondem a tipos de campo de contêiner, campos de resumo, campos que têm a opção de armazenamento global ou campos de cálculo não armazenados em um arquivo de banco de dados do FileMaker.

A criação de um índice para uma coluna de texto seleciona automaticamente a opção de armazenamento **Mínimo** em **Indexação** para o campo correspondente no arquivo de banco de dados do FileMaker. A criação de um índice para uma coluna que não é de texto (ou uma coluna formatada para texto em japonês) seleciona automaticamente a opção de armazenamento **Tudo** em **Indexação** para o campo correspondente no arquivo de banco de dados do FileMaker.

A criação de um índice para qualquer coluna seleciona automaticamente a opção de armazenamento **Criar índices automaticamente conforme necessário** em **Indexação** para o campo correspondente no arquivo de banco de dados do FileMaker.

O FileMaker cria índices automaticamente quando necessário. O uso de `CREATE INDEX` faz com que o índice seja criado imediatamente, e não sob demanda.

### Exemplo

```
CREATE INDEX ON Salespeople.Salesperson_ID
```

## Instrução DROP INDEX

Use a instrução `DROP INDEX` para remover um índice de um arquivo de banco de dados. O formato da instrução `DROP INDEX` é:

```
DROP INDEX ON table_name.column_name
DROP INDEX ON table_name (column_name)
```

Remova um índice quando o arquivo de banco de dados for muito grande ou quando você não usar um campo com frequência nas consultas.

Se as consultas apresentarem desempenho insatisfatório e você estiver trabalhando com um arquivo de banco de dados do FileMaker extremamente grande, com vários campos de texto indexados, é recomendável remover os índices de alguns campos. Além disso, considere a remoção dos índices dos campos que você raramente usa nas instruções `SELECT`.

A remoção de um índice para qualquer coluna seleciona automaticamente a opção de armazenamento **Nenhum** e desmarca **Criar índices automaticamente conforme necessário** em **Indexação** para o campo correspondente no arquivo de banco de dados do FileMaker.

O atributo `PREVENT INDEX CREATION` não é suportado.

**Exemplo**

```
DROP INDEX ON Salespeople.Salesperson_ID
```

# Expressões SQL

Use expressões nas cláusulas `WHERE`, `HAVING` e `ORDER BY` das instruções `SELECT` para formar consultas de banco de dados detalhadas e sofisticadas. Os elementos de expressão válidos são:

- Nomes de campo
- Constantes
- Notação exponencial/científica
- Operadores numéricos
- Operadores de caractere
- Operadores de data
- Operadores relacionais
- Operadores lógicos
- Funções

## Nomes de campo

A maioria das expressões comuns são um nome de campo simples, como `calc` ou `Sales_Data.Invoice_ID`.

## Constantes

As constantes são valores que não são alterados. Por exemplo, na expressão `PRICE * 1.05`, o valor 1.05 é uma constante. Ou você poderia atribuir o valor 30 à constante `Number_Of_Days_In_June`.

Você deve colocar as constantes de caractere entre aspas simples ('). Para incluir uma aspa simples em uma constante de caractere que já está entre aspas simples, use duas aspas simples juntas (por exemplo, `'Don''t'`).

Para aplicativos ODBC e JDBC, o FileMaker aceita as constantes de data, hora e carimbo de data/hora ODBC/JDBC entre chaves ({}), por exemplo:

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

O FileMaker permite que o especificador de tipo (`D`, `T`, `TS`) esteja em maiúsculas ou minúsculas. Você pode usar qualquer quantidade de espaços após os especificados de tipo ou, até mesmo, omitir o espaço.

O FileMaker também aceita os formatos de data e hora ISO da sintaxe SQL-92 sem chaves:

- `DATE 'YYYY-MM-DD'`
- `TIME 'HH:MM:SS'`
- `TIMESTAMP 'YYYY-MM-DD HH:MM:SS'`

A função ExecuteSQL do FileMaker Pro aceita apenas os formatos de data e hora ISO da sintaxe SQL-92 sem chaves.

