Claris FileMaker

SQL リファレンスガイド

Claris

Claris International Inc.
5201 Patrick Henry Drive
Santa Clara, California 95054

第 01 版

目次

第 1 章
はじめに

Claris™ FileMaker® Pro を使用すると SQL の知識のないデータベース開発者でもデータベースソリューションを作成できます。SQL の知識がある場合は FileMaker Pro データベースファイルを ODBC または JDBC データソースとして使用し、ODBC および JDBC を使用してその他のアプリケーションとデータを共有できます。また、FileMaker Pro の ExecuteSQL 関数を使用して、FileMaker Pro データベース内の任意のテーブルオカレンスからデータを取得することもできます。

このリファレンスでは FileMaker ソフトウェアでサポートされる SQL ステートメントおよび標準について説明します。FileMaker ODBC および JDBC クライアントドライバは、このリファレンスに記載されているすべての SQL ステートメントをサポートします。FileMaker Pro の ExecuteSQL 関数は SELECT ステートメントのみをサポートします。

このリファレンスについて

- 旧バージョンの FileMaker Pro での ODBC および JDBC の使用については、プロダクトドキュメンテーションセンターを参照してください。
- このリファレンスは、ユーザが FileMaker Pro 関数の使用、ODBC および JDBC アプリケーションのコーディング、および SQL クエリーの構築に関する基本を理解していることを想定しています。これらのトピックの詳細については、他社の書籍を参照してください。

SQL について

SQL (構造化照会言語) は、リレーショナルデータベースからデータを照会 (クエリー) するために設計されたプログラミング言語です。データベースを照会するために使用する主なステートメントは SELECT ステートメントです。

データベースを照会する言語に加えて、SQL はデータの操作 (データの追加、更新、および削除) を実行するステートメントも提供します。

また、SQL はデータの定義を実行するステートメントも提供します。これらのステートメントを使用すると、テーブルとインデックスを作成および変更することができます。

FileMaker ソフトウェアでサポートされる SQL ステートメントおよび標準については、第 2 章「サポートされている標準」を参照してください。

データソースとしての FileMaker Pro データベースの使用

FileMaker Pro データベースを ODBC または JDBC データソースとして使用すると、FileMaker データを ODBC および JDBC に準拠したアプリケーションと共有できます。アプリケーションは FileMaker クライアントドライバを使用して FileMaker データソースに接続し、ODBC または JDBC を使用して SQL クエリーを構築および実行し、FileMaker Pro データベースソリューションから取得したデータを処理します。

FileMaker ソフトウェアを ODBC および JDBC アプリケーションのデータソースとして使用する方法の詳細については、『FileMaker ODBC と JDBC ガイド』を参照してください。

FileMaker ODBC および JDBC クライアントドライバは、このリファレンスに記載されているすべての SQL ステートメントをサポートします。

ExecuteSQL 関数の使用

FileMaker Pro の ExecuteSQL 関数を使用すると、リレーションシップグラフで指定したテーブルオカレンスからデータを取得できます。この操作は定義済みのリレーションシップから独立して行うことができます。複数のテーブルからデータを取得する場合でも、テーブル結合やテーブル間のリレーションシップを作成する必要はありません。ExecuteSQL 関数を使用して、リレーションシップグラフの複雑さを軽減できる場合もあります。

ExecuteSQL 関数で照会するフィールドはいずれのレイアウト上にある必要もないので、ExecuteSQL 関数を使用して、レイアウトコンテキストから独立してデータを取得できます。コンテキストに依存しないので、スクリプトで ExecuteSQL 関数を使用するとスクリプトの移植性を向上させることができます。ExecuteSQL 関数は、グラフやレポートを始めとして、計算を指定できるすべての場所で使用できます。

ExecuteSQL 関数は、SELECT ステートメントのみをサポートします。詳細については、8 ページの「SELECT ステートメント」を参照してください。

また、ExecuteSQL 関数は、中カッコ ({}) を使用しない SQL-92 構文の ISO 形式の日付と時刻のみを処理します。ExecuteSQL 関数は、中カッコを使用する ODBC/JDBC 形式の日付、時刻、およびタイムスタンプ定数を処理しません。

ExecuteSQL 関数の構文および使用方法については、FileMaker Pro ヘルプを参照してください。

第 2 章
サポートされている標準

FileMaker ODBC および JDBC クライアントドライバは、ODBC または JDBC 準拠のアプリケーションから FileMaker Pro データベースソリューションにアクセスする場合に使用します。FileMaker Pro データベースソリューションは、FileMaker Pro または FileMaker Server により共有されます。

- ODBC クライアントドライバは、ODBC 3.0 Level 1 をサポートします。
- JDBC クライアントドライバは、JDBC 3.0 仕様を部分的にサポートします。
- ODBC および JDBC クライアントドライバは、両方とも SQL-92 エントリレベルに準拠しており、SQL-92 中間レベルの機能も部分的にサポートしています。

Unicode 文字のサポート

ODBC および JDBC クライアントドライバでは、Unicode API がサポートされています。ただし、これらのクライアントドライバを使用するカスタムアプリケーションを作成するときは、Unicode に対応していないクエリーツールやアプリケーションが使用される場合に備えて、フィールド名、テーブル名、およびファイル名には ASCII を使用してください。

メモ　Unicode データの挿入および取得には、`SQL_C_WCHAR` を使用します。

SQL ステートメント

ODBC および JDBC クライアントドライバでは、次の SQL ステートメントに対するサポートが提供されています:

- SELECT (8 ページ)
- DELETE (17 ページ)
- INSERT (17 ページ)
- UPDATE (20 ページ)
- CREATE TABLE (21 ページ)
- TRUNCATE TABLE (23 ページ)
- ALTER TABLE (23 ページ)
- CREATE INDEX (23 ページ)
- DROP INDEX (24 ページ)

また、クライアントドライバでは、FileMaker データタイプと ODBC SQL および JDBC SQL データタイプのマッピングもサポートされています。データタイプの変換については、『FileMaker ODBC と JDBC ガイド』を参照してください。SQL クエリーの構築の詳細については、他社の書籍を参照してください。

メモ　ODBC および JDBC クライアントドライバでは、FileMaker Pro ポータルはサポートされていません。

SELECT ステートメント

SELECT ステートメントを使用して、要求する列を指定します。SELECT ステートメントの後に、フィールド名に相当する、抽出する列式を指定します (たとえば、姓)。列式には、数式演算または文字操作 (たとえば、給与 * 1.05) を入れることができます。

SELECT ステートメントでは、次のようなさまざまな句を使用できます:

```
SELECT [DISTINCT] {* | 列式 [[AS] 列エイリアス],...}
FROM テーブル名 [テーブルエイリアス], ...
[ WHERE 式1 リレーショナル演算子 式2 ]
[ GROUP BY {列式, ...}]
[ HAVING 式1 リレーショナル演算子 式2 ]
[ UNION [ALL] (SELECT...)]
[ ORDER BY {ソート式 [DESC | ASC]}, ...]
[ OFFSET n {ROWS | ROW} ]
[ FETCH FIRST [ n [ PERCENT ] ] { ROWS | ROW } {ONLY | WITH TIES } ]
[ FOR UPDATE [OF {列式, ...}]]
```

角カッコ ([]) 内の項目は省略可能です。

列エイリアスを使用して、列によりわかりやすい名前を付ける、または長い列名を短縮できます。

例

列「部門」にエイリアス「所属部門」を割り当てます。

```
SELECT "部門" AS "所属部門" FROM "従業員名簿"
```

フィールド名に、テーブル名またはテーブルエイリアスの接頭辞を付けることができます。たとえば、「従業員名簿.姓」または「名簿.姓」のように指定します。この場合、「名簿」はテーブル「従業員名簿」のエイリアスです。

