Claris FileMaker

Referência SQL

Claris

Claris International Inc.
One Apple Park Way
Cupertino, CA 95014

Para obter mais informações, visite nosso site.

Edição: Fevereiro de 2022

Conteúdo

Capítulo 1
Introdução

Como desenvolvedor de banco de dados, você pode usar o Claris® FileMaker Pro® para criar soluções de banco de dados sem saber SQL. Entretanto, se você tiver algum conhecimento em SQL, pode usar um arquivo de banco de dados do FileMaker Pro 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 software Claris FileMaker®. 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 Documentação dos Produtos.
- 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.

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 aceitos pelo software FileMaker estão descritos no capítulo 2, “Padrões suportados“.

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

Ao hospedar um banco de dados do FileMaker Pro 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 Pro.

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.

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 ExecuteSQLnão 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 8.

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

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

- O driver cliente ODBC oferece suporte a ODBC 3.0 Nível 1.
- 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 (cas uma ferramenta ou 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 8)
- DELETE (página 17)
- INSERT (página 17)
- UPDATE (página 19)
- CREATE TABLE (página 20)
- TRUNCATE TABLE (página 22)
- ALTER TABLE (página 22)
- CREATE INDEX (página 23)
- DROP INDEX (página 23)

Os drivers cliente também suportam o mapeamento de tipo de dados do FileMaker para tipos de dados SQL ODBC e JDBC. Consulte no Guia ODBC e JDBC do FileMaker as 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 Pro.

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, `sobrenome`). As expressões podem incluir operações matemáticas ou manipulação de cadeias (por exemplo, `SALÁRIO * 1.05`).

A instrução `SELECT` pode usar diversas cláusulas:

```
SELECT [DISTINCT] {* | expressão_coluna [[AS] alias_coluna],...}
FROM nome_tabela [alias_tabela], ...
[ WHERE expr1 operador_rel expr2 ]
[ GROUP BY {expressão_coluna, ...} ]
[ HAVING expr1 operador_rel expr2 ]
[ UNION [ALL] (SELECT...) ]
[ ORDER BY {expressão_classificação [DESC | ASC]}, ... ]
[ OFFSET n {ROWS | ROW} ]
[ FETCH FIRST [ n [ PERCENT ] ] { ROWS | ROW } {ONLY | WITH TIES } ]
[ FOR UPDATE [OF {expressão_coluna, ...}] ]
```

Os itens entre colchetes são opcionais.

`alias_coluna` pode ser usado para atribuir à coluna um nome mais descritivo ou abreviar um nome de coluna mais longo.

Exemplo

Atribua o alias `departamento` à coluna `dept`.

```
SELECT dept AS departamento FROM func
```

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

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

Exemplo

```
SELECT DISTINCT dept FROM func
```

Cláusulas SQL

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

Use esta cláusula SQL	Para
FROM (página 9)	Indicar quais tabelas são usadas na instrução `SELECT`.
WHERE (página 11)	Especificar as condições que os registros devem atender para serem recuperados (como uma solicitação de busca do FileMaker Pro).
GROUP BY (página 11)	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 12)	Especificar condições de grupos de registros (por exemplo, exibir somente os departamentos com salários que totalizam mais de US$ 200.000).
UNION (página 12)	Combinar os resultados de duas ou mais instruções `SELECT` em um único resultado.
ORDER BY (página 13)	Indicar como os registros são classificados.
OFFSET (página 13)	Informar o número de linhas a serem ignoradas antes de começar a recuperar linhas.
FETCH FIRST (página 13)	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 14)	Executar atualizações ou exclusões posicionadas através 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 nome_tabela [alias_tabela] [, nome_tabela [alias_tabela]]
```

`nomeDaTabela` 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).

`alias_tabela` 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).

Exemplo

A instrução `ExecuteSQL` para o campo nomeado `_SOBRENOME` é:

```
SELECT "_SOBRENOME" from func
```

Os nomes de campo podem ser prefixados com o nome de tabela ou o alias de tabela.

Exemplo

Dada a especificação de tabela `FROM funcionário E`, você pode fazer referência ao campo `SOBRENOME` como `F.SOBRENOME`. Os alias de tabela deverão ser usados se a instrução `SELECT` associar uma tabela a si mesmo.

