TIBCO FOCUS®

Db2 Web Query 関数ガイド

バージョン2.3.0
January 2022

目次

1

関数の使用

次のトピックでは、関数の概要および使用例について説明します。関数を使用することで、特定の計算や操作を簡単に実行することができます。

このマニュアルで説明している関数のいずれを使用する場合でも、一時項目を作成する必要があります。一時項目については後述します。

トピックス

関数カテゴリ

アクセス可能な関数の種類は次のとおりです。

- **文字列関数**　文字フィールドまたは文字列を操作します。
- **簡略文字列関数**　SQL 関数で使用されるパラメータリストに類似した、簡略化されたパラメータリストが使用されます。
- **データソースおよびデコード関数**　データソース値を取得し、入力フィールドの値に基づいて値を割り当てます。
- **日付時間関数**　日付および時間を操作します。
- **簡略日付時間関数**　SQL 関数で使用されるパラメータリストに類似した、簡略化されたパラメータリストが使用されます。

- **フォーマット変換関数**　フィールドのフォーマットを変換します。
- **数値関数**　数値定数と数値フィールドの計算を実行します。
- **システム関数**　オペレーティングシステムを呼び出して、オペレーティング環境に関する情報を取得します。

文字列関数

次の関数は、文字フィールドまたは文字列を操作します。

ARGLEN
フィールド内の末尾のブランクを除いた文字列の長さを取得します。

BITSON
文字列内の特定のビットを評価し、オンかオフかを返します。

BYTVAL
文字列を ASCII または EBCDIC の 10 進数に変換します。

CHKFMT
文字列内の文字または文字種が正しいかを確認します。

CTRAN
文字列内の文字を 10 進数に基づいて他の文字に変換します。

CTRFLD
文字をフィールド内で中央揃えにします。

EDIT
文字列から文字を抽出、または文字列に文字を追加します。

GETTOK
文字列内の特殊文字 (区切り文字と呼ばれる) に基づいて、文字列をトークンと呼ばれるサブ文字列に分割します。

LCWORD
文字列内の先頭文字を大文字、それ以外を小文字に変換します。

LCWORD2
各単語の先頭の文字を大文字、それ以外をすべて小文字に変換します。

LCWORD3
文字列内の文字を先頭大文字に変換します。この関数で、各単語の先頭文字が大文字に、その他すべての文字が小文字に変換されます。

LJUST
文字をフィールド内で左揃えにします。

LOCASE
文字列を小文字に変換します。

OVRLAY
文字列内のサブ文字列を他のサブ文字列で上書きします。

PARAG
テキスト行を区切り文字で分割します。

POSIT
文字列内のサブ文字列の開始位置を検索します。

PTOA
数値形式を文字形式に変換します。数値の小数点の位置を保持し、先頭にブランクを挿入することにより、数値を右揃えします。PTOA によって変換された数値には、編集オプションを追加することができます。

REVERSE
文字列内の文字を逆にします。

RJUST
文字列を右揃えにします。

SOUNDEX
文字列を、綴りとは無関係に音声で検索します。

SPELLNUM
小数点以下 2 桁の文字また数値をドルとセントの文字表記に書き替えます。

SUBSTR
サブ文字列の開始位置および長さに基づいて、ソース文字列からサブ文字列を抽出します。

UPCASE
文字列を大文字に変換します。

簡略文字列関数

次の文字列関数では、SQL 関数で使用されるパラメータリストに類似した、簡略化されたパラメータリストが使用されます。

CHAR_LENGTH

文字列の長さをバイト数で返します。

DIGITS

数値を文字列に変換します。

LOWER

文字列をすべて小文字で取得します。

LPAD

文字列の左側にブランクを挿入します。

LTRIM

文字列の左端からブランクを削除します。

POSITION

ソース文字列内のサブ文字列の開始位置を取得します。

RTRIM

文字列の右端からブランクを削除します。

RPAD

文字列の右側にブランクを挿入します。

SUBSTRING

ソース文字列からサブ文字列を抽出します。

TOKEN

文字列からトークンを抽出します。

TRIM_

文字列から先頭、末尾、またはこれらの両方の文字を削除します。

UPPER

文字列をすべて大文字で取得します。

データソースおよびデコード関数

次の関数は、データソースの値を取得するか、値を割り当てます。

DB_EXPR

Web Query または SQL 言語のリクエストで生成されるネイティブ SQL に、ネイティブ SQL 式を入力されたとおりに挿入します。

DECODE

コード化された入力フィールドの値に基づいて値を割り当てます。

LAST

フィールドの前の値を取得します。

日付時間関数

次の関数は、日付および時間を操作します。

標準日付時間関数

AYM

年月フォーマットの日付と指定した月数の和または差を計算します。

AYMD

年月日フォーマットの日付と指定した日数の和または差を計算します。

CHGDAT

文字フォーマットの日付の年月日部分を再配置し、長いフォーマットと短いフォーマット間の変換を実行します。

DA

日付を 1899 年 12 月 31 日から数えた経過日数に変換します。

DADMY - 日月年フォーマットに変換します。

DADYM - 日年月フォーマットに変換します。

DAMDY - 月日年フォーマットに変換します。

DAMYD - 月年日フォーマットに変換します。

DAYDM - 年日月フォーマットに変換します。

DAYMD - 年月日フォーマットに変換します。

DATEADD

日付フォーマットに単位を追加、または日付フォーマットから単位を削除します。

DATECVT

日付フォーマットを変換します。

DATEDIF

2 つの日付の差を単位で返します。

DATEMOV

日付を有効な位置に移動します。

DATEPATTERN

データ値を特別な標準に基づかない文字フォーマットとして格納します。年、四半期、月などの構成要素の任意の組み合わせ、および任意の区切り文字を使用することができます。

DATETRAN

日付を国際フォーマットに変換します。

DMY、MDY、YMD

2 つの日付の差を計算します。

DOWK および DOWKL

日付に対応する曜日を見つけます。

DT

1899 年 12 月 31 日から数えた経過日数を日付に変換します。

DTDMY - 数値を日月年の日付に変換します。

DTDYM - 数値を日年月の日付に変換します。

DTMDY - 数値を月日年の日付に変換します。

DTMYD - 数値を月年日の日付に変換します。

DTYDM - 数値を年日月の日付に変換します。

DTYMD - 数値を年月日の日付に変換します。

FIYR

会計年度の開始日および会計年度の算定方式に基づいて、特定のカレンダー日付に対応する会計年度を返します。

FIQTR

会計年度の開始日および会計年度の算定方式に基づいて、特定のカレンダー日付に対応する会計四半期を返します。

FIYYQ

指定したカレンダー日付に対応する会計日付を返します。この日付には、会計年度および会計四半期が含まれます。

GREGDT

ユリウス暦の日付を年月日フォーマットに変換します。

HADD

日付時間フィールドを指定した単位数で増加します。

HCNVRT

日付時間フィールドを文字列に変換します。

HDATE

日付時間フィールドの日付部分を抽出し、日付フォーマットに変換後、結果を YYMD フォーマットで返します。

HDIFF

2 つの日付時間値の単位での日数差を計算します。

HDTTM

日付フィールドを日付時間フィールドに変換します。時間部分は午前零時に設定されます。

HGETC

現在の日付と時間を日付時間フィールドに格納します。

HHMMSS

システムから現在の時間を取得します。

HINPUT

文字列を日付時間値に変換します。

HMIDNT

日付時間フィールドの時間部分を午前零時 (すべてゼロ) に変更します。

HNAME

日付時間フィールドから特定の構成要素を抽出し、文字フォーマットで返します。

HPART

日付時間フィールドから特定の構成要素を抽出し、数値フォーマットで返します。

HSETPT

指定した構成要素の数値を日付時間値に挿入します。

HTIME

日付時間フィールドの時間部分をミリ秒またはマイクロ秒に変換します。

JULDAT

年月日フォーマットの日付をユリウス暦 (年月) フォーマットに変換します。

TIMETOTS

時間をタイムスタンプに変換します。

TODAY

システムから現在の日付を取得します。

YM

日付から日付までの経過月数を計算します。日付は年月フォーマットである必要があります。

簡略日付関数および日付時間関数

次の関数では、SQL 関数で使用されるパラメータリストに類似した、簡略化されたパラメータリストが使用されます。

DTADD

日付または日付時間構成要素に増分値の加算します。

DTDIFF

日付値または日付時間値の構成要素の差分を取得します。

DTPART

日付または日付時間構成要素を整数フォーマットで取得します。

DTRUNC

特定の日付が属する日付範囲の開始日を取得します。

フォーマット変換関数

次の関数は、フィールドのフォーマットを変換します。

ATODBL

文字フォーマットの数値を倍精度フォーマットに変換します。

EDIT

数値を含む文字フィールドを数値フォーマットに変換、または数値フィールドを文字フォーマットに変換します。

FTOA

数値フォーマットの数値を文字フォーマットに変換します。

HEXBYT

ASCII または EBCDIC の 10 進コードに対応する文字を取得します。

ITONUM

データソースの整数を倍精度フォーマットに変換します。

ITOPACK
データソースの整数をパック 10 進数フォーマットに変換します。

ITOZ
数値フォーマットの数値をゾーン 10 進数フォーマットに変換します。

PCKOUT
抽出ファイルに指定した長さでパック 10 進数を書き込みます。

数値関数

次の関数は、数値定数または数値フィールドに対する計算を実行します。

ABS

数値の絶対値を返します。

BAR

縦棒グラフを生成します。

CHKPCK

フィールド内のデータがパック 10 進フォーマットかどうかを確認します。

DMOD、FMOD、IMOD

除算の剰余を計算します。

EXP

値「e」を指数でべき乗します。

INT

数値の整数部分を返します。

LOG

数値の自然対数を返します。

MAX、MIN

値リストから最大値、最小値をそれぞれ取得します。

PRDNOR および PRDUNI

再生可能な乱数を生成します。

RDNORM および RDUNIF

乱数を生成します。

SQRT

数値の平方根を計算します。

システム関数

次の関数は、オペレーティングシステムを呼び出して、オペレーティング環境に関する情報を取得します。

FGETENV

環境変数値を取得し、文字列として返します。

GETUSER

接続ユーザの ID を取得します。

HHMMSS

システムから現在の時間を取得します。

TODAY

システムから現在の日付を取得します。

関数の引数指定

関数に引数を指定する際は、使用可能な引数の種類、フォーマットと長さ、数と順序を理解する必要があります。

引数の種類

関数では、次の引数を使用できます。

- 数値定数 (例、6 や 15 など)。
- 日付定数 (例、022802)。
- 日付 (文字、数値、日付フォーマット)。
- 文字リテラル (例、STEVENS、NEW YORK NY)。リテラルは一重引用符 (') で囲みます。
- 文字フォーマットの数値。
- フィールド名 (例、FIRST_NAME や HIRE_DATE)。フィールドには、データソースフィールドまたは一時項目も使用できます。フィールドは 66 バイト以内のフィールド名または完全修飾名、一意の省略名、あるいはエイリアスです。

- 式 (例、数値、日付、または文字で構成される式)。式には算術演算子と連結記号 (|) が使用できます。有効な式の例は次のとおりです。

  ```
  CURR_SAL * 1.03
  ```

  および

  ```
  FN || LN
  ```

- 一重引用符 (') で囲んだ出力値フォーマット。
- 他の関数。

関数引数の増加

ユーザが記述するサブルーチンでサポートされる引数の数が、28 個から 200 個に増加しています。ユーザが記述するサブルーチンの引数に関するその他の規則に、変更はありません。

引数のフォーマット

引数のフォーマットは関数ごとに異なります。フォーマットには、文字、数値、日付があります。引数のフォーマットに誤りがあると、エラーが発生したり、正しい結果が得られない原因となります。有効なフォーマットを取得するには、ツールの [フォーマット] ボタンをクリックし、有効なタイプおよび長さのリストを表示します。引数フォーマットの種類は次のとおりです。

- **文字引数** 内部的には 1 バイトにつき 1 英数文字として格納されます。文字引数には、リテラル、文字フィールド、文字フォーマットで格納された数値または日付、文字式、文字フィールドのフォーマットがあります。
- **数値引数** 内部的には 2 進数またはパック 10 進数として格納されます。数値引数のフォーマットには、整数 (I)、単精度浮動小数点数 (F)、倍精度浮動小数点数 (D)、パック 10 進数 (P) などがあります。数値引数は、数値定数、フィールド、式、または数値フィールドのフォーマットをとることもできます。数値引数は、関数で使用する際、すべて倍精度浮動小数点数に変換されますが、結果は出力フィールドで指定したフォーマットで返されます。
- **日付引数** 日付引数のフォーマットは、文字、数値、日付のいずれかです。関数で使用可能なフォーマットの種類は、個々の関数の引数リストで指定します。日付引数のフォーマットは、文字、数値、日付のいずれかです。日付フィールド、式、または日付フィールドのフォーマットをとることもできます。引数に 2 桁の年を指定すると、世紀の値は、DATEFNS、YRTHRESH、DEFCENT パラメータ設定に基づいて割り当てられます。

2

文字列関数

文字列関数は、文字フィールドおよび文字列を操作します。

トピックス

ARGLEN - 文字列の長さを取得

ARGLEN 関数は、フィールド内の末尾のブランクを除いた文字列の長さを取得します。マスターファイル内のフィールドフォーマットでは、末尾のブランクを含めた長さを指定します。

構文　文字列の長さを取得

```
ARGLEN(inlength, infield, 'outfield')
```

説明

`inlength`
　整数

文字列を含むフィールドの長さです。長さを含むフィールドを指定することもできます。

`infield`
文字

文字列を含むフィールド名です。

`outfield`
整数

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例　文字列の長さを取得

ARGLEN は LAST_NAME の文字列の長さを取得し、結果を NAME_LEN に格納します。

```
COMPUTE NAME_LEN/I3 = ARGLEN(15, LAST_NAME, 'I3');
```

BITSON - ビットのオンとオフを返す

BITSON 関数は、文字列内の特定のビットを評価し、オンかオフかを返します。BITSON は、ビットがオンの場合は値 1 を返し、オフの場合は値 0 (ゼロ) を返します。この関数は、マルチパンチデータの解釈に役立ちます。ここで、各パンチは情報の 1 項目を表します。

構文　ビットのオンとオフを返す

```
BITSON(bitnumber, string, 'outfield')
```

説明

`bitnumber`
整数

評価するビット数です。ビット数は文字列の最も左にあるビットから数えられます。

`string`
文字

評価する文字列です。文字列は一重引用符 (') で囲みます。文字列を含むフィールドを指定することもできます。文字列は、複数の 8 ビットブロックで構成されます。

`outfield`
整数

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例　フィールド内のビットを評価

BITSON は LAST_NAME の 24 ビット目を評価し、結果を BIT_24 に格納します。

```
COMPUTE BIT_24/I1 = BITSON(24, LAST_NAME, 'I1');
```

BYTVAL - 文字を 10 進数に変換

BYTVAL 関数は、文字列をオペレーティングシステムに対応する ASCII、EBCDIC、または Unicode のいずれかの 10 進数に変換します。

構文　　文字を変換

```
BYTVAL(character, 'outfield')
```

説明

文字列[NODATA もじれつ]
: 文字

  変換される文字です。文字を含むフィールドまたは変数、あるいは一重引用符 (') で囲んだ文字を指定することができます。複数の文字を指定すると、先頭の文字が評価されます。

outfield
: 整数

  出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例　　フィールドの 1 文字目を変換

BYTVAL 関数は、LAST_NAME の 1 文字目を対応する ASCII または EBCDIC の 10 進数に変換し、結果を LAST_INIT_CODE に格納します。入力文字列には複数の文字が含まれているため、BYTVAL 関数は 1 文字目を評価します。

```
COMPUTE LAST_INIT_CODE/I3 = BYTVAL(LAST_NAME, 'I3');
```

CHKFMT - 文字列のフォーマットを確認

CHKFMT 関数は、文字列内の文字または文字種が正しいかを確認します。第 1 文字列を第 2 文字列 (mask) と比較し、第 1 文字列の各文字を mask 内の対応する各文字と比較します。文字列内のすべての文字が mask 内の文字列または文字種と一致する場合、値 0 (ゼロ) を返します。それ以外の場合、CHKFMT は mask と一致しない文字列の先頭文字の位置と同一の値を返します。

mask が文字列よりも短い場合、この関数は mask に対応する部分の文字列のみを確認します。たとえば、4 バイトの mask を使用して 9 バイトの文字列をテストする場合、最初の 4 バイトのみが確認され、それ以外は不一致と見なされ、不一致部分の先頭位置が返されます。

構文　　文字列のフォーマットを確認

```
CHKFMT(numchar, string, 'mask', 'outfield')
```

説明

`numchar`
整数

mask と比較する文字のバイト数です。

`string`
文字

文字列です。文字列は一重引用符 (') で囲みます。文字列を含むフィールドを指定することもできます。

`mask`
文字

比較に使用する mask 文字列です。mask 文字列は一重引用符 (') で囲みます。

mask 内には、文字種のみを表す汎用文字を含めることができます。文字列内の文字の 1 つがこれらの文字列の 1 つと比較され、文字種が同一である場合、文字は一致したと見なされます。汎用文字は次のとおりです。

`A` - A から Z (大文字または小文字)

`9` - 0 から 9 の任意の数字

`X` - A から Z の文字または 0 から 9 の数字

`$` - 任意の文字

mask 内のその他の文字は、その文字自体を表します。たとえば、mask 内の 3 つ目の文字が「B」の場合、文字列内の 3 つ目の文字が一致するためには「B」である必要があります。

`outfield`
整数

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例　フィールドのフォーマットを変換

CHKFMT 関数は、「11」で始まる 9 バイトの EMP_ID を検証し、結果を CHK_ID に格納します。

```
COMPUTE CHK_ID/I3 = CHKFMT(9, EMP_ID, '119999999', 'I3');
```

CTRAN - 文字を他の文字に変換

CTRAN 関数は、文字列内の文字を 10 進数に基づいて他の文字に変換します。この関数は、利用できない文字、入力が難しい文字、またはキーボードにない文字を置換文字列によって入力するときに役立ちます。

CTRAN を使用するには、対応する内部マシンの 10 進表現の知識が必要です。

Unicode 構成の場合、この関数は次の範囲の値を使用します。

- 1 バイト文字 - 0 (ゼロ) から 255
- 2 バイト文字 - 256 から 65535
- 3 バイト文字 - 65536 から 16777215
- 4 バイト文字 - 16777216 から 4294967295 (主として EBCDIC 用)

構文　文字を他の文字に変換

```
CTRAN(charlen, string, decimal, decvalue, 'outfield')
```

説明

`charlen`
　整数

　文字列のバイト数です。長さを含むフィールドを指定することもできます。

`string`
　文字

　変換される文字列です。文字列は一重引用符 (') で囲みます。文字列を含むフィールドを指定することもできます。

`decimal`
　整数

　変換する文字の ASCII たは EBCDIC の 10 進コードです。

`decvalue`
　整数

decimal の代替文字として使用する文字の ASCII または EBCDIC の 10 進コードです。

`outfield`
文字

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例 ブランクをアンダースコア (_) に変換 (EBCDIC プラットフォーム)

CTRAN 関数は、ADDRESS_LN3 (EBCDIC 10 進数は 64) のブランクをアンダースコア (EBCDIC 10 進数は 109) に変換し、ALT_ADDR に格納します。

```
COMPUTE ALT_ADDR/A20 = CTRAN(20, ADDRESS_LN3, 64, 109, 'A20');
```

CTRFLD - 文字列を中央揃え

CTRFLD 関数は、文字列をフィールド内で中央揃えにします。先頭のブランクの数は末尾のブランクの数と同一またはそれよりも 1 つ少ない数になります。

フィールドの内容または埋め込みフィールドのみで構成される見出しを中央揃えにするときに役立ちます。HEADING CENTER は、各フィールド値を末尾のブランクを含めて中央揃えにします。末尾のブランクを含めずに中央揃えにするには、CTRFLD を使用します。

制限：スタイルシートを使用するレポートで CTRFLD を使用する場合、対象項目を中央揃え要素としてスタイル設定しない限り、CTRFLD は無効になります。また、デフォルトフォントがプロポーショナルであるプラットフォームで CTRFLD 関数を使用する場合、固定フォントを使用するか、リクエストの実行前に SET STYLE=OFF を発行します。

構文 文字列を中央揃え

```
CTRFLD(string, length, 'outfield')
```

説明

`string`
文字

文字列です。文字列は一重引用符 (') で囲みます。文字列を含むフィールドを指定することもできます。

`length`
整数

string および *outfield* のバイト数です。長さを含むフィールドを指定することもできます。この引数には正の数を指定します。length に負の数を指定すると、予期しない結果の原因となります。

outfield
文字

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例 フィールドを中央揃え

CTRFLD 関数は、LAST_NAME を中央揃えにし、結果を CENTER_NAME に格納します。

```
COMPUTE CENTER_NAME/A12 = CTRFLD(LAST_NAME, 12, 'A12');
```

EDIT - 文字を抽出または追加

EDIT 関数は、文字列から文字を抽出、または文字列に文字を追加します。文字列の任意の部分から文字列を抽出することができます。また、文字列から抽出した文字列を他の文字列に挿入することもできます。たとえば、文字列の先頭の 2 バイトと末尾の 2 バイトを抽出し、単一文字列を生成することができます。

EDIT 関数は、mask 内の文字とソースフィールド内の文字を比較します。mask 内で「9」が見つかると、EDIT はソース文字列の対応する文字を新しいフィールドにコピーします。mask 内にドル記号 ($) が見つかると、EDIT はソースフィールド内の対応する文字を無視します。mask 内でその他の文字が見つかると、EDIT は新しいフィールドの対応する位置に、その文字をコピーします。EDIT 関数では、結果は文字に限定され、サイズは mask 引数の値が決定するため、outfield 引数を指定する必要はありません。

EDIT 関数は、フィールドのフォーマットを変換することもできます。

構文 文字列を抽出または追加

```
EDIT(fieldname, 'mask');
```

説明

fieldname
文字

ソースフィールドです。

文字列の抽出元の文字列です。mask と同じか、mask よりも長い文字列を指定します。

mask
文字

文字列です。文字列は一重引用符 (') で囲みます。mask の長さ (9 と $ を除く) により出力フィールドの長さが決定されます。

例　EDIT マスク文字を使用したフィールドへの文字の抽出と追加

EDIT 関数は FIRST_NAME フィールドの 1 文字目を抽出し、FIRST_INIT に格納します。また、EMP_ID フィールドにハイフン (-) を追加し、結果を EMPIDEDIT に格納します。

```
COMPUTE FIRST_INIT/A1 = EDIT(FIRST_NAME, '9$$$$$$$$$$'); AND
COMPUTE EMPIDEDIT/A11 = EDIT(EMP_ID, '999-99-9999');
```

GETTOK - サブ文字列 (トークン) を抽出

GETTOK 関数は、ソース文字列を解析し、「トークン」と呼ばれる文字列を検索します。「区切り文字」と呼ばれる文字列内の特定の文字により、文字列をトークンに分割します。GETTOK 関数は、token_number により指定されたトークンを返します。文字列内の先頭と末尾のブランクは無視されます。

たとえば、文字列の 4 つ目の単語を抽出することを想定します。区切り文字にブランクを使用します。token_number には 4 を指定します。GETTOK は、この区切り文字により文を単語に分割し、4 つ目の単語を抽出します。文字列が区切り文字で分割されていない場合は、PARAG を使用します。

構文　文字列 (トークン) を抽出

```
GETTOK(infield, inlen, token_number, 'delim', outlen, 'outfield')
```

説明

`infield`
　文字

　元の文字列を含むフィールドです。

`inlen`
　整数

　ソース文字列の長さをバイト数で指定します。この引数が 0 (ゼロ) 以下の場合、ブランクが返されます。

`token_number`
　整数

　抽出するトークンの番号です。この引数が正の数の場合、トークンは左から右へ数えられます。この引数が負の数の場合、トークンは右から左へ数えられます。たとえば、-2 の場合、右から 2 つ目のトークンが抽出されます。この引数が 0 (ゼロ) の場合、ブランクが返されます。先頭と末尾のブランクのトークンは無視されます。

delim
文字

一重引用符 (') で囲まれた元の文字列の区切り文字です。複数の文字を指定した場合、先頭の文字のみが使用されます。

outlen
整数

抽出するトークンのサイズです。この引数が 0 (ゼロ) 以下の場合、ブランクが返されます。この引数よりも長い場合、トークンは切り捨てられます。短い場合には、トークンの末尾にブランクが追加されます。

outfield
文字

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。抽出されたトークンに、区切り文字は含まれません。

例　フィールドからトークンを抽出

GETTOK 関数は、ADDRESS_LN3 から最後のトークンを抽出し、LAST_TOKEN に格納します。

```
COMPUTE LAST_TOKEN/A10 = GETTOK(ADDRESS_LN3, 20, -1, ' ', 10, 'A10');
```