| Constante | Sintaxe aceitável (exemplos) |
|---|---|
| Texto | `'Paris'` |
| Número | `1,05` |
| Data | `DATE '05-06-2012'`<br>`{ D '2012-06-05' }`<br>`{05/06/2012}`<br>`{05/06/12}`<br>Nota A sintaxe de ano de dois dígitos não é suportada no formato ODBC/JDBC ou SQL-92. |
| Hora | `TIME '14:35:10'`<br>`{ T '14:35:10' }`<br>`{14:35:10}` |
| Carimbo de data/hora | `TIMESTAMP '05-06-2012 14:35:10'`<br>`{ TS '2012-06-05 14:35:10'}`<br>`{05/06/2012 14:35:10}`<br>`{05/06/12 14:35:10}`<br>Verifique se **Tipo de dados rígido: Data com ano de 4 dígitos** não está selecionada como uma opção de validação no arquivo de banco de dados do FileMaker para um campo que usa a sintaxe de ano de dois dígitos.<br>Nota A sintaxe de ano de dois dígitos não é suportada no formato ODBC/JDBC ou SQL-92. |

Ao inserir valores de data e hora, faça a correspondência do formato da localidade do arquivo de banco de dados. Por exemplo, se o banco de dados tiver sido criado em um sistema com idioma italiano, use formatos de data e hora italianos.

## Notação exponencial/científica

Os números podem ser expressos por meio de notação científica.

### Exemplo

```
SELECT column1 / 3.4E+7 FROM table1 WHERE calc < 3.4E-6 * column2
```

## Operadores numéricos

Você pode incluir os seguintes operadores em expressões numéricas: +, -, *, /, e ^ ou ** (exponenciação).

Você pode preceder expressões numéricas com um sinal de adição (+) ou subtração (-) unário.

## Operadores de caractere

Você pode concatenar caracteres.

### Exemplos

Nos exemplos a seguir, `last_name` é `'JONES '` e first_name é `'ROBERT '`:

| Operador | Concatenação | Exemplo | Resultado |
|---|---|---|---|
| + | Mantém os caracteres em branco à direita | first_name + last_name | 'ROBERT JONES ' |
| - | Move os caracteres em branco à direita para o fim | first_name - last_name | 'ROBERTJONES ' |

## Operadores de data

Você pode modificar datas.

### Exemplos

Nos exemplos a seguir, `hire_date` é `DATE '30-01-2013'`.

| Operador | Efeito na data | Exemplo | Resultado |
|---|---|---|---|
| + | Adiciona um número de dias a uma data | `hire_date` + 5 | DATE '04-02-2013' |
| - | Encontra o número de dias entre duas datas | `hire_date - DATE '01-01-2013'` | 29 |
| | Subtrai um número de dias de uma data | `hire_date - 10` | DATE '20/01/2013' |

Exemplos adicionais:

```
SELECT Date_Sold, Date_Sold + 30 AS agg FROM Sales_Data
SELECT Date_Sold, Date_Sold - 30 AS agg FROM Sales_Data
```

## Operadores relacionais

| Operador | Significado |
|---|---|
| = | Igual a |
| <> | Diferente de |
| > | Maior que |
| >= | Maior ou igual a |
| < | Menor que |
| <= | Menor ou igual a |
| LIKE | Corresponde a um padrão |
| NOT LIKE | Não corresponde a um padrão |
| IS NULL | Igual a nulo |
| IS NOT NULL | Diferente de nulo |
| BETWEEN | Intervalo de valores entre um limite inferior e superior |
| IN | Um membro de um conjunto de valores especificados ou um membro de uma subconsulta |

| Operador | Significado |
|---|---|
| NOT IN | Não é um membro de um conjunto de valores especificados ou um membro de uma subconsulta |
| EXISTS | 'True' se uma subconsulta tiver retornado pelo menos um registro |
| ANY | Compara um valor com cada valor retornado por uma subconsulta (o operador deve ser precedido por =, <>, >, >=, < ou <=); =Any é equivalente a In |
| ALL | Compara um valor com cada valor retornado por uma subconsulta (o operador deve ser precedido por =, <>, >, >=, < ou <=) |