```
SELECT * FROM funcionário E, funcionário F WHERE F.id_gerente =
F.id_funcionário
```

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`.

Exemplo

```
SELECT *
   FROM Vendedores INNER JOIN Dados_vendas
   ON Vendedores.Vendedores_ID = Dados_vendas.Vendedores_ID
```

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

Exemplo

```
SELECT *
   FROM Vendedores LEFT OUTER JOIN Dados_vendas
   ON Vendedores.Vendedores_ID = Dados_vendas.Vendedores_ID
```

Cada linha da tabela “Vendedores” 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 operador_rel expr2
```

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

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

Exemplo

Recupere os nomes dos funcionários que recebem salário superior a US$ 20.000.

```
SELECT sobrenome,nome FROM func WHERE salário >= 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 colunas
```

O escopo da cláusula `GROUP BY` é a expressão de tabela na cláusula `FROM`. Como resultado, as expressões de coluna especificadas por `colunas` devem ser das tabelas especificadas na cláusula `FROM`. Uma expressão de coluna pode ser um ou mais nomes de campo da tabela de banco de dados separada por vírgulas.

Exemplo

Some os salários em cada departamento

```
SELECT id_dept, SUM (salário) FROM func GROUP BY id_dept
```

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 operador_rel 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`.

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

Exemplo

Retorne somente os departamentos cujas somas salariais são superiores a US$ 200.000.

```
SELECT id_dept, SUM (salário) FROM func
   GROUP BY id_dept HAVING SUM (salário) > 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 declaração UNION [ALL] SELECT declaração
```

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.

Exemplo

```
SELECT sobrenome, salário, data_contratação FROM func UNION SELECT nome,
pago, data_nascimento FROM pessoa
```

O exemplo a seguir não é válido porque os tipos de dados das expressões de coluna são diferentes (`SALÁRIO` em `FUNC` tem um tipo de dados diferente de `SOBRENOME` em `AUMENTOS`). 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.

Exemplo

```
SELECT sobrenome, salário FROM func UNION SELECT salário, sobrenome FROM
aumentos
```

Cláusula ORDER BY

A cláusula `ORDER BY` indica como os registros serão classificados. Se a instrução `SELECT` não incluir uma cláusula `ORDER BY`, os registros poderão ser retornados em qualquer ordem.

O formato é:

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

`expressão_classificação` pode ser o nome de campo ou o número posicional da expressão de coluna a ser usada. O padrão é realizar a classificação em ordem crescente (`ASC`).

Exemplos

Realize a classificação por `sobrenome` e depois por `nome`.

```
SELECT id_funcionário, sobrenome, nome FROM func ORDER BY sobrenome, nome
```

O segundo exemplo usa os números posicionais 2 e 3 para obter a mesma ordem do que o exemplo anterior que especificou `sobrenome` e `nome` explicitamente.

```
SELECT id_funcionário, sobrenome, nome FROM func ORDER BY 2,3
```

Nota O FileMaker Server usa uma ordem de classificação binária unicode, que é diferente da ordem de classificação de idioma no FileMaker Pro ou com a ordem de classificação neutra com relação a idiomas.

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 as cláusulas `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 as linhas pares, as que não são diferenciadas com base na cláusula `ORDER BY`, também são retornadas.

Exemplos

Retorne informações de vinte e seis linhas do conjunto de resultados classificado por `sobrenome` em seguida por `nome`.

```
SELECT id_funcionário, sobrenome, nome FROM func ORDER BY sobrenome, nome
OFFSET 25 ROWS
```

Especifique que você deseja retornar apenas dez linhas.

```
SELECT id_funcionário, sobrenome, nome FROM func ORDER BY sobrenome, nome
OFFSET 25 ROWS FETCH FIRST 10 ROWS ONLY
```

Retorne as dez linhas e suas linhas pares (linhas que não são diferenciadas com base na cláusula `ORDER BY`).

```
SELECT id_funcionário, sobrenome, nome FROM func ORDER BY sobrenome, nome
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 expressões_coluna]
```

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

Exemplo

Retorne todos os registros do banco de dados de funcionários que têm um valor de campo `SALÁRIO` superior a US$ 20.000.

```
SELECT * FROM func WHERE salário > 20000
   FOR UPDATE OF sobrenome, nome, salário
```

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.

