ibi™ WebFOCUS®

WebFOCUS 関数リファレンス

バージョン 9.0.0 以降 | January 2024

目次

1

このマニュアルの使用方法

このマニュアルは、TIBCO WebFOCUS 製品で提供される関数について説明しており、プログラムからこれらの関数を呼び出して、計算またはデータ操作を実行するアプリケーション開発者を対象としています。開発者以外のユーザは、企業データにアクセスしてレポートを生成する際に関数を呼び出すことができます。

このマニュアルは、個々のニーズに応じて使用する関数 (サブルーチン) を作成する方法についても説明します。

トピックス

利用可能な言語

関数は、レポート言語で使用できます。

- レポート言語には、レポートを作成するためのすべてのコマンドが含まれます。レポート言語はすべての WebFOCUS 製品のユーザが使用できます。

関数の説明は、利用可能な言語に対応するもの、またはカテゴリ別の一覧から参照します。カテゴリ別一覧は、19 ページの 「 関数の概要 」 を参照してください。

オペレーティングシステム

特に記述がない限り、すべての関数がサーバのサポート対象のオペレーティングシステムで実行できます。

2

関数の概要

この章では、関数の概要と種類について説明します。

トピックス

関数の使用

関数は 1 つ以上の引数を使用して計算を実行し、単一の値を返します。返された値は、フィールドへの格納、ダイアログマネージャ変数への割り当て、式またはその他の処理での使用、選択または有効性のテストでの使用が可能です。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。

関数には、次の 3 種類があります。

- **内部関数** WebFOCUS 言語に組み込まれています。アクセスや使用のために、特別な作業は必要ありません。次の関数は内部関数です。これらの内部関数を同名の独自の関数で置き換えることはできません。これらを除く関数は、すべて外部関数です。
  - ABS
  - ASIS
  - DMY、MDY、YMD
  - DECODE
  - EDIT
  - FIND
  - LAST
  - LOG
  - LOOKUP
  - MAX、MIN

- SQRT

- **外部関数**　外部ライブラリに格納されます。外部ライブラリにアクセスする必要があります。これらの関数を呼び出す際は、出力フィールドまたは結果のフォーマットを指定する引数が必要です。外部関数は WebFOCUS とともに提供されています。これらの関数は、同名の独自の関数で置き換えることができます。ただし、この場合、USERFNS=LOCAL を設定する必要があります。
- **サブルーチン**　ユーザが記