LCWORD - 文字列を先頭大文字に変換

LCWORD 関数は、文字列内の先頭文字を大文字、それ以外を小文字に変換します。単語の先頭文字、および一重引用符 (') と二重引用符 (") の後の先頭文字を除き、すべてのアルファベットを小文字に変換します。たとえば、「O'CONNOR」は「O'Connor」に変換され、「JACK'S」は「Jack'S」に変換されます。

LCWORD 関数は、文字列内の数字を大文字として処理するため、その後に続く文字は小文字に変換されます。その結果、先頭文字が大文字で残りの文字が小文字の文字列が生成されます。

構文　文字列を先頭大文字に変換

```
LCWORD(length, string, 'outfield')
```

説明

length
整数

変換される文字列またはフィールドの長さです。長さを含むフィールドを指定することもできます。

```
string
```
文字

変換される文字列です。文字列は一重引用符 (') で囲みます。文字列を含むフィールドを指定することもできます。

```
outfield
```
文字

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。このフィールドの長さは、length フィールドの値以上である必要があります。

例　文字列を先頭大文字に変換

LCWORD 関数は、LAST_NAME フィールド内の先頭文字を大文字、それ以外を小文字に変換し、MIXED_CASE に格納します。

```
COMPUTE MIXED_CASE/A15 = LCWORD(15, LAST_NAME, 'A15');
```

LCWORD2 - 文字列を先頭大文字に変換

LCWORD2 関数は、文字列の先頭の文字以外をすべて小文字に変換します。LCWORD2 では、単一の一重引用符 (') が検出された場合、その次の文字は、小文字に変換されます。たとえば、「SMITH」は「Smith」に、「JACK'S」は「Jack's」に、それぞれ変換されます。

構文　文字列を先頭大文字に変換

```
LCWORD2(length, string, 'outfield')
```

説明

```
length
```
整数

変換される文字列またはフィールドの長さです。長さを含むフィールドを指定することもできます。

```
string
```
文字

変換する文字列です。文字列を含む一時項目を指定することもできます。

```
outfield
```
文字

結果を格納するフィールド名、または出力フォーマットです。フォーマットは一重引用符 (') で囲みます。このフィールドの長さは、length フィールドの値以上である必要があります。

例　文字列を先頭大文字に変換

レポート開発ツールのダイアログボックスまたは Developer Workbench のシノニムエディタを使用して、LCWORD2 により「O'CONNOR's」を先頭大文字に変換します。

```
MYVAL1/A10='O'CONNOR'S';
LC2/A10 = LCWORD2(10, MYVAL1, 'A10');
```

これらのフィールドのレポート出力は、次のとおりです。

```
MYVAL1       LC2
------       ---
O'CONNOR'S   O'Connor's
```

LCWORD3 - 文字列を先頭大文字に変換

LCWORD3 関数は、文字列内の文字を先頭大文字に変換します。この関数で、各文字列の先頭文字が大文字に、その他すべての文字が小文字に変換されます。また、一重引用符 (') の後にブランクが続く場合および入力文字列の最後の文字が一重引用符の場合を除いて、一重引用符に続く文字が大文字に変換されます。

たとえば、「SMITH」は「Smith」に、「JACK'S」は「Jack's」に、それぞれ変換されます。

構文　LCWORD3 - 文字列を先頭大文字に変換

```
LCWORD3(length, string, output)
```

説明

`length`

整数

変換する文字列の長さです。長さが定義されたフィールドを指定することもできます。

`source_string`

文字

変換する文字列、または文字列を含むフィールドです。

```
output
```

文字

結果を格納するフィールド名、または出力フォーマットです。フォーマットは一重引用符 (') で囲みます。このフィールドの長さは、length フィールドの値以上である必要があります。

例　LCWORD3 を使用して文字列を先頭大文字に変換

レポート開発ツールの COMPUTE または DEFINE ダイアログボックスまたは Developer Workbench のシノニムエディタを使用して、LCWORD3 により「O'CONNOR's」を「O'CONNOR's」、「o'connor's」を先頭大文字に変換します。

```
MYVAL1/A10='O'CONNOR'S';
MYVAL2/A10='o'connor's';
LC1/A10 = LCWORD3(10, MYVAL1, 'A10');
LC2/A10 = LCWORD3(10, MYVAL2, 'A10');
```

出力結果では、最初の一重引用符 (') に続く文字「C」は大文字で出力されています。これは、この文字の後の文字がブランクでもなく、入力文字列の最後の文字でもないためです。2 つ目の一重引用符 (') に続く文字「s」は小文字で出力されています。これは、この文字が入力文字列の最後の文字であるためです。

```
MYVAL1      LC1          MYVAL2      LC2
------      ---          ------      ---
O'CONNOR'S  O'Connor's   o'connor's  O'Connor's
```

LJUST - 文字列を左揃え

LJUST 関数は、文字列をフィールド内で左揃えにします。左側のブランクはすべて右側に移動します。

スタイルシート (SET STYLE=ON) を使用するレポートでは、項目を中央揃えにしない限り、LJUST 関数の視覚効果は有効になりません。

構文　文字列を左揃え

```
LJUST(length, string, 'outfield')
```

説明

```
length
```

整数

string および *outfield* の長さです。長さを含むフィールドを指定することもできます。

`string`
文字

左揃えされる文字列です。文字列を含むフィールドを指定することもできます。

`outfield`
文字

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例　フィールドを左揃え

LJUST 関数は XNAME フィールドを左揃えに変更し、結果を YNAME に格納します。

```
COMPUTE YNAME/A25 = LJUST(15, XNAME, 'A25');
```

LOCASE - テキストを小文字に変換

LOCASE 関数は、テキストを小文字に変換します。

構文　テキストを小文字に変換

```
LOCASE(length, string, 'outfield')
```

説明

`length`
整数

string および *outfield* の長さです。長さを含むフィールドを指定することもできます。 長さは 1 以上で、string と outfield に同一の値が指定されている必要があります。それ以外の値が指定されている場合はエラーが発生します。

`string`
文字

変換される文字列です。文字列は一重引用符 (') で囲みます。文字列を含むフィールドを指定することもできます。

`outfield`
文字

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。フィールド名には *string* と同一の値を指定することもできます。

例　フィールドを小文字に変換

LOCASE 関数は、LAST_NAME フィールドを小文字に変換し、結果を LOWER_NAME に格納します。

```
COMPUTE LOWER_NAME/A15 = LOCASE(15, LAST_NAME, 'A15');
```

OVRLAY - 文字列を上書き

OVRLAY 関数は、文字列内のサブ文字列を別のサブ文字列で上書きします。

構文　文字列を上書き

```
OVRLAY(string1, stringlen, string2, sublen, position, 'outfield')
```

説明

`string1`
文字

ソース文字列です。

`length`
整数

string1 および *outfield* の長さです。長さを含むフィールドを指定することもできます。引数に 0 (ゼロ) 以下を指定すると、予期しない結果が発生します。

`string2`
文字

string1 を上書きする文字列です。

`sublen`
整数

string2 の長さです。長さを含むフィールドを指定することもできます。この引数が 0 (ゼロ) 以下の場合、ブランクが返されます。

`position`
整数

元の文字列内で上書きを開始する位置です。この引数が 0 (ゼロ) 以下の場合、ブランクが返されます。この引数が *stringlen* よりも大きい場合、この関数は元の文字列を返します。

`outfield`
文字

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。上書き文字列が出力フィールドよりも長い場合、文字列はフィールドの長さに切り捨てられます。

例　文字列を上書き

OVRLAY 関数は、EMP_ID の末尾の 3 バイトを CURR_JOBCODE の値で置換して新しいセキュリティ ID を作成し、結果を NEW_ID に格納します。

```
COMPUTE NEW_ID/A9 = OVRLAY(EMP_ID, 9, CURR_JOBCODE, 3, 7, 'A9');
```

PARAG - テキストを行に分割

PARAG 関数は、区切り文字を使用して 1 行の文字列を複数の文字列に分割します。行の開始位置から特定のバイト数をスキャンし、その文字列グループ末尾のブランクを区切り文字で置換します。その後、区切り文字から開始して、行の次の文字列グループをスキャンし、このグループ末尾のブランクを 2 つ目の区切り文字と置換することにより、2 つ目のトークンを作成します。行の末尾に到達するまで、この処理を繰り返します。

区切り文字で分割された各文字列グループは個別の行になります。その後、GETTOK 関数がこれらの行を異なるフィールドに指定します。PARAG 関数は、スキャンしたグループ内にブランクが見つからない場合、グループの先頭文字を区切り文字で置換します。このため、すべての文字列がスキャンされたバイト数以下であることが重要です。

入力された行の長さがほぼ同じである場合、均等に表示されるように分割行の長さを指定することができます。たとえば、文字列が 120 バイトの場合、各行を 60 バイト (2 行) または 40 バイト (3 行) に分割することができます。この手法を使用すると、テキスト行を段落形式で表示することが可能です。

ただし、行を等しいサイズに分割する場合、意図する数よりも多くの行が作成される可能性があります。たとえば、120 バイトのテキスト行を最大 60 バイトの 2 行に分割しようとして、1 つ目の行が 50 バイト、2 つ目の行が 55 バイトに分割されたことを想定します。分割後の 3 つ目の行は 15 バイトになります。これを修正するには、3 つ目の行の先頭にブランクを挿入 (弱連結を使用) し、この行を 2 つ目の行末に追加 (強連結を使用) します。

構文　テキストを行に分割

```
PARAG(length, string, 'delim', subsize, 'outfield')
```

説明

`length`

整数

string および *outfield* の長さです。長さを含むフィールドを指定することもできます。

`string`
文字

文字列です。文字列は一重引用符 (') で囲みます。文字列を含むフィールドを指定することもできます。

`delim`
文字

一重引用符 (') で囲まれた区切り文字です。テキストで使用されていない文字を選択します。

`subsize`
整数

各行の長さの最大値です。

`outfield`
文字

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例　テキストを行に分割

PARAG 関数は、カンマ (,) を区切り文字として使用し、ADDRESS_LN2 を 10 バイト以下の行に分割します。その結果を PARA_ADDR に格納します。

```
COMPUTE PARA_ADDR/A20 = PARAG(20, ADDRESS_LN2, ',', 10, 'A20');
```

POSIT - サブ文字列の開始位置を検索

POSIT 関数は、文字列の中から特定の文字列の開始位置を見つけます。たとえば、文字列 PRODUCTION 内の文字列 DUCT の開始位置は 4 です。指定した文字列が元の文字列内に存在しない場合、この関数は値 0 (ゼロ) を返します。

構文　サブ文字列の開始位置を検索

```
POSIT(parent, inlength, substring, sublength, 'outfield')
```

説明

`parent`
文字

文字列です。文字列は一重引用符 (') で囲みます。文字列を含むフィールドを指定することもできます。

`inlength`
整数

元の文字列のバイト数です。長さを含むフィールドを指定することもできます。この引数が 0 (ゼロ) 以下の場合、0 (ゼロ) が返されます。

`substring`
文字

位置を検索する文字列です。 一重引用符 (') で囲まれた文字列またはその文字列を含むフィールドを指定することもできます。

`sublength`
整数

substring の長さです。引数が 0 (ゼロ) 以下の場合、または *inlength* よりも大きいときは、0 (ゼロ) が返されます。

`outfield`
整数

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例　文字列の開始位置を検索

POSIT 関数は、LAST_NAME の 1 つ目の大文字「I」の位置を特定し、結果を I_IN_NAME に格納します。

```
COMPUTE I_IN_NAME/I2 = POSIT(LAST_NAME, 15, 'I', 1, 'I2');
```

PTOA - パック 10 進数を文字に変換

PTOA 関数は、数値を数値フォーマットから文字フォーマットに変換します。数値の小数点の位置を保持し、先頭にブランクを挿入することにより、数値を右揃えします。PTOA によって変換された数値には、編集オプションを追加することができます。

PTOA を使用して 10 進数を含む数値を文字列に変換するときは、数値の整数部分と小数点以下を格納するために十分な大きさの文字フォーマットを指定する必要があります。たとえば、P12.2C フォーマットは A14 に変換されます。出力フォーマットの長さが不十分な場合、右端の文字が切り捨てられます。

参照　パック 10 進数を文字に変換

```
PTOA(number, '(format)', output)
```

説明

`number`
数値 P (パック 10 進数)、F または D (単精度または倍精度浮動小数点数)
変換される数値です。

`format`
文字
括弧で囲まれた数値のフォーマットです。

`output`
文字

この引数の長さは数値の長さよりも大きくする必要があります。編集オプションおよび負の符号が追加される可能性も考慮します。次の例では、パック 10 進数から文字フォーマットへの変換により、PGROSS がパック 10 進数から文字フォーマットに変換されます。

```
PTOA(PGROSS, '(P12.2)', 'A14');
```

REVERSE - 文字列の順序を入れ替え

REVERSE 関数は、文字列内の文字の順序を逆にします。

構文　文字列の順序を入れ替え

```
REVERSE(length, string, 'outfield')
```

説明

`length`
整数

string および *outfield* の長さです。長さを含むフィールドを指定することもできます。

`string`
文字

文字列です。文字列は一重引用符 (') で囲みます。文字列を含むフィールドを指定することもできます。

`outfield`
文字

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例　文字列の順序を入れ替え

REVERSE 関数は、PRODCAT 内の文字列の順序を入れ替え、結果を REVERSE_NAME に格納します。

```
COMPUTE REVERSE_NAME/A15 = REVERSE(15, PRODCAT, 'A15');
```

RJUST - 文字列を右揃え

RJUST 関数は、文字列を右揃えにします。末尾のブランクは、すべて先頭のブランクになります。数値を含む文字フィールドを表示するときに役立ちます。

スタイルシート (SET STYLE=ON) を使用するレポートでは、項目を中央揃えにしない限り、RJUST 関数の視覚効果は有効になりません。また、スタイルシートがデフォルト設定でオンに設定されているプラットフォームで RJUST 関数を使用する場合は、リクエストの実行前に SET STYLE=OFF を発行します。

構文　文字列を右揃え

```
RJUST(length, string, 'outfield')
```

説明

`length`
整数

string および *outfield* の長さです。長さを含むフィールドを指定することもできます。 文字揃えの問題を回避するため、必ず同一の長さを指定します。

`string`
文字

文字列です。一重引用符 (') で囲まれた文字列を含むフィールドを指定することもできます。

`outfield`
文字

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例　フィールドを右揃え

RJUST 関数は、LAST_NAME フィールドを右揃えに変更し、結果を RIGHT_NAME に格納します。

```
COMPUTE RIGHT_NAME/A15 = RJUST(15, LAST_NAME, 'A15');
```

SOUNDEX - 文字列を音声的に比較

SOUNDEX 関数は、綴りに関わらず、文字列を音声的に解析します。文字列を 4 バイトのコードに変換します。先頭の文字は、常に文字列内の先頭文字です。残りの 3 バイトは、文字列内で音声的に有意な次の 3 バイトを表します。

音声検索を実行するには、次の手順を実行します。

1. SOUNDEX 関数を使用して、検索するフィールドのデータ値を音声コードに変換します。
2. SOUNDEX 関数を使用して、最も妥当と想定されるターゲット文字列を音声コードに変換します。ターゲット文字列の綴りは正確である必要はありませんが、先頭文字は一致する必要があります。
3. WHERE 条件または IF 条件を使用して、手順 1 で作成した一時項目と手順 2 で作成した一時項目を比較します。

構文　文字列を音声的に比較

```
SOUNDEX(inlength, string, 'outfield')
```

説明

`inlength`
文字 (2 バイト)

string のバイト数です。長さを含むフィールドを指定することもできます。一重引用符 (') で囲まれた数値、または数値を含むフィールドを指定することもできます。01 から 99 の 2 桁の数値 (例、'01') を指定する必要があります。99 よりも大きい数値を指定すると、出力結果としてアスタリスク (*) が返されます。

`string`
文字

文字列です。文字列は一重引用符 (') で囲みます。文字列を含むフィールドを指定することもできます。

`outfield`
文字

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例　文字列を音声的に比較

以下のリクエストは次の 3 つのフィールドを作成します。

- PHON_NAME には、従業員の LAST_NAME の音声コードが含まれます。

- PHON_COY には、想定した音声コード「MICOY」が含まれます。
- PHON_MATCH は、想定した音声コードを実際の音声コードと比較します。

```
COMPUTE PHON_NAME/A4 = SOUNDEX('15', LAST_NAME, 'A4'); AND
COMPUTE PHON_COY/A4 WITH LAST_NAME = SOUNDEX('15', 'MICOY', 'A4'); AND
COMPUTE PHON_MATCH/A3 = IF PHON_NAME IS PHON_COY THEN 'YES' ELSE 'NO';
```

SPELLNM - ドルとセントの通貨表記を文字表記に書き替え

SPELLNM 関数は、2 桁の英数文字列または小数点以下 2 桁の数値をドルとセントの文字表記に変換します。たとえば、「32.50」という値は「THIRTY TWO DOLLARS AND FIFTY CENTS」に変換されます。

構文　ドルとセントの通貨表記を文字表記に書き替え

```
SPELLNM(outlength, number, 'outfield')
```

説明

`outlength`
整数

outfield の長さです。長さを含むフィールドを指定することもできます。

数値の最大値が分かっている場合、下表に従い *outlength* の値を決定します。

最大値 (より小さい)	outlength の値
$10	37
$100	45
$1,000	59
$10,000	74
$100,000	82
$1,000,000	96

`number`
文字または 10 進数の数値 (9.2)

文字表記に書き換える数値です。

`outfield`
文字

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例　ドルとセントの通貨表記を文字表記に書き替え

SPELLNM 関数は CURR_SAL の値の綴りを書き出し、結果を AMT_IN_WORDS に格納します。

```
COMPUTE AMT_IN_WORDS/A82 = SPELLNM(82, CURR_SAL, 'A82');
```

SUBSTR - サブ文字列を抽出

SUBSTR 関数は、文字列内の開始位置および長さに基づいて、サブ文字列を抽出します。SUBSTR 関数は、他のフィールドの値に基づいてサブ文字列の位置を変更することができます。

構文　サブ文字列を抽出

```
SUBSTR(inlength, parent, start, end, sublength, 'outfield')
```

説明

`inlength`
整数

元の文字列の長さです。長さを含むフィールドを指定することもできます。

`parent`
文字

一重引用符 (') で囲まれた文字列です。文字列を含むフィールドを指定することもできます。

`start`
整数

parent 内の抽出文字列の開始位置です。この引数が 0 (ゼロ) 以下の場合、ブランクが返されます。

`end`
整数

抽出する文字列の終了位置です。この引数が *start* より小さい、または *inlength* より大きい場合、この関数はブランクを返します。

`sublength`
整数

文字列の長さです。この値は通常、end - start + 1 (end と start の値の差に 1 を加えたもの) です。*sublength* が *end - start* +1 よりも長い場合、抽出文字列の末尾にブランクが追加されます。短い場合は、抽出文字列は切り捨てられます。この値は *outfield* で宣言した長さと同一にします。sublength の長さ分の文字のみが処理されます。

`outfield`
文字

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例　文字列を抽出

POSIT 関数は、LAST_NAME の 1 つ目の「I」の位置を特定し、結果を I_IN_NAME に格納します。その後、SUBSTR 関数は、LAST_NAME から「I」で開始する 3 バイトを抽出し、結果を I_SUBSTR に格納します。

```
COMPUTE I_IN_NAME/I2 = POSIT(LAST_NAME, 15, 'I', 1, 'I2'); AND
COMPUTE I_SUBSTR/A3 = SUBSTR(15, LAST_NAME, I_IN_NAME, I_IN_NAME+2, 3,
'A3');
```

UPCASE - テキストを大文字に変換

UPCASE 関数は、文字列を大文字に変換します。大文字と小文字が混在する値、および大文字のみの値が混在するフィールドをソートする際に役立ちます。大文字と小文字が混在するフィールドをソートすると、誤った結果が発生します。これは、EBCDIC によるソートは常に大文字の前に小文字を配置し、ASCII によるソートは常に小文字の前に大文字を配置するためです。正しい結果を取得するには、値すべてを大文字にした新しいフィールドを定義し、そのフィールドをソートします。

構文　テキストを大文字に変換

```
UPCASE(length, input, 'outfield')
```

説明

`length`
整数

input および *outfield* のバイト数です。

`input`
文字

文字列です。文字列は一重引用符 (') で囲みます。文字列を含むフィールドを指定することもできます。

outfield
文字

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例 大文字と小文字が混在するフィールドを大文字に変換

UPCASE 関数により LAST_NAME_MIXED フィールドが大文字に変換され、結果が LAST_NAME_UPPER に格納されます。

```
COMPUTE LAST_NAME_UPPER/A15 = UPCASE(15, LAST_NAME_MIXED, 'A15') ;
```

3

簡略文字列関数

簡略文字列関数では、SQL 関数で使用されるパラメータリストに類似した、簡略化されたパラメータリストが使用されます。ただし、これらの簡略関数の機能は、以前のバージョンの同様の関数と若干異なる場合があります。

簡略関数には、出力引数はありません。各関数は、特定のデータタイプを持つ値を返します。

これらの関数をリレーショナルデータソースに対するリクエストで使用すると、関数が最適化された上で、RDBMS に渡されて処理されます。

トピックス

CHAR_LENGTH - 文字列の長さ (文字数) の取得

CHAR_LENGTH 関数は、文字列の長さを文字数で返します。Unicode 環境では、この関数はキャラクタセマンティクスを使用するため、文字数で表す長さとバイト数で表す長さが一致しない場合があります。文字列の末尾にブランクが含まれている場合、これらのブランク数も計算された上で文字数が返されます。そのため、ソース文字列のタイプが An の場合、返される値は常に n になります。

構文　文字列の長さ (文字数) の取得

```
CHAR_LENGTH(string)
```

説明

`string`
　文字

　長さを取得する文字列です。

長さの値は、整数データタイプで返されます。

例　文字列の長さの取得

次のリクエストは、EMPLOYEE データソースを使用して、データタイプが A15V の LASTNAME という一時項目 (DEFINE) を作成し、LAST_NAME 値から末尾のブランクを除外した値を格納します。次に、CHAR_LENGTH 関数を使用して、文字数を返します。

```
DEFINE FILE EMPLOYEE
LASTNAME/A15V = RTRIM(LAST_NAME);
END
TABLE FILE EMPLOYEE
SUM LAST_NAME NOPRINT AND COMPUTE
NAME_LEN/I3 = CHAR_LENGTH(LASTNAME);
BY LAST_NAME
ON TABLE SET PAGE NOPAGE
END
```

出力結果は次のとおりです。

```
 LAST_NAME        NAME_LEN
---------        --------
BANNING                 7
BLACKWOOD               9
CROSS                   5
GREENSPAN               9
IRVING                  6
JONES                   5
MCCOY                   5
MCKNIGHT                8
ROMANS                  6
SMITH                   5
STEVENS                 7
```

DIGITS - 数値を文字列に変換

DIGITS 関数は、指定された数値を特定の長さの文字列に変換します。数値が格納されているフィールドは、整数フォーマットである必要があります。

構文　数値を文字列に変換

```
DIGITS(number,length)
```

説明

`number`
整数

整数データタイプのフィールドに格納された変換元の数値です。

`length`
1 から 10 までの整数

返される文字列の長さです。length で指定した長さが、変換する数値の桁数より大きい場合、返される値の左側に 0 (ゼロ) がパディングされます。length で指定した長さが、変換する数値の桁数より小さい場合、返される値の左側が切り取られます。