Exemplos

Uso	SQL de amostra
constante de texto	`SELECT 'CatDog' FROM Vendedores`
constante numérica	`SELECT 999 FROM Vendedores`
constante de data	`SELECT DATE '2021-06-05' FROM Vendedores`
constante de hora	`SELECT TIME '02:49:03' FROM Vendedores`
constante de carimbo de data/hora	`SELECT TIMESTAMP '2021-06-05 02:49:03' FROM Vendedores`
coluna de texto	`SELECT Nome_Empresa FROM Dados_Venda` `SELECT DISTINCT Nome_Empresa FROM Dados_Venda`
coluna numérica	`SELECT Montante FROM Dados_Venda` `SELECT DISTINCT Montante FROM Dados_Venda`
coluna de data	`SELECT Data_Venda FROM Dados_Venda` `SELECT DISTINCT Data_Venda FROM Dados_Venda`
coluna de hora	`SELECT Tempo_Venda FROM Dados_Venda` `SELECT DISTINCT Tempo_Venda FROM Dados_Venda`
coluna de carimbo de data/hora	`SELECT Timestamp_Sold FROM Dados_Venda` `SELECT DISTINCT Timestamp_Sold FROM Dados_Venda`
coluna BLOB[a]	`SELECT Brochuras_Empresa FROM Dados_Venda` `SELECT GETAS(Logo_Empresa, 'JPEG') FROM Dados_Venda`
Curinga *	`SELECT * FROM Vendedores` `SELECT DISTINCT * FROM Vendedores`

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

Notas dos exemplos

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

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

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

  ```
  SELECT * FROM teste WHERE c =''
  SELECT * FROM teste WHERE c =''
  ```

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

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

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

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

Exemplo

Use a função `CAST()` com uma instrução `SELECT` para recuperar informações de referência de arquivo.

```
SELECT CAST(Brochuras_Empresa AS VARCHAR) FROM Dados_Venda
```

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 container de outro aplicativo, a instrução `SELECT` exibirá '?' (o arquivo é exibido como **Sem nome.dat** no FileMaker Pro).

É possível usar a instrução `SELECT` com a função `GetAs()` para recuperar dados em formato binário das seguintes maneiras:

- Se você usar a função `GetAs()` com a opção `DEFAULT`, recuperará o fluxo padrão para o container sem a necessidade de definir explicitamente o tipo de fluxo.

  #### Exemplo

  ```
  SELECT GetAs(Brochuras_Empresa, DEFAULT) FROM Dados_Venda
  ```

- 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.

  #### Exemplo

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

  ```
  SELECT GetAs(Brochuras_Empresa, 'FILE') FROM Dados_Venda
  ```

Exemplo

Se os dados tiverem sido inseridos por meio do comando **Inserir** > **Imagem**, arraste e solte ou cole da Área de transferência, especifique um dos tipos de arquivo listados na tabela a seguir, por exemplo, `'JPEG'`.

```
SELECT GetAs(Logo_Empresa, 'JPEG') FROM Ícones_Empresa
```

Tipo de arquivo	Descrição
`'GIFf'`	Graphics Interchange Format
`'JPEG'`	Imagens fotográficas
`'TIFF'`	Formato de arquivo raster para imagens digitais
`'PDF '`	Formato de documento portátil
`'PNGf'`	Formato de imagem de bitmap

Instrução DELETE

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

```
DELETE FROM nome_tabela [ WHERE { condições } ]
```

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

Exclua um registro da tabela `func`.

```
DELETE FROM func WHERE id_funcionário = '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 Funcionário, 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 nome_tabela [(nome_coluna, ...)] VALUES (expr, ...)
```

nome_coluna 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 nome_coluna, 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.nome_coluna também pode especificar uma repetição de campo, como, por exemplo datasRecentes[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.

Exemplo

Insira uma lista de expressões.

```
INSERT INTO func (sobrenome, nome, id_funcionário, salário,
data_contratação)
   VALUES ('Smith', 'John', 'E22345', 27500, DATE '2019-06-05')
```

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, func. São especificados valores para cinco colunas. As colunas restantes da tabela recebem um valor em branco, que significa nulo.