## Exemplos

```
SELECT Sales_Data.Invoice_ID FROM Sales_Data
   WHERE Sales_Data.Salesperson_ID = 'SP-1'

SELECT Sales_Data.Amount FROM Sales_Data WHERE Sales_Data.
Invoice_ID <> 125

SELECT Sales_Data.Amount FROM Sales_Data WHERE Sales_Data.Amount > 3000

SELECT Sales_Data.Time_Sold FROM Sales_Data
   WHERE Sales_Data.Time_Sold < '12:00:00'

SELECT Sales_Data.Company_Name FROM Sales_Data
   WHERE Sales_Data.Company_Name LIKE '%University'

SELECT Sales_Data.Company_Name FROM Sales_Data
   WHERE Sales_Data.Company_Name NOT LIKE '%University'

SELECT Sales_Data.Amount FROM Sales_Data WHERE Sales_Data.Amount IS NULL

SELECT Sales_Data.Amount FROM Sales_Data WHERE Sales_Data.Amount IS NOT
NULL

SELECT Sales_Data.Invoice_ID FROM Sales_Data
   WHERE Sales_Data.Invoice_ID BETWEEN 1 AND 10

SELECT COUNT(Sales_Data.Invoice_ID) AS agg
   FROM Sales_Data WHERE Sales_Data.INVOICE_ID IN (50,250,100)

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

SELECT COUNT(Sales_Data.Invoice_ID) AS agg FROM Sales_Data
   WHERE Sales_Data.INVOICE_ID NOT IN (SELECT Sales_Data.Invoice_ID
   FROM Sales_Data WHERE Sales_Data.Salesperson_ID = 'SP-4')

SELECT *
   FROM Sales_Data WHERE EXISTS (SELECT Sales_Data.Amount
   FROM Sales_Data WHERE Sales_Data.Salesperson_ID IS NOT NULL)
```

```
SELECT *
   FROM Sales_Data WHERE Sales_Data.Amount = ANY (SELECT
Sales_Data.Amount
   FROM Sales_Data WHERE Sales_Data.Salesperson_ID = 'SP-1')

SELECT *
   FROM Sales_Data WHERE Sales_Data.Amount = ALL (SELECT
Sales_Data.Amount
   FROM Sales_Data WHERE Sales_Data.Salesperson_ID IS NULL)
```

## Operadores lógicos

Você combina duas ou mais condições. As condições devem ser relacionadas por `AND` ou `OR`; por exemplo:

```
salary = 40000 AND exempt = 1
```

O operador lógico NOT é usado para reverter o significado; por exemplo:

```
NOT (salary = 40000 AND exempt = 1)
```

### Exemplos

```
SELECT * FROM Sales_Data WHERE Sales_Data.Company_Name
   NOT LIKE '%University' AND Sales_Data.Amount > 3000

SELECT * FROM Sales_Data WHERE (Sales_Data.Company_Name
   LIKE '%University' OR Sales_Data.Amount > 3000)
   AND Sales_Data.Salesperson_ID = 'SP-1'
```

## Precedência do operador

À medida que as expressões se tornam mais complexas, a ordem em que elas são avaliadas se tornam mais importantes. Esta tabela mostra a ordem em que os operadores são avaliados. Os operadores da primeira linha são avaliados primeiro e assim por diante. Os operadores da mesma linha são avaliados da esquerda para a direita na expressão.

| Precedência | Operador |
|---|---|
| 1 | Unary `'-'`, Unary `'+'` |
| 2 | ^, ** |
| 3 | *, / |
| 4 | +, - |
| 5 | =, <>, <, <=, >, >=, Like, Not Like, Is Null, Is Not Null, Between, In, Exists, Any, All |
| 6 | Not |
| 7 | AND |
| 8 | OR |

O exemplo a seguir mostra a importância da precedência:

```
WHERE salary > 40000 OR hire_date > (DATE '30-01-2008') AND dept =
'D101'
```

Como AND é avaliado primeiro, essa consulta recupera os funcionários do departamento D101 admitidos após 30 de janeiro de 2008, assim como cada funcionário que ganha mais de US$ 40.000, independentemente do departamento ou da data de admissão.

Para forçar a avaliação da cláusula em uma ordem diferente, use parênteses para delimitar as condições que serão avaliadas primeiro. Por exemplo:

```
WHERE (salary > 40000 OR hire_date > DATE '30-01-2008') AND dept =
'D101'
```

recupera os funcionários do departamento D101 que ganham mais de US$ 40.000 ou que foram admitidos após 30 de janeiro de 2008.

# funções SQL

A SQL do FileMaker suporta várias funções que você pode usar nas expressões. Algumas funções retornam cadeias de caracteres, algumas retornam números, algumas retornam datas e algumas retornam valores que dependem das condições atendidas pelos argumentos da função.