DISTINCT 演算子は、最初の列式の前に配置できます。この演算子は、クエリー結果から重複する行を除去します。

例

```
SELECT DISTINCT "部門" FROM "従業員名簿"
```

SQL 句

ODBC および JDBC クライアントドライバでは、次の SQL 句に対するサポートが提供されています。

SQL 句	目的
FROM (9 ページ)	SELECT ステートメントで使用するテーブルを指定します。
WHERE (11 ページ)	FileMaker Pro の検索条件のように、抽出するレコードの条件を指定します。
GROUP BY (11 ページ)	返された値をグループ化するための、1 つまたは複数のフィールド名を指定します。この句は、FileMaker Pro の小計のように、それぞれのグループについて 1 行を返すことによって統計値のセットを返すときに使用します。
HAVING (12 ページ)	グループ化の条件を指定します。たとえば、給与総額が 20,000,000 円以上の部門だけを表示する場合などです。
UNION (12 ページ)	2 つ以上の SELECT ステートメントの結果を 1 つの結果に結合します。
ORDER BY (13 ページ)	レコードのソート方法を指定します。
OFFSET (13 ページ)	行の取得を開始する前に、スキップする行の数を示します。
FETCH FIRST (13 ページ)	取得する行の数を指定します。指定した数以上の行は返されませんが、指定した行数よりも少ない行が取得された場合、指定した数よりも少ない行が返されます。
FOR UPDATE (14 ページ)	SQL カーソルで位置付け更新または位置付け削除を実行します。

メモ 列なしのテーブルからデータを取得しようとしても、`SELECT` ステートメントは何も返しません。

FROM 句

`FROM` 句は、`SELECT` ステートメントで使用されるテーブルを指定します。形式は次のとおりです:

```
FROM テーブル名 [テーブルエイリアス] [, テーブル名 [テーブルエイリアス]]
```

テーブル名は、現在のデータベースのテーブルの名前です。テーブル名は、アルファベット文字で始まる必要があります。テーブル名がアルファベット文字以外で始まる場合は、ダブルクォーテーションマークで囲む必要があります (クォーテーションマークで囲まれた識別子)。

テーブルエイリアスを使用して、テーブルによりわかりやすい名前を付ける、長いテーブル名を短縮、または同じテーブルにクエリーを複数回含めることができます (たとえば、自己連結など)。

フィールド名はアルファベット文字で始めます。フィールド名がアルファベット文字以外で始まる場合は、ダブルクォーテーションマークで囲む必要があります (クォーテーションマークで囲まれた識別子)。

例

「姓」という名前のフィールドに対する `ExecuteSQL` ステートメントは、次のようになります :

```
SELECT "姓" from "従業員名簿"
```

フィールド名に、テーブル名またはテーブルエイリアスの接頭辞を付けることができます。

例

テーブルの指定が FROM 従業員 E の場合は、E. 姓で姓フィールドを参照できます。SELECT ステートメントでテーブルをそれ自身に結合する場合は、テーブルエイリアスを使用する必要があります。

```
SELECT * FROM "従業員" "E", "従業員" "F" WHERE "E"."管理者番号" = "F"."従業員番号"
```

等号記号 (=) を指定すると、一致する行のみが結果に含まれます。

複数のテーブルを結合して、両方のソーステーブルに対応する行が存在しない行をすべて破棄する場合は、INNER JOIN を使用できます。

例

```
SELECT *
   FROM "営業社員" INNER JOIN "営業データ"
   ON "営業社員"."営業社員番号" = "営業データ"."営業社員番号"
```

2 つのテーブルを結合して、最初のテーブル (「左の」テーブル) の行を破棄しない場合は、LEFT OUTER JOIN を使用できます。

例

```
SELECT *
   FROM "営業社員" LEFT OUTER JOIN "営業データ"
   ON "営業社員"."営業社員番号" = "営業データ"."営業社員番号"
```

「営業社員」テーブルのすべての行が、結合されたテーブルに表示されます。

メモ

- RIGHT OUTER JOIN については現在サポートされていません。
- FULL OUTER JOIN については現在サポートされていません。

WHERE 句

WHERE 句は、抽出するレコードの条件を指定します。WHERE 句には、次の形式で条件を含めます:

```
WHERE 式1 リレーショナル演算子 式2
```

式1 および式2 には、フィールド名、定数値、または式を指定できます。

リレーショナル演算子は、2 つの式をリンクするリレーショナル演算子です。

例

給与が 2,000,000 円以上の従業員の名前を取得します。

```
SELECT "姓","名" FROM "従業員名簿" WHERE "給与" >= 2000000
```

WHERE 句では、次のような式を使用することもできます:

```
WHERE 式1 IS NULL
WHERE NOT 式2
```

メモ SELECT リスト (射影リスト) で完全修飾名を使用する場合は、関連する WHERE 句でも完全修飾名を使用する必要があります。

GROUP BY 句

GROUP BY 句は、返された値をグループ化するための、1 つまたは複数のフィールド名を指定します。この句は、統計値のセットを返すときに使用します。この句の形式は次のとおりです:

```
GROUP BY 列
```

GROUP BY 句の有効範囲は FROM 句のテーブル式です。そのため、列で指定される列式は FROM 句で指定されるテーブルのものである必要があります。列式には、データベーステーブルの 1 つまたは複数のフィールド名をコンマで区切って指定できます。

例

各部門の給与の合計を求めます。

```
SELECT "部門番号", SUM ("給与") FROM "従業員名簿" GROUP BY "部門番号"
```

このステートメントは、固有の各部門番号に対して 1 つの行を返します。各行には、部門番号、およびその部門の従業員の給与の合計が含まれます。

HAVING 句

HAVING 句を使用することで、グループ化の条件を指定できます。たとえば、給与総額が 20,000,000 円を超える部門だけを表示する場合などです。この句の形式は次のとおりです:

```
HAVING 式1 リレーショナル演算子 式2
```

式1 および式2 には、フィールド名、定数値、または式を指定できます。これらの式は、SELECT 句の列式に一致する必要はありません。

リレーショナル演算子は、2 つの式をリンクするリレーショナル演算子です。

例

給与の合計が 20,000,000 円を超える部門のみを返します。

```
SELECT "部門番号", SUM ("給与") FROM "従業員名簿"
   GROUP BY "部門番号" HAVING SUM ("給与") > 20000000
```

UNION 演算子

UNION 演算子は、複数の SELECT ステートメントの結果を 1 つの結果に結合します。この 1 つの結果には、SELECT ステートメントから返されたレコードがすべて入ります。ただし、デフォルトでは、重複したレコードは返されません。重複レコードを返すには、ALL キーワードを使用します (UNION ALL)。形式は次のとおりです:

```
SELECT ステートメント UNION [ALL] SELECT ステートメント
```

UNION 演算子を使用する場合、各 SELECT ステートメントの選択リストには、同じデータタイプで同じ数の列式が含まれていて、同じ順序で指定されている必要があります。

例

```
SELECT "姓", "給与", "入社年月日" FROM "従業員名簿" UNION SELECT "名前",
"給料", "生年月日" FROM "従業員"
```

次の例は列式のデータタイプが異なる (従業員名簿の給与と昇給の姓のデータタイプが異なる) ため、有効ではありません。この例では各 SELECT ステートメントの列式の数は同じですが、データタイプごとの式の順序が同じではありません。

例

```
SELECT "姓", "給与" FROM "従業員名簿" UNION SELECT "給与", "姓" FROM "昇給"
```

ORDER BY 句

ORDER BY 句は、レコードのソート方法を指定します。SELECT ステートメントに ORDER BY 句が入っていない場合、任意の順序でレコードが返される可能性があります。

形式は次のとおりです:

```
ORDER BY {ソート式 [DESC | ASC]}, ...
```

ソート式にはフィールド名、または使用する列式の位置を示す数値を指定できます。デフォルトではソートは昇順 (ASC) で実行されます。

例

姓でソートしてから名でソートします。

```
SELECT "従業員番号", "姓", "名" FROM "従業員名簿" ORDER BY "姓", "名"
```

2 番目の例では、列式の位置を示す数値として 2 と 3 を指定しているため姓と名を明示的に指定した 1 番目の例と同じ順序で返すことができます。

```
SELECT "従業員番号", "姓", "名" FROM "従業員名簿" ORDER BY 2, 3
```

メモ FileMaker Server では、FileMaker Pro の言語のソートやデフォルトの言語に中立なソート順とは異なる Unicode バイナリのソート順を使用します。

OFFSET 句と FETCH FIRST 句

OFFSET 句と FETCH FIRST 句は、結果セットの特定の開始ポイントから始まる行の指定範囲を返すために使用します。大きな結果セットから取得する行の範囲を制限できるので、データを「ページ」で区切って効率を向上させることができます。