Nota Nos campos de container, 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 nome_tabela (nome_container) VALUES(? AS 'nome do arquivo.extensão
do arquivo')
```

Tipos de arquivo não suportados serão inseridos como tipo FILE.

Ao usar a função PutAs() especifique o tipo: PutAs(col, 'tipo'), em que o valor de tipo é um tipo conforme descrito em “Recuperação do conteúdo de um campo de container: CAST() function and GetAs() function” na página 16.

A instrução SELECT é uma consulta que retorna valores para cada valor nome_coluna 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.

Exemplo

Insira usando a instrução SELECT.

```
INSERT INTO func1 (nome, sobrenome, id_funcionário, dept, Salário)
   SELECT nome, sobrenome, id_funcionário, dep, salário do func
   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 nome_tabela SET nome_coluna = expr, ... [ WHERE { condições } ]
```

nome_coluna 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.

Exemplo

Instrução UPDATE na tabela func.

```
UPDATE func SET salário=32000, isento=1 WHERE id_funcionário = '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 Funcionário, somente um registro será atualizado.

Exemplo

Instrução UPDATE na tabela func com uma subconsulta.

```
UPDATE func SET salário = (SELECT avg(salário) from func) WHERE
id_funcionário = '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 container, 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 nome_tabela SET (nome_container) = ? AS 'nome do arquivo.extensão do arquivo'
```

Tipos de arquivo não suportados serão inseridos como tipo FILE.

Ao usar a função `PutAs()` especifique o tipo: `PutAs(col, 'tipo')`, em que o valor de tipo é um tipo conforme descrito em “Recuperação do conteúdo de um campo de container: CAST() function and GetAs() function” na página 16.

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 nome_tabela ( lista_elemento_tabela [, lista_elemento_tabela...] )
```

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

- `nome_tabela` é o nome da tabela. `nome_tabela` 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 `lista_elemento_tabela` é:

  ```
  nome_campo tipo_campo [[repetições]]
  [DEFAULT expr] [UNIQUE | NOT NULL | PRIMARY KEY | GLOBAL]
  [EXTERNAL string_caminho_relativo [SECURE | OPEN string_calc_relativo]]
  ```

  - `nome_campo` é o nome do campo. Os nomes de campo devem ser exclusivos. 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).

    Exemplo

    A instrução `CREATE TABLE` para o campo nomeado `_SOBRENOME` é:

    ```
    CREATE TABLE "_FUNCIONÁRIO" (ID INT PRIMARY KEY, "_NOME" VARCHAR(20),
    "_SOBRENOME" VARCHAR(20))
    ```

  - Para a instrução `CREATE TABLE repetições`, especifique uma repetição de campo usando um número de 1 a 32000 entre colchetes após o tipo do campo.

    Exemplo

    ```
    ID_FUNCIONÁRIO INT[4]
    SOBRENOME VARCHAR(20)[4]
    ```

- `tipo_campo` 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.

  ### 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 Pro.
- 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 Pro. 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 container, use `BLOB`, `VARBINARY` ou `BINARY VARYING` em `field_type`.
- Para definir uma coluna como um campo de container que armazena dados externamente, use a palavra-chave `EXTERNAL`. `string_caminho_relativo` define a pasta em que os dados são armazenados externamente, relativa ao local do banco de dados do FileMaker Pro. 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, `string_calc_relativo` será a subpasta da pasta `string_caminho_relativo` 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)`

Uso	SQL de amostra
coluna para campo de container	`CREATE TABLE T6 (C1 BLOB, C2 BLOB, C3 BLOB, C4 BLOB)`
coluna para campo de container de armazenamento externo	`CREATE TABLE T7 (C1 BLOB EXTERNAL 'Files/MyDatabase/' SECURE)` `CREATE TABLE T8 (C1 BLOB EXTERNAL 'Files/MyDatabase/' OPEN 'Objects')`

Instrução TRUNCATE TABLE

Use a instrução `TRUNCATE TABLE` para excluir todos os registros na tabela especificada rapidamente, esvaziando todos os dados da tabela.