例　数値を文字列に変換

次のリクエストは、WF_RETAIL_LITE データソースを使用し、-123.45 および ID_PRODUCT を文字列に変換します。

```
DEFINE FILE WF_RETAIL_LITE
MEAS1/I8=-123.45;
DIG1/A6=DIGITS(MEAS1,6) ;
DIG2/A6=DIGITS(ID_PRODUCT,6) ;
END
TABLE FILE WF_RETAIL_LITE
PRINT MEAS1 DIG1
ID_PRODUCT DIG2
BY PRODUCT_SUBCATEG
WHERE PRODUCT_SUBCATEG EQ 'Flat Panel TV'
ON TABLE SET PAGE NOPAGE
END
```

出力結果は次のとおりです。

Product Subcategory	MEAS1	DIG1	ID Product	DIG2
Flat Panel TV	-123	000123	4012	004012
	-123	000123	4017	004017
	-123	000123	4018	004018
	-123	000123	4017	004017
	-123	000123	4017	004017
	-123	000123	4018	004018
	-123	000123	4018	004018
	-123	000123	4017	004017
	-123	000123	4014	004014
	-123	000123	4016	004016
	-123	000123	4016	004016
	-123	000123	4018	004018
	-123	000123	4017	004017
	-123	000123	4018	004018
	-123	000123	4018	004018
	-123	000123	4017	004017
	-123	000123	4016	004016
	-123	000123	4018	004018
	-123	000123	4016	004016
	-123	000123	4018	004018
	-123	000123	4017	004017
	-123	000123	4018	004018
	-123	000123	4017	004017
	-123	000123	4017	004017
	-123	000123	4014	004014
	-123	000123	4018	004018

参照　DIGITS 使用上の注意

- 整数 (I) フォーマットの数値のみが変換されます。 倍精度浮動小数点数 (D)、単精度浮動小数点数 (F)、パック 10 進数 (P) フォーマットの場合、エラーメッセージが生成されます。これらのフォーマットは、DIGITS 関数を使用する前に、整数 (I) フォーマットに変換する必要があります。変換可能な数値の最大値は 2 ギガバイトです。
- 負の整数は正の整数に変換されます。
- 整数フォーマットで小数点以下の桁数が存在する場合、小数部が切り取られます。
- ダイアログマネージャでは DIGITS はサポートされません。

LOWER - 文字列をすべて小文字で取得

LOWER 関数は、ソース文字列のすべての文字を小文字に変換し、変換後の文字列を同一のデータタイプで返します。

構文　文字列をすべて小文字で取得

```
LOWER(string)
```

説明

`string`

文字

小文字に変換する文字列です。

文字列は、ソース文字列と同一のデータタイプと長さで返されます。

例　文字列を小文字に変換

次のリクエストでは EMPLOYEE データソースが使用され、LOWER 関数が、LAST_NAME フィールドを小文字に変換し、結果を LOWER_NAME フィールドに格納します。

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
LOWER_NAME/A15 = LOWER(LAST_NAME);
ON TABLE SET PAGE NOPAGE
END
```

出力結果は次のとおりです。

```
LAST_NAME       LOWER_NAME
---------       ----------
STEVENS         stevens
SMITH           smith
JONES           jones
SMITH           smith
BANNING         banning
IRVING          irving
ROMANS          romans
MCCOY           mccoy
BLACKWOOD       blackwood
MCKNIGHT        mcknight
GREENSPAN       greenspan
CROSS           cross
```

LPAD - 文字列の左パディング

LPAD 関数は、指定された文字および出力長に基づいて、その文字が左側にパディングされた文字列を返します。

構文　　文字列の左パディング

```
LPAD(string, out_length, pad_character)
```

説明

`source_string`
固定長の文字

左パディングを追加する文字列です。

`out_length`
整数

パディング追加後の出力文字列の長さです。

`pad_character`
固定長の文字

パディングに使用する単一文字です。

例　文字列の左パディング

次のリクエストでは WF_RETAIL データソースが使用され、LPAD 関数が、PRODUCT_CATEGORY フィールドの左側にパディング文字「@」を追加します。

```
DEFINE FILE WF_RETAIL
LPAD1/A25 = LPAD(PRODUCT_CATEGORY,25,'@');
DIG1/A4 = DIGITS(ID_PRODUCT,4);
END
TABLE FILE WF_RETAIL
SUM DIG1 LPAD1
BY PRODUCT_CATEGORY
ON TABLE SET PAGE NOPAGE
ON TABLE SET STYLE *
TYPE=DATA,FONT=COURIER,SIZE=11,COLOR=BLUE,$
END
```

出力結果は次のとおりです。

Product Category	DIG1	LPAD1
Accessories	5005	@@@@@@@@@@@@@@@Accessories
Camcorder	3006	@@@@@@@@@@@@@@@@@@Camcorder
Computers	6016	@@@@@@@@@@@@@@@@@@Computers
Media Player	1003	@@@@@@@@@@@@@@@Media Player
Stereo Systems	2155	@@@@@@@@@@@@Stereo Systems
Televisions	4018	@@@@@@@@@@@@@@@@Televisions
Video Production	7005	@@@@@@@@@@Video Production

参照　LPAD 使用上の注意

- 一重引用符 (') をパディング文字として使用するには、2 つの一重引用符 (') を 2 つの一重引用符 (') で囲む必要があります (例、(LPAD(COUNTRY, 20,''''))。このパラメータの一重引用符 (') 内に変数を使用することはできますが、一時項目 (DEFINE) または実フィールドに関係なく、フィールドを使用することはできません。
- 入力は、固定長または可変長の文字にすることができます。
- 出力は (SQL に最適化された場合)、常に VARCHAR データタイプになります。

- 指定した出力文字列の長さが元の入力文字列より短い場合、元のデータが切り取られ、パディング文字のみが残ります。出力文字列の長さは、正の整数として指定することも、引用符で囲まれていない & 変数 (数値を表す) として指定することもできます。

LTRIM - 文字列の左端からブランクを削除

LTRIM 関数は、文字列の左端からブランクをすべて削除します。

構文　文字列の左端からブランクを削除

```
LTRIM(string)
```

説明

`string`

文字

左端からブランクを削除する文字列です。

返される文字列のデータタイプは AnV で、ソース文字列と同一の最大長になります。

例　文字列の左端からブランクを削除

次のリクエストでは MOVIES データソースが使用され、DIRECTOR フィールドを右揃えにし、結果を RDIRECTOR という一時項目 (DEFINE) に格納します。次に、LTRIM 関数が、RDIRECTOR フィールドから左端のブランクを削除します。

```
DEFINE FILE MOVIES
RDIRECTOR/A17 = RJUST(17, DIRECTOR, 'A17');
 END
TABLE FILE MOVIES
PRINT RDIRECTOR AND
COMPUTE
TRIMDIR/A17 = LTRIM(RDIRECTOR);
WHERE DIRECTOR CONTAINS 'BR'
ON TABLE SET PAGE NOPAGE
END
```

出力結果は次のとおりです。

```
  RDIRECTOR          TRIMDIR
  ---------          -------
        ABRAHAMS J.  ABRAHAMS J.
          BROOKS R.  BROOKS R.
        BROOKS J.L.  BROOKS J.L.
```

POSITION - 文字列内のサブ文字列の開始位置を取得

POSITION 関数は、ソース文字列内のサブ文字列の開始位置を文字数で返します。

構文　文字列内のサブ文字列の開始位置を取得

```
POSITION(pattern, string)
```

説明

`pattern`
文字

開始位置を特定するサブ文字列です。この文字列は、単一の文字 (ブランクでも可) にすることも、複数の文字にすることもできます。

`string`
文字

パターンを検索する文字列です。

返される値のデータタイプは整数です。

例　サブ文字列の開始位置の取得

次のリクエストでは EMPLOYEE データソースが使用され、POSITION 関数が、LAST_NAME フィールドで最初の大文字「I」が出現する位置を特定し、結果を I_IN_NAME フィールドに格納します。

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
I_IN_NAME/I2 = POSITION('I', LAST_NAME);
ON TABLE SET PAGE NOPAGE
END
```

出力結果は次のとおりです。

```
LAST_NAME        I_IN_NAME
---------        ---------
STEVENS                  0
SMITH                    3
JONES                    0
SMITH                    3
BANNING                  5
IRVING                   1
ROMANS                   0
MCCOY                    0
BLACKWOOD                0
MCKNIGHT                 5
GREENSPAN                0
CROSS                    0
```

RPAD - 文字列の右パディング

RPAD 関数は、指定された文字および出力長に基づいて、その文字が右側にパディングされた文字列を返します。

構文　　文字列の右パディング

```
RPAD(string, out_length, pad_character)
```

説明

`string`
　文字

　右パディングを追加する文字列です。

`out_length`
　整数

　パディング追加後の出力文字列の長さです。

`pad_character`
　文字

　パディングに使用する単一文字です。

例　文字列の右パディング

次のリクエストでは WF_RETAIL データソースが使用され、RPAD 関数が、PRODUCT_CATEGORY フィールドの右側にパディング文字「@」を追加します。

```
DEFINE FILE WF_RETAIL
RPAD1/A25 = RPAD(PRODUCT_CATEGORY,25,'@');
DIG1/A4 = DIGITS(ID_PRODUCT,4);
END
TABLE FILE WF_RETAIL
SUM DIG1 RPAD1
BY PRODUCT_CATEGORY
ON TABLE SET PAGE NOPAGE
ON TABLE SET STYLE *
TYPE=DATA,FONT=COURIER,SIZE=11,COLOR=BLUE,$
END
```

出力結果は次のとおりです。

Product Category	DIG1	RPAD1
Accessories	5005	Accessories@@@@@@@@@@@@@@@
Camcorder	3006	Camcorder@@@@@@@@@@@@@@@@@
Computers	6016	Computers@@@@@@@@@@@@@@@@@
Media Player	1003	Media Player@@@@@@@@@@@@@
Stereo Systems	2155	Stereo Systems@@@@@@@@@@@
Televisions	4018	Televisions@@@@@@@@@@@@@@
Video Production	7005	Video Production@@@@@@@@@

参照　RPAD 使用上の注意

- 入力文字列のデータタイプは、AnV、VARCHAR、TX、An のいずれかにすることができます。
- 出力文字列のデータタイプは、AnV または An のみです。

- リレーショナル VARCHAR フィールドを使用する場合、特別な理由がない限り、フィールドから末尾のブランクを削除する必要はありません。ただし、An フィールドから得られた An および AnV フィールドの場合、末尾のブランクはデータの一部であり、これらの位置の右側にパディングが追加されて出力されます。RPAD 関数を適用する前に、TRIM または TRIMV 関数を使用することで、これらの末尾のブランクを削除することができます。

RTRIM - 文字列の右端からブランクを削除

RTRIM 関数は、文字列の右端からブランクをすべて削除します。

構文　文字列の右端からブランクを削除

```
RTRIM(string)
```

説明

`string`

文字

右端からブランクを削除する文字列です。

返される文字列のデータタイプは AnV で、ソース文字列と同一の最大長になります。

例　文字列の右端からブランクを削除

次のリクエストは、MOVIES データソースを使用して、DIRSLASH フィールドを作成し、DIRECTOR フィールドの末尾にスラッシュ (/) を追加した値を格納します。次に、TRIMDIR フィールドを作成し、DIRECTOR フィールドから末尾のブランクを削除した後、末尾にスラッシュ (/) を追加した値を格納します。

```
TABLE FILE MOVIES
PRINT DIRECTOR NOPRINT AND
COMPUTE
DIRSLASH/A18 = DIRECTOR|'/';
TRIMDIR/A17V = RTRIM(DIRECTOR)|'/';
WHERE DIRECTOR CONTAINS 'BR'
ON TABLE SET PAGE NOPAGE
END
```

出力結果のスラッシュ (/) の位置から分かるように、DIRECTOR フィールドには末尾のブランクが表示されていますが、TRIMDIR フィールドではブランクが削除されています。

```
  DIRSLASH            TRIMDIR
  --------            -------
  ABRAHAMS J.       /  ABRAHAMS J./
  BROOKS R.         /  BROOKS R./
  BROOKS J.L.       /  BROOKS J.L./
```

SUBSTRING - ソース文字列からサブ文字列を抽出

SUBSTRING 関数は、ソース文字列からサブ文字列を抽出します。サブ文字列に指定した終了位置がソース文字列の末尾を超える場合、ソース文字列の最終文字の位置は、サブ文字列の終了位置になります。

構文　　ソース文字列からサブ文字列を抽出

```
SUBSTRING(string, position, length)
```

説明

`string`
文字

サブ文字列を抽出する文字列です。この文字列には、フィールド、一重引用符 (') で囲んだリテラル、または変数のいずれかを指定することができます。

`position`
正の整数

string で指定した文字列内のサブ文字列の開始位置です。

`length`
整数

サブ文字列の長さ制限値です。サブ文字列の終了位置は、「position + length - 1」で計算されます。計算された位置がソース文字列の末尾を超える場合、string で指定した文字列の最終文字の位置が、サブ文字列の終了位置になります。

返されるサブ文字列のデータタイプは AnV です。

例　ソース文字列からサブ文字列を抽出

次のリクエストでは、POSITION 関数が LAST_NAME フィールドで最初に出現する文字「I」の位置を特定し、結果を I_IN_NAME フィールドに格納します。次に、SUBSTRING 関数が LAST_NAME フィールドで「I」から始まる 3 文字を抽出し、その結果を I_SUBSTR フィールドに格納します。

```
TABLE FILE EMPLOYEE
PRINT
COMPUTE
I_IN_NAME/I2 = POSITION('I', LAST_NAME); AND
COMPUTE
I_SUBSTR/A3 =
SUBSTRING(LAST_NAME, I_IN_NAME, I_IN_NAME+2);
BY LAST_NAME
ON TABLE SET PAGE NOPAGE
END
```

出力結果は次のとおりです。

```
LAST_NAME        I_IN_NAME  I_SUBSTR
---------        ---------  --------
BANNING                  5  ING
BLACKWOOD                0  BL
CROSS                    0  CR
GREENSPAN                0  GR
IRVING                   1  IRV
JONES                    0  JO
MCCOY                    0  MC
MCKNIGHT                 5  IGH
ROMANS                   0  RO
SMITH                    3  ITH
                         3  ITH
STEVENS                  0  ST
```

TOKEN - 文字列からトークンを抽出

TOKEN 関数は、文字列からトークン (サブ文字列) を抽出します。トークンは、区切り文字で区切られ、トークンの文字列内の位置を表すトークン番号が指定されます。

構文　文字列からトークンを抽出

```
TOKEN(string, delimiter, number)
```

説明

`source_string`
固定長の文字

トークンを抽出する文字列です。

delimiter
固定長の文字

単一の区切り文字です。

number
整数

抽出するトークン番号です。

例　文字列からトークンを抽出

TOKEN 関数は、PRODUCT_SUBCATEG フィールドから 2 つ目のトークンを抽出します。ここで、区切り文字は P です。

```
DEFINE FILE WF_RETAIL_LITE
TOK1/A20 =TOKEN(PRODUCT_SUBCATEG,'P',2);
END
TABLE FILE WF_RETAIL_LITE
SUM TOK1 AS Token
BY PRODUCT_SUBCATEG
ON TABLE SET PAGE NOPAGE
END
```

出力結果は次のとおりです。

Product Subcategory	Token
Blu Ray	
Boom Box	
CRT TV	
Charger	
DVD Players	layers
DVD Players - Portable	layers -
Flat Panel TV	anel TV
Handheld	
Headphones	hones
Home Theater Systems	
Portable TV	ortable TV
Professional	rofessional
Receivers	
Smartphone	hone
Speaker Kits	eaker Kits
Standard	
Streaming	
Tablet	
Universal Remote Controls	
Video Editing	
iPod Docking Station	od Docking Station

TRIM_ - 文字列から先頭、末尾、または両方の文字を削除

TRIM_ 関数は、文字列の先頭、末尾、または先頭と末尾の両方に出現する単一文字をすべて削除します。

注意

- 先頭および末尾のブランクは文字と見なされます。削除する文字の前にブランクがある場合 (先頭のブランク) または削除する文字の後にブランクがある場合 (末尾のブランク)、その文字は削除されません。文字フィールドの長さが、そのフィールドに格納されている文字より長い場合、フィールドの末尾にブランクがパディングされます。
- 指定した文字および位置での文字の削除がサポートされるリレーショナル DBMS に対してこの関数を実行する場合、関数が最適化されます。

構文　文字列から先頭、末尾、または両方の文字を削除

```
TRIM_(where, pattern, string)
```

説明

`where`
キーワード

ソース文字列から文字を削除する位置を指定します。有効な値には、次のものがあります。

- **LEADING**　先頭に出現する文字を削除します。
- **TRAILING**　末尾に出現する文字を削除します。
- **BOTH**　先頭と末尾の両方に出現する文字を削除します。

`pattern`
文字

単一文字を一重引用符 (') で囲みます。この文字が、string で指定した文字列から削除されます。たとえば、この文字を単一のブランク (' ') にすることができます。

`string`
文字

先頭または末尾から削除する文字列です。

返される文字列のデータタイプは AnV です。

例　文字列から文字を削除

次のリクエストでは、TRIM_ 関数が DIRECTOR フィールドの先頭に出現する「B」という文字を削除します。

```
TABLE FILE MOVIES
PRINT DIRECTOR AND
COMPUTE
TRIMDIR/A17 = TRIM_(LEADING, 'B', DIRECTOR);
WHERE DIRECTOR CONTAINS 'BR'
ON TABLE SET PAGE NOPAGE
END
```

出力結果は次のとおりです。

```
DIRECTOR           TRIMDIR
--------           -------
ABRAHAMS J.        ABRAHAMS J.
BROOKS R.          ROOKS R.
BROOKS J.L.        ROOKS J.L.
```

UPPER - 文字列をすべて大文字で取得

UPPER 関数は、ソース文字列のすべての文字を大文字に変換し、変換後の文字列を同一のデータタイプで返します。

構文　文字列をすべて大文字で取得

```
UPPER(string)
```

説明

`string`

文字

大文字に変換する文字列です。

文字列は、ソース文字列と同一のデータタイプと長さで返されます。

例　文字を大文字に変換

次のリクエストでは、LCWORD 関数が LAST_NAME フィールドを先頭大文字に変換します。次に、UPPER 関数が LAST_NAME_MIXED フィールドをすべて大文字に変換します。

```
DEFINE FILE EMPLOYEE
LAST_NAME_MIXED/A15=LCWORD(15, LAST_NAME, 'A15');
LAST_NAME_UPPER/A15=UPPER(LAST_NAME_MIXED) ;
END
TABLE FILE EMPLOYEE
PRINT LAST_NAME_UPPER AND FIRST_NAME
BY LAST_NAME_MIXED
WHERE CURR_JOBCODE EQ 'B02' OR 'A17' OR 'B04';
ON TABLE SET PAGE NOPAGE
END
```

出力結果は次のとおりです。

```
LAST_NAME_MIXED  LAST_NAME_UPPER  FIRST_NAME
---------------  ---------------  ----------
Banning          BANNING          JOHN
Blackwood        BLACKWOOD        ROSEMARIE
Cross            CROSS            BARBARA
Mccoy            MCCOY            JOHN
Mcknight         MCKNIGHT         ROGER
Romans           ROMANS           ANTHONY
```

4

データソースおよびデコード関数

データソースおよびデコード関数は、データソース値を取得し、入力フィールドの値に基づいて値を割り当てます。

トピックス

DB_EXPR - リクエストへの SQL 式の挿入

DB_EXPR 関数は、Web Query または SQL 言語のリクエストで生成されるネイティブ SQL に、ネイティブ SQL 式を入力されたとおりに挿入します。

DB_EXPR 関数は、DEFINE コマンド、マスターファイルの DEFINE、WHERE 句、SQL ステートメントで使用することができます。また、リクエストが集計リクエスト (SUM、WRITE、または ADD コマンドを使用) であり、1 つの表示コマンドが含まれている場合は、この関数を COMPUTE コマンドで使用することができます。この式は、単一値を返す必要があります。

構文　DB_EXPR によるリクエストへの SQL 式の挿入

```
DB_EXPR(native_SQL_expression)
```

説明

`native_SQL_expression`

リクエストで生成される SQL に挿入可能な部分的なネイティブ SQL 文字列です。SQL 文字列では、各フィールド参照の前後に二重引用符 (") を入力する必要があります。

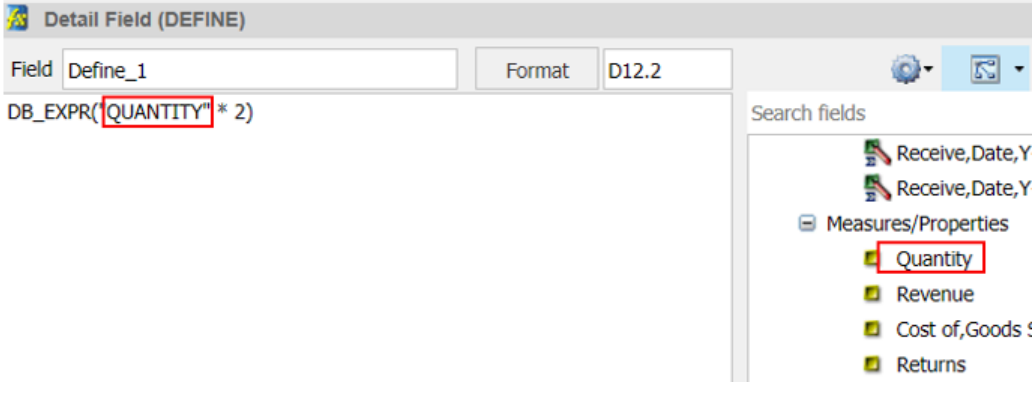
参照　DB_EXPR 関数使用上の注意

- 1 つまたは複数の DB_EXPR 関数が含まれたリクエストは、リレーショナル SUFFIX が存在するシノニムに使用する必要があります。
- ネイティブ SQL 式のフィールド参照は、現在のシノニムコンテキスト内に存在する必要があります。

- ネイティブ SQL 式は、インラインでコーディングする必要があります。ファイルから読み取られた SQL はサポートされません。
- DB_EXPR 関数で FIELDNAME を参照する場合は、WITH を使用するか、FIELDNAME を二重引用符 (") で囲む必要があります。ISQL スニペット式の FIELDNAME が適切に修飾されるよう、リクエストで変換済み SQL を作成する際は、二重引用符 (") の使用をお勧めします。
- SQL 式で指定する FIELDNAME は、大文字と小文字が区別されます。

例 QUANTITY を 2 で乗算

次の式は、「QUANTITY」という名前のフィールドを 2 で乗算します。FIELDNAME の大文字と小文字は区別され、FIELDNAME は二重引用符 (") で囲みます。下図のように、式では FIELDNAME が使用されますが、これは [DEFINE] ダイアログボックス右側パネルの列見出しとは異なることに注意してください。



例 TABLE リクエストへの Db2 BIGINT および CHAR 関数の挿入

次の構文は、DB_EXPR 関数を使用して 2 つの Db2 関数を呼び出します。この関数は、BIGINT 関数を呼び出して売上の 2 乗値のフィールドを BIGINT データタイプに変換し、次に CHAR 関数を使用してその値を文字に変換します。

```
DB_EXPR(CHAR(BIGINT("REVENUE" * "REVENUE") ) )
```

この式は、DEFINE フィールドまたは COMPUTE フィールドに入力することも、InfoAssist+ または Developer Workbench のシノニムエディタで入力することもできます。

DECODE - 値を置き換え

DECODE 関数は、コード化された入力フィールドの値に基づいて値を割り当てます。コード化されたフィールドの値に、より意味のある値を指定する場合に役立ちます。たとえば、フィールド GENDER は、女性従業員に F コード、男性従業員用に M コードを割り当てることにより、効率的にデータを格納します (女性従業員を表す 6 バイト分の文字 female の代わりに 1 バイト)。DECODE は、これらの値を置き換え、レポート上に正しく表示します。

DECODE 関数を使用するには、値を関数内に直接指定、または個別のファイルから読み取ります。

構文　　関数に値を指定

```
DECODE fieldname(code1 result1 code2 result2...[ELSE default]);
```

説明

`fieldname`
文字または数値

入力フィールド名です。

`code`
文字または数値

DECODE 関数が検索するコード化された値です。この値にブランク、カンマ (,)、または他の特殊文字が埋め込まれている場合、一重引用符 (') で囲みます。DECODE が指定した値を検出すると、対応する結果が返されます。