OFFSET 句は、データを返す前にスキップする行の数を示します。OFFSET 句を SELECT ステートメントで使用しない場合、開始行は 0 になります。FETCH FIRST 句は、OFFSET 句で指定した開始ポイントから開始する返す行の数を 1 以上の符号なしの整数またはパーセンテージとして指定します。OFFSET 句と FETCH FIRST 句は両方とも SELECT ステートメントで使用します。OFFSET 句を先に指定します。

OFFSET 句と FETCH FIRST 句は、サブクエリーでサポートされていません。

OFFSET の形式

OFFSET の形式は次のとおりです:

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

n は符号なしの整数です。結果セットで返される行数よりも n が大きい場合、何の結果も返されず、エラーメッセージも表示されません。

ROWS は ROW と同じです。

FETCH FIRST の形式

FETCH FIRST の形式は次のとおりです:

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

n は返される行の数です。n を省略した場合のデフォルト値は 1 です。

PERCENT が後に続く場合を除き、n は 1 以上の符号なし整数です。n の後に PERCENT が続く場合、値は正の小数値または符号なしの整数のいずれかです。

ROWS は ROW と同じです。

WITH TIES は、ORDER BY 句と共に使用する必要があります。

WITH TIES を使用すると、ピア行 (ORDER BY 句に基づいて区別されない行) も返されるので、FETCH カウント値で指定された数よりも多くの行を返すことができます。

例

姓、名の順でソートされた結果セットの 26 行目から情報を返します。

```
SELECT "従業員番号", "姓", "名" FROM "従業員名簿" ORDER BY "姓", "名" OFFSET 25
ROWS
```

10 行のみを返すように指定します。

```
SELECT "従業員番号", "姓", "名" FROM "従業員名簿" ORDER BY "姓", "名" OFFSET 25
ROWS FETCH FIRST 10 ROWS ONLY
```

10 行とそのピア行 (ORDER BY 句に基づいて区別されない行) を返します。

```
SELECT "従業員番号", "姓", "名" FROM "従業員名簿" ORDER BY "姓", "名" OFFSET 25
ROWS FETCH FIRST 10 ROWS WITH TIES
```

FOR UPDATE 句

FOR UPDATE 句は、SQL カーソルによる位置付け更新または位置付け削除を実行します。形式は次のとおりです:

```
FOR UPDATE [OF 列式]
```

列式は、更新するデータベーステーブル内のフィールド名をコンマで区切ったリストです。列式はオプションで省略可能です。

例

給与フィールドの値が 2,000,000 円を超える、従業員名簿データベース内のすべてのレコードを返します。

```
SELECT * FROM "従業員番号" WHERE "給与" > 2000000
   FOR UPDATE OF "姓", "名", "給与"
```

各レコードは取得時にロックされます。レコードを更新または削除する場合、ロックは変更を確定するまで維持されます。その他の場合は次のレコードを取得するとロックは解除されます。

例

使用する列	SQL の例
文字列定数	`SELECT 'CatDog' FROM "営業社員"`
数値定数	`SELECT 999 FROM "営業社員"`
日付定数	`SELECT DATE '2021-06-05' FROM "営業社員"`
時刻定数	`SELECT TIME '02:49:03' FROM "営業社員"`
タイムスタンプ定数	`SELECT TIMESTAMP '2021-06-05 02:49:03' FROM "営業社員"`
テキスト列	`SELECT "会社名" FROM "営業データ"` `SELECT DISTINCT "会社名" FROM "営業データ"`
数字列	`SELECT "金額" FROM "営業データ"` `SELECT DISTINCT "金額" FROM "営業データ"`
日付列	`SELECT "売上日" FROM "営業データ"` `SELECT DISTINCT "売上日" FROM "営業データ"`
時刻列	`SELECT "売上時刻" FROM "営業データ"` `SELECT DISTINCT "売上時刻" FROM "営業データ"`
タイムスタンプ列	`SELECT "売上タイムスタンプ" FROM "営業データ"` `SELECT DISTINCT "売上タイムスタンプ" FROM "営業データ"`
BLOB[a] 列	`SELECT "会社パンフレット" FROM "営業データ"` `SELECT GETAS("会社ロゴ", 'JPEG') FROM "営業データ"`
ワイルドカード *	`SELECT * FROM "営業社員"` `SELECT DISTINCT * FROM "営業社員"`

a. BLOB は、FileMaker Pro データベースファイルのオブジェクトフィールドです。

これらの例に関する注意

列は FileMaker Pro データベースファイルのフィールドの参照です (フィールドには複数の値が含まれている場合があります)。

アスタリスク (*) のワイルドカード文字は、「すべて」を簡単に指定する方法です。たとえば、`SELECT * FROM "営業社員"`では、結果は営業社員テーブル内のすべての行になります。`SELECT DISTINCT * FROM "営業社員"`の例では、結果は営業社員テーブル内にある固有な (重複しない) 行すべてになります。

- FileMaker ソフトウェアは空の文字列のデータを保存しないため、次のクエリーでは常にレコードが返されません:

```
SELECT * FROM テスト WHERE c =''
SELECT * FROM テスト WHERE c <>''
```

- バイナリデータで `SELECT` を使用している場合は、返すストリームを指定する `GetAs()` 関数を使用する必要があります。詳細については、「オブジェクトフィールドの内容の取得: CAST() 関数と GetAs() 関数」を参照してください。

オブジェクトフィールドの内容の取得: CAST() 関数と GetAs() 関数

オブジェクトフィールドから、ファイル参照情報、バイナリデータ、または特定のファイルタイプのデータを取得することができます。

- ファイルへのファイルパス、ピクチャ、または QuickTime ムービーなど、オブジェクトフィールドからファイル参照情報 (ファイルパスなど) を取得するには、`CAST()` 関数を使用した `SELECT` ステートメントを使用します。
- ファイルデータまたは JPEG バイナリデータが存在する場合、`GetAS( "フィールド名", 'JPEG')` を使用した `SELECT` ステートメントによってデータがバイナリ形式で取得されます。存在しない場合は、フィールド名を使用した `SELECT` ステートメントは `NULL` を返します。

例

ファイル参照情報を取得するには、`CAST()` 関数を使用した `SELECT` ステートメントを使用します。

```
SELECT CAST("会社パンフレット" AS VARCHAR) FROM "営業データ"
```

この例で取得する内容は次のとおりです :

- FileMaker Pro を使用してオブジェクトフィールドにファイルを挿入してファイルへの参照のみを保存する場合、`SELECT` ステートメントによってタイプ `SQL_VARCHAR` というファイル参照情報が取得されます。
- FileMaker Pro を使用してオブジェクトフィールドにファイルの内容を挿入すると、`SELECT` ステートメントによってファイル名が取得されます。
- 別のアプリケーションからオブジェクトフィールドにファイルがインポートされると、`SELECT` ステートメントは「?」を表示します (FileMaker Pro では、ファイルは **名称未設定.dat** と表示されます)。

データをバイナリ形式で取得するには、次の方法で `SELECT` ステートメントと `GetAs()` 関数を使用できます:

- `GetAs()` 関数を `DEFAULT` オプションと使用する場合、ストリームタイプを明示的に定義する必要なく、オブジェクトのマスタストリームを取得します。

  例

  ```
  SELECT GetAs("会社パンフレット", 'DEFAULT') FROM "営業データ"
  ```

- オブジェクトから個々のストリームタイプを取得するには、FileMaker Pro のオブジェクトフィールドにデータが挿入された方法に基づいたファイルタイプを指定した `GetAs()` 関数を使用します。

  例

  `[`**挿入**`]` > `[`**ファイル...**`]` コマンドでデータが挿入された場合、`GetAs()` 関数で `'FILE'` を指定します。

  ```
  SELECT GetAs("会社パンフレット", 'FILE') FROM "営業データ"
  ```

例

[**挿入**] > [**ピクチャ...**] コマンド、ドラッグ & ドロップ、またはクリップボードから貼り付け (ペースト) してデータが挿入された場合、次の一覧の 'JPEG' などのファイルタイプの 1 つを指定します。

```
SELECT GetAs("会社ロゴ", 'JPEG') FROM "会社アイコン"
```