```
TRUNCATE TABLE nome_tabela
```

Não é possível especificar uma cláusula `WHERE` com a instrução `TRUNCATE TABLE`. A instrução `TRUNCATE TABLE` exclui todos os registros.

Somente os registros na tabela especificada por `nome_tabela` são excluídos. Os registros de quaisquer tabelas relacionadas não são afetados.

A instrução `TRUNCATE TABLE` deve poder bloquear todos os registros na tabela para excluir os dados de registro. Se algum registro na tabela estiver bloqueado por outro usuário, o software FileMaker retornará o código de erro 301 (“Registro em uso por outro usuário”).

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 nome_tabela ADD [COLUMN] definição_coluna
ALTER TABLE nome_tabela DROP [COLUMN] nome_coluna_nao_qualificado
ALTER TABLE nome_tabela ALTER [COLUMN] definição_coluna SET DEFAULT expr
ALTER TABLE nome_tabela ALTER [COLUMN] definição_coluna 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 Vendedores ADD C1 VARCHAR`
remover colunas	`ALTER TABLE Vendedores DROP C1`
definir o valor padrão de uma coluna	`ALTER TABLE Vendedores ALTER Empresa SET DEFAULT 'Claris'`
remover o valor padrão de uma coluna	`ALTER TABLE Vendedores ALTER Empresa 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 nome_tabela.nome_coluna
CREATE INDEX ON nome_tabela (nome_coluna)
```

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 container, 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 Pro.

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 Pro. 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 Pro.

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 Pro.

O software 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 Vendedores.Vendedor_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 nome_tabela.nome_coluna
DROP INDEX ON nome_tabela (nome_coluna)
```

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 Pro 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 Pro.

O atributo `PREVENT INDEX CREATION` não é suportado.

Exemplo

```
DROP INDEX ON Vendedores.Vendedor_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 `Dados_Venda.Fatura_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 `Número_de_Dias_em_Junho`.

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 software FileMaker aceita as constantes de data, hora e carimbo de data/hora ODBC/JDBC entre chaves ({}).

Exemplos

```
{D '2019-06-05'}
{T '14:35:10'}
{TS '2019-06-05 14:35:10'}
```

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

Exemplos

```
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 '2019-06-05'`
	`{ D '2019-06-05' }`
	`{06/05/2019}`
	`{06/05/19}`
	Nota a sintaxe de ano de dois dígitos não é suportada no formato ODBC/JDBC ou SQL-92.

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

Carimbo de data/hora	`TIMESTAMP '2019-06-05 14:35:10'`
	`{ TS '2019-06-05 14:35:10'}`
	`{06/05/2019 14:35:10}`
	`{06/05/19 14:35:10}`
	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 Pro para um campo que usa a sintaxe de ano de dois dígitos.
	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. Nos exemplos a seguir, `sobrenome` é `'JONES '` e nome é `'ROBERT '`.

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

Operadores de data

Você pode modificar datas. Nos exemplos a seguir, `data_contratação` é `DATE '2019-01-30'`.

Operador	Efeito na data	Exemplo	Resultado
+	Adiciona um número de dias a uma data	`data_contratação` + 5	DATE '2019-02-04'
-	Encontra o número de dias entre duas datas	`data_contratação - DATE '2019-01-01'`	`29`
	Subtrai um número de dias de uma data	`data_contratação - 10`	DATE '2019-01-20'

Exemplos adicionais

```
SELECT Data_Venda, Data_Venda + 30 AS agr FROM Dados_Venda
SELECT Data_Venda, Data_Venda - 30 AS agr FROM Dados_Venda
```

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 NULL
IS NOT NULL	Diferente de NULL
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
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 =, <>, >, >=, <, or <=); =Any é equivalente a In
ALL	Compara um valor com cada valor retornado por uma subconsulta (o operador deve ser precedido por =, <>, >, >=, < ou <=)

Exemplo