`result`
文字または数値

コードに割り当てられた値です。この値にブランクまたはカンマ (,) が埋め込まれている場合、あるいは負の数が含まれるときは、一重引用符 (') で囲みます。

`default`
文字または数値

コードが見つからない場合に割り当てられる値です。デフォルト値を省略する場合、DECODE は不一致コードにブランクまたは 0 (ゼロ) を割り当てます。

DECODE 関数のコードと結果の組み合わせには、通常 40 行まで、ELSE キーワードを含める場合は、39 行まで使用することができます。結果からコードを区別するには、カンマ (,) またはブランクを使用します。

例　関数に値を指定

まず、EDIT 関数が CURR_JOBCODE フィールドの 1 文字目を抽出します。次に、DECODE 関数は、抽出された値に基づいて ADMINISTRATIVE または DATA PROCESSING を返します。

```
COMPUTE DEPX_CODE/A1 = EDIT(CURR_JOBCODE, '9$$'); AND
COMPUTE JOB_CATEGORY/A15 = DECODE DEPX_CODE(A 'ADMINISTRATIVE'
B 'DATA PROCESSING') ;
```

構文　ファイルから値を読み込み

```
DECODE fieldname(ddname [ELSE default]);
```

説明

`fieldname`
文字または数値

入力フィールド名です。

`ddname`
文字

置き換えた値を含む物理ファイルを指定する論理名または短縮名です。

`default`
文字または数値

コードが見つからない場合に割り当てられる値です。デフォルト値を省略する場合、DECODE はブランクまたは 0 (ゼロ) を不一致コードに割り当てます。

参照　ファイルからの値の読み込みについてのガイドライン

- ファイル内の各レコードには、カンマ (,) またはブランクで区切られた要素の組み合わせが含まれていることが想定されます。
- ファイル内の各レコードが 1 つの要素のみで構成されている場合、この要素はコードとして解釈され、結果としてブランクまたは 0 (ゼロ) が生成されます。

  これにより、ファイルに次の選別条件で参照するリテラルを含めることができます。

  ```
  IF field IS (filename)
  ```

  また、このファイルを計算式で指定する IF 条件のリテラルファイルとして使用することもできます。以下はその例です。

  ```
  TAKE = DECODE SELECT (filename ELSE 1);
  VALUE = IF TAKE IS 0 THEN... ELSE...;
  ```

TAKE は、リテラルファイルに SELECT 値が見つかった場合には 0 (ゼロ)、それ以外の場合には 1 です。式の計算と同様に VALUE の計算が実行されます。

```
IF SELECT (filename) THEN... ELSE...;
```

- ファイルの最大容量は、32767 バイトです。
- 先頭と末尾のブランクは無視されます。
- 各レコードの残りの部分は無視されますが、これらはコメントや他のデータに使用可能です。この規則は、ファイル名が HOLD である場合以外に適用されます。ファイル名が HOLD の場合、ファイルはフィールドに内部フォーマットで書き込む HOLD コマンドにより作成されたと見なされ、DECODE の組はそれに基づいて解釈されます。この場合、レコードの残りの部分は無視されます。

例　ファイルから値を読み込み

DECODE 関数は、HOLD ファイルに EMP_ID が存在する従業員に値 0 (ゼロ)、EMP_ID が存在しない従業員に値 1 を割り当てます。

```
COMPUTE NOT_IN_LIST/I1 = DECODE EMP_ID(HOLD ELSE 1);
```

LAST - 前の値を抽出

LAST 関数は、フィールドの前の値を抽出します。

LAST の影響は、DEFINE コマンドと COMPUTE コマンドのどちらに使用するかによって異なります。

- DEFINE コマンドでは、LAST 値は、ソートを実行する前に、データソースから抽出した前のレコードに適用されます。
- COMPUTE コマンドでは、LAST 値は、内部マトリックスの前の行のレコードに適用されます。

構文　前の値を抽出

```
LAST fieldname
```

説明

`fieldname`
　文字または数値

フィールド名です。

例　前の値を抽出

LAST は、DEPARTMENT フィールドの前の値を抽出し、部署ごとの給与合計を再計算するかどうかを決定します。前の値が現在の値と同一である場合、CURR_SAL が RUN_TOT に追加され、各部署の給与の合計が生成されます。

```
COMPUTE RUN_TOT/D12.2M = IF DEPARTMENT EQ LAST DEPARTMENT THEN
                (RUN_TOT + CURR_SAL) ELSE CURR_SAL ;
```

5

日付時間関数

日付と時間の値を操作する日付および時間関数について説明します。

標準日付時間関数を使用する際、これらの関数の動作を変更する設定、使用可能なフォーマット、値の指定方法について理解する必要があります。

日付時間関数では、営業日を定義することで、動作を変更することができます。これにより、営業日に対して日付関数を使用すると、営業日以外の日付は無視されます。

AYM - 基準となる日付に月数を加えて、新たな年月を求める

AYM 関数は、年月フォーマットの日付と指定した月数の和または差を計算します。CHGDAT または EDIT 関数を使用することで、日付をこのフォーマットに変換することができます。

構文　　日付に月数を加算または減算

```
AYM(indate, months, 'outfield')
```

説明

`indate`
整数 (I4、I4YM、I6、または I6YYM)

年月フォーマットの元の日付、日付を含むフィールド名、日付を返す式のいずれかを指定します。日付が有効でない場合、この関数は 0 (ゼロ) を返します。

`months`
整数

日付に加算、または日付から減算する月数です。月を減算するには、負の値を使用します。

`outfield`
整数 (I4YM または I6YYM)

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

ヒント：入力日付が整数年月日フォーマット (I6YMD または I8YYMD) の場合、日付を 100 で除算することにより、年月フォーマットに変換し、結果を整数に設定します。これにより日付の日の部分が削除されます。日は、小数部分で表されます。

例　日付に月数を追加

COMPUTE コマンドは、HIRE_DATE の日付を年月日から年月フォーマットに変換し、結果を HIRE_MONTH に格納します。AYM は、HIRE_MONTH に 6 か月加算し、結果を AFTER6MONTHS に格納します。

```
COMPUTE HIRE_MONTH/I4YM = HIRE_DATE/100; AND
COMPUTE AFTER6MONTHS/I4YM = AYM(HIRE_MONTH, 6, 'I4YM');
```

AYMD - 基準となる日付に日数を加えて、新たな日付を求める

AYMD 関数は、年月フォーマットの日付と指定した日数の和または差を計算します。CHGDAT または EDIT 関数を使用することで、日付をこのフォーマットに変換することができます。

日数の加算または減算により、世紀が前後に変更される場合、出力年の世紀の桁は調整されます。

構文　基準となる日付に日数を加えて、新たな日付を求める

```
AYMD(indate, days, 'outfield')
```

説明

`indate`
整数 (I6、I6YMD、I8、 I8YYMD)

日付が有効でない場合、この関数は 0 (ゼロ) を返します。

`days`
整数

indate に加算または indate から減算する日数です。日を減算するには、負の値を使用します。

`outfield`
整数 (I6、I6YMD、I8、I8YYMD)

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。*indate* がフィールドの場合、*outfield* とフォーマットが同一である必要があります。

例　日付に日数を追加

AYMD 関数は、HIRE_DATE フィールド内の各値に 35 日加算し、結果を AFTER35DAYS に格納します。

```
COMPUTE AFTER35DAYS/I6YMD = AYMD(HIRE_DATE, 35, 'I6YMD');
```

CHGDAT - 日付文字列の表示を変更

CHGDAT 関数は、日付を表す入力文字列の年、月、日の部分を再編成します。この関数により、長い日付入力文字列を短く、短い日付入力文字列を長くすることもできます。長い表現には、3 つの日付構成要素である年、月、日がすべて含まれ、短い表現では、年、月、日の日付構成要素の 1 つまたは 2 つが省略されます。入力および出力日付文字列は、日付文字列内の構成要素 (年、月、日) の順序、および年に 2 桁と 4 桁のどちらを使用するか (例、97 または 1997) を記述します。CHGDAT は、入力日付文字列を読み取り、同一の日付を異なる方法で表示する出力日付文字列を作成します。

注意：CHGDAT には、実際の日付ではなく入力文字列としての日付が必要です。CHGDAT を適用する前に、入力文字列を日付文字列に変換 (例、EDIT または DATECVT を使用) します。

日付文字列内の日付構成要素の順序は、次の文字で構成される表示オプションで記述します。

文字	説明
D	日 (01 から 31)
M	月 (01 から 12)
Y[Y]	年 Y - 2 桁の年を示します (例、94)。YY - 4 桁の年を示します (例、 1994)。

結果文字列に数字ではなく月名を表示するには、結果文字列の表示オプションに次の文字を追加します。

文字	説明
T	月を 3 バイトの略語で表示します。
X	月を完全な名前で表示します。

表示オプションには、表示する文字を 5 バイト以下で指定することができます。これらの表示オプション以外の文字列は、無視されます。

たとえば、表示オプション「DMYY」は、2 桁の日、2 桁の月、4 桁の年を指定します。

注意：表示オプションは、日付フォーマットとは異なります。

参照　日付の長さを変換

下表のように、日付を短い表現から長い表現 (例、年月から年月日) に変換すると、短い表現内に存在しない日付部分は、この関数により指定されます。

存在しない日付部分	関数により指定
日 (例、YM から YMD)	月の最終日
月 (例、Y から YM)	年の最終月 (12 月)
年 (例、MD から YMD)	99 年
2 桁の年から 4 桁の年への変換 (例、YMD から YYMD)	DATEFNS=ON の場合、世紀は DEFCENT および YRTHRESH で定義する 100 年で決定されます。 DATEFNS=OFF の場合、19xx 年が指定されます。ここで、xx は年の末尾の 2 桁です。

構文　日付文字列の表示を変更

```
CHGDAT('in_display_options', 'out_display_options', date_string,
'outfield')
```

説明

`in_display_options`
文字 (A1 から A5)

date_string のレイアウトを記述する 5 バイト以内の一連の表示オプションです。これらのオプションは、文字フィールドに格納するか、リテラルとして指定します。リテラルは一重引用符で囲みます。

`out_display_options`
文字 (A1 から A5)

変換後の文字列のレイアウトを記述する 5 バイト以内の一連の表示オプションです。これらのオプションは、文字フィールドに格納するか、リテラルとして指定します。リテラルは一重引用符で囲みます。

`date_string`
文字 (A2 から A8)

入力日付文字列です。日付構成要素の順序は *in_display_options* で指定します。

元の日付が数値フォーマットの場合、日付文字列に変換する必要があります。*date_string* の日付表現が正しくない (日付が無効) 場合、関数はブランクを返します。

`outfield`
文字

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

out_display_options で指定する日付文字列を格納するために十分なバイト数を指定します。A17 を指定すると、最長の日付文字列を格納することができます。

参照 CHGDAT 関数使用上の注意

CHGDAT は、日付ではなく日付文字列を使用し、17 バイト以下の日付文字列を返すため、EDIT 関数、DATECVT 関数、またはそれ以外の手段により、日付文字列に変換、あるいは日付文字列から変換する必要があります。

例 日付表示を YMD から MDYYX に変換

EDIT 関数は、HIRE_DATE の表示を数値から文字フォーマットに変換します。CHGDAT 関数は、ALPHA_HIRE の各値の構成要素の表示を YMD から MDYYX に変換し、結果を A17 フォーマットで HIRE_MDY に格納します。出力値のオプション X により、完全な月名が表示されます。

```
COMPUTE ALPHA_HIRE/A17 = EDIT(HIRE_DATE); AND
COMPUTE HIRE_MDY/A17 = CHGDAT('YMD', 'MDYYX', ALPHA_HIRE, 'A17');
```

DA - 日付を整数に変換

DA 関数は、日付を 1899 年 12 月 31 日との日数の差に変換します。日付を日数に変換することにより、日付の和または差の計算を実行し、日付の間隔を計算することができます。

DA 関数は 6 つあります。各関数では、異なるフォーマットの日付が使用可能です。

構文 レガシー日付を整数に変換

```
function(indate, 'outfield')
```

説明

`function`
次のいずれかです。

DADMY - 日月年フォーマットの日付を変換します。

DADYM - 日年月フォーマットの日付を変換します。

DAMDY - 月日年フォーマットの日付を変換します。

DAMYD - 月年日フォーマットの日付を変換します。

DAYDM - 年日月フォーマットの日付を変換します。

DAYMD - 年月日フォーマットの日付を変換します。

indate
整数またはパック 10 進数

日付表示オプションを含む I または P フォーマットです。

変換される日付、または日付を含むフィールド名のいずれかを指定します。変換前に日付の端数が切り捨てられ、整数になります。

年を指定するには、最後の 2 桁のみを入力します。世紀構成要素は、関数により入力されます。日付が有効でない場合、この関数は 0 (ゼロ) を返します。

outfield
整数

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。返される日付のフォーマットは、関数により異なります。

例　日付を変換した後に差を計算

DAYMD 関数は、DAT_INC および HIRE_DATE フィールドの値を 1899 年 12 月 31 日との日数の差に変換します。その後、これらの値の差が計算されます。

```
COMPUTE DAYS_HIRED/I8 = DAYMD(DAT_INC, 'I8') - DAYMD(HIRE_DATE, 'I8');
```

DATEADD - 日付単位数を日付に加算または日付から減算

DATEADD 関数は、日付単位数 (例、日数、月数、年数) を日付に加算または日付から減算します。単位は次のいずれかです。

- **年**
- **月**　月単位を使用する計算が無効な日付を作成する場合、DATEADD は、この日付を月の最終日に修正します。たとえば、10 月 31 日に 1 か月加算すると、11 月は 30 日間であるため、11 月 30 日になります。

- 日
- **平日**　平日単位を使用すると、DATEADD は土曜日と日曜日を計算しません。たとえば、金曜日に 1 日加算すると、結果は月曜日になります。
- **営業日**　営業日単位を使用すると、DATEADD は BUSDAYS パラメータおよび祝日ファイルにより、営業日を決定します。これ以外は無視されます。月曜日が営業日でない場合、日曜日の次の営業日は火曜日になります。

日を基準としない日付 (例、YM または YQ) の和または差の計算は、DATEADD 関数を使用せずに、直接実行してください。

DATEADD 関数は、年月日を含む完全な日付形式でのみ有効です。

構文　日付単位数を日付に加算または日付から減算

```
DATEADD(date, 'unit', #units)
```

説明

`date`
日付

年月日を含む完全な日付形式です。

`unit`
文字

次のいずれかを一重引用符 (') で囲んで指定します。

`Y` - 構成要素「年」を示します。

`M` - 構成要素「月」を示します。

`D` - 構成要素「日」を示します。

`WD` - 構成要素「平日」を示します。

`BD` - 構成要素「営業日」を示します。

`#units`
整数

date で指定した日付に加算する、または日付から減算する日付単位数です。この単位数が整数でない場合、小数点以下が切り捨てられて次に大きい整数になります。

例　　日付に平日を追加

DATEADD は、平日 3 日を NEW_DATE に追加します。3 日追加すると HIRE_DATE_PLUS_THREE が週末になることがあるため、4 日以上追加する場合があります。

```
COMPUTE NEW_DATE/YYMD = HIRE_DATE; AND
COMPUTE HIRE_DATE_PLUS_THREE/YYMD = DATEADD(NEW_DATE, 'WD', 3);
```

DATECVT - 日付フォーマットを変換

DATECVT 関数は、中間の計算を実行せずに、アプリケーションの日付のフォーマットを変換します。無効なフォーマットを指定すると、DATECVT は 0 (ゼロ) またはブランクを返します。

日付フォーマットの A8YYMD、A8MDYY、A8DMYY、I8YYMD、I8MDYY、I8DMYY、P8YYMD、P8MDYY、P8DMYY を使用する場合、DATECVT 関数は最適化されます。日付の変換は、最適化済み SQL が Db2 エンジンに渡されて、処理されます。

注意：この関数を呼び出す代わりに、単純な割り当てを使用することができます。

構文　　日付フォーマットを変換

```
DATECVT(date, 'infmt', 'outfmt')
```

説明

`date`
　日付

　変換される日付です。無効な日付を指定すると、0 (ゼロ) が返されます。変換を実行する際、レガシー日付は、このフィールドに指定された DEFCENT および YRTHRESH パラメータの設定に従います。

`infmt`
　文字

　日付のフォーマットです。フォーマットは一重引用符 (') で囲みます。次のいずれかです。

- 標準、非レガシー、または日付フォーマット (例、YYMD、YQ、M、DMY、JUL)。
- レガシー日付フォーマット (例、I6YMD、A8MDYY)。
- 非日付フォーマット (例、I8、A6)。YYMD フィールドの基準日 (1900/12/31) のオフセットとしての *infmt* 関数の非日付フォーマットです。

`outfmt`
　文字

出力のフォーマットです。フォーマットは一重引用符 (') で囲みます。次のいずれかです。

- 標準、非レガシー、または日付フォーマット (例、YYMD、YQ、M、DMY、JUL)。
- レガシー日付フォーマット (例、I6YMD、A8MDYY)。
- 非日付フォーマット (例、I8、A6)。YYMD フィールドの基準日 (1900/12/31) のオフセットとしての *infmt* 関数の非日付フォーマットです。

例　日付フォーマットを YYMD から DMY に変換

DATECVT 関数は、19991231 を 311299 に変換し、結果を CONV_FIELD に格納します。

```
COMPUTE CONV_FIELD/DMY = DATECVT(19991231, 'I8YYMD', 'DMY');
```

または

```
COMPUTE CONV_FIELD/DMY = DATECVT('19991231', 'A8YYMD', 'DMY');
```

例　レガシー日付を日付フォーマットに変換

DATECVT 関数は、HIRE_DATE のフォーマットを I6YMD レガシー日付フォーマットから YYMD 日付フォーマットに変換します。

```
COMPUTE NEW_HIRE_DATE/YYMD = DATECVT(HIRE_DATE, 'I6YMD', 'YYMD');
```

DATEDIF - 2 つの日付の差を計算

DATEDIF 関数は、指定した単位で 2 つの日付の差を返します。単位は次のいずれかです。

- **年**　DATEDIF で年単位を使用すると、DATEADD の逆の結果が返されます。日付 X から 1 年を減算して日付 Y を作成する場合、X と Y の年差は 1 です。2 月 29 日から 1 年を減算すると、結果は 2 月 28 日になります。
- **月**　DATEDIF で月単位を使用すると、DATEADD の逆の結果が返されます。日付 X から 1 か月を減算して日付 Y を作成する場合、X と Y の月差は 1 です。to_date が月の最終日である場合、逆数計算規則を保障するために、月差は絶対数で切り上げられる可能性があります。

  入力日付の 1 つまたは両方が月の最終日である場合、この規則が適用されます。これにより、1 月 31 日と 4 月 30 日の月差は、2 か月ではなく 3 か月になります。
- **日**

- **平日**　平日単位を使用すると、DATEDIF は日付の計算から土曜日と日曜日を除外します。これにより、金曜日と月曜日の差は 1 日になります。
- **営業日**　営業日単位を使用すると、DATEDIF は BUSDAYS パラメータおよび祝日ファイルにより、営業日を決定します。これ以外は無視されます。これにより、月曜日が営業日でない場合、金曜日と火曜日の差は 1 日になります。

DATEDIF は、整数を返します。2 つの日付の差が整数でない場合、DATEDIF は値を切り捨て、次の最大整数値を返します。たとえば、2001 年 3 月 2 日と 2002 年 3 月 1 日の年差は 0 (ゼロ) です。終了日が開始日よりも前である場合、DATEDIF は負の値を返します。

日を基準としない日付 (例、YM または YQ) の差の計算は、DATEDIF 関数を使用せずに、直接実行してください。

構文　2 つの日付の差を計算

```
DATEDIF(from_date, to_date, 'unit')
```

説明

`from_date`
日付

差を計算する開始日です。年月日を含む完全な日付形式です。

`to_date`
日付

差を計算する終了日です。

`unit`
文字

次のいずれかを一重引用符 (') で囲んで指定します。

`Y` - 構成要素「年」を示します。

`M` - 構成要素「月」を示します。

`D` - 構成要素「日」を示します。

`WD` - 構成要素「平日」を示します。

`BD` - 構成要素「営業日」を示します。

例　2つの日付の差に基づき平日日数を計算

DATECVT 関数は、HIRE_DATE と DAT_INC 内のレガシー日付を日付フォーマット YYMD に変換します。DATEDIF 関数は、これらの日付フォーマットで、NEW_HIRE_DATE と NEW_DAT_INC の差から平日日数を計算します。

```
COMPUTE NEW_HIRE_DATE/YYMD = DATECVT(HIRE_DATE, 'I6YMD', 'YYMD'); AND
COMPUTE NEW_DAT_INC/YYMD = DATECVT(DAT_INC, 'I6YMD', 'YYMD'); AND
COMPUTE WDAYS_HIRED/I8 = DATEDIF(NEW_HIRE_DATE, NEW_DAT_INC, 'WD');
```

DATEMOV - 日付を有効な位置に移動

DATEMOV 関数は、日付を有効な位置に移動します。

DATEMOV 関数は、年月日を含む完全な日付形式でのみ有効です。

構文　日付を指定の位置に移動

```
DATEMOV(date, 'move-point')
```

説明

`date`
日付

年月日を含む完全な日付形式です。 移動する日付です。

`move-point`
文字

日付を移動する有効な位置です。一重引用符 (') で囲みます。無効な位置を指定すると、リターンコード 0 (ゼロ) が返されます。有効な値には、次のものがあります。

`EOM` - 月の終わりです。

`BOM` - 月のはじめです。

`EOQ` - 四半期の終わりです。

`BOQ` - 四半期のはじめです。

`EOY` - 年の終わりです。

`BOY` - 年のはじめです。

`EOW` - 週の終わりです。

`BOW` - 週のはじめです。

NWD - 次の平日です。

NBD - 次の営業日です。

PWD - 先週の平日です。

PBD - 前回の営業日です。

WD- - 平日またはそれ以前です。

BD- - 営業日またはそれ以前です。

WD+ - 平日またはそれ以降です。

BD+ - 営業日またはそれ以降です。

営業日の計算には、BUSDAYS および HDAY パラメータの設定が反映されます。

例　週の最終日を特定

DATEMOV 関数は、NEW_DATE 内の各日付の週の最終日を特定し、結果を EOW に格納します。

```
COMPUTE NEW_DATE/YYMDWT = DATECVT(HIRE_DATE, 'I6YMD', 'YYMDWT'); AND
COMPUTE EOW/YYMDWT = DATEMOV(NEW_DATE, 'EOW');
```

日付構成要素を整数として取得

DPART 関数は、日付フィールドから、指定した構成要素を抽出し、数値フォーマットで返します。

構文　日付構成要素を抽出して整数を取得

```
DPART(datevalue, 'component', outfield)
```

説明

datevalue

日付

年月日を含む完全な日付形式です。

component

文字

取得される構成要素名です。文字列は一重引用符 (') で囲みます。有効な値には、次のものがあります。

年 - YEAR、YY

月 - MONTH、MM

日 - DAY、DAY-OF-MONTH

四半期 - QUARTER、QQ

`outfield`
整数

結果を格納するフィールド名、または出力値の整数フォーマットです。フォーマットの場合は一重引用符 (') で囲みます。

例 日付構成要素を整数フォーマットで抽出

次のリクエストは、VIDEOTRK データソースが使用され、DPART 関数によって TRANSDATE フィールドから年、月、日付コンポーネントを抽出します。

```
DEFINE FILE
 VIDEOTRK
 YEAR/I4 = DPART(TRANSDATE, 'YEAR', 'I4');
 MONTH/I4 = DPART(TRANSDATE, 'MM', 'I4');
 DAY/I4 = DPART(TRANSDATE, 'DAY', 'I4');
END

TABLE FILE VIDEOTRK
PRINT TRANSDATE YEAR MONTH DAY
BY LASTNAME BY FIRSTNAME
WHERE LASTNAME LT 'DIAZ'
END
```