ファイルタイプ	説明
'GIFf'	Graphics Interchange Format
'JPEG'	写真イメージ
'TIFF'	デジタルイメージのラスタファイル形式
'PDF'	Portable Document Format
'PNGf'	Bitmap イメージ形式

DELETE ステートメント

DELETE ステートメントを使用して、データベーステーブルからレコードを削除します。DELETE ステートメントの形式は次のとおりです:

```
DELETE FROM テーブル名 [ WHERE { 条件 } ]
```

メモ WHERE 句は削除するレコードを決定します。WHERE キーワードを省略すると、テーブル内のすべてのレコードが削除されますが、テーブルには影響ありません。

例

従業員名簿テーブルからレコードを削除します。

```
DELETE FROM "従業員名簿" WHERE "従業員番号" = 'E10001'
```

それぞれの DELETE ステートメントは、WHERE 句の条件を満たすすべてのレコードを取り除きます。この例では、従業員番号 「E10001」を含むすべてのレコードが削除されます。従業員番号は従業員名簿テーブル内の固有な値なので、1 レコードだけが削除されます。

INSERT ステートメント

INSERT ステートメントを使用して、データベーステーブルにレコードを作成します。次のいずれかを指定できます:

- 新しいレコードとして挿入する値のリスト
- 新しいレコードのセットとして挿入するために、他のテーブルのデータをコピーする SELECT ステートメント

INSERT ステートメントの形式は次のとおりです:

```
INSERT INTO テーブル名 [(列名, ...)]VALUES (値式, ...)
```

列名は、省略可能な列名の一覧で、VALUES 句で値が指定される列の名前と順序を示します。列名を省略した場合は、値式 (値式) に、テーブルで定義されているすべての列の値を、テーブルで定義されている順序と同じ列順で指定する必要があります。列名では、最終日[4] などのフィールドの繰り返しも指定することができます。

値式は、新しいレコードの列の値を指定する式のリストです。通常、この式は列について定数値ですが、サブクエリーを指定することもできます。文字列の値は、シングルクォーテーション (') の組で囲む必要があります。シングルクォーテーションで囲まれた文字列の値にシングルクォーテーションを含めるには、同時に 2 つのシングルクォーテーションを使用します (例: 'Don''t')。

サブクエリーはカッコ (()) で囲む必要があります。

例

式のリストを挿入します。

```
INSERT INTO "従業員名簿" ("姓", "名", "従業員番号", "給与", "入社年月日")
   VALUES ('小田', '弘', 'E22345', 2750000, DATE '2019-06-05')
```

それぞれの INSERT ステートメントにより、データベーステーブルに 1 レコードが追加されます。この例では、従業員名簿データベーステーブルに 1 レコードが追加されました。5 つの列に値が指定されます。テーブルの残りの列には、NULL を意味する空白の値が割り当てられます。

メモ オブジェクトフィールドでは、引数化されたステートメントを準備し、アプリケーションからデータをストリームしていない限り、INSERT 処理を行うことができるのはテキストのみです。バイナリデータを使用するには、シングルクォーテーションでファイル名を囲んでファイル名を割り当てるか、PutAs() 関数を使用します。ファイル名を指定すると、ファイル拡張子からファイルタイプが推定されます:

```
INSERT INTO "テーブル名" ("オブジェクト名") VALUES(? AS 'ファイル名.ファイル拡張子')
```

サポートされていないファイルタイプは、タイプ FILE として挿入されます。

PutAs() 関数を指定する場合、PutAs(col, 'タイプ') の形式でタイプを指定します。タイプ値は、サポートされているファイルタイプです。サポートされているファイルタイプの詳細については、16 ページの「オブジェクトフィールドの内容の取得: CAST() 関数と GetAs() 関数」を参照してください。

SELECT ステートメントは、列名のリストで指定されたそれぞれの列名の値を返すクエリーです。値式のリストの代わりに SELECT ステートメントを使用すると、あるテーブルから行のセットを選択し、INSERT ステートメントでそれを別のテーブルに挿入することができます。

例

SELECT ステートメントを使用して挿入します。

```
INSERT INTO "従業員名簿1" ("姓", "名", "従業員番号", "部門", "給与")
   SELECT "姓", "名", "従業員番号", "部門", "給与" FROM "従業員名簿"
   WHERE "部門" = 'D050'
```

このタイプの INSERT ステートメントでは、挿入する列の数が SELECT ステートメントの列の数と一致する必要があります。また、他のタイプの INSERT ステートメントで列のリストが値式のリストに対応するのと同様に、挿入する列のリストが SELECT ステートメントの列のリストに対応する必要があります。たとえば、挿入される先頭の列は選択された先頭の列に対応し、挿入される 2 番目の列は選択された 2 番目の列に対応するというようになります。

これらの対応する列のサイズとデータタイプは、互換性がある必要があります。SELECT ステートメント内のそれぞれの列は、INSERT ステートメント内の対応する列に対する通常の INSERT/UPDATE ステートメントで ODBC または JDBC クライアントドライバが処理できるデータタイプにしてください。SELECT ステートメント内の列の値のサイズが対応する INSERT ステートメント内の列のサイズより大きい場合は値が制限されます。

SELECT ステートメントは、値が挿入される前に評価されます。

UPDATE ステートメント

UPDATE ステートメントを使用して、データベーステーブル内のレコードを変更します。UPDATE ステートメントの形式は次のとおりです:

```
UPDATE テーブル名 SET 列名 = 式, ...[ WHERE { 条件 } ]
```

列名は、値を変更する列の名前です。1 つのステートメントで複数の列を変更できます。

式は、列の新しい値です。

通常、この式は列について定数値ですが、サブクエリーを指定することもできます。文字列の値は、シングルクォーテーション (') の組で囲む必要があります。シングルクォーテーションで囲まれた文字列の値にシングルクォーテーションを含めるには、同時に 2 つのシングルクォーテーションを使用します (例: 'Don''t')。

サブクエリーはカッコ (()) で囲む必要があります。

WHERE 句は、任意の有効な句です。これによって更新するレコードを決定します。

例

従業員名簿テーブルに対する UPDATE ステートメント。

```
UPDATE "従業員名簿" SET "給与" =32000, "控除" =1 WHERE "従業員番号" = 'E10001'
```

それぞれの UPDATE ステートメントは、WHERE 句の条件を満たすすべてのレコードを変更します。この例では、従業員番号「E10001」を含むすべての従業員について、給与と控除が変更されます。従業員番号は従業員名簿テーブル内の固有な値なので、1 レコードだけが更新されます。

例

サブクエリーを使用した従業員名簿テーブルに対する UPDATE ステートメント。

```
UPDATE "従業員名簿" SET "給与" = (SELECT avg("給与") from "従業員名簿") WHERE
"従業員番号" = 'E10001'
```

この例では、従業員番号 E10001 を含む従業員について、給与を会社の平均給与に変更します。

メモ オブジェクトフィールドでは、引数化されたステートメントを準備し、アプリケーションからデータをストリームしていない限り、UPDATE 処理を行うことができるのはテキストのみです。バイナリデータを使用するには、シングルクォーテーションでファイル名を囲んでファイル名を割り当てるか、PutAs() 関数を使用します。ファイル名を指定すると、ファイル拡張子からファイルタイプが推定されます:

```
UPDATE "テーブル名" SET("オブジェクト名") = ? AS 'ファイル名.ファイル拡張子'
```

サポートされていないファイルタイプは、タイプ FILE として挿入されます。

PutAs() 関数を指定する場合: PutAs(col, 'タイプ') の形式でタイプを指定します。タイプ値は、サポートされているファイルタイプです。サポートされているファイルタイプの詳細については、16 ページの「オブジェクトフィールドの内容の取得: CAST() 関数と GetAs() 関数」を参照してください。

CREATE TABLE ステートメント

CREATE TABLE ステートメントを使用して、データベースファイル内にテーブルを作成します。CREATE TABLE ステートメントの形式は次のとおりです:

```
CREATE TABLE テーブル名 (テーブル要素リスト [, テーブル要素リスト...])
```