```
SELECT Dados_Venda.Fatura_ID FROM Dados_Venda
   WHERE Dados_Venda.Vendedor_ID = 'SP-1'
SELECT Dados_Venda.Montante FROM Dados_Venda WHERE Dados_Venda.Fatura_ID
<> 125
SELECT Dados_Venda.Montante FROM Dados_Venda WHERE Dados_Venda.Montante
> 3000
SELECT Dados_Venda.Tempo_Venda FROM Dados_Venda
   WHERE Dados_Venda.Tempo_Venda < '12:00:00'
SELECT Dados_Venda.Nome_Empresa FROM Dados_Venda
   WHERE Dados_Venda.Nome_Empresa LIKE '%Universidade'
SELECT Dados_Venda.Nome_Empresa FROM Dados_Venda
   WHERE Dados_Venda.Nome_Empresa NOT LIKE '%Universidade'
SELECT Dados_Venda.Montante FROM Dados_Venda WHERE Dados_Venda.Montante
IS NULL
SELECT Dados_Venda.Montante FROM Dados_Venda WHERE Dados_Venda.Montante
IS NOT NULL
SELECT Dados_Venda.Fatura_ID FROM Dados_Venda
   WHERE Dados_Venda.Fatura_ID BETWEEN 1 AND 10
SELECT COUNT(Dados_Venda.Fatura_ID) AS agr
   FROM Dados_Venda WHERE Dados_Venda.FATURA_ID IN (50,250,100)
SELECT COUNT(Dados_Venda.Fatura_ID) AS agr
   FROM Dados_Venda WHERE Dados_Venda.FATURA_ID NOT IN (50,250,100)
SELECT COUNT(Dados_Venda.Fatura_ID) AS agr FROM Dados_Venda
   WHERE Dados_Venda.FATURA_ID NOT IN (SELECT Dados_Venda.Fatura_ID
   FROM Dados_Venda WHERE Dados_Venda.Vendedor_ID = 'SP-4')
SELECT *
   FROM Dados_Venda WHERE EXISTS (SELECT Dados_Venda.Montante
   FROM Dados_Venda WHERE Dados_Venda.Vendedor_ID IS NOT NULL)
SELECT *
   FROM Dados_Venda WHERE Dados_Venda.Montante = ANY (SELECT
Dados_Venda.Montante
   FROM Dados_Venda WHERE Dados_Venda.Vendedor_ID = 'SP-1')
SELECT *
   FROM Dados_Venda WHERE Dados_Venda.Montante = ALL (SELECT
Dados_Venda.Montante
   FROM Dados_Venda WHERE Dados_Venda.Vendedor_ID IS NULL)
```

Operadores lógicos

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

```
salário = 40000 AND isento = 1
```

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

```
NOT (salário = 40000 AND isento = 1)
```

Exemplo

```
SELECT * FROM Dados_Venda WHERE Dados_Venda.Nome_Empresa
   NOT LIKE '%Universidade' AND Dados_Venda.Montante > 3000
SELECT * FROM Dados_Venda WHERE (Dados_Venda.Nome_Empresa
   LIKE '%Universidade' OR Dados_Venda.Montante > 3000)
   AND Dados_Venda.Vendedor_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	Unário `'-'`, Unário `'+'`
2	^, **
3	*, /
4	+, -
5	=, <>, <, <=, >, >=, Like, Not Like, Is Null, Is Not Null, Between, In, Exists, Any, All
6	Not
7	AND
8	OR

Exemplos

```
WHERE salário > 40000 OR data_contratação > (DATE '2008-01-30') 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.

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

Esse exemplo 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 Claris fornece uma implementação do padrão SQL para a Plataforma FileMaker e oferece suporte a várias funções que você pode usar em 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(SALÁRIO)`) ou em conjunto com uma expressão de coluna (por exemplo, `AVG(SALÁRIO * 1.07)`).

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

Exemplo

```
COUNT (DISTINCT sobrenome)
```

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(SALÁRIO)` 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(SALÁRIO)` 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 NULL. Um exemplo especial é `COUNT(*)`, que retorna o número de registros no conjunto, incluindo os registros com valores NULL.
`MAX`	O valor máximo em qualquer expressão de campo. Por exemplo, `MAX(SALÁRIO)` retorna o valor de campo de salário máximo.
`MIN`	O valor mínimo em qualquer expressão de campo. Por exemplo, `MIN(SALÁRIO)` retorna o valor de campo de salário mínimo.

Exemplo

```
SELECT SUM (Dados_Venda.Montante) AS agr FROM Dados_Venda
SELECT AVG (Dados_Venda.Montante) AS agr FROM Dados_Venda
SELECT COUNT (Dados_Venda.Montante) AS agr FROM Dados_Venda
SELECT MAX (Dados_Venda.Montante) AS agr FROM Dados_Venda
   WHERE Dados_Venda.Montante < 3000
SELECT MIN (Dados_Venda.Montante) AS agr FROM Dados_Venda
   WHERE Dados_Venda.Montante > 3000
```

Não é possível usar uma função agregada como argumento para outras funções. Se você o fizer, o software FileMaker retornará o código de erro 8309 (“Expressões envolvendo agregações não são suportadas”). Por exemplo, a seguinte instrução não é válida, pois a função agregada `SUM` não pode ser usada como argumento para a função `ROUND`:

Exemplo