出力結果は次のとおりです。

```
LASTNAME        FIRSTNAME   TRANSDATE  YEAR  MONTH   DAY
--------        ---------   ---------  ----  -----   ---
ANDREWS         NATALIA     91/06/19   1991      6    19
                            91/06/18   1991      6    18
BAKER           MARIE       91/06/19   1991      6    19
                            91/06/17   1991      6    17
BERTAL          MARCIA      91/06/23   1991      6    23
                            91/06/18   1991      6    18
CHANG           ROBERT      91/06/28   1991      6    28
                            91/06/27   1991      6    27
                            91/06/26   1991      6    26
COLE            ALLISON     91/06/24   1991      6    24
                            91/06/23   1991      6    23
CRUZ            IVY         91/06/27   1991      6    27
DAVIS           JASON       91/06/24   1991      6    24
```

DATETRAN - 日付を国際フォーマットに変換

DATETRAN 関数は、日付を国際フォーマットに変換します。

構文　　日付を国際フォーマットに変換

```
DATETRAN (indate, '(intype)', '([formatops])', 'lang', outlen, 'outfield')
```

説明

`indate`
フォーマットを設定する入力日付です。日付フォーマットには、日付表示オプション付きの文字または数値フォーマットは使用できません。

`intype`
入力日付構成要素とその表示順序を指定する次のいずれかの文字列です。文字列は、一重引用符 (') と括弧で囲みます。

入力構成要素が 1 つの場合

1 つの入力構成要素	説明
`'(W)'`	曜日構成要素のみ (元のフォーマットは「W」のみ)
`'(M)'`	月構成要素のみ (元のフォーマットは「M」のみ)

入力構成要素が 2 つの場合

2 つの入力構成要素	説明
`'(YYM)'`	4 桁の西暦年、月
`'(YM)'`	2 桁の西暦年、月
`'(MYY)'`	月、4 桁の西暦年
`'(MY)'`	月、2 桁の西暦年

入力構成要素が 3 つの場合

3 つの構成要素の入力タイプ	説明
'(YYMD)'	4 桁の西暦年、月、日
'(YMD)'	2 桁の西暦年、月、日
'(DMYY)'	日、月、4 桁の西暦年
'(DMY)'	日、月、2 桁の西暦年
'(MDYY)'	月、日、4 桁の西暦年
'(MDY)'	月、日、2 桁の西暦年
'(MD)'	月、日 (年月日の日付から抽出。西暦年は無視)
'(DM)'	日、月 (年月日の日付から抽出。西暦年は無視)

`formatops`

0 (ゼロ) 以上のフォーマットオプションを表す文字列です。文字列は括弧および一重引用符 (') で囲みます。括弧と引用符 (') は、フォーマットオプションを指定しない場合でも必要です。次のフォーマットオプションがあります。

- 月または日の数値の先頭の 0 (ゼロ) を非表示にするオプション。
- 月または日構成要素を完全な名前または略名に変換するオプション。変換先の文字は、すべて大文字に指定することも、言語のデフォルト値 (先頭大文字、またはすべて小文字) に指定することも可能です。
- 日付の区切り文字オプション、および日付にカンマ (,) を付けるオプション。

数値の先頭の 0 (ゼロ) を非表示にするオプションには次のものがあります。

フォーマットオプション	説明
m	月部分の 0 (ゼロ) を省略 (1 月から 9 月を 01 から 09 ではなく 1 から 9 で表示) します。
d	1 桁の日を 01 から 09 ではなく、1 から 9 として表示します。
dp	1 桁の日を 01 から 09 ではなく、1 から 9 として表示します。数値の後にはピリオド (.) が追加されます。
do	1 桁の日を 1 から 9 として表示します。英語 (言語コード EN) でのみ、数値の後に序数を表す接尾語 (st、nd、rd、th) が追加されます。

有効な月名および日付オプションは次のとおりです。

フォーマットオプション	説明
T	月の略名がピリオド (.) なしで表示されます。すべて大文字です。
TR	完全な月名が表示されます。すべて大文字です。
Tp	月の略名が末尾にピリオド (.) を伴って表示されます。すべて大文字です。
t	月の略名がピリオド (.) なしで表示されます。言語コードによって、すべて小文字または先頭大文字で表示されます。
tr	完全な月名が表示されます。言語コードによって、すべて小文字または先頭大文字で表示されます。
tp	月の略名が末尾にピリオド (.) を伴って表示されます。名前の大文字、小文字は選択する言語のデフォルト値が使用されます (例、フランス語およびスペイン語ではすべて小文字、英語およびドイツ語では先頭大文字)。

フォーマットオプション	説明
W	日付の先頭に曜日の略名が表示されます。すべて大文字で、ピリオド (.) は使用されません。
WR	日付の先頭に完全な曜日名が表示されます。すべて大文字です。
Wp	日付の先頭に曜日の略名が表示されます。すべて大文字で、末尾にはピリオド (.) が追加されます。
w	日付の先頭に曜日の略名が区切り記号なしで表示されます。ピリオド (.) は使用されません。名前の大文字、小文字は選択する言語のデフォルト値が使用されます (例、フランス語およびスペイン語ではすべて小文字、英語およびドイツ語では先頭大文字)。
wr	日付の先頭に完全な曜日名が表示されます。名前の大文字、小文字は選択する言語のデフォルト値が使用されます (例、フランス語およびスペイン語ではすべて小文字、英語およびドイツ語では先頭大文字)。
wp	日付先頭に曜日の略名が表示されます。末尾にはピリオド (.) が追加されます。名前の大文字、小文字は選択する言語のデフォルト値が使用されます (例、フランス語およびスペイン語ではすべて小文字、英語およびドイツ語では先頭大文字)。
X	日付の末尾に曜日の略名が表示されます。すべて大文字で、ピリオド (.) は使用されません。
XR	日付の末尾に完全な曜日名が表示されます。すべて大文字です。
Xp	日付の末尾に曜日の略名が表示されます。すべて大文字で、末尾にはピリオド (.) が追加されます。
x	日付の末尾に曜日の略名がピリオド (.) なしで表示されます。名前の大文字、小文字は選択する言語のデフォルト値が使用されます (例、フランス語およびスペイン語ではすべて小文字、英語およびドイツ語では先頭大文字)。

フォーマットオプション	説明
xr	日付の末尾に完全な曜日名が表示されます。名前の大文字、小文字は選択する言語のデフォルト値が使用されます (例、フランス語およびスペイン語ではすべて小文字、英語およびドイツ語では先頭大文字)。
xp	日付の末尾に曜日の略名がピリオド (.) を伴って表示されます。名前の大文字、小文字は選択する言語のデフォルト値が使用されます (例、フランス語およびスペイン語ではすべて小文字、英語およびドイツ語では先頭大文字)。

有効な日付の区切り文字オプションは、次のとおりです。

フォーマットオプション	説明
B	構成要素の区切り文字にブランクを 1 つ使用します。このオプションは、月名および曜日が文字に変換されている場合、またはカンマ (,) が使用されている場合のデフォルト値です。
.	構成要素の区切り文字にピリオド (.) を使用します。
-	構成要素の区切り文字にマイナス記号 (-) を使用します。このオプションは、ブランクがデフォルトの区切り文字として使用できない場合のデフォルト値です。
/	構成要素の区切り文字にスラッシュ (/) を使用します。
|	構成要素の区切り文字を省略します。
K	構成要素の区切り文字に適切なアジア言語の文字を使用します。

フォーマットオプション	説明
c	月の末尾にカンマ (,) を追加します (T、Tp、TR、t、tp、tr の後ろに追加)。 日の末尾にカンマ (,) とブランクを 1 つずつ追加します (W、Wp、WR、w、wp、wr の後ろに追加)。 日の前にカンマ (,) とブランクを 1 つずつ追加します (X、XR、x、xr の後ろに追加)。
e	スペイン語やポルトガル語の「de」または「DE」を日と月の間、および月と年の間に表示します。大文字と小文字の使用は、月名に一致します。月名が大文字の場合は「DE」、小文字の場合は「de」が表示されます。DMY、DMYY、MY、MYY フォーマットで役立ちます。
D	日と指定された区切り文字の間にカンマ (,) を挿入します。
Y	年と指定された区切り文字の間にカンマ (,) を挿入します。

lang
日付が変換される言語の 2 バイトの標準 ISO コードです。文字は一重引用符 (') で囲みます。以下は、有効な言語コードです。

'AR' アラビア語

'CS' チェコ語

'DA' デンマーク語

'DE' ドイツ語

'DU' オランダ語

'EN' 英語

'ES' スペイン語

'FI' フィンランド語

'FR' フランス語

'EL' ギリシャ語

'IW' ヘブライ語

'IT' イタリア語

'JA' 日本語

'KO' 韓国語

'LT' リトアニア語

'NO' ノルウェー語

'PO' ポーランド語

'PT' ポルトガル語

'RU' ロシア語

'SV' スウェーデン語

'TH' タイ語

'TR' トルコ語

'TW' 中国語 (繁体字)

'ZH' 中国語 (簡体字)

`outlen`
数値

出力フィールドの長さをバイト数で指定します。フィールドの長さが値よりも小さい場合は、すべてがブランクの結果が返されます。長さが大きすぎる場合は、右側にブランクが挿入されます。

`outfield`
文字

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

参照 DATETRAN 関数使用上の注意

- 月名および曜日名の長さが異なるため、タイプ A 出力フィールドには、可変長の情報を含めることができます。また、ゼロサプレスオプションを選択した場合、月番号および曜日番号の長さは 1 バイトまたは 2 バイトのいずれかになります。使用されないバイトの部分には、ブランクが挿入されます。
- 入力が無効または不整合の場合は、0 (ゼロ) が出力されます。データが欠落している場合はブランクが出力されます。

- 基準日 (1900-12-31 および 1900-12、または 1901-01) は、DATEDISPLAY 設定が ON である状態として処理され、自動的にはブランクとして表示されません。基準日 (内部整数値 0 を含む) の出力を非表示にするには、DATETRAN 関数を呼び出す前に 0 (ゼロ) をテストします。以下はその例です。

```
RESULT/A40 = IF DATE EQ 0 THEN ' ' ELSE
                DATETRAN (DATE, '(YYMD)', '(.t)', 'FR', 40, 'A40');
```

- 変換後の日付構成要素の有効値は、「DTLNG*lng*」という名前のファイルに含まれています。*lng* は、言語を指定する 3 バイトのコードです。 これらのファイルは、日付の変換先の各言語からアクセス可能である必要があります。

例　DATETRAN 関数の使用

次の DATETRAN 関数は、フランス語でデフォルト設定の大文字小文字表記で曜日を表示します。

```
COMPUTE OUT/A8=DATETRAN(DATEW, '(W)', '(wr)', 'FR', 8 , 'A8') ;
```

日付時間値の精度

以前のバージョンでは、日付時間値の秒コンポーネントは 0 (ゼロ)、3、6 桁で表示することができました。現在のバージョンでは、0 (ゼロ) 桁から 9 桁で表示することができます。

日付時間フォーマットの秒コンポーネントを制御するには、フォーマットに 1 から 9 までの桁数を入力します。

構文　日付時間値の精度

3 桁 (ミリ秒、s) 、6 桁 (マイクロ秒、m) 、9 桁 (ナノ秒、n) 用に、特殊なフォーマットコンポーネントが用意されています。表示する精度に応じて、次のように指定します。

- **時間値のみ**　フォーマットに複数の時間表示コンポーネントが含まれる場合、次のように指定します。
  - 時間、分、秒、ミリ秒、マイクロ秒、ナノ秒の順にコンポーネントを指定します。
  - 最初のコンポーネントには、時、分、秒のいずれかを指定する必要があります。
  - 中間の構成要素を省略することはできません。時を指定した場合、次のコンポーネントは分にしなければならず、秒にすることはできません。
  - 秒 (S) のフォーマットの後に表示桁数を入力します。ここで、ミリ秒 (s)、マイクロ秒 (m)、ナノ秒 (n) コンポーネントを 1 つかそれ以上指定することもできます。

- **日付コンポーネントと時間コンポーネントの両方**　時間コンポーネントは 1 文字で、表示する時間の最小単位を指定します。この文字は次のいずれかです。
  - 秒コンポーネントに表示する桁数を指定する 1 から 9 までの数字。
  - サポートされる時間コンポーネントのいずれか。この場合、上位の時間構成要素もすべて表示されます。

例　日付時間値精度の指定

これらの変換を実行するには、レポート開発ツールの COMPUTE または DEFINE ダイアログボックス、あるいは Developer Workbench のシノニムエディタを使用します。日付が 1999 年 2 月、時間が午前 02:05:25.123456789 であることを想定します。次のリクエストは、表示フォーマットおよび関数呼び出しにナノ秒の時間構成要素 (精度) を使用しています。

```
TRANSDT/HYYMDn      = DT(19990205 02:05:25.123456789);
 O_HSsmn/HSsmn       = TRANSDT;
 O_HHIS2/HHIS2       = TRANSDT;
 O_HYYMDn/HYYMDn     = TRANSDT;
 O_HYYMD1/HYYMD1     = TRANSDT;
 O_HADD/HYYMD9       = HADD(TRANSDT, 'NS', 2, 12, 'HYYMD9');
 O_HCNVRT/A26        = HCNVRT(TRANSDT, '(H23)', 23, 'A26');
 O_HDIFF/D12.2       = HDIFF(O_HADD, TRANSDT, 'NS', 'D12.2');
 TRANSDATE_DATE/YYMD = HDATE(TRANSDT, 'YYMD');
 O_HDTTM/HYYMDn      = HDTTM(TRANSDATE_DATE,12, 'HYYMDn');
 O_HEXTR/HHIS9       = HEXTR(TRANSDT, 'n',12, 'HHIS9');
 O_HGETC/HYYMDn      = HGETC(12, 'HYYMDn');
 O_HINPUT/HYYMDn     = HINPUT(14, O_HCNVRT,12, 'HYYMDn');
 O_HMASK/HYYMDn      = HMASK(O_HEXTR, 'HISsmn', TRANSDT, 12, 'HYYMDn');
 O_HMIDNT/HYYMDn     = HMIDNT(TRANSDT,12, 'HYYMDn');
 O_HNAME/A10         = HNAME(TRANSDT, 'NANOSECOND', 'A10');
 O_HPART/I10         = HPART(TRANSDT, 'NANOSECOND', 'I10');
 O_HSETPT/HYYMDn     = HSETPT(TRANSDT, 'NS', 28, 12,'HYYMDn');
 O_HTIME/P20.2C      = HTIME(12,TRANSDT, 'D16');
```

これらの DEFINE フィールドのレポートでの出力は、次のとおりです。

```
TRANSDT   1999/02/05 02:05:25.123456789
O_HSsmn   25.123456789
O_HHIS2   02:05:25.12
O_HYYMDn  1999/02/05 02:05:25.123456789
O_HYYMD1  1999/02/05 02:05:25.1
O_HADD    1999/02/05 02:05:25.123456791
O_HCNVRT  19990205020525123456789
O_HDIFF                            2.00
O_HDTTM   1999/02/05 00:00:00.000000000
O_HEXTR   00:00:00.000000789
O_HGETC   2008/06/25 10:26:52.343644000
O_HINPUT  1999/02/05 02:05:25.000000000
O_HMASK   1999/02/05 00:00:00.000456789
O_HMIDNT  1999/02/05 00:00:00.000000000
O_HNAME   123456789
O_HPART                       123456789
O_HSETPT  1999/02/05 02:05:25.000000028
O_HTIME            7,525,123,456,789.00
```

注意

- O_HSsmn フィールドには、TRANSDT の秒 (S)、ミリ秒 (s)、マイクロ秒 (m)、ナノ秒 (n) が表示されます。
- O_HHIS2 フィールドには、TRANSDT の時間 (H)、分 (I)、秒 (S) が表示されます。秒は 2 桁で表示されます (2)。
- O_HYYMDn フィールドには、TRANSDT の YYMD フォーマットの日付とナノ秒 (n) までの時間が表示されます。
- O_HYYMD1 フィールドには、TRANSDT の YYMD フォーマットの日付と 1 桁の秒までの時間が表示されます。
- O_HADD フィールドは、HADD 関数を呼び出し、TRANSDT の日付時間値に 2 つのナノ秒を加算して作成されます。
- O_HCNVRT フィールドは、HCNVRT 関数を呼び出し、TRANSDT の日付時間値を文字フォーマットに変換して作成されます。
- O_HDIFF フィールドは、HDIFF 関数を呼び出し、O_HADD の日付時間値から TRANSDT の日付時間値を減算して作成されます。
- O_HDTTM フィールドは、HDTTM 関数を呼び出して TRANSDATE_DATE から日付を取り出し、時間コンポーネントを「0 (ゼロ)」に設定して作成されます。
- O_HEXTR フィールドは、HEXTR 関数を呼び出して TRANSDT からナノ秒 (9 桁のうち下 3 桁) を抽出し、残りのコンポーネントを 0 (ゼロ) に設定して作成されます。

- HGETC フィールドは、HGETC 関数を呼び出して現在の日付時間を取り出し、秒コンポーネントに 9 桁で表示して作成されます。オペレーティングシステムの種類によっては、9 桁すべてを返すとは限らないため、その場合、返されなかった桁は「0」と表示されます。
- O_HINPUT フィールドは、HINPUT 関数を呼び出して O_HCNVRT フィールドに保存された日付時間文字列を日付時間値に変換し、ナノ秒まで表示して作成されます。
- O_HMASK フィールドは、HMASK 関数を呼び出して O_HEXTR から時間、分、秒、ミリ秒、ナノ秒を抽出し、残りのコンポーネントを TRANSDT から抽出して作成されます。
- O_HMIDNT フィールドは、HMIDNT 関数を呼び出して TRANSDT から日付を抽出し、時間を午前零時に設定して作成されます。
- O_HNAME フィールドは、HNAME 関数を呼び出し、TRANSDT から文字フォーマットのナノ秒コンポーネントを抽出して作成されます。
- O_HPART フィールドは、HPART 関数を呼び出し、TRANSDT から文字フォーマットのナノ秒コンポーネントを抽出して作成されます。
- O_HPART フィールドは、HSETPT 関数を呼び出し、ナノ秒コンポーネントの値を 28 に設定して作成されます。
- O_HTIME フィールドは、HTIME 関数を呼び出し、TRANSDT の時間部分をナノ秒の数値に変換して作成されます。

参照　ナノ秒日付時間フォーマットコンポーネント使用上の注意

- 日付フィールドでは、ACTUAL フォーマットの最大値は H12 です。USAGE フォーマットの最大値は H23 です。
- 日付時間関数 HADD、HDIFF、HNAME、HPART、HSETPT は、引数としてコンポーネント名をとることができます。これらの日付時間関数で使用されるナノ秒のコンポーネント名は「nanosecond」で、「ns」という省略形を使用することもできます。さらに、HEXTR 関数と HMASK 関数には、9 桁のうち下 3 桁を表す「n」という新しいコンポーネントが追加されています。
- 次の関数には、日付時間フォーマットの精度によって長さが決定される引数があります。
  - HADD、HDIFF、HINPUT、HMIDNT、HSETPT、HTIME は長さの引数をとります。長さは 8、10、または 12 で、秒の値が 6 桁を超える場合は、必ず 12 を指定します。
  - HDTTM と HGETC は日付時間値の長さ引数をとり、8、10、または 12 を指定することが可能ですが、秒の値が 6 桁を超える場合は、必ず 12 を指定します。

- HCNVRT は、変換される日付時間フィールドのフォーマットと長さを指定する引数を 1 つずつとります。これらの引数の長さの上限は 23 です。出力フォーマットには、返される文字すべてを格納するために十分な長さを指定する必要があります。
- 最初の引数が 12 である場合、HTIME は日付時間値の時間部分をナノ秒に変換します。

マスターファイルの DATEPATTERN

データソースには、日付の値が文字フォーマットで格納され、特別な規格が存在せず、年、四半期、月などの構成要素の任意の組み合わせ、および任意の区切り文字を使用するものがあります。レポートにソートが設定されている場合、このようなデータはアルファベット順にソートされ、結果は実務上の意味を持ちません。データフィールドのソート、集計、レポート実行を適切に行うため、Db2 Web Query では、「DATEPATTERN」というマスターファイル属性で指定する変換パターンを使用して、文字フォーマットの日付を標準 Db2 Web Query 日付フォーマットに変換することができます。

パターンの各要素は、実際に入力される特定の文字、または日付構成要素を表す変数です。マスターファイルの USAGE 属性を編集して、日付構成要素の日付パターンを記述する必要があります。DATEPATTERN 文字列の最大長は 64 バイトです。

日付パターン変数の指定

有効な日付構成要素 (変数) は、年、四半期、月、日、曜日です。日付パターン内の変数は、大括弧 ([ ]) で囲みます。これらの大括弧は、入力または出力の一部ではありません。データに大括弧が含まれる場合は、日付パターン内でエスケープ文字を使用して、データの大括弧を変数を囲む大括弧と区別する必要があります。

構文　日付パターンに年を指定

`[YYYY]`
4 桁の年を指定します。

`[YY]`
2 桁の年を指定します。

`[yy]`
0 (ゼロ) を非表示にする 2 桁の年を指定します (例、2008 年の場合は 8)。

`[by]`
ブランクでパディングされた 2 桁の年を指定します。

構文　日付パターンに月を表す数値を指定

`[MM]`
2 桁の月を表す数値を指定します。

[mm]
0 (ゼロ) を非表示にする月を表す数値を指定します。

[bm]
ブランクでパディングされた月を表す数値を指定します。

構文　日付パターンに月名を指定

[MON]
3 バイトの月名を大文字で指定します。

[mon]
3 バイトの月名を小文字で指定します。

[Mon]
3 バイトの月名を先頭大文字で指定します。

[MONTH]
月の完全名を大文字で指定します。

[か月]
月の完全名を小文字で指定します。

[月]
月の完全名を先頭大文字で指定します。

構文　日付パターンに日付を指定

[DD]
2 桁の月単位の日付を指定します。

[dd]
0 (ゼロ) を非表示にする月単位の日付を指定します。

[bd]
ブランクでパディングされた月単位の日付を指定します。

構文　日付パターンにユリウス暦の日付を指定

[DDD]
3 桁の年単位の日付を指定します。

[ddd]
0 (ゼロ) を非表示にする年単位の日付を指定します。

[bdd]
ブランクでパディングされた年単位の日付を指定します。

構文　日付パターンに曜日を指定

`[WD]`
1 桁の曜日を指定します。

`[DAY]`
3 バイトの曜日を大文字で指定します。

`[day]`
3 バイトの曜日を小文字で指定します。

`[日]`
3 バイトの曜日を先頭大文字で指定します。

`[WDAY]`
曜日の完全名を大文字で指定します。

`[wday]`
曜日の完全名を小文字で指定します。

`[Wday]`
曜日の完全名を先頭大文字で指定します。

曜日の場合、WEEKFIRST の設定で、週の開始日を定義します。

構文　日付パターンに四半期を指定

`[Q]`
1 桁の四半期番号を指定します (1、2、3、4 のいずれか)。

Q2 や Q02 などの文字列の場合は、[Q] の前に定数を使用します (例、Q0[Q])。

日付パターン定数の指定

変数の間に、任意の定数値を挿入することができます。

通常は変数の一部として解釈される文字を挿入する場合は、円記号 (¥) を使用します。以下はその例です。

- 定数値として左大括弧 ([) を指定するには、¥[を使用します。
- 定数値として円記号 (¥) を指定するには、¥¥ を使用します。

一重引用符 (') の場合は、連続する 2 つの一重引用符 ('') を使用します。

例　日付パターンサンプル

データソースの日付が「CY 2001 Q1」の形式の場合、DATEPATTERN 属性は次のとおりです。

```
DATEPATTERN = 'CY [YYYY] Q[Q]'
```

データソースの日付が「Jan 31, 01」の形式の場合、DATEPATTERN 属性は次のとおりです。

```
DATEPATTERN = '[Mon] [DD], [YY]'
```

データソースの日付が「APR-06」の形式の場合、DATEPATTERN 属性は次のとおりです。

```
DATEPATTERN = '[MON]-[YY]'
```

データソースの日付が「APR - 06」の形式の場合、DATEPATTERN 属性は次のとおりです。

```
DATEPATTERN = '[MON] - [YY]'
```

データソースの日付が「APR '06」の形式の場合、DATEPATTERN 属性は次のとおりです。

```
DATEPATTERN = '[MON] ''[YY]'
```