ステートメント内で、各列の名前とデータタイプを指定します。

- テーブル名は、テーブルの名前です。テーブル名には、100文字という制限があります。同じ名前のテーブルがすでに定義されていないようにしてください。テーブル名は、アルファベット文字で始まる必要があります。テーブル名がアルファベット文字以外で始まる場合は、ダブルクォーテーションマークで囲む必要があります (クォーテーションマークで囲まれた識別子)。
- テーブル要素リストの形式は次の通りです:

```
フィールド名フィールドタイプ [[繰り返し]]
[DEFAULT数式] [UNIQUE | NOT NULL | PRIMARY KEY | GLOBAL]
[EXTERNAL 相対パス文字列 [SECURE | OPEN 計算パス文字列]]
```

  - フィールド名はフィールドの名前です。ファイル名は固有である必要があります。フィールド名はアルファベット文字で始めます。フィールド名がアルファベット文字以外で始まる場合は、ダブルクォーテーションマークで囲む必要があります (クォーテーションマークで囲まれた識別子)。

    例

    「姓」という名前のフィールドに対する CREATE TABLE ステートメントは、次のようになります :

    ```
    CREATE TABLE "従業員名簿" (ID INT PRIMARY KEY, "名" VARCHAR(20), "姓"
    VARCHAR(20))
    ```

  - CREATE TABLE ステートメントの 繰り返しには、フィールドタイプの後に角カッコで囲んだ 1 から 32000 の数値を使用して指定します。

    例

    ```
    "従業員番号" INT[4]
    "姓" VARCHAR(20)[4]
    ```

  - フィールドタイプは、NUMERIC、DECIMAL、INT、DATE、TIME、TIMESTAMP、VARCHAR、CHARACTER VARYING、BLOB、VARBINARY、LONGVARBINARY、または BINARY VARYINGです。NUMERIC と DECIMAL の場合は、桁数とスケールを指定できます。例: DECIMAL(10,0)。TIME と TIMESTAMP の場合は、桁数を指定できます。例: TIMESTAMP(6)。VARCHAR と CHARACTER VARYING の場合は、文字列の長さを指定できます。

例

```
VARCHAR(255)
```

- DEFAULT キーワードでは列のデフォルト値を設定できます。値式では、定数値または式を使用できます。使用可能な式は、USER、USERNAME、CURRENT_USER、CURRENT_DATE、CURDATE、CURRENT_TIME、CURTIME、CURRENT_TIMESTAMP、CURTIMESTAMP、および NULL です。
- 列を UNIQUE に定義すると、自動的に、FileMaker Pro データベースファイル内の対応するフィールドの入力値の制限オプション [**ユニークな値**] が選択されます。
- 列を NOT NULL に定義すると、自動的に、FileMaker Pro データベースファイル内の対応するフィールドの入力値の制限オプション [**空欄不可**] が選択されます。このフィールドには、FileMaker Pro の [データベースの管理] ダイアログボックスの [**フィールド**] タブで [**空欄不可**] としてフラグが付けられます。
- 列をオブジェクトフィールドとして定義するには、フィールドタイプに BLOB、VARBINARY、または BINARY VARYING を使用します。
- データを外部に保存するオブジェクトフィールドとして列を定義するには、EXTERNAL キーワードを使用します。相対パス文字列は、FileMaker Pro データベースの場所に関連し、データが外部に保存されるフォルダを定義します。このパスは、FileMaker Pro の [オブジェクトの管理] ダイアログボックスの基本ディレクトリとして指定する必要があります。セキュア格納の場合は 「SECURE」、オープン格納の場合は「OPEN」を指定します。オープン格納を使用する場合、計算パス文字列はオブジェクトが保存される相対パス文字列内のフォルダです。パスのフォルダ名にはスラッシュ (/) を使用する必要があります。

例

使用する列	SQL の例
テキスト列	`CREATE TABLE "T1" ("C1" VARCHAR, "C2" VARCHAR (50), "C3" VARCHAR (1001), "C4"VARCHAR (500276))`
テキスト列、NOT NULL	`CREATE TABLE "T1NN" ("C1" VARCHAR NOT NULL, "C2" VARCHAR (50) NOT NULL, "C3" VARCHAR (1001) NOT NULL, "C4" VARCHAR (500276) NOT NULL)`
数字列	`CREATE TABLE "T2" ("C1" DECIMAL, "C2" DECIMAL (10,0), "C3" DECIMAL (7539,2), "C4" DECIMAL (497925,301))`
日付列	`CREATE TABLE "T3" ("C1" DATE, "C2" DATE, "C3" DATE, "C4" DATE)`
時刻列	`CREATE TABLE "T4" ("C1" TIME, "C2" TIME, "C3" TIME, "C4" TIME)`
タイムスタンプ列	`CREATE TABLE "T5" ( "C1" TIMESTAMP, "C2" TIMESTAMP, "C3" TIMESTAMP, "C4" TIMESTAMP)`
オブジェクトフィールドの列	`CREATE TABLE "T6" ("C1" BLOB, "C2" BLOB, "C3" BLOB," C4" BLOB )`
外部に保存するオブジェクトフィールドの列	`CREATE TABLE "T7" ("C1" BLOB EXTERNAL 'Files/MyDatabase/' SECURE)` `CREATE TABLE "T8" ("C1" BLOB EXTERNAL 'Files/MyDatabase/' OPEN 'Objects' )`

TRUNCATE TABLE ステートメント

TRUNCATE TABLE ステートメントは指定したテーブルのレコードを素早く削除し、すべてのデータのテーブルを空にします。

TRUNCATE TABLE テーブル名

TRUNCATE TABLE ステートメントで WHERE 句を指定することはできません。TRUNCATE TABLE ステートメントはすべてのレコードを削除します。

テーブル名で指定されたテーブルのレコードのみを削除します。関連テーブルのレコードは削除されません。

レコードデータを削除するには TRUNCATE TABLE ステートメントによってテーブルの全レコードをロックできる必要があります。他のユーザによってテーブルのレコードがロックされていると、FileMaker ソフトウェアはエラーコード 301 (「別のユーザがレコードを使用中です」) を返します。

ALTER TABLE ステートメント

ALTER TABLE ステートメントを使用して、データベースファイル内の既存のテーブルの構造を変更します。各ステートメントで変更できる列は 1 つだけです。ALTER TABLE ステートメントの形式は次のとおりです:

ALTER TABLE テーブル名 ADD [COLUMN] 列定義

ALTER TABLE テーブル名 DROP [COLUMN] 非修飾の列名

ALTER TABLE テーブル名 ALTER [COLUMN] 列定義 SET DEFAULT 式

ALTER TABLE テーブル名 ALTER [COLUMN] 列定義 DROP DEFAULT

ALTER TABLE ステートメントを使用する前に、テーブルの構造と変更の内容を確認する必要があります。

例

目的	SQL の例
列を追加する	ALTER TABLE "営業社員" ADD "列 1" VARCHAR
列を削除する	ALTER TABLE "営業社員" DROP "列 1"
列のデフォルト値を設定する	ALTER TABLE "営業社員" ALTER "会社" SET DEFAULT 'Claris'
列のデフォルト値を削除する	ALTER TABLE "営業社員" ALTER "会社" DROP DEFAULT

メモ SET DEFAULT および DROP DEFAULT はテーブルの既存の行には影響しませんが、その後テーブルに追加される行のデフォルト値を変更します。

CREATE INDEX ステートメント

CREATE INDEX ステートメントを使用して、データベースファイル内での検索を高速化します。CREATE INDEX ステートメントの形式は次のとおりです:

CREATE INDEX ON テーブル名.列名
CREATE INDEX ON テーブル名 (列名)

CREATE INDEX は、1 つの列に対してサポートされています (複数の列の索引はサポートされていません)。FileMaker Pro データベースファイルのオブジェクトフィールドタイプ、集計フィールド、グローバル格納オプションが設定されているフィールド、または非保存の計算フィールドに対応する列に対しては、索引を作成できません。

テキスト列の**索引**を作成すると、FileMaker Pro データベースファイルの対応するフィールドに対して、自動的に [格納] のオプション [**最小限**] が選択されます。テキスト以外の列 (または日本語のテキストとして書式設定されている列) の索引を作成すると、FileMaker Pro データベースファイルの対応するフィールドに対して、自動的に [**格納**] のオプション [**すべて**] が選択されます。