## Funções de agregação

As funções agregadas retornam um valor único em um conjunto de registros. Você pode usar uma função agregada como parte de uma instrução `SELECT`, com um nome de campo (por exemplo, `AVG(SALARY)`) ou em conjunto com uma expressão de coluna (por exemplo, `AVG(SALARY * 1.07)`).

Você pode preceder a expressão de coluna com o operador `DISTINCT` para eliminar valores duplicados. Por exemplo:

```
COUNT (DISTINCT last_name)
```

Neste exemplo, somente os valores de sobrenome exclusivos são contabilizados.

| Função de agregação | Retorna |
|---|---|
| `SUM` | O total dos valores em uma expressão de campo numérica. Por exemplo, `SUM(SALARY)` retorna a soma de todos os valores de campo de salário. |
| `AVG` | A média dos valores em uma expressão de campo numérica. Por exemplo, `AVG(SALARY)` retorna a média de todos os valores de campo de salário. |
| `COUNT` | O número de valores em qualquer expressão de campo. Por exemplo, `COUNT(NAME)` retorna o número de valores de nome. Quando a função `COUNT` é usada com um nome de campo, ela retorna o número de valores de campo não nulos. Um exemplo especial é `COUNT(*)`, que retorna o número de registros no conjunto, incluindo os registros com valores nulos. |
| `MAX` | O valor máximo em qualquer expressão de campo. Por exemplo, `MAX(SALARY)` retorna o valor de campo de salário máximo. |
| `MIN` | O valor mínimo em qualquer expressão de campo. Por exemplo, `MIN(SALARY)` retorna o valor de campo de salário mínimo. |

### Exemplos

```
SELECT SUM (Sales_Data.Amount) AS agg FROM Sales_Data

SELECT AVG (Sales_Data.Amount) AS agg FROM Sales_Data

SELECT COUNT (Sales_Data.Amount) AS agg FROM Sales_Data

SELECT MAX (Sales_Data.Amount) AS agg FROM Sales_Data
   WHERE Sales_Data.Amount < 3000

SELECT MIN (Sales_Data.Amount) AS agg FROM Sales_Data
   WHERE Sales_Data.Amount > 3000
```

## Funções que retornam cadeias de caracteres

| Funções que retornam cadeias de caracteres | Descrição | Exemplo |
|---|---|---|
| CHR | Converte um código ASCII em uma cadeia de um caractere | CHR(67) retorna **C** |
| CURRENT_USER | Retorna a ID de login especificada no momento da conexão | |
| DAYNAME | Retorna o nome do dia que corresponde a uma data especificada | |
| RTRIM | Remove os espaços em branco à direita de uma cadeia | RTRIM('  ABC ') retorna '  ABC' |
| TRIM | Remove os espaços em branco à esquerda e à direita de uma cadeia | TRIM(' ABC ') retorna 'ABC' |
| LTRIM | Remove os espaços em branco à esquerda de uma cadeia | LTRIM(' ABC') retorna 'ABC' |
| UPPER | Altera cada letra de uma cadeia para maiúscula | UPPER('Allen') retorna 'ALLEN' |
| LOWER | Altera cada letra de uma cadeia para minúscula | LOWER('Allen') retorna 'allen' |
| LEFT | Retorna os caracteres da extrema esquerda de uma cadeia | LEFT('Mattson',3) retorna 'Mat' |
| MONTHNAME | Retorna os nomes do mês do calendário | |
| RIGHT | Retorna os caracteres da extrema direita de uma cadeia | RIGHT('Mattson',4) retorna 'tson' |
| SUBSTR<br>SUBSTRING | Retorna uma subcadeia de uma cadeia, com os parâmetros da cadeia, o primeiro caractere a ser extraído e o número de caracteres a ser extraído (opcional) | SUBSTR('Conrad',2,3) retorna 'onr'<br>SUBSTR('Conrad',2) retorna 'onrad' |
| SPACE | Gera uma cadeia de espaços em branco | SPACE(5) retorna '  ' |
| STRVAL | Converte um valor de qualquer tipo em uma cadeia de caracteres | STRVAL('Woltman') retorna 'Woltman'<br>STRVAL(5 * 3) retorna '15'<br>STRVAL(4 = 5) retorna 'False'<br>STRVAL(DATE '25-12-2008') retorna '25-12-2008' |
| TIME<br>TIMEVAL | Retorna a hora do dia como uma cadeia | Às 9:49 PM, TIME() retorna **21:49:00** |
| USERNAME<br>USER | Retorna a ID de login especificada no momento da conexão | |

Nota A função TIME() foi preterida. Use o padrão SQL CURRENT_TIME.