データソースの日付が「APR [06]」の形式の場合、DATEPATTERN 属性は次のとおりです。

```
DATEPATTERN = '[MON] ¥[[YY]¥]' (or '[MON] ¥[[YY]]'
```

右大括弧 (]) に、エスケープ文字は必要ありません。

例　文字の日付によるソート

次の例で、 .ftmDATE1 は、以下のデータが格納されたシーケンシャルファイルです。

```
June 1, '02
June 2, '02
June 3, '02
June 10, '02
June 11, '02
June 12, '02
June 20, '02
June 21, '02
June 22, '02
June 1, '03
June 2, '03
June 3, '03
June 10, '03
June 11, '03
June 12, '03
June 20, '03
June 21, '03
June 22, '03
June 1, '04
June 2, '04
June 3, '04
June 4, '04
June 10, '04
June 11, '04
June 12, '04
June 20, '04
June 21, '04
June 22, '04
```

DATE1 マスターファイルの DATE1 フィールドには、USAGE フォーマットおよび ACTUAL フォーマットが設定されます。これらは両者とも文字 (A18) フォーマット です。

```
FILENAME=DATE1, SUFFIX=FIX,
  DATASET = c:¥tst¥date1.ftm, $
  SEGMENT=FILE1, SEGTYPE=S0, $
    FIELDNAME=DATE1, ALIAS=E01, USAGE=A18, ACTUAL=A18, $
```

次のリクエストは、DATE1 FIELD でソートします。

```
TABLE FILE DATE1
PRINT DATE1 NOPRINT
BY DATE1
ON TABLE SET PAGE NOPAGE
END
```

出力で、文字の日付は、日付順ではなくアルファベット順にソートされています。

```
DATE1
-----
June 1, '02
June 1, '03
June 1, '04
June 10, '02
June 10, '03
June 10, '04
June 11, '02
June 11, '03
June 11, '04
June 12, '02
June 12, '03
June 12, '04
June 2, '02
June 2, '03
June 2, '04
June 20, '02
June 20, '03
June 20, '04
June 21, '02
June 21, '03
June 21, '04
June 22, '02
June 22, '03
June 22, '04
June 3, '02
June 3, '03
June 3, '04
June 4, '04
```

日付を正しくソートするため、マスターファイルに DATEPATTERN 属性を追加して、Db2 Web Query によって日付が Db2 Web Query の日付フィールドに変換されるようにします。また、USAGE フォーマットを編集し、そのフィールドを Db2 Web Query の日付フォーマットにする必要もあります。適切なパターンを構成するため、格納されている日付のすべての要素を記述する必要があります。文字の日付には、次の変数と定数があります。

- 変数 - 先頭大文字の月の完全名 [Month]。
- 定数 - ブランク。
- 変数 - ゼロを非表示にした月を表す数値 [dd]。
- 定数 - カンマ (,)、ブランク、アポストロフィ (') (パターンでは 2 つのアポストロフィとして記述) の順に入力。
- 変数 - 2 桁の年 [YY]。

編集後のマスターファイルは、次のようになります。DEFCENT 属性を追加して、2 桁の年を 4 桁の年に変換します。

```
FILENAME=DATE1, SUFFIX=FIX,
  DATASET = c:¥tst¥date1.ftm, $
  SEGMENT=FILE1, SEGTYPE=S0, $
    FIELDNAME=DATE1, ALIAS=E01, USAGE=A18, ACTUAL=A18, DEFCENT=20,
    DATEPATTERN = '[Month] [dd], ''[YY]', $
```

同一のリクエストを発行すると、出力は次のようになります。DATE1 は、USAGE で指定した MtrDYY フォーマットの Db2 Web Query の日付に変換されています。

```
DATE1
-----
June  1, 2002
June  2, 2002
June  3, 2002
June 10, 2002
June 11, 2002
June 12, 2002
June 20, 2002
June 21, 2002
June 22, 2002
June  1, 2003
June  2, 2003
June  3, 2003
June 10, 2003
June 11, 2003
June 12, 2003
June 20, 2003
June 21, 2003
June 22, 2003
June  1, 2004
June  2, 2004
June  3, 2004
June  4, 2004
June 10, 2004
June 11, 2004
June 12, 2004
June 20, 2004
June 21, 2004
June 22, 2004
```

DMY、MDY、YMD - 2 つの日付の差を計算

DMY、MDY、および YMD 関数は、整数、文字、またはパック 10 進数フォーマットの 2 つの日付の差を計算します。

構文　2 つの日付の差を計算

```
function(begin, end)
```

説明

`function`
次のいずれかです。

`DMY` - 日月年フォーマットの 2 つの日付の差を計算します。

`MDY` - 月日年フォーマットの 2 つの日付の差を計算します。

`YMD` - 年月日フォーマットの 2 つの日付の差を計算します。

`begin`
整数、パック 10 進数、または文字

日付表示オプションを含む I、P、A フォーマットです。

開始日、または日付を含むフィールド名のいずれかを指定します。

`end`
整数、パック 10 進数、または文字

日付表示オプションを含む I、P、A フォーマットです。

終了日、または日付を含むフィールド名のいずれかを指定します。

例　2 つの日付の日数差を計算

YMD 関数は、HIRE_DATE から DAT_INC までの日数を計算します。

```
COMPUTE DIFF/I4 = YMD(HIRE_DATE, FST.DAT_INC);
```

DOWK および DOWKL - 曜日を検索

DOWK および DOWKL 関数は、対応する曜日を検索します。DOWK は曜日の 3 バイトの省略形を返し、DOWKL は完全な曜日名を表示します。

構文　曜日を検索

```
{DOWK|DOWKL}(indate, 'outfield')
```

説明

`indate`
整数 (I6YMD または I8YMD)

年月日フォーマットの入力日付です。日付が無効の場合、関数はブランクを返します。日付が 2 桁の年を指定し、DEFCENT および YRTHRESH の値が設定されていない場合は、20 世紀と見なされます。

`outfield`

DOWK - 文字

DOWKL - 文字

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例　曜日を検索

DOWK 関数は、HIRE_DATE フィールドの値に対応する曜日を特定し、結果を DATED に格納します。

```
COMPUTE DATED/A3 = DOWK(HIRE_DATE, 'A3');
```

DT 関数 - 整数を日付に変換

DT 関数は、1899 年 12 月 31 日との日数の差を表す整数を対応する日付に変換します。これらは、日数に変換された日付に対して演算を行う場合に役立ちます。DT 関数は、結果を日付に変換します。

DT 関数は 6 つあります。それぞれの関数は、数値を異なるフォーマットの日付に変換します。

注意：USERFNS が LOCAL に設定されると、DT 関数は 6 桁の日付のみを表示します。

構文　整数を日付に変換

```
function(number, 'outfield')
```

説明

`function`

次のいずれかです。

`DTDMY` - 数値を日月年の日付に変換します。

`DTDYM` - 数値を日年月の日付に変換します。

`DTMDY` - 数値を月日年の日付に変換します。

`DTMYD` - 数値を月年日の日付に変換します。

`DTYDM` - 数値を年日月の日付に変換します。

`DTYMD` - 数値を年月日の日付に変換します。

`number`
整数

1899 年 12 月 31 日以降の日数です。数字は整数に切り捨てられます。

`outfield`
整数

I6xxx です。xxx は、上記リストの DTxxx 関数に対応したフォーマットを表します。

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。出力フォーマットは、使用されている関数により異なります。

例　整数を日付に変換

DTMDY 関数は、NEWF フィールド (DAYMD によりに数に変換されたフィールド) を対応する日付に変換し、結果を NEW_HIRE_DATE に格納します。

```
COMPUTE NEWF/I8 WITH EMP_ID = DAYMD(HIRE_DATE, NEWF); AND
COMPUTE NEW_HIRE_DATE/I8MDYY WITH EMP_ID = DTMDY(NEWF, NEW_HIRE_DATE);
```

FIYR - 会計年度の取得

FIYR 関数は、会計年度の開始日および会計年度の算定方式に基づいて、特定のカレンダー日付に対応する会計年度を返します。

構文　会計年度の取得

```
FIYR(inputdate, lowcomponent, startmonth, startday, yrnumbering, output)
```

説明

`inputdate`

日付

会計年度を取得する日付です。この日付には、基準日からのオフセットとして格納される標準の日付を指定する必要があります。

会計年度が月の初日以外から始まる場合、日付の要素は Y(Y)、M、D または Y(Y)、JUL で構成する必要があります (JUL は YJUL と同等)。会計年度が月の初日から始まる場合、日付の要素は Y(Y)、M または Y(Y)、Q で構成する必要があります。

`lowcomponent`

文字

次のいずれかです。

- D - 日付に D または JUL 構成要素が含まれている場合。
- M - 日付に M 構成要素は含まれているが、D 構成要素が含まれていない場合。
- Q - 日付に Q 構成要素が含まれている場合。

`startmonth`

数値

1 から 12 までの数字を使用して、会計年度の開始月を表します (例、1 は 1 月、12 は 12 月)。下位構成要素が Q の場合、開始月には 1、4、7、10 のいずれかを指定する必要があります。

`startday`

数値

開始月の開始日です。通常は 1 を指定します。下位構成要素が M または Q の場合、1 を指定する必要があります。

`yrnumbering`

文字

有効な値には、次のものがあります。

FYE - 会計年度の終了日を基準にする方式を使用します。 会計年度値は、会計年度の最終日のカレンダー年になります。たとえば、会計年度が 2008 年 10 月 1 日から始まる場合、「2008 年 11 月 1 日」という日付は会計年度 2009 年の第 1 四半期に分類されます。これは、この日付が 2009 年 9 月 30 日に終了する会計年度の範囲内にあるためです。

FYS - 会計年度の開始日を基準にする方式を使用します。 この会計年度値は、会計年度の開始日のカレンダー年になります。たとえば、会計年度が 2008 年 4 月 6 日から始まる場合、「2008 年 7 月 6 日」という日付は会計年度 2008 年の第 2 四半期に分類されます。これは、この日付が 2008 年 4 月 6 日に始まる会計年度の範囲内にあるためです。

`output`

I、Y、YY

結果は、整数フォーマット、あるいは Y または YY になります。この関数は、年の値を返します。エラーが発生した場合は、0 (ゼロ) が返されます。

注意：会計年度の開始日として 2 月 29 日を使用することはできません。

例　会計年度の取得

次の例は、特定の会計期間 (PERIOD フィールド、YYM フォーマット) に対応する会計年度を取得し、サポートされているフォーマット (Y、YY、I4) で値を返します。

```
FISCALYY/YY=FIYR(PERIOD,'M', 4,1,'FYE',FISCALYY);
FISCALY/Y=FIYR(PERIOD,'M', 4,1,'FYE',FISCALY);
FISCALI/I4=FIYR(PERIOD,'M', 4,1,'FYE',FISCALI);
END
```

出力結果では、2002 年 4 月 (2002/04) は会計年度 2003 年に分類されています。これは、開始月が 4 月であり、会計年度の算定方式として FYE が使用されているためです。

```
Ledger
Account  PERIOD   FISCALYY  FISCALY  FISCALI
-------  ------   --------  -------  -------
1000     2002/01  2002      02          2002
         2002/02  2002      02          2002
         2002/03  2002      02          2002
         2002/04  2003      03          2003
         2002/05  2003      03          2003
         2002/06  2003      03          2003
2000     2002/01  2002      02          2002
         2002/02  2002      02          2002
         2002/03  2002      02          2002
         2002/04  2003      03          2003
         2002/05  2003      03          2003
         2002/06  2003      03          2003
```

FIQTR - 会計四半期の取得

FIQTR 関数は、会計年度の開始日および会計年度の算定方式に基づいて、特定のカレンダー日付に対応する会計四半期を返します。

構文　会計四半期の取得

```
FIQTR(inputdate, lowcomponent, startmonth, startday, yrnumbering, output)
```

説明

`inputdate`

日付

会計年度を取得する日付です。この日付には、基準日からのオフセットとして格納される標準の日付を指定する必要があります。

会計年度が月の初日以外から始まる場合、日付の要素は Y(Y)、M、D または Y(Y)、JUL で構成する必要があります (JUL は YJUL と同等)。会計年度が月の初日から始まる場合、日付の要素は Y(Y)、M または Y(Y)、Q で構成する必要があります。

`lowcomponent`

文字

次のいずれかです。

- `D` - 日付に D または JUL 構成要素が含まれている場合。
- `M` - 日付に M 構成要素は含まれているが、D 構成要素が含まれていない場合。
- `Q` - 日付に Q 構成要素が含まれている場合。

`startmonth`

数値

1 から 12 までの数字を使用して、会計年度の開始月を表します (例、1 は 1 月、12 は 12 月)。下位構成要素が Q の場合、開始月には 1、4、7、10 のいずれかを指定する必要があります。

`startday`

数値

開始月の開始日です。通常は 1 を指定します。下位構成要素が M または Q の場合、1 を指定する必要があります。

`yrnumbering`

文字

有効な値には、次のものがあります。

FYE - 会計年度の終了日を基準にする方式を使用します。 会計年度値は、会計年度の最終日のカレンダー年になります。たとえば、会計年度が 2008 年 10 月 1 日から始まる場合、「2008 年 11 月 1 日」という日付は会計年度 2009 年の第 1 四半期に分類されます。これは、この日付が 2009 年 9 月 30 日に終了する会計年度の範囲内にあるためです。

FYS - 会計年度の開始日を基準にする方式を使用します。 この会計年度値は、会計年度の開始日のカレンダー年になります。たとえば、会計年度が 2008 年 4 月 6 日から始まる場合、「2008 年 7 月 6 日」という日付は会計年度 2008 年の第 2 四半期に分類されます。これは、この日付が 2008 年 4 月 6 日に始まる会計年度の範囲内にあるためです。

`output`

I または Q

結果は、整数フォーマット、または Q になります。この関数は 1 から 4 までの値を返します。エラーが発生した場合は、0 (ゼロ) が返されます。

注意：会計年度の開始日として 2 月 29 日を使用することはできません。

例　会計四半期の取得

次の例は、特定の従業員の開始日 (START_DATE フィールド、YYMD フォーマット) に対応する会計四半期を取得し、サポートされているフォーマット (Q および I1) で値を返します。

```
FISCALQ/Q=FIQTR(START_DATE,'D',10,1,'FYE',FISCALQ);
FISCALI/I1=FIQTR(START_DATE,'D',10,1,'FYE',FISCALI);
```

出力結果では、1998 年 11 月 12 日 (1998/11/12) は第 1 四半期 (Q1) に分類されています。これは、開始月が 10 月であるためです。

```
Last            First         Starting
Name            Name          Date        FISCALQ  FISCALI
----            -----         --------    -------  -------
CHARNEY         ROSS          1998/09/12  Q4             4
CHIEN           CHRISTINE     1997/10/01  Q1             1
CLEVELAND       PHILIP        1996/07/30  Q4             4
CLINE           STEPHEN       1998/11/12  Q1             1
COHEN           DANIEL        1997/10/05  Q1             1
CORRIVEAU       RAYMOND       1997/12/05  Q1             1
COSSMAN         MARK          1996/12/19  Q1             1
CRONIN          CHRIS         1996/12/03  Q1             1
CROWDER         WESLEY        1996/09/17  Q4             4
CULLEN          DENNIS        1995/09/05  Q4             4
CUMMINGS        JAMES         1993/07/11  Q4             4
CUTLIP          GREGG         1997/03/26  Q2             2
```

FIYYQ - カレンダー日付を会計日付に変換

FIYYQ 関数は、指定したカレンダー日付に対応する会計日付を返します。この日付には、会計年度および会計四半期が含まれます。返された会計日付は、会計年度の開始日および会計年度の算定方式に基づいています。

構文　カレンダー日付を会計日付に変換

```
FIYYQ(inputdate, lowcomponent, startmonth, startday, yrnumbering, output)
```

説明

`inputdate`

日付

会計年度を取得する日付です。この日付には、基準日からのオフセットとして格納される標準の日付を指定する必要があります。

会計年度が月の初日以外から始まる場合、日付の要素は Y(Y)、M、D または Y(Y)、JUL で構成する必要があります (JUL は YJUL と同等)。会計年度が月の初日から始まる場合、日付の要素は Y(Y)、M または Y(Y)、Q で構成する必要があります。

`lowcomponent`

文字

次のいずれかです。

- `D` - 日付に D または JUL 構成要素が含まれている場合。
- `M` - 日付に M 構成要素は含まれているが、D 構成要素が含まれていない場合。
- `Q` - 日付に Q 構成要素が含まれている場合。

`startmonth`

数値

1 から 12 までの数字を使用して、会計年度の開始月を表します (例、1 は 1 月、12 は 12 月)。下位構成要素が Q の場合、開始月には 1、4、7、10 のいずれかを指定する必要があります。

`startday`

数値

開始月の開始日です。通常は 1 を指定します。下位構成要素が M または Q の場合、1 を指定する必要があります。

`yrnumbering`

文字

有効な値には、次のものがあります。

FYE - 会計年度の終了日を基準にする方式を使用します。 会計年度値は、会計年度の最終日のカレンダー年になります。たとえば、会計年度が 2008 年 10 月 1 日から始まる場合、「2008 年 11 月 1 日」という日付は会計年度 2009 年の第 1 四半期に分類されます。これは、この日付が 2009 年 9 月 30 日に終了する会計年度の範囲内にあるためです。

FYS - 会計年度の開始日を基準にする方式を使用します。 この会計年度値は、会計年度の開始日のカレンダー年になります。たとえば、会計年度が 2008 年 4 月 6 日から始まる場合、「2008 年 7 月 6 日」という日付は会計年度 2008 年の第 2 四半期に分類されます。これは、この日付が 2008 年 4 月 6 日に始まる会計年度の範囲内にあるためです。

`output`

Y[Y]Q または QY[Y]

エラーが発生した場合は、0 (ゼロ) が返されます。

注意：会計年度の開始日として 2 月 29 日を使用することはできません。

例　カレンダー日付を会計日付に変換

次の例は、各従業員の開始日 (START_DATE フィールド、YYMD フォーマット) を、年および四半期構成要素を含む会計日付に変換し、サポートされているすべてのフォーマット (YQ、YYQ、QY、QYY) で値を返します。

```
FISYQ/YQ=FIYYQ(START_DATE,'D',10,1,'FYE',FISYQ);
FISYYQ/YYQ=FIYYQ(START_DATE,'D',10,1,'FYE',FISYYQ);
FISQY/QY=FIYYQ(START_DATE,'D',10,1,'FYE',FISQY);
FISQYY/QYY=FIYYQ(START_DATE,'D',10,1,'FYE',FISQYY);
```

出力結果では、1998 年 11 月 12 日 (1998/11/12) が第 1 四半期 (Q1) に変換されています。これは、開始月が 10 月であり、会計年度の算定方式として FYE が使用されているためです。

```
Last            First         Starting
Name            Name          Date        FISYQ  FISYYQ   FISQY  FISQYY
----            -----         --------    -----  ------   -----  ------
CHARNEY         ROSS          1998/09/12  98 Q4  1998 Q4  Q4 98  Q4 1998
CHIEN           CHRISTINE     1997/10/01  98 Q1  1998 Q1  Q1 98  Q1 1998
CLEVELAND       PHILIP        1996/07/30  96 Q4  1996 Q4  Q4 96  Q4 1996
CLINE           STEPHEN       1998/11/12  99 Q1  1999 Q1  Q1 99  Q1 1999
COHEN           DANIEL        1997/10/05  98 Q1  1998 Q1  Q1 98  Q1 1998
CORRIVEAU       RAYMOND       1997/12/05  98 Q1  1998 Q1  Q1 98  Q1 1998
COSSMAN         MARK          1996/12/19  97 Q1  1997 Q1  Q1 97  Q1 1997
CRONIN          CHRIS         1996/12/03  97 Q1  1997 Q1  Q1 97  Q1 1997
CROWDER         WESLEY        1996/09/17  96 Q4  1996 Q4  Q4 96  Q4 1996
CULLEN          DENNIS        1995/09/05  95 Q4  1995 Q4  Q4 95  Q4 1995
CUMMINGS        JAMES         1993/07/11  93 Q4  1993 Q4  Q4 93  Q4 1993
CUTLIP          GREGG         1997/03/26  97 Q2  1997 Q2  Q2 97  Q2 1997
```

GREGDT - ユリウス暦から太陽暦フォーマットに変換

GREGDT 関数は、日付をユリウス暦フォーマットから太陽暦フォーマット (年月日) に変換します。

ユリウス暦フォーマットの日付は、5 桁または 7 桁の数値です。先頭の 2 桁または 4 桁は年、末尾の 3 桁は 1 月 1 日から数えた日数です。たとえば、ユリウス暦フォーマットの 1999 年 1 月 1 日は、99001 と 1999001 のいずれかです。

参照　GREGDT の DATEFNS 設定

GREGDT 関数は、ユリウス暦の日付を YMD または YYMD フォーマットに変換します。必要に応じて、DEFCENT および YRTHRESH パラメータ設定を使用して世紀を決定します。GREGDT 関数は、日付を次のように返します。

DATEFNS 設定	I6 または I7 フォーマット	I8 フォーマット、またはそれ以上
ON	YMD	YYMD
OFF	YMD	YMD

構文　ユリウス暦から太陽暦フォーマットに変換

```
GREGDT(indate, 'outfield')
```

説明

`indate`
整数 (I5 または I7)

ユリウス暦の日付です。変換前に整数に切り捨てられます。切り捨て後、各値は 5 桁または 7 桁の数値である必要があります。日付が有効でない場合、この関数は 0 (ゼロ) を返します。

`outfield`
整数 (I6、I8、I6YMD、I8YYMD)

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例　ユリウス暦から太陽暦フォーマットに変換

GREGDT 関数は、JULIAN フィールドを YYMD (太陽暦) フォーマットに変換します。

```
COMPUTE GREG_DATE/I8 = GREGDT(JULIAN, 'I8');
```

HADD - 日付時間値を増加

HADD 関数は、指定した単位数分、日付時間値を増加します。

構文　日付時間値を増加

```
HADD(value, 'component', increment, length, 'outfield')
```

説明

`value`
日付時間

増加される日付時間値です。値を含む日付時間フィールドの名前、または値を返す式を指定することもできます。

`component`
文字

増加される構成要素名です。文字列は一重引用符 (') で囲みます。

注意：WEEKDAY は、HADD で有効な構成要素ではありません。

`increment`
整数

構成要素の増加に使用する単位数です。値を含む数値フィールド名、または値を返す式を指定することもできます。

`length`
整数

返された日付時間値の長さです。有効な値には、次のものがあります。

`8` - ミリ秒を含む時間値です。

`10` - マイクロ秒を含む時間値です。

`outfield`
日付時間

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例　日付時間フィールドの月構成要素を追加

HADD 関数は、TRANSDATE の各値に 2 か月加算し、結果を ADD_MONTH に格納します。必要に応じて、日の部分が調整されます。

```
COMPUTE ADD_MONTH/HYYMDS = HADD(TRANSDATE, 'MONTH', 2, 8, 'HYYMDS');
```

HCNVRT - 日付時間値を文字フォーマットに変換

HCNVRT 関数は、日付時間値を演算子 EDIT、CONTAINS、LIKE などで使用する文字フォーマットに変換します。

構文　日付時間値を文字フォーマットに変換

```
HCNVRT(value, '(fmt)', length, 'outfield')
```

説明

`value`
日付時間

変換される日付時間値です。値を含む日付時間フィールド名、または値を返す式を指定することもできます。

`fmt`
文字

日付時間フィールドのフォーマットです。フォーマットは一重引用符 (') と括弧で囲みます。

`length`
整数

返される文字フィールド内のバイト数です。実際の値、値を含む文字フィールド名、値を返す式のいずれかを指定します。*length* の値が文字フィールドを表示するために必要なバイト数よりも小さい場合、関数はブランクを返します。

`outfield`
文字

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例　日付時間フィールドを文字フォーマットに変換

HCNVRT 関数は、TRANSDATE フィールドを文字フォーマットに変換します。1 つ目の関数には、フィールドの日付時間表示オプションは含まれません。このオプションは 2 つ目の関数で指定します。入力フィールドの秒の表示も指定します。

```
COMPUTE ALPHA_DATE_TIME1/A20 = HCNVRT(TRANSDATE, '(H17)', 17, 'A20'); AND
COMPUTE ALPHA_DATE_TIME2/A20 = HCNVRT(TRANSDATE, '(HYYMDS)', 20, 'A20');
```