どの列の索引を作成した場合でも、FileMaker Pro データベースファイルの対応するフィールドに対して、自動的に**索引**オプション [**必要時に索引を自動設定**] が選択されます。

FileMaker ソフトウェアは必要に応じて索引を自動的に作成します。CREATE INDEX を使用すると、索引はオンデマンドではなくただちに構築されます。

例

```
CREATE INDEX ON "営業社員"."営業社員番号"
```

DROP INDEX ステートメント

DROP INDEX ステートメントを使用して、データベースファイルから索引を削除します。DROP INDEX ステートメントの形式は次のとおりです:

```
DROP INDEX ON テーブル名.列名
DROP INDEX ON テーブル名 (列名)
```

データベースファイルの容量が大きすぎる場合や、特定のフィールドをクエリーで使用する頻度が高くない場合は、索引を削除します。

索引設定された多くのテキストフィールドが含まれる非常に容量の大きい FileMaker Pro データベースファイルを操作している場合に、クエリーのパフォーマンスが低いときは、一部のフィールドから索引を削除することを検討してください。また、SELECT ステートメントでほとんど使用しないフィールドからも索引を削除することを検討します。

どの列の**索引**を削除した場合でも、FileMaker Pro データベースファイルの対応するフィールドに対して、自動的に [格納] のオプション [**なし**] が選択され、[**必要時に索引を自動設定**] の選択は解除されます。

PREVENT INDEX CREATION 属性はサポートされていません。

例

```
DROP INDEX ON "営業社員"."営業社員番号"
```

SQL 式

SELECT ステートメントの WHERE 句、HAVING 句、および ORDER BY 句で式を使用して、高度で詳細なデータベースクエリーを作成します。有効な式の要素を次に示します:

- フィールド名
- 定数
- 指数または科学表記
- 数値演算子
- 文字演算子
- 日付演算子
- リレーショナル演算子
- 論理演算子
- 関数

フィールド名

最も一般的な式は、「計算」や「営業データ.請求書番号」などの単純なフィールド名です。

定数

定数とは、変わらない値です。たとえば、価格 `* 1.05` という式では、値 1.05 が定数です。または、「`6` 月の日数」という定数に値 30 を割り当てることができます。

文字定数は、シングルクォーテーション (') の組で囲む必要があります。シングルクォーテーションで囲まれた文字定数にシングルクォーテーションを含めるには、同時に 2 つのシングルクォーテーションを使用します (例: `'Don''t'`)。

ODBC および JDBC アプリケーションの場合、FileMaker ソフトウェアは中カッコ ({}) で囲まれた ODBC/JDBC 形式の日付、時刻、およびタイムスタンプ定数を処理します。

例

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

FileMaker ソフトウェアでは、型指定子 (D、T、TS) に大文字、小文字が使用できます。型指定子の後にスペースをいくつでも入れることができますし、スペースを省略することもできます。

FileMaker ソフトウェアは、SQL-92 の構文で中カッコ ({}) で囲まれていない ISO 形式の日付と時刻も処理します。

例

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

FileMaker Pro の ExecuteSQL 関数は、中カッコ ({}) を使用しない SQL-92 構文の ISO 日付と時刻の書式のみを処理します。

定数	使用可能な構文の例
テキスト	`'大阪市'`
数字	`1.05`

日付	`DATE '2019-06-05'`
	`{ D '2019-06-05' }`
	`{06/05/2019}`
	`{06/05/19}`
	メモ: 2 桁の西暦の構文は、ODBC/JDBC 形式、または SQL-92 形式ではサポートされていません。

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

タイムスタンプ	`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}`
	この 2 桁の西暦の構文を使用したフィールドに対しては、FileMaker Pro データベースファイルの入力値の**制限オプションとして、**［**西暦4桁の日付**］が選択されていないことを確認してください。
	メモ: 2 桁の西暦の構文は、ODBC/JDBC 形式、または SQL-92 形式ではサポートされていません。

日付と時刻の値を入力する際は、データベースファイルのロケールの書式と同じ書式を使用します。たとえば、データベースがイタリア語のシステムで作成された場合は、イタリア語の日付書式と時刻書式を使用します。

指数または科学表記

数字は科学表記を使用して表記できます。

例

```
SELECT "列 1" / 3.4E+7 FROM "テーブル 1" WHERE "計算" < 3.4E-6 * "列 2"
```

数値演算子

数値式には: +、-、*、/、および ^、または ** (指数) を含めることができます。
数値式の前に、単項のプラス (+) またはマイナス (-) を付けることができます。

文字演算子

複数の文字を連結することができます。次の例では姓は「田中」で名は「一郎」です。

演算子	連結	例	結果
+	後部の空白文字を保持します。	姓 + 名	'田中 一郎 '
-	後部の空白文字を末尾に移動します。	姓 - 名	'田中一郎 '

日付演算子

日付を変更することができます。次の例の入社年月日は、`DATE '2019-01-30'` です。

演算子	日付の処理	例	結果
+	日付に日数を加算します。	`"入社年月日" + 5`	DATE '2019-02-04'
-	2 つの日付の間の日数を調べます。	`"入社年月日" - DATE'2019-01-01'`	29
	日付から日数を減算します。	`"入社年月日" - 10`	DATE '2019-01-20'

その他の例

```
SELECT "売上日", "売上日" + 30 AS "統計" FROM "営業データ"
SELECT "売上日", "売上日" - 30 AS "統計" FROM "営業データ"
```

リレーショナル演算子

演算子	意味
=	等しい
<>	等しくない
>	大きい
>=	大きいか等しい
<	小さい
<=	小さいか等しい
LIKE	パターンに一致する
NOT LIKE	パターンに一致しない
IS NULL	NULL に等しい
IS NOT NULL	NULL に等しくない
BETWEEN	下限と上限の間の値の範囲
IN	指定された値のセットの 1 つ、またはサブクエリーの 1 つ
NOT IN	指定された値のセットの 1 つでない、またはサブクエリーの 1 つでない
EXISTS	サブクエリーによって少なくとも 1 つのレコードが返された場合は「真」
ANY	サブクエリーによって返された各値をある値と比較する (演算子の前に、=、<>、>、>=、<、または <= を付ける必要があります) =Any は In と同じです
ALL	サブクエリーによって返された各値をある値と比較する (演算子の前に、=、<>、>、>=、<、または <= を付ける必要があります)

例

```
SELECT "営業データ"."請求書番号" FROM "営業データ"
   WHERE "営業データ"."営業社員番号" = 'SP-1'
SELECT "営業データ"."金額" FROM "営業データ" WHERE "営業データ"."請求書番号" <> 125
SELECT "営業データ"."金額" FROM "営業データ" WHERE "営業データ"."金額" > 300000
SELECT "営業データ"."売上時刻" FROM "営業データ"
   WHERE "営業データ"."売上時刻" < '12:00:00'
SELECT "営業データ"."会社名" FROM "営業データ"
   WHERE "営業データ"."会社名" LIKE '%大学'
SELECT "営業データ"."会社名" FROM "営業データ"
   WHERE "営業データ"."会社名" NOT LIKE '%大学'
SELECT "営業データ"."金額" FROM "営業データ" WHERE "営業データ"."金額" IS NULL
SELECT "営業データ"."金額" FROM "営業データ" WHERE "営業データ"."金額" IS NOT NULL
SELECT "営業データ"."請求書番号" FROM "営業データ"
   WHERE "営業データ"."請求書番号" BETWEEN 1 AND 10
SELECT COUNT ("営業データ"."請求書番号") AS "統計"
   FROM "営業データ" WHERE "営業データ"."請求書番号" IN (50,250,100)
SELECT COUNT ("営業データ"."請求書番号") AS "統計"
   FROM "営業データ" WHERE "営業データ"."請求書番号" NOT IN (50,250,100)
SELECT COUNT ("営業データ"."請求書番号") AS "統計" FROM "営業データ"
   WHERE "営業データ"."請求書番号" NOT IN (SELECT "営業データ"."請求書番号"
   FROM "営業データ" WHERE "営業データ"."営業社員番号" = 'SP-4')
SELECT *
   FROM "営業データ" WHERE EXISTS (SELECT "営業データ"."金額"
   FROM "営業データ" WHERE "営業データ"."営業社員番号" IS NOT NULL)
SELECT *
   FROM "営業データ" WHERE "営業データ"."金額" = ANY (SELECT "営業データ"."金額"
   FROM "営業データ" WHERE "営業データ"."営業社員番号" = 'SP-1')
SELECT *
   FROM "営業データ" WHERE "営業データ"."金額" = ALL (SELECT "営業データ"."金額"
   FROM "営業データ" WHERE "営業データ"."営業社員番号" IS NULL)
```