### Exemplos

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

SELECT RTRIM(' ' + Salespeople.Salesperson_ID) AS agg FROM Salespeople

SELECT TRIM(SPACE(1) + Salespeople.Salesperson_ID) AS agg FROM
Salespeople

SELECT LTRIM(' ' + Salespeople.Salesperson_ID) AS agg FROM Salespeople

SELECT UPPER(Salespeople.Salesperson) AS agg FROM Salespeople

SELECT LOWER(Salespeople.Salesperson) AS agg FROM Salespeople

SELECT LEFT(Salespeople.Salesperson, 5) AS agg FROM Salespeople

SELECT RIGHT(Salespeople.Salesperson, 7) AS agg FROM Salespeople

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

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

SELECT SPACE(2) + Salespeople.Salesperson_ID AS Salesperson_ID FROM
Salespeople

SELECT STRVAL('60506') AS agg FROM Sales_Data WHERE Sales_Data.Invoice = 1
```

## Funções que retornam números

| Funções que retornam números | Descrição | Exemplo |
|---|---|---|
| ABS | Retorna o valor absoluto da expressão numérica | |
| ATAN | Retorna a tangente do arco do argumento como um ângulo expresso em radianos | |
| ATAN2 | Retorna a tangente do arco das coordenadas x e y como um ângulo expresso em radianos | |
| CEIL<br>CEILING | Retorna o valor inteiro menor que é maior ou igual ao argumento | |
| DEG<br>DEGREES | Retorna o número de graus do argumento, que é um ângulo expresso em radianos | |
| DAY | Retorna a parte de dia de uma data | `DAY(DATE '30-01-2012')` retorna **30** |
| DAYOFWEEK | Retorna o dia da semana (1-7) de uma expressão de data | `DAYOFWEEK(DATE '01-05-2004')` retorna **7** |
| MOD | Divide dois números e retorna o restante da divisão | `MOD(10,3)` retorna **1** |
| EXP | Retorna um valor que é a base do logaritmo natural (e) elevado a uma potência especificada pelo argumento | |

| Funções que retornam números | Descrição | Exemplo |
|---|---|---|
| FLOOR | Retorna o valor inteiro maior que é menor ou igual ao argumento | |
| HOUR | Retorna a parte de hora de um valor | |
| INT | Retorna a parte de inteiro de um número | INT(6.4321) retorna **6** |
| LENGTH | Retorna o tamanho de uma cadeia | LENGTH('ABC') retorna **3** |
| MONTH | Retorna a parte de mês de uma data | MONTH(DATE '30-01-2012') retorna **1** |
| LN | Retorna o logaritmo natural do argumento | |
| LOG | Retorna o logaritmo comum do argumento | |
| MAX | Retorna o maior de dois números | MAX(66,89) retorna **89** |
| MIN | Retorna o menor de dois números | MIN(66,89) retorna **66** |
| MINUTE | Retorna a parte de minuto de um valor | |
| NUMVAL | Converte uma cadeia de caracteres em um número A função falhará se a cadeia de caracteres não for um número válido. | NUMVAL('123') retorna **123** |
| PI | Retorna o valor de constante da constante matemática pi | |
| RADIANS | Retorna o número de radianos de um argumento expresso em graus | |
| ROUND | Arredonda um número | ROUND(123.456,0) retorna **123**<br>ROUND(123.456,2) retorna **123,46**<br>ROUND(123.456,-2) retorna **100** |
| SECOND | Retorna a parte de segundos de um valor | |
| SIGN | Um indicador do sinal do argumento: -1 para negativo, 0 para 0 e 1 para positivo | |
| SIN | Retorna o seno do argumento | |
| SQRT | Retorna a raiz quadrada do argumento | |
| TAN | Retorna a tangente do argumento | |
| YEAR | Retorna a parte de ano de uma data | YEAR(DATE '30-01-2013') retorna **2013** |