HDATE - 日付時間値の日付部分を日付フォーマットに変換

HDATE 関数は、日付時間値の日付の部分を日付フォーマット YYMD に変換します。この結果は、別の日付フォーマットに変換することができます。

構文　日付時間値の日付部分を日付フォーマットに変換

```
HDATE(value, 'YYMD')
```

説明

`value`
日付時間

変換される日付時間値です。値を含む日付時間フィールド名、または値を返す式を指定することもできます。

`YYMD`
日

出力日付フォーマットです。値は常に YYMD です。YYMD は定数値で、この構文では変更できません。このフォーマットは、DEFINE および COMPUTE を実行して変更することができます。

例　日付時間フィールドの日付部分を日付フォーマットに変換

HDATE 関数は、TRANSDATE フィールドの日付部分を日付フォーマット YYMD に変換します。

```
COMPUTE TRANSDATE_DATE/YYMD = HDATE(TRANSDATE, 'YYMD');
```

HDIFF - 2 つの日付時間値の差を計算

HDIFF 関数は、指定した日付単位で 2 つの日付時間値の差を計算します。

構文　2 つの日付時間値の差を計算

```
HDIFF(value1, value2, 'component', 'outfield')
```

説明

`value1`
日付時間

終了日付時間値、値を含む日付時間フィールド名、値を返す式のいずれかを指定します。

`value2`
日付時間

開始日付時間値、値を含む日付時間フィールド名、値を返す式のいずれかを指定します。

`component`
文字

計算に使用される構成要素名です。文字は一重引用符 (') で囲みます。構成要素が週の場合、計算に WEEKFIRST パラメータの設定が使用されます。

`outfield`
単精度浮動小数点数または 10 進数

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例　2 つの日付時間フィールドの日数差を計算

HDIFF 関数は、TRANSDATE と ADD_MONTH フィールドの日数の差を計算し、結果を D12.2 フォーマットで DIFF_PAYS に格納します。

```
COMPUTE ADD_MONTH/HYYMDS = HADD(TRANSDATE, 'MONTH', 2, 8, 'HYYMDS'); AND
COMPUTE DIFF_DAYS/D12.2 = HDIFF(ADD_MONTH, TRANSDATE, 'DAY', 'D12.2');
```

HDTTM - 日付値を日付時間値に変換

HDTTM 関数は、日付値を日付時間値に変換します。時間部分は午前零時に設定されます。

構文　日付値を日付時間値に変換

```
HDTTM(date, length, 'outfield')
```

説明

`date`
日付

変換される日付値、値を含む日付フィールド名、値を返す式のいずれかを指定します。

`length`
整数

返された日付時間値の長さです。有効な値には、次のものがあります。

`8` - ミリ秒を含む時間値です。

`10` - マイクロ秒を含む時間値です。

`outfield`
日付時間

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例　日付フィールドを日付時間フィールドに変換

HDTTM 関数は、日付フィールド TRANSDATE_DATE を日付時間フィールドに変換します。

```
COMPUTE TRANSDATE_DATE/YYMD = HDATE(TRANSDATE, 'YYMD'); AND
COMPUTE DT2/HYYMDIA = HDTTM(TRANSDATE_DATE, 8, 'HYYMDIA');
```

HGETC - 現在の日付および時間を日付時間フィールドに格納

HGETC 関数は、現在の日付と時間を日付時間フィールドに格納します。オペレーティングシステム環境で、ミリ秒やマイクロ秒値が利用できない場合、これらの構成要素には、0 (ゼロ) が取得されます。

構文　現在の日付および時間を日付時間フィールドに格納

```
HGETC(length, 'outfield')
```

説明

`length`
整数

返された日付時間値の長さです。有効な値には、次のものがあります。

`8` - ミリ秒を含む時間値です。

`10` - マイクロ秒を含む時間値です。

`outfield`
日付時間

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例　現在の日付および時間を日付時間フィールドに格納

HGETC 関数は、現在の日付および時間を DT2 に格納します。

```
COMPUTE DT2/HYYMDm = HGETC(10, 'HYYMDm');
```

HHMMSS - 現在の時間を取得

HHMMSS 関数は、オペレーティングシステムから現在の時間を取得します。時間は、時、分、秒をピリオド (.) で区切った 8 バイトの文字列として取得されます。

構文　現在の時間を取得

```
HHMMSS('outfield')
```

説明

`outfield`
　文字

　出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。フィールドフォーマットは A8 以上にする必要があります。

例　現在の時間を取得

HHMMSS 関数は、現在の時間を取得します。

```
COMPUTE NOWTIME/A8 = HHMMSS('A8');
```

HINPUT - 文字列を日付時間値に変換

HINPUT 関数は、文字列を日付時間値に変換します。

構文　文字列を日付時間値に変換

```
HINPUT(inputlength, 'inputstring', length, 'outfield')
```

説明

`inputlength`
　整数

　変換される文字列の長さです。実際の値、値を含む文字フィールド名、値を返す式のいずれかを指定します。

`inputstring`
　文字

　変換される文字列です。文字列は一重引用符 (') で囲みます。文字列を含む文字フィールド名、または文字列を返す式を指定することもできます。この文字列には、入力値として有効な任意の日付時間の組み合わせを使用することができます。

`length`
　整数

　返された日付時間値の長さです。有効な値には、次のものがあります。

　`8` - ミリ秒を含む時間値です。

　`10` - マイクロ秒を含む時間値です。

`outfield`
日付時間

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例　文字列を日付時間値に変換

HCNVRT 関数は、TRANSDATE フィールドを文字フォーマットに変換します。その後、HINPUT 関数が文字列を日付時間値に変換します。

```
COMPUTE ALPHA_DATE_TIME/A20 = HCNVRT(TRANSDATE, '(H17)', 17, 'A20'); AND
COMPUTE DT_FROM_ALPHA/HYYMDS = HINPUT(14, ALPHA_DATE_TIME, 8, 'HYYMDS');
```

HMIDNT - 日付時間値の時間部分を午前零時に設定

HMIDNT 関数は、日付時間値の時間部分を午前零時 (デフォルト値はすべて 0 (ゼロ)) に変更します。これにより、日付フィールドを日付時間フィールドと比較することができます。

構文　日付時間値の時間部分を午前零時に設定

```
HMIDNT(value, length, 'outfield')
```

説明

`value`
日付時間

時間部分が午前零時に設定される日付時間値です。値を含む日付時間フィールド名、または値を返す式を指定することもできます。

`length`
整数

返された日付時間値の長さです。有効な値には、次のものがあります。

`8` - ミリ秒を含む時間値です。

`10` - マイクロ秒を含む時間値です。

`outfield`
日付時間

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例　時間を午前零時に設定

HMIDNT 関数は、TRANSDATE フィールドの時間部分を午前零時に、最初は 24 時間制、次に 12 時間制で設定します。

```
COMPUTE TRANSDATE_MID_24/HYYMDS  = HMIDNT(TRANSDATE, 8, 'HYYMDS'); AND
COMPUTE TRANSDATE_MID_12/HYYMDSA = HMIDNT(TRANSDATE, 8, 'HYYMDSA');
```

HNAME - 日付時間構成要素を文字フォーマットで取得

HNAME 関数は、日付時間値から指定された構成要素を文字フォーマットで抽出します。

構文　日付時間構成要素を文字フォーマットで取得

```
HNAME(value, 'component', 'outfield')
```

説明

`value`
日付時間

構成要素の抽出元の日付時間値です。値を含む日付時間フィールド名、または値を返す式を指定することもできます。

`component`
文字

取得される構成要素名です。文字列は一重引用符 (') で囲みます。

`outfield`
文字

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。フィールドフォーマットは A2 以上にする必要があります。

関数は、他のすべての構成要素を数値にのみ変換します。年は常に 4 桁であり、時間は 24 時間制と見なされます。

例　週構成要素を文字フォーマットで取得

HNAME 関数は、TRANSDATE フィールドから週を文字フォーマットで取得します。WEEKFIRST パラメータ設定の変更は、構成要素の値を変更します。

```
COMPUTE WEEK_COMPONENT/A10 = HNAME(TRANSDATE, 'WEEK', 'A10');
```

例　日構成要素を数値フォーマットで取得

HNAME 関数は、TRANSDATE フィールドから日構成要素を文字フォーマットで取得します。

```
COMPUTE DAY_COMPONENT/A2 = HNAME(TRANSDATE, 'DAY', 'A2');
```

HPART - 日付時間構成要素を数値フォーマットで取得

HPART 関数は、指定した構成要素を日付時間値から抽出し、数値フォーマットで返します。

構文　日付時間構成要素を数値フォーマットで取得

```
HPART(value, 'component', 'outfield')
```

説明

`value`
日付時間

日付時間値、値を含む日付時間フィールド名、値を返す式のいずれかを指定します。

`component`
文字

取得される構成要素名です。文字列は一重引用符 (') で囲みます。

`outfield`
整数

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例　日構成要素を数値フォーマットで取得

HPART 関数は、TRANSDATE フィールドから日構成要素を整数フォーマットで取得します。

```
COMPUTE DAY_COMPONENT/I2 = HPART(TRANSDATE, 'DAY', 'I2');
```

HSETPT - 日付時間値に構成要素を挿入

HSETPT 関数は、指定した構成部分の数値を日付時間値に挿入します。

構文　日付時間値に構成要素を挿入

```
HSETPT(dtfield, 'component', value, length, 'outfield')
```

説明

`dtfield`
日付時間

日付時間値、値を含む日付時間フィールド名、値を返す式のいずれかを指定します。

`component`
文字

挿入される構成要素名です。文字列は一重引用符 (') で囲みます。

`value`
整数

要求された構成要素に挿入される数値、値を含む数値フィールド名、値を返す式のいずれかを指定します。

`length`
整数

返された日付時間値の長さです。有効な値には、次のものがあります。

`8` - ミリ秒を含む時間値です。

`10` - マイクロ秒を含む時間値です。

`outfield`
日付時間

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例　日付時間フィールドに日構成要素を挿入

HSETPT 関数は、日の数値「28」を ADD_MONTH フィールドに挿入し、結果を INSERT_DAY に格納します。

```
COMPUTE ADD_MONTH/HYYMDS = HADD(TRANSDATE, 'MONTH', 2, 8, 'HYYMDS'); AND
COMPUTE INSERT_DAY/HYYMDS = HSETPT(ADD_MONTH, 'DAY', 28, 8, 'HYYMDS');
```

HTIME - 日付時間値の時間部分を数値に変換

HTIME 関数は、1 つ目の引数が 8 の場合、日付時間値の時間部分をミリ秒の数値に変換し、1 つ目の引数が 10 の場合、マイクロ秒の数値に変換します。マイクロ秒を含めるには、入力日付時間値は 10 バイトでなければなりません。

構文　日付時間値の時間部分を数値に変換

```
HTIME(length, value, 'outfield')
```

説明

`length`
整数

入力日付時間値の長さです。有効な値には、次のものがあります。

8 - ミリ秒を含む時間値です。

10 - マイクロ秒を含む時間値です。

value
日付時間

時間に変換される日付時間値、値を含む日付時間フィールド名、値を返す式のいずれかを指定します。

outfield
単精度浮動小数点数または 10 進数

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例　日付時間フィールドの時間部分を数値に変換

HTIME 関数は、TRANSDATE フィールドの時間部分をミリ秒数に変換します。

```
COMPUTE MILLISEC/D12.2 = HTIME(8, TRANSDATE, 'D12.2');
```

JULDAT - 太陽暦からユリウス暦フォーマットに変換

JULDAT 関数は、日付を太陽暦フォーマット (年月日) からユリウス暦フォーマット (year-number_of_the_day) に変換します。ユリウス暦フォーマットの日付は、5 桁または 7 桁の数値です。先頭の 2 桁または 4 桁は年、末尾の 3 桁は 1 月 1 日から数えた日数です。たとえば、ユリウス暦フォーマットの 1999 年 1 月 1 日は、99001 と 1999001 のいずれかです。

参照　JULDAT の DATEFNS 設定

JULDAT 関数は、太陽暦の日付を YYNNN または YYYYNNN フォーマットに変換します。必要に応じて、DEFCENT および YRTHRESH パラメータ設定を使用して世紀を決定します。

JULDAT 関数は、次のデータを返します。

DATEFNS 設定	I6 または I7 フォーマット	I8 フォーマット、またはそれ以上
ON	YYNNN	YYYYNNN
OFF	YYNNN	YYNNN

構文　太陽暦からユリウス暦フォーマットに変換

```
JULDAT(indate, 'outfield')
```

説明

`indate`
整数 (I6、I8、I6YMD、I8YYMD)

日付、または年月日フォーマット (YMD または YYMD) の日付を含むフィールド名のいずれかを指定します。

`outfield`
整数 (I5 または I7)

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例　太陽暦からユリウス暦フォーマットに変換

JULDAT は、HIRE_DATE フィールドを太陽暦フォーマットに変換します。

```
COMPUTE JULIAN/I7 = JULDAT(HIRE_DATE, 'I7');
```

TIMETOTS - 時間をタイムスタンプに変換

TIMETOTS 関数は、時間をタイムスタンプに変換します。値の日付構成要素には、現在の日付が使用されます。1 つ目の引数は、H (日付時間) フォーマットである必要があります。日付構成要素は、現在の日付に設定されます。

構文　時間をタイムスタンプに変換

```
TIMETOTS (time, length, 'outfield')
```

説明

`time`
日付時間

日付時間フォーマットで表される時間です。

`length`
整数

結果の長さです。次のいずれかを指定することができます。

`8` - ミリ秒を含む時間値です。

`10` - マイクロ秒を含む入力時間値です。

```
outfield
```
日付時間

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例　時間をタイムスタンプに変換

TIMETOTS 関数は、時間引数をタイムスタンプに変換します。

```
COMPUTE TSTMPSEC/HYYMDS = TIMETOTS(TMSEC, 8, 'HYYMDS'); AND
COMPUTE TSTMPMILLI/HYYMDm = TIMETOTS(TMMILLI, 10, 'HYYMDm');
```

TODAY - 現在の日付を取得

TODAY 関数は、MM/DD/YY または MM/DD/YYYY フォーマットでオペレーティングシステムから現在の日付を取得します。常に現在の日付が返されます。このため、深夜にアプリケーションを実行する場合は、TODAY を使用することをお勧めします。デフォルトで挿入されたスラッシュ記号 (/) を削除するには、EDIT 関数を使用します。

構文　現在の日付を取得

```
TODAY('outfield')
```

説明

```
outfield
```
文字

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。フィールドフォーマットは A8 以上にする必要があります。以下のように適用します。

- DATEFNS=ON でフォーマットが A8 または A9 の場合、TODAY は 2 桁の年を返します。
- DATEFNS=ON でフォーマットが A10 以上の場合、TODAY は 4 桁の年を返します。
- DATEFNS=OFF の場合、*outfield* のフォーマットに関わらず、TODAY は 2 桁の年を返します。

例　現在の日付を取得

TODAY は、現在の日付を取得し、DATE フィールドに格納します。

```
COMPUTE DATE/A10 = TODAY('A10');
```

YM - 経過月数を計算

YM 関数は、日付から日付までの経過月数を計算します。日付は年月フォーマットである必要があります。CHGDAT または EDIT 関数を使用することにより、日付をこのフォーマットに変換することができます。

構文　経過月数を計算

```
YM(fromdate, todate, 'outfield')
```

説明

`fromdate`
整数 (I4YM または I6YYM)

年月日の開始日付です (例、I4YM)。日付が有効でない場合、この関数は 0 (ゼロ) を返します。

`todate`
整数 (I4YM または I6YYM)

整数年月フォーマットのレガシー日付です。日付が有効でない場合、この関数は 0 (ゼロ) を返します。

`outfield`
整数

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

ヒント：*fromdate* または *todate* が整数の年月日フォーマット (I6YMD または I8YYMD) の場合、年月フォーマットを変換し、結果を整数に設定するには、100 で除算します。これにより日付の日の部分が削除されます。日は、小数部分で表されます。

例　経過月数を計算

COMPUTE コマンドは、年月日の日付を年月フォーマットに変換します。その後、YM は HIRE_DATE/100 と DAT_INC/100 フィールドの値の差を計算します。

```
COMPUTE HIRE_MONTH/I4YM = HIRE_DATE/100; AND
COMPUTE MONTH_INC/I4YM = DAT_INC/100; AND
COMPUTE MONTHS_HIRED/I3 = YM(HIRE_MONTH, MONTH_INC, 'I3');
```

6

簡略日付関数および日付時間関数

簡略日付関数および日付時間関数では、SQL 関数で使用されるパラメータリストに類似した、簡略化されたパラメータリストが使用されます。ただし、これらの簡略関数の機能は、以前のバージョンの同様の関数と若干異なる場合があります。

簡略関数には、出力引数はありません。各関数は、特定のデータタイプを持つ値を返します。

これらの関数をリレーショナルデータソースに対するリクエストで使用すると、関数が最適化された上で、RDBMS に渡されて処理されます。

標準の日付および日付時間フォーマットは、YYMD および HYYMD 構文で表されます (文字フィールドおよび数値フィールドには格納されない日付)。これらのフォーマット以外の日付は、簡略関数で使用する前に、変換する必要があります。リテラル日付時間値は、DT 関数で使用することができます。

すべての引数は、リテラル、フィールド名、変数のいずれかにすることができます。

トピックス

DTADD - 日付または日付時間構成要素への増分値の加算

DTADD 関数は、標準の日付または日付時間フォーマットで指定された日付から、有効な構成要素の増分値を加算した上で、新しい日付を返します。返される日付フォーマットは、入力日付フォーマットと同一になります。

構文　日付または日付時間構成要素への増分値の加算

```
DTADD(date, component, increment)
```

説明

`date`
日付または日付時間

増分値を加算する日付値または日付時間値です。

`component`
キーワード

増分値を加算する構成要素です。有効な構成要素 (および受容可能な値) は次のとおりです。

- YEAR (1-9999)
- QUARTER (1-4)
- MONTH (1-12)
- WEEK (1-53)　この構成要素は、WEEKFIRST 設定の影響を受けます。
- DAY (日付、1-31)
- HOUR (0-23)
- MINUTE (0-59)
- SECOND (0-59)

`increment`
整数

構成要素に加算する値 (正または負) です。

例　日付の DAY 構成要素への増分値の加算

次のリクエストは、WF_RETAIL データソースを使用し、従業員の誕生日に 3 日を加算します。

```
DEFINE FILE WF_RETAIL
NEWDATE/YYMD = DTADD(DATE_OF_BIRTH, DAY, 3);
MGR/A3 = DIGITS(ID_MANAGER, 3);
END
TABLE FILE WF_RETAIL
SUM MGR NOPRINT DATE_OF_BIRTH NEWDATE
BY MGR
ON TABLE SET PAGE NOPAGE
END
```

出力結果は次のとおりです。

MGR	Date of Birth	NEWDATE
001	1985/01/29	1985/02/01
101	1982/04/01	1982/04/04
201	1976/11/14	1976/11/17
301	1980/05/15	1980/05/18
401	1975/10/19	1975/10/22
501	1985/04/11	1985/04/14
601	1967/02/03	1967/02/06
701	1977/10/16	1977/10/19
801	1970/04/18	1970/04/21
901	1972/03/29	1972/04/01
999	1976/10/21	1976/10/24

参照 DTADD 使用上の注意

- 各要素は、それぞれ個別に操作する必要があります。たとえば、日付に 1 年と 1 日を加算する場合は、この関数を 2 回呼び出す必要があります。1 回目は YEAR (うるう年を考慮する必要あり) に加算し、2 回目は DAY に加算します。複数の簡略関数は、単一の式にネストすることも、それぞれを個別の DEFINE 式または COMPUTE 式に適用することもできます。
- DTADD 関数でのパラメータの検証に関しては、最初のパラメータで使用される標準の日付値または日付時間値のみが対象になります。
- 増分値は確認されません。また、小数点以下の桁数はサポートされず、小数部は切り取られます。複数の増分値を任意に組み合わせて加算した結果、年の値が 9999 を超えた場合、入力日付が返されます。この場合、メッセージは表示されません。入力日付以外の値が返される状況で、入力日付が返された場合は、エラーが発生した可能性があります。

DTDIFF - 2 つの日付値または日付時間値の構成要素の差分を取得

DTDIFF 関数は、標準の日付または日付時間フォーマットで指定された 2 つの日付間の構成要素の差分を返します。返される値は、カレンダー構成要素には整数フォーマット、時間構成要素には倍精度浮動小数点数フォーマットが使用されます。

構文　構成要素の差分の取得

```
DTDIFF(end_date, start_date, component)
```

説明

`end_date`
日付または日付時間

標準の日付または日付時間フォーマットで指定する終了日です。この日付が標準の日付フォーマットで指定された場合、すべての時間構成要素は 0 (ゼロ) と見なされます。

`start_date`
日付または日付時間

標準の日付または日付時間フォーマットで指定する開始日です。この日付が標準の日付フォーマットで指定された場合、すべての時間構成要素は 0 (ゼロ) と見なされます。

`component`
キーワード

差分を計算する構成要素です。たとえば、QUARTER を指定すると、2 つの日付間の四半期の差分が計算されます。有効な構成要素 (および受容可能な値) は次のとおりです。

- YEAR (1-9999)
- QUARTER (1-4)
- MONTH (1-12)
- WEEK (1-53)　この構成要素は、WEEKFIRST 設定の影響を受けます。
- DAY (日付、1-31)
- HOUR (0-23)
- MINUTE (0-59)
- SECOND (0-59)

例　2 つの日付の年数差を取得

次のリクエストは、WF_RETAIL データソースを使用し、従業員の雇用時の年齢を計算します。

```
DEFINE FILE WF_RETAIL
YEARS/I9 = DTDIFF(START_DATE, DATE_OF_BIRTH, YEAR);
END
TABLE FILE WF_RETAIL
PRINT START_DATE DATE_OF_BIRTH YEARS AS 'Hire,Age'
BY  EMPLOYEE_NUMBER
WHERE EMPLOYEE_NUMBER CONTAINS 'AA'
ON TABLE SET PAGE NOPAGE
END
```

出力結果は次のとおりです。

Employee Number	Start Date	Date of Birth	Hire Age
AA100	2008/11/14	1991/06/04	17
AA12	2008/11/19	1985/07/13	23
AA137	2013/01/15	1988/12/24	25
AA174	2013/01/15	1980/08/30	33
AA195	2013/01/15	1977/12/11	36
AA427	2008/12/23	1969/08/08	39
AA820	2013/10/29	1983/11/27	30
AA892	2013/10/27	1981/04/24	32

DTPART - 日付または日付時間構成要素を整数フォーマットで取得

DTPART 関数は、標準の日付または日付時間フォーマットで指定された日付および構成要素から、構成要素の値を整数フォーマットで返します。

構文　日付または日付時間構成要素を整数フォーマットで取得

```
DTPART(date, component)
```

説明

`date`
日付または日付時間

標準の日付または日付時間フォーマットで指定する日付です。

`component`
キーワード

整数フォーマットで抽出する構成要素です。有効な構成要素 (および値) は次のとおりです。

- YEAR (1-9999)
- QUARTER (1-4)
- MONTH (1-12)
- WEEK (年の週番号、1-53)　この構成要素は、WEEKFIRST 設定の影響を受けます。
- DAY (日付、1-31)
- DAY_OF_YEAR (1-366)
- WEEKDAY (曜日番号、1-7)　この構成要素は、WEEKFIRST 設定の影響を受けます。
- HOUR (0-23)
- MINUTE (0-59)
- SECOND (0-59)
- MILLISECOND (0-999)
- MICROSECOND (0-999999)

例　四半期構成要素を整数フォーマットで抽出

次のリクエストは、WF_RETAIL データソースを使用し、従業員の勤務開始日から四半期構成要素を抽出します。

```
DEFINE FILE WF_RETAIL
QTR/I2 = DTPART(START_DATE, QUARTER);
END
TABLE FILE WF_RETAIL
PRINT START_DATE QTR AS Quarter
BY  EMPLOYEE_NUMBER
WHERE EMPLOYEE_NUMBER CONTAINS 'AH'
ON TABLE SET PAGE NOPAGE
END
```