論理演算子

2 つ以上の条件を結合することができます。次に示すように、AND または ORを使用して条件を関連させる必要があります:

給与 = 4000000 AND 控除 = 1

論理 NOT 演算子は、次に示すように反対の意味にするために使用します:

NOT (給与 = 4000000 AND 控除 = 1)

例

```
SELECT * FROM "営業データ" WHERE "営業データ"."会社名"
   NOT LIKE '%大学' AND "営業データ"."金額" > 300000
SELECT * FROM "営業データ" WHERE ( "営業データ"."会社名"
    LIKE '%大学' OR "営業データ"."金額" > 300000)
    AND "営業データ"."営業社員番号" = 'SP-1'
```

演算子の優先順位

式が複雑になるに連れて、式が評価される順序が重要になってきます。次の表は、演算子が評価される順序を示します。最初に 1 行目の演算子、続いて 2 行目の演算子というように評価されます。同じ行にある演算子は、式の左から右に評価されます。

優先順位	演算子
1	単項の「-」、単項の「+」
2	^, **
3	*, /
4	+, -
5	=, <>, <, <=, >, >=, Like, Not Like, Is Null, Is Not Null, Between, In, Exists, Any, All
6	Not
7	AND
8	OR

例

```
WHERE ("給与" > 4000000 OR "入社年月日" > (DATE '2008-01-30') AND
"部門" = 'D101'
```

AND が最初に評価されるため、このクエリーでは、2008 年 1 月 30 日より後に入社した部門 D101 の従業員の他に、部門や入社年月日に関係なく給与が 4,000,000 円を超えるすべての従業員も抽出されます。

この句を強制的に異なる順序で評価するには、カッコを使用して、最初に評価する条件を囲みます。

```
WHERE ("給与" > 4000000 OR "入社年月日" > DATE '2008-01-30') AND
"部門" = 'D101'
```

この例では、給与が 4,000,000 円を超えるか、または 2008 年 1 月 30 日より後に入社した部門 D101 の従業員が抽出されます。

SQL 関数

Claris では FileMaker プラットフォームに SQL 標準を実装して式で使用できるさまざまな関数をサポートしています。文字列を返す関数、数字を返す関数、日付を返す関数、および関数の引数によって満たされる条件に応じた値を返す関数があります。

統計関数

統計関数は、レコードのセットから 1 つの値を返します。統計関数は、SELECT ステートメントの一部として使用するか、AVG("給与") のようにフィールド名とともに使用するか、または AVG ("給与" * 1.07) のように列式と組み合わせて使用することができます。

列式の前に DISTINCT 演算子を指定して、重複する値を取り除くことができます。

例

```
COUNT (DISTINCT "姓" )
```

この例では、固有な姓の値のみがカウントされます。

統計関数	返される値
SUM	数値フィールド式の値の合計。たとえば、SUM ("給与") は、すべての給与フィールドの値の合計を返します。
AVG	数値フィールド式の値の平均。たとえば、AVG ("給与") は、すべての給与フィールドの値の平均を返します。
COUNT	任意の数値式の値の数。たとえば、COUNT ("名前") は、名前の値の数を返します。COUNT をフィールド名とともに使用した場合、COUNT は NULL 以外のフィールドの値の数を返します。COUNT(*) は特別な例で、NULL 値が含まれるレコードを含む、セット内のレコードの数を返します。
MAX	任意のフィールド式の最大値。たとえば、MAX ("給与") は、給与フィールドの最大値を返します。
MIN	任意のフィールド式の最小値。たとえば、MIN ("給与") は、給与フィールドの最小値を返します。

例

```
SELECT SUM ("営業データ"."金額") AS "統計" FROM "営業データ"
SELECT AVG ("営業データ"."金額") AS "統計" FROM "営業データ"
SELECT COUNT ("営業データ"."金額") AS "統計" FROM "営業データ"
SELECT MAX ("営業データ"."金額") AS "統計" FROM "営業データ"
   WHERE "営業データ"."金額" < 300000
SELECT MIN ("営業データ"."金額") AS "統計" FROM "営業データ"
   WHERE "営業データ"."金額" > 300000
```

他の関数に対する引数として統計関数を使用することはできません。他の関数に対する引数として統計関数を使用した場合、FileMaker ソフトウェアはエラーコード 8309 ([Expressions involving aggregations are not supported]) を返します。たとえば、SUM という統計関数は ROUND 関数の引数として使用できないため、次のステートメントは無効になります:

例

```
SELECT ROUND(SUM("給与"), 0) FROM "給与支払名簿"
```

ただし、統計関数では引数として数値を返す関数を使用することは可能です。次のステートメントは有効になります。

例

```
SELECT SUM(ROUND("給与", 0)) FROM "給与支払名簿"
```

文字列を返す関数

文字列を返す関数	説明	例
CHR	ASCII コードを 1 文字の文字列に変換します。	CHR(67) は **C** を返します。
CURRENT_USER	接続時に指定されたログイン ID を返します。	
DAYNAME	指定した日付に対応する曜日の名前を返します。	
RTRIM	文字列から後部の空白を削除します。	RTRIM(' ABC ') は「 ABC」を返します。
TRIM	文字列から前部および後部の空白を削除します。	TRIM(' ABC ') は「ABC」を返します。
LTRIM	文字列から前部の空白を削除します。	LTRIM(' ABC') は「ABC」を返します。
UPPER	文字列の各文字を大文字に変更します。	UPPER('Allen') は「ALLEN」を返します。
LOWER	文字列の各文字を小文字に変更します。	LOWER('Allen') は「allen」を返します。
LEFT	文字列の最も左側の文字を返します。	LEFT('Mattson',3) は「Mat」を返します。
MONTHNAME	歴月の名前を返します。	
RIGHT	文字列の最も右側の文字を返します。	RIGHT(Mattson', 4) は「tson」を返します。
SUBSTR SUBSTRING	文字列のサブ文字列を返します。文字列、抽出する最初の文字、抽出する文字数 (オプション) の引数を指定します。	SUBSTR('Conrad', 2, 3) は「onr」を返します。 SUBSTR('Conrad', 2) は「onrad」を返します。
SPACE	空白の文字列を生成します。	SPACE(5) は「 」を返します。
STRVAL	任意のタイプの値を文字列に変換します。	STRVAL('Woltman') は「Woltman」を返します。 STRVAL(5 * 3) は「15」を返します。 STRVAL(4 = 5) は「False」を返します。 STRVAL(DATE '2019-12-25') は「2019-12-25」を返します。

文字列を返す関数	説明	例
TIME TIMEVAL	時刻を文字列として返します。	午後 9:49 の場合、TIME() は「**21:49:00**」を返します。
USERNAME USER	接続時に指定されたログイン ID を返します。	

メモ TIME() 関数は将来の対応が保証されていません。SQL 標準の CURRENT_TIME を使用してください。

例

```
SELECT CHR(67) + SPACE(1) + CHR(70) FROM "営業社員"
SELECT RTRIM(' ' + "営業社員"."営業社員番号") AS "統計" FROM "営業社員"
SELECT TRIM(SPACE(1) + "営業社員"."営業社員番号") AS "統計" FROM "営業社員"
SELECT LTRIM(' ' + "営業社員"."営業社員番号") AS "統計" FROM "営業社員"
SELECT UPPER("営業社員"."営業社員番号") AS "統計" FROM "営業社員"
SELECT LOWER("営業社員"."営業社員") AS "統計" FROM "営業社員"
SELECT LEFT("営業社員"."営業社員", 5) AS "統計" FROM "営業社員"
SELECT RIGHT("営業社員"."営業社員", 7) AS "統計" FROM "営業社員"
SELECT SUBSTR("営業社員"."営業社員番号", 2, 2) + SUBSTR("営業社員"."営業社員番号",
4, 2) AS "統計" FROM "営業社員"
SELECT SUBSTR("営業社員"."営業社員番号", 2) + SUBSTR("営業社員"."営業社員番号", 4)
AS "統計" FROM "営業社員"
SELECT SPACE(2) + "営業社員"."営業社員番号" AS "営業社員番号" FROM "営業社員"
SELECT STRVAL('60506') AS "統計" FROM "営業データ" WHERE "営業データ"."請
求書番号" = 1
```