## Funções que retornam datas

| Funções que retornam datas | Descrição | Exemplo |
|---|---|---|
| CURDATE<br>CURRENT_DATE | Retorna a data de hoje | |
| CURTIME<br>CURRENT_TIME | Retorna a hora atual | |
| CURTIMESTAMP<br>CURRENT_TIMESTAMP | Retorna o valor de carimbo de data/hora atual | |
| TIMESTAMPVAL | Converte uma cadeia de caracteres em carimbo de data/hora | TIMESTAMPVAL('30-01-2013 14:00:00') returna o valor do carimbo de data/hora. |
| DATE<br>TODAY | Retorna a data de hoje | Se hoje for 21/11/2013, DATE() retornará **21-11-2013** |
| DATEVAL | Converte uma cadeia de caracteres em data | DATEVAL('30-01-2013') retorna **30-01-2013** |

Nota A função DATE() foi preterida. Use o padrão SQL CURRENT_DATE.

## Funções condicionais

| Funções condicionais | Descrição | Exemplo |
|---|---|---|
| CASE WHEN | **formato Simple CASE**<br><br>Compara o valor de *input_exp* aos valores de argumentos *value_exp* para determinar o resultado.<br><br>CASE *input_exp*<br>{WHEN *value_exp* THEN *result*...} [ELSE *result*]<br>END | SELECT<br>Invoice_ID,<br>CASE Company_Name<br>WHEN 'Exports UK' THEN 'Exports UK Found'<br>WHEN 'Home Furniture Suppliers' THEN 'Home Furniture Suppliers Found'<br>ELSE 'Neither Exports UK nor Home Furniture Suppliers'<br>END,<br>Salesperson_ID<br>FROM<br>Sales_Data |
| | **formato Searched CASE**<br><br>Retorna um resultado com base em se a condição especificada pela expressão WHEN é verdadeira.<br><br>CASE<br>{WHEN *boolean_exp* THEN *result...*}<br>[ELSE *result*]<br>END | SELECT<br>Invoice_ID,<br>Amount,<br>CASE<br>WHEN Amount > 3000 THEN 'Above 3000'<br>WHEN Amount < 1000 THEN 'Below 3000'<br>ELSE 'Between 1000 and 3000'<br>END,<br>Salesperson_ID<br>FROM<br>Sales_Data |
| COALESCE | Retorna o primeiro valor que não é NULL | SELECT<br>Salesperson_ID,<br>COALESCE(Sales_Manager, Salesperson)<br>FROM<br>Salespeople |
| NULLIF | Compara dois valores e retorna NULL se os dois valores forem iguais; caso contrário, retorna o primeiro valor. | SELECT<br>Invoice_ID,<br>NULLIF(Amount, -1),<br>Salesperson_ID<br>FROM<br>Sales_Data |

# Palavras-chave SQL reservadas

Esta seção lista as palavras-chave reservadas que não devem ser usadas como nomes de colunas, tabelas, alias ou outros objetos definidos pelo usuário. Se você está recebendo erros de sintaxe, eles possivelmente estão ocorrendo devido ao uso de uma dessas palavras reservadas. Se quiser usar uma dessas palavras-chave, precisará usar aspas para evitar que a palavra seja tratada como uma palavra-chave.

Por exemplo, a instrução `CREATE TABLE` a seguir mostra como usar a palavra-chave `DEC` como nome do elemento de dados.

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

| | | |
|---|---|---|
| ABSOLUTE | CHAR | CURTIME |
| ACTION | CHARACTER | CURTIMESTAMP |
| ADD | CHARACTER_LENGTH | DATE |
| ALL | CHAR_LENGTH | DATEVAL |
| ALLOCATE | CHECK | DAY |
| ALTER | CHR | DAYNAME |
| AND | 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
FALSE
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
LOWER
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
OR
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
TRUE
UNION
UNIQUE
UNKNOWN
UPDATE
UPPER
USAGE
USER
USERNAME
USING
VALUE
VALUES
VARBINARY
VARCHAR
VARYING
VIEW
WHEN
WHENEVER
WHERE
WITH
WORK
WRITE
YEAR
ZONE

# Índice

## G



## H



## I



## L



## N



## O



## P



## R

## S



## T



## U



## V



## W