出力結果は次のとおりです。

Employee Number	Start Date	Quarter
AH118	2013/01/15	1
AH288	2013/11/11	4
AH42	2008/11/13	4
AH928	2009/04/11	2

DTRUNC - 特定の日付が属する日付範囲の開始日を取得

DTRUNC 関数は、指定された日付またはタイムスタンプおよび構成要素から、その構成要素で指定された日付範囲の開始日を返します。

構文　　日付範囲の開始日または最終日を取得

```
DTRUNC(date_or_timestamp, date_period)
```

説明

`date_or_timestamp`
日付または日付時間

特定の日付またはタイムスタンプです。

`date_period`
開始日または最終日を特定する日付範囲です。次のいずれかの値です。

- DAY - 入力日の日付を返します (時間が含まれる場合は省略)。
- YEAR - 年の開始日の日付を返します。
- MONTH - 月の開始日の日付を返します。
- QUARTER - 四半期の開始日の日付を返します。
- WEEK - 特定の週の開始日の日付を返します。

  デフォルト設定では、開始曜日は日曜日になりますが、WEEKFIRST パラメータを使用してデフォルト値を変更することができます。
- YEAR_END - 年の最終日の日付を返します。

- QUARTER_END - 四半期の最終日の日付を返します。
- MONTH_END - 月の最終日の日付を返します。
- WEEK_END - 週の最終日の日付を返します。

例　日付範囲の開始日を取得

次のリクエストでは WF_RETAIL データソースが使用され、DTRUNC 関数が、従業員の勤務開始日が属する四半期の開始日を返します。

```
DEFINE FILE WF_RETAIL
QTRSTART/YYMD = DTRUNC(START_DATE, QUARTER);
END
TABLE FILE WF_RETAIL
PRINT START_DATE QTRSTART AS 'Start,of Quarter'
BY EMPLOYEE_NUMBER
WHERE EMPLOYEE_NUMBER CONTAINS 'AH'
ON TABLE SET PAGE NOPAGE
END
```

出力結果は次のとおりです。

Employee Number	Start Date	Start of Quarter
AH118	2013/01/15	2013/01/01
AH288	2013/11/11	2013/10/01
AH42	2008/11/13	2008/10/01
AH928	2009/04/11	2009/04/01

7

フォーマット変換関数

フォーマット変換関数は、フィールドのフォーマットを変換します。

トピックス

ATODBL - 文字列を倍精度浮動小数点数フォーマットに変換

ATODBL 関数は、文字列を実数 (倍精度浮動小数点数) フォーマットに変換します。

構文　　文字列を倍精度浮動小数点数フォーマットに変換

```
ATODBL(string, length, 'outfield')
```

説明

`string`
文字

変換する文字列、または文字列を含むフィールドです。

`length`
文字

infield の 2 バイトの文字列です。数値定数を指定することも、値を含むフィールドを指定することもできます。数値定数を指定する場合は、一重引用符 (') で囲みます。最大値は 15 です。

outfield
10 進コード

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例 ATODBL - 文字列を倍精度浮動小数点数フォーマットに変換

ATODBL 関数は、EMP_ID フィールドを倍精度小数点数フォーマットに変換し、結果を D_EMP_ID に格納します。

```
COMPUTE D_EMP_ID/D12.2 = ATODBL(EMP_ID, '09', 'D12.2');
```

EDIT - フィールドのフォーマットを変換

EDIT 関数は、数値を含む文字フィールドを数値フォーマットに変換、または数値フィールドを文字フォーマットに変換します。特定のフォーマットが必要なコマンドを使用してフィールドを操作する場合に役立ちます。

EDIT で変換後の値を新しいフィールドに割り当てるときは、新しいフィールドのフォーマットが返された値のものと同一である必要があります。たとえば、EDIT で数値フィールドを文字フォーマットに変換する場合、新しいフィールドは文字フォーマットである必要があります。

```
DEFINE ALPHAPRICE/A6 = EDIT(PRICE);
```

EDIT 関数は、特殊文字を次のように処理します。

- 文字フィールドを数値フィールドに変換する場合、フィールドの符号や小数点は受容可能で、数値フィールドに格納されます。
- 浮動小数点数またはパック 10 進数フィールドを文字フォーマットに変換するときは、符号、小数点、および小数点以下のすべての数字は削除されます。さらに残りの数字を右揃えし、指定されたフィールドの長さに達するまで、先頭に 0 (ゼロ) を追加します。浮動小数点数またはパック 10 進数フォーマットの 10 桁以上の数値を変換すると、結果に誤りが生じる可能性があります。

また、EDIT 関数を使用して、文字列から文字を抽出したり、文字列に文字を追加したりすることもできます。

構文　フィールドのフォーマットを変換

```
EDIT(fieldname);
```

説明

`fieldname`
文字または数値

フィールド名です。

例　数値を文字フォーマットに変換

EDIT 関数は、HIRE_DATE (レガシー日付フォーマット) を文字フォーマットに変換します。これにより、文字フォーマットをとる CHGDAT 関数でこのフィールドが使用できるようになります。

```
COMPUTE ALPHA_HIRE/A17 = EDIT(HIRE_DATE); AND
COMPUTE HIRE_MDY/A17 = CHGDAT('YMD', 'MDYYX', ALPHA_HIRE, 'A17');
```

FTOA - 数値を文字フォーマットに変換

FTOA 関数は、16 桁以内の数値を文字フォーマットに変換します。数値の小数点の位置を保持し、先頭にブランクを追加して右揃えします。FTOA で変換する数値には、編集オプションを追加することができます。

FTOA を使用して小数部を含む値を文字列に変換するときは、数値の整数部分と小数点以下を格納するために十分な大きさの文字フォーマットを指定する必要があります。たとえば、D12.2 は、A14 に変換されます。出力フォーマットの大きさが十分でない場合、小数点以下は切り捨てられます。

構文　数値を文字フォーマットに変換

```
FTOA(number, '(format)', 'outfield')
```

説明

`number`
数値 F または D (単精度浮動小数点数または倍精度浮動小数点数)

変換される数値です。数値を含むフィールド名を指定することもできます。

`format`
文字

数値の出力フォーマットです。文字列は一重引用符 (') および括弧で囲みます。浮動小数点数は、単精度および倍精度フォーマットのみがサポートされています。出力に表示する編集オプションをすべて含めます。D (倍精度浮動小数点数) フォーマットを指定すると、カンマ (,) が自動的に追加されます。

この引数にフィールド名を使用する場合、引用符や括弧を使用せずに名前を入力します。フォーマットを指定する場合、括弧で囲む必要があります。

`outfield`
文字

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。この引数の長さは数値の長さよりも大きくする必要があります。編集オプションおよび負の符号が追加される可能性も考慮します。

例　数値を文字フォーマットに変換

FTOA 関数は、GROSS フィールドのフォーマットを倍精度浮動小数点数フォーマットから文字フォーマットに変換し、結果を ALPHA_GROSS に格納します。

```
COMPUTE ALPHA_GROSS/A15 = FTOA(GROSS, '(D12.2)', 'A15');
```

HEXBYT - 10 進数を文字に変換

HEXBYT 関数は、10 進数の整数に対応する ASCII、EBCDIC、または Unicode のいずれかの文字を取得します。取得される文字は、構成およびオペレーティングシステムにより異なります。この関数は、ASCII、EBCDIC、または Unicode 文字セットのいずれかで、単一文字を返します。この関数を使用することにより、CTRAN 関数と同様、使用するキーボードにはない文字を生成することができます。

Unicode 構成の場合、この関数は次の範囲の値を使用します。

- 1 バイト文字 - 0 (ゼロ) から 255
- 2 バイト文字 - 256 から 65535
- 3 バイト文字 - 65536 から 16777215
- 4 バイト文字 - 16777216 から 4294967295 (主として EBCDIC 用)

特殊文字の表示は、ソフトウェアとハードウェアにより異なります。特殊文字には表示されないものがあります。

構文　10 進数を文字に変換

```
HEXBYT(input, 'outfield')
```

説明

`source_string`
　整数

　文字に変換される 10 進数です。Unicode 以外の環境では、255 より大きい値は、*input* の値を 256 で除算した剰余として扱われます。

`outfield`
　文字

　出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例　10 進数を ASCII および Unicode の文字に変換

HEXBYT 関数は、LAST_INIT_CODE を対応する文字に変換し、結果を LAST_INIT に格納します。

```
COMPUTE LAST_INIT_CODE/I3 = BYTVAL(LAST_NAME, 'I3'); AND
COMPUTE LAST_INIT/A1 = HEXBYT(LAST_INIT_CODE, 'A1');
```

ITONUM - 整数を倍精度小数点数フォーマットに変換

ITONUM 関数は、データソースの整数を倍精度浮動小数点数フォーマットに変換します。プログラミング言語やデータストレージシステムには、整数フォーマットを使用するものがあります。ただし、(5 バイト長以上の) 整数はマスターファイルではサポートされません。このため、倍精度フォーマットへの変換が必要になります。

入力フィールド内の最も右側から数えた有効バイト数を指定する必要があります。結果は、8 バイトの倍精度浮動小数点数フィールドです。

構文　整数を倍精度浮動小数点数フォーマットに変換

```
ITONUM(maxbytes, infield, 'outfield')
```

説明

`maxbytes`
　数値

2 進数符号を含めた有効数値データの 8 バイトの infield の最大数です。有効な値には、次のものがあります。

5 - 左から 3 バイトを無視します。

6 - 最も左側の 2 バイトを無視します。

7 - 左から 7 バイトを無視します。

infield
文字

2 進数を含むフィールドです。このフィールドの USAGE および ACTUAL フォーマットは、どちらも A8 である必要があります。

outfield
10 進コード

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。フォーマットは D*n* である必要があります。

例　整数を倍精度小数点数フォーマットに変換

ITONUN 関数は、BINARYFLD フィールドを倍精度浮動小数点数フォーマットに変換します。

```
COMPUTE MYFLD/D14 = ITONUM(6, BINARYFLD, 'D14');
```

ITOPACK - 整数をパック 10 進数フォーマットに変換

ITOPACK 関数はデータソースの整数をパック 10 進数フォーマットに変換します。プログラミング言語やデータストレージシステムには、倍長整数フォーマットを使用するものがあります。(5 バイト長以上の) 整数はマスターファイルではサポートされていません。このため、パック 10 進数フォーマットへの変換が必要になります。

入力フィールド内の最も右側から数えた有効バイト数を指定する必要があります。結果は、有効桁数が 15 桁以内の 8 バイトのパック 10 進数フィールド (例、P15 または P16.2) です。

制限：「PIC 9(15) COMP」として定義したフィールド、またはこれに準ずるフィールド (有効桁数 15 桁) において、変換可能な最大数は、167,744,242,712,576 です。

構文　整数をパック 10 進数フォーマットに変換

```
ITOPACK(maxbytes, infield, 'outfield')
```

説明

`maxbytes`
数値

2 進数符号を含めた有効数値データの 8 バイトの infield の最大数です。

有効な値には、次のものがあります。

`5` - 左から 3 バイトを無視します (11 桁以内の有効な位置)。

`6` - 左から 2 バイトを無視します (14 桁以内の有効な位置)。

`7` - 左から 7 バイトを無視します (15 桁以内の有効な位置)。

`infield`
文字

2 進数を含むフィールドです。このフィールドの USAGE および ACTUAL フォーマットは、どちらも A8 である必要があります。

`outfield`
パック 10 進数

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。フォーマットは P*n* または P*n.d* である必要があります。

例　整数をパック 10 進数フォーマットに変換

ITOPACK 関数は、BINARYFLD フィールドをパック 10 進数フォーマットに変換します。

```
COMPUTE PACKFLD/P14.4 = ITOPACK(6, BINARYFLD, 'P14.4');
```

ITOZ - 数値をゾーン 10 進数フォーマットに変換

ITOZ 関数は、数値フォーマットの数をゾーン 10 進数フォーマットに変換します。リクエストはゾーン 10 進数を処理することはできませんが、ゾーン 10 進数フィールドを抽出ファイルに書き込み、外部プログラムで使用することができます。

構文　ゾーン 10 進数フォーマットに変換

```
ITOZ(outlength, number, 'outfield')
```

説明

`outlength`
整数

number のバイト数です。最大バイト数は 15 です。末尾バイトには、符号が含まれます。

`number`
数値

変換される数値です。数値を含むフィールドを指定することもできます。数値は変換前に切り捨てられて整数になります。

`outfield`
文字

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例　数値をゾーン 10 進数フォーマットに変換

ITOZ 関数は、CURR_SAL フィールドをゾーン 10 進数に変換します。

```
COMPUTE ZONE_SAL/A8 = ITOZ(8, CURR_SAL, 'A8');
```

PCKOUT - 指定した長さでパック 10 進数を書き込み

PCKOUT 関数は、抽出ファイルに指定した長さでパック 10 進数を書き込みます。リクエストが抽出ファイルにパック 10 進数を保存する際、フォーマットの指定に関わらず、通常 8 バイトまたは 16 バイトのフィールドとして書き込みます。PCKOUT を使用することで、フィールドを 1 から 16 バイトの指定した長さに変更することができます。

構文　指定した長さでパック 10 進数を書き込み

```
PCKOUT(infield, outlength, 'outfield')
```

説明

`infield`
数値

数値を含むフィールドです。このフィールドには、パック 10 進数、整数、浮動小数点数、倍精度フォーマットが使用できます。フィールドが整数フォーマットではない場合、端数処理されて、値は最も近い整数になります。

`outlength`
数値

outfield の長さ (1 から 16 バイト) です。

`outfield`
文字

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。フィールドはパック 10 進数を含んでいますが、この関数はフィールドを文字として返します。

例　指定した長さでパック 10 進数を書き込み

PCKOUT 関数は、CURR_SAL フィールドを 5 バイトのパック 10 進数フィールドに変換し、結果を SHORT_SAL に格納します。

```
COMPUTE SHORT_SAL/A5 = PCKOUT(CURR_SAL, 5, 'A5');
```

8

数値関数

数値関数は、数値定数と数値フィールドの計算を実行します。

トピックス

ABS - 絶対値を計算

ABS 関数は、数値の絶対値を返します。

構文　絶対値を計算

```
ABS(argument)
```

説明

`argument`
数値

絶対値を返す値、値を含むフィールド名、あるいは値を返す式です。式を指定する場合は、評価の順序を正しくするため、必要に応じて括弧を使用します。

例　絶対値を計算

最初の COMPUTE コマンドが DIFF フィールドを作成し、次に、ABS 関数が DIFF の絶対値を計算します。

```
COMPUTE DIFF/I5 = DELIVER_AMT - UNIT_SOLD; AND
COMPUTE ABS_DIFF/I5 = ABS(DIFF);
```

BAR - 棒グラフを作成

BAR 関数は、横棒グラフを作成します。棒には、繰り返し文字が使用されます。必要に応じて、棒グラフを明確にするために目盛りを作成することができます。これには、棒を含むカラムタイトルを目盛りで置き換えます。

構文　棒グラフを作成

```
BAR(barlength, infield, maxvalue, 'char', 'outfield')
```

説明

`barlength`
　数値

　棒の長さの最大値をバイト数で指定します。この値が 0 (ゼロ) 以下の場合、関数は棒グラフを返しません。

`infield`
　数値

　棒グラフとして描くデータフィールドです。

`maxvalue`
　数値

　棒グラフの最大値です。この値は、`infield` に格納された最大値より大きくなければなりません。`infield` の値が `maxvalue` の値よりも大きい場合、関数は `maxvalue` を使用し、最大長の棒グラフを返します。

`char`
　文字

　棒グラフを作成する繰り返し文字です。文字列は一重引用符 (') で囲みます。複数の文字を指定した場合、先頭の文字のみが使用されます。

`outfield`
　文字

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。出力フィールドには、barlength に定義された最大値の長さを持つ棒グラフの表示に十分な長さが必要です。

例　棒グラフを作成

BAR 関数は、CURR_SAL フィールドの棒グラフを作成し、出力結果を SAL_BAR に格納します。作成された棒の長さは 30 バイト以下で、表す値は 30,000 以下である必要があります。

```
COMPUTE SAL_BAR/A30 = BAR(30, CURR_SAL, 30000, '=', SAL_BAR);
```

CHKPCK - パック 10 進数フィールドを検査

CHKPCK 関数は、プラットフォームで使用可能な場合、パック 10 進数フィールドとして記述されるフィールド内のデータを検査します。この関数は、リクエストがフィールドを読み取る際に、有効なパック 10 進数が含まれていることを期待して実際に含まれていない場合に、データ例外が発生することを防止します。

CHKPCK の使用方法は、次のとおりです。

1. マスターファイル (USAGE および ACTUAL 属性) で、フィールドがパック 10 進数ではなく、文字として定義されていることを確認します。これにより、フィールドデータは変更されずにパック 10 進数のままになりますが、リクエストによるデータの読み取り時にデータ例外の発生を防止することができます。
2. CHKPCK を呼び出してフィールドを検査します。この関数は、パック 10 進数として定義したフィールドに出力結果を返します。検査する値が有効なパック 10 進数の場合、関数は値を返します。パック 10 進数ではない場合は、エラーコードを返します。

構文　パック 10 進数フィールドを検査

```
CHKPCK(inlength, infield, error, 'outfield')
```

説明

`inlength`
数値

引数の長さです。1 から 16 バイトの値を指定します。

`infield`
文字

入力フィールド名です。フィールドは、パック 10 進数ではなく、文字として記述されます。

`error`
数値

値がパック 10 進数ではない場合に関数が返すエラーコードです。データの範囲外のエラーコードを選択します。エラーコードは、整数に切り捨てられた後、パック 10 進数フォーマットに変換されます。ただし、出力フィールドのフォーマットによっては、レポートに小数点付きで表示される場合があります。

`outfield`
パック 10 進数

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例　パック 10 進数を検査

CHKPCK 関数は、PACK_SAL フィールドの値を検証し、結果を GOOD_PACK フィールドに格納します。フォーマットがパック 10 進数以外の値は、エラーコード -999 を返します。パック 10 進数フォーマットの値は正しく表示されます。

```
COMPUTE GOOD_PACK/P8CM = CHKPCK(8, PACK_SAL, -999, GOOD_PACK);
```

DMOD、FMOD、IMOD - 除算の剰余を計算

MOD 関数は、除算の剰余を計算します。各関数は、異なるフォーマットで剰余を返します。

関数は次の公式を使用します。

```
remainder = dividend - INT(dividend/divisor) * divisor
```

- `DMOD` - 剰余を倍精度浮動小数点数で返します。
- `FMOD` - 剰余を単精度浮動小数点数で返します。
- `IMOD` - 剰余を整数で返します。

構文　除算の剰余を計算

```
function(dividend, divisor, 'outfield')
```

説明

`function`
次のいずれかです。

DMOD - 剰余を倍精度浮動小数点数で返します。

FMOD - 剰余を単精度浮動小数点数で返します。

IMOD - 剰余を整数で返します。

dividend
数値

被除数です。

divisor
数値

除数です。

outfield
数値

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。フォーマットは、特定の関数により返された結果により決定されます。

例　除算の剰余を計算

IMOD 関数は、ACCTNUMBER を 1000 で除算し、その剰余を LAST3_ACCT に返します。

```
COMPUTE LAST3_ACCT/I3L = IMOD(ACCTNUMBER, 1000, LAST3_ACCT);
```

EXP - 「e」を N でべき乗

EXP 関数は、値 「e」 (およそ 2.72) を指数でべき乗します。この関数は、引数の対数を返す LOG 関数の逆です。

EXP は、無限級数の項の和を計算します。項が合計に加算する値が 0.000001 パーセントよりも小さくなったところで、関数は計算を終了し、結果を倍精度小数点数で返します。

構文　「e」 を N でべき乗

```
EXP(power, 'outfield')
```

説明

power
数値

「e」 をべき乗する指数です。

outfield
単精度浮動小数点数または 10 進数

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例　「e」 を N でべき乗

EXP は、「e」を N でべき乗します。

```
COMPUTE E2/D12.2 = EXP(2, 'D12.2');
```

INT - 整数を検索

INT 関数は、数値の整数構成要素を返します。

構文　整数を検索

```
INT(argument)
```

説明

`argument`
数値

整数構成要素が返される値です。値を含むフィールド名または値を返す式を指定することもできます。式を指定する場合は、評価の順序を正しくするため、必要に応じて括弧を使用します。

例　整数を検索

INT は DED_AMT フィールドの最大整数を検索し、結果を INT_DED_AMT に格納します。

```
COMPUTE INT_DED_AMT/I9 = INT(DED_AMT);
```

LOG - 自然対数を計算

LOG 関数は、数値の自然対数を返します。

構文　自然対数を計算

```
LOG(argument)
```

説明

`argument`
数値

自然対数が計算される値です。値を含むフィールド名、または値を返す式を指定することもできます。式を指定する場合は、評価の順序を正しくするため、必要に応じて括弧を使用します。argument の値が 0 (ゼロ) 以下の場合、LOG は 0 (ゼロ) を返します。

例　自然対数を計算

LOG 関数は、CURR_SAL フィールドの対数を計算します。

```
COMPUTE LOG_CURR_SAL/D12.2 = LOG(CURR_SAL);
```

MAX および MIN - 最大値または最小値を検索

MAX および MIN 関数は、値リストからそれぞれ最大値と最小値を返します。

構文　最大値または最小値を検索

```
{MAX|MIN}(argument1, argument2, ...)
```

説明

MAX
最大値を返します。

MIN
最小値を返します。

argument1, argument2
数値

最大値または最小値を返す値、値を含むフィールド名、あるいは値を返す式です。式を指定する場合は、評価の順序を正しくするため、必要に応じて括弧を使用します。

例　最小値を抽出

MIN 関数は、ED_HRS フィールドの値と定数 30 のうちの、小さい方の値を返します。

```
COMPUTE MIN_EDHRS_30/D12.2 = MIN(ED_HRS, 30);
```

SQRT - 平方根を計算

SQRT 関数は、数値の平方根を計算します。

構文　　平方根を計算

```
SQRT(argument)
```

説明

`argument`
数値

平方根が計算される値です。値を含むフィールド名、または値を返す式を指定することもできます。式を指定する場合は、評価の順序を正しくするため、必要に応じて括弧を使用します。負の数値を指定すると、結果は 0 (ゼロ) になります。

例　　平方根を計算

SQRT 関数は、LISTPR の平方根を計算します。

```
COMPUTE SQRT_LISTPR/D12.2 = SQRT(LISTPR);
```

9

システム関数

システム関数は、オペレーティングシステムを呼び出して、オペレーティング環境に関する情報を取得します。

トピックス

FGETENV - 環境変数値を取得

FGETENV 関数は、オペレーティングシステムの環境変数値を取得し、文字列として返します。

構文　環境変数の値を取得

```
FGETENV(varlength, 'varname', outfieldlen, 'outfield')
```

説明

`varlength`
整数

環境変数名の長さです。

`varname`
文字

環境変数名です。

`outfieldlen`
整数

環境変数の値が格納されているフィールドの長さです。

`outfield`
文字

出力値のフォーマットです。フォーマットは一重引用符 (') で囲みます。

例　言語ロケールを取得

FGETENV 関数は、LANG 環境変数を使用して、言語ロケールのオブジェクトの位置を取得します。

```
COMPUTE LANG_LOCALE/A40 = FGETENV(4, 'LANG', 40, 'A40');
```

GETUSER - ユーザ ID を取得

GETUSER 関数は、接続ユーザの ID を取得します。

構文　ユーザ ID を取得

```
GETUSER('outfield')
```

説明

`outfield`
文字

出力値のフォーマットです (A8 以上)。フォーマットは一重引用符 (') で囲みます。長さは、関数を発行するプラットフォームに応じて異なります。プラットフォームで要求される最大長を指定します。最大長を指定しない場合、出力の末尾が切り取られる場合があります。

例　ユーザ ID を取得

GETUSER 関数は、リクエストを実行しているユーザの ID を取得します。

```
COMPUTE USERID/A8 = GETUSER('A8');
```

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE "LICENSE" FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

TIBCO, the TIBCO logo, the TIBCO O logo, FOCUS, iWay, Omni-Gen, Omni-HealthData, and WebFOCUS are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

Java and all Java based trademarks and logos are trademarks or registered trademarks of Oracle Corporation and/or its affiliates.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

This software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the readme file for the availability of this software version on a specific operating system platform.

THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.