数字を返す関数

数字を返す関数	説明	例
ABS	数値式の絶対値を返します。	
ATAN	引数のアークタンジェントをラジアンで表された角度として返します。	
ATAN2	x および y 座標のアークタンジェントをラジアンで表された角度として返します。	
CEIL CEILING	引数以上で最小の整数値を返します。	
DEG DEGREES	引数の角度の数字、つまりラジアンで表された角度を返します。	
DAY	日付の日の部分を返します。	DAY(DATE '2019-01-30') は「**30**」を返します。
DAYOFWEEK	日付式の曜日を 1 から 7 の数字で返します。	DAYOFWEEK(DATE '2004-05-01') は「**7**」を返します。
MOD	2 つの数字を除算して、除算の余りを返します。	MOD(10, 3) は「**1**」を返します。
EXP	引数で指定された乗数で累乗した自然対数 (e) のベースとなる値を返します。	
FLOOR	引数以下で最大の整数値を返します。	
HOUR	値の時間の部分を返します。	
INT	数字の整数の部分を返します。	INT(6.4321) は「**6**」を返します。
LENGTH	文字列の長さを返します。	LENGTH('ABC') は「**3**」を返します。
MONTH	日付の月の部分を返します。	MONTH(DATE '2019-01-30') は「**1**」を返します。
LN	引数の自然対数を返します。	
LOG	引数の常用対数を返します。	
MAX	2 つの数字の大きい方を返します。	MAX(66, 89) は「**89**」を返します。
MIN	2 つの数字の小さい方を返します。	MIN(66, 89) は「**66**」を返します。
MINUTE	値の分の部分を返します。	
NUMVAL	文字列を数字に変換します。文字列が有効な数字でない場合、この関数は失敗します。	NUMVAL('123') は「**123**」を返します。
PI	数学的定数 pi の定数値を返します。	
RADIANS	角度で表された引数のラジアンの数字を返します。	
ROUND	数字を四捨五入します。	ROUND(123.456,0) は「**123**」を返します。 ROUND(123.456,2) は「**123.46**」を返します。 ROUND(123.456, -2) は「**100**」を返します。
SECOND	値の秒の部分を返します。	
SIGN	引数の記号を示します: -1 は負、0 は 0、1 は正です。	

数字を返す関数	説明	例
SIN	引数のサインを返します。	
SQRT	引数の平方根を返します。	
TAN	引数のタンジェントを返します。	
YEAR	日付の年の部分を返します。	YEAR(DATE '2019-01-30') は「**2019**」を返します。

日付を返す関数

日付を返す関数	説明	例
CURDATE CURRENT_DATE	現在の日付を返します。	
CURTIME CURRENT_TIME	現在の時刻を返します。	
CURTIMESTAMP CURRENT_TIMESTAMP	現在のタイムスタンプの値を返します。	
TIMESTAMPVAL	文字列をタイムスタンプに変換します。	TIMESTAMPVAL('2019-01-30 14:00:00') は、そのタイムスタンプ値を返します。
DATE TODAY	現在の日付を返します。	今日が 2019/11/21 の場合、DATE() は「**2019-11-21**」を返します。
DATEVAL	文字列を日付に変換します。	DATEVAL('2019-01-30') は「**2019-01-30**」を返します。

メモ DATE() 関数は将来の対応が保証されていません。SQL 標準の CURRENT_DATE を使用してください。

条件関数

条件関数	説明	例
CASE WHEN	**シンプルな CASE 形式** 入力式の値を値式の値と比較して、結果を求めます。 CASE 入力式 {WHEN 値式 THEN 結果...}[ELSE 結果] END	SELECT "請求書番号", CASE "会社名" 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, "営業社員番号" FROM "営業データ"
	検索された CASE 形式 WHEN 式によって指定された条件が真かどうかに基づいて結果を返します。 CASE {WHEN ブール式 THEN 結果...}[ELSE 結果] END	SELECT "請求書番号", "金額", CASE WHEN "金額" > 3000 THEN 'Above 3000' WHEN "金額" < 1000 THEN 'Below 3000' ELSE 'Between 1000 and 3000' END, "営業社員番号" FROM "営業データ"
COALESCE	NULL でない最初の値を返します。	SELECT "営業社員番号", COALESCE("営業部長","営業社員") FROM 営業社員
NULLIF	2 つの値を比較して、2 つの値が等しい場合に NULL を返します。それ以外の場合は、最初の値を返します。	SELECT "請求書番号", NULLIF("金額", -1)、 "営業社員番号" FROM "営業データ"

FileMaker システムオブジェクト

FileMaker Pro データベースファイルには次のシステムオブジェクトが含まれます。これらのシステムオブジェクトには、SQL クエリーを使用してアクセスできます。

FileMaker システムテーブル

各 FileMaker Pro データベースファイルには 2 つのシステムテーブルがあります: FileMaker_Tables および FileMaker_Fields。ODBC アプリケーションの場合、これらのテーブルは SQLTables のカタログ関数が返す情報の中に含まれます。JDBC アプリケーションの場合、これらのテーブルは DatabaseMetaData メソッドの getTables が返す情報の中に含まれます。テーブルは ExecuteSQL 関数でも使用できます。

FileMaker_Tables

FileMaker_Tables テーブルには FileMaker Pro ファイル内で定義されたデータベーステーブルに関する情報が含まれます。

FileMaker_Tables テーブルは次の列を持つリレーションシップグラフの各テーブルオカレンスの行を含みます:

- TableName - テーブルオカレンス名です。
- TableId - テーブルオカレンスの固有の ID です。
- BaseTableName - テーブルオカレンスの作成元の基本テーブル名です。
- BaseFileName - 基本テーブルが含まれるデータベースファイルの FileMaker Pro ファイル名です。
- ModCount - このテーブルの定義で確定された変更の合計回数です。

例

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

FileMaker_Fields テーブル

FileMaker_Fields テーブルには FileMaker Pro ファイル内で定義されたフィールドに関する情報が含まれます。

FileMaker_Fields テーブルは次の列を含みます:

- TableName - フィールドが含まれるテーブルの名前を指定します。
- FieldName - フィールドの名前です。
- FieldType - フィールドの SQL データタイプです。
- FieldId - フィールドの固有の ID です。
- FieldClass - 次の 3 つの値のいずれかが入ります: Summary (集計フィールド)、Calculated (計算結果)、または Normal。
- FieldReps - フィールドの繰り返しの回数です。
- ModCount - このテーブルの定義で確定された変更の合計回数です。

例

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

FileMaker システム列

FileMaker ソフトウェアでは、FileMaker Pro ファイルで定義された全テーブルのすべての行 (レコード) に対してシステム列 (フィールド) を追加します。ODBC アプリケーションの場合、これらの列は SQLSpecialColumns のカタログ関数が返す情報の中に含まれます。JDBC アプリケーションの場合、これらの列は DatabaseMetaData メソッドの getVersionColumns が返す情報の中に含まれます。列は ExecuteSQL 関数でも使用できます。

ROWID 列

`ROWID` システム列にはレコード固有の ID 番号が含まれます。この ID 番号は FileMaker Pro Get (レコード ID) 関数が返す値と同じです。

ROWMODID 列

`ROWMODID` システム列には現在のレコードに対して確定した編集の合計回数が含まれます。この値は FileMaker Pro Get (レコード編集回数) 関数が返す値と同じです。

例

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

予約 SQL キーワード

次の表に、列、テーブル、エイリアス、またはその他のユーザ定義のオブジェクトの名前に使用できない予約キーワードを一覧にします。構文エラーが表示される場合、エラーはこれらの予約語のいずれかを使用したことによる可能性があります。これらのキーワードのいずれかを使用する場合、クォーテーションマークを使用してその語がキーワードとして処理されるのを防ぐ必要があります。

例

`DEC` キーワードをデータ要素名として使用します。

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

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

索引

M

N

O

P

R

S

T