```
SELECT ROUND(SUM(Salário), 0) FROM Folha de pagamento
```

Entretanto, funções agregadas podem usar funções que retornam números como argumentos. A instrução a seguir é válida.

Exemplo

```
SELECT SUM(ROUND(Salário, 0)) FROM Folha de pagamento
```

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 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'` `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'`
		`STRVAL(5 * 3)` retorna `'15'`
		`STRVAL(4 = 5)` retorna `'False'`
		`STRVAL(DATE '2019-12-25')` retorna `'2019-12-25'`

TIME TIMEVAL	Retorna a hora do dia como uma cadeia	Às 9:49 PM, `TIME()` retorna **21:49:00**
USERNAME USER	Retorna a ID de login especificada no momento da conexão	

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

Exemplo

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

SELECT RTRIM(' ' + Vendedores.Vendedor_ID) AS agr FROM Vendedores

SELECT TRIM(SPACE(1) + Vendedores.Vendedor_ID) AS agr FROM Vendedores

SELECT LTRIM(' ' + Vendedores.Vendedor_ID) AS agr FROM Vendedores

SELECT UPPER(Vendedores.Vendedor) AS agr FROM Vendedores

SELECT LOWER(Vendedores.Vendedor) AS agr FROM Vendedores

SELECT LEFT(Vendedores.Vendedor, 5) AS agr FROM Vendedores

SELECT RIGHT(Vendedores.Vendedor, 7) AS agr FROM Vendedores

SELECT SUBSTR(Vendedores.Vendedor_ID, 2, 2) +
SUBSTR(Vendedores.Vendedor_ID, 4, 2) AS agr FROM Vendedores

SELECT SUBSTR(Vendedores.Vendedor_ID, 2) + SUBSTR(Vendedores.Vendedor_ID,
4) AS agr FROM Vendedores

SELECT SPACE(2) + Vendedores.Vendedor_ID AS Vendedor_ID FROM Vendedores

SELECT STRVAL('60506') AS agr FROM Dados_Venda WHERE
Dados_Venda.Fatura = 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 CEILING	Retorna o valor inteiro menor que é maior ou igual ao argumento	
DEG 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 '2019-01-30')` retorna **30**
DAYOFWEEK	Retorna o dia da semana (1-7) de uma expressão de data	`DAYOFWEEK(DATE '2004-05-01')` 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	
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 '2019-01-30')` retorna **1**
LN	Retorna o logaritmo natural do argumento	
LOG	Retorna o logaritmo comum do argumento	
MAX	Retorna o maior dois dois números	`MAX(66,89)` retorna **89**
MIN	Retorna o menor dois dois números	`MIN(66,89)` retorna **66**
MINUTE	Retorna a parte de minuto de um valor	
NUMVAL	Converte uma cadeia de caracteres em número. A função falha 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** `ROUND(123.456.2)` retorna **123,46** `ROUND(123.456,-2)` retorna **100**
SECOND	Retorna a parte de segundos de um valor	

Funções que retornam números	Descrição	Exemplo
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 '2019-01-30') retorna **2019**

Funções que retornam datas

Funções que retornam datas	Descrição	Exemplo
CURDATE CURRENT_DATE	Retorna a data de hoje	
CURTIME CURRENT_TIME	Retorna a hora atual	
CURTIMESTAMP CURRENT_TIMESTAMP	Retorna o valor de carimbo de data/hora atual	
TIMESTAMPVAL	Converte uma cadeia de caracteres em carimbo de data/hora	TIMESTAMPVAL('2019-01-30 14:00:00') retorna um valor de carimbo de data/hora
DATE TODAY	Retorna a data de hoje	Se hoje for 11/21/2019, DATE() retornará **2019-11-21**
DATEVAL	Converte uma cadeia de caracteres em data	DATEVAL('2019-01-30') retorna **2019-01-30**

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

Funções condicionais

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

Objetos de sistema do FileMaker

Os arquivos de banco de dados do FileMaker Pro incluem os seguintes objetos de sistema que você pode acessar usando consultas SQL.

Tabelas de sistema do FileMaker

Cada arquivo de banco de dados do FileMaker Pro inclui essas tabelas de sistema: FileMaker_Tables, FileMaker_Fields e FileMaker_BaseTableFields. Para aplicativos ODBC, essas tabelas são incluídas nas informações retornadas pela função de catálogo SQLTables. Para aplicativos JDBC, essas tabelas são incluídas nas informações retornadas pelo método DatabaseMetaData getTables. As tabelas também podem ser usadas em funções ExecuteSQL.

FileMaker_Tables

A tabela FileMaker_Tables contém informações sobre as tabelas de banco de dados definidas no arquivo do FileMaker Pro.

A tabela FileMaker_Tables inclui uma linha para cada ocorrência de tabela no gráfico de relacionamentos com as seguintes colunas:

- TableName - O nome da ocorrência de tabela.
- TableId - A ID exclusiva para a ocorrência de tabela.
- BaseTableName - O nome da tabela de base da qual a ocorrência de tabela foi criada.
- BaseFileName - O nome de arquivo do FileMaker Pro para o arquivo de banco de dados que contém a tabela de base.
- ModCount - O número total de vezes que as alterações na definição dessa tabela foram confirmadas.

Exemplo

```
SELECT TableName FROM FileMaker_Tables WHERE TableName LIKE 'Sales%'
```

Tabela FileMaker_Fields

A tabela FileMaker_Fields contém informações sobre os campos definidos no arquivo do FileMaker Pro para todas as ocorrências de tabela.

A tabela FileMaker_Fields inclui as seguintes colunas:

- TableName - O nome da tabela que contém o campo.
- FieldName - O nome do campo.
- FieldType - O tipo de dados SQL do campo.
- FieldId - A ID exclusiva para o campo.
- FieldClass - Um de três valores: Summary (Resumo), para campos resumidos; Calculated (Calculado), para resultados calculados; ou Normal.
- FieldReps - O número de repetições do campo.
- ModCount - O número total de vezes que as alterações na definição dessa tabela foram confirmadas.

Exemplo

```
SELECT * FROM FileMaker_Fields WHERE TableName='Sales'
```

Tabela FileMaker_BaseTableFields

Incluída na plataforma FileMaker na versão 19.4.1, a tabela FileMaker_BaseTableFields contém informações sobre os campos definidos no arquivo do FileMaker Pro apenas para as tabelas de origem (ou de base).

A tabela FileMaker_BaseTableFields inclui as seguintes colunas:

- BaseTableName - O nome da tabela de base que contém o campo.
- FieldName - O nome do campo.
- FieldType - O tipo de dados SQL do campo.
- FieldId - A ID exclusiva para o campo.
- FieldClass - Um de três valores: Summary (Resumo), para campos resumidos; Calculated (Calculado), para resultados calculados; ou Normal.
- FieldReps - O número de repetições do campo.
- ModCount - O número total de vezes que as alterações na definição dessa tabela de base foram confirmadas.

Exemplo

```
SELECT * FROM FileMaker_BaseFields WHERE BaseTableName='Sales'
```

Colunas de sistema do FileMaker

O software FileMaker adiciona colunas de sistema (campos) a todas as linhas (registros) em todas as tabelas definidas no arquivo do FileMaker. Para aplicativos ODBC, essas colunas são incluídas nas informações retornadas pela função de catálogo SQLSpecialColumns. Para aplicativos JDBC, essas colunas são incluídas nas informações retornadas pelo método DatabaseMetaData getVersionColumns. As colunas também podem ser usadas em funções ExecuteSQL.

Coluna ROWID

A coluna de sistema `ROWID` contém o número de ID exclusiva do registro. Esse é o mesmo valor que a função Get(RecordID) do FileMaker Pro retorna.

Coluna ROWMODID

A coluna de sistema `ROWMODID` contém o número total de vezes que as alterações no registro atual foram confirmadas. Esse é o mesmo valor que a função Get(RecordModificationCount) do FileMaker Pro retorna.

Exemplo

```
SELECT ROWID, ROWMODID FROM MyTable WHERE ROWMODID > 3
```

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.

Exemplo

Use a palavra-chave `DEC` como nome do elemento de dados.

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

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

END_EXEC
ESCAPE
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
INSERT
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
TRUNCATE
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

M

N

O

P

R

S

T

U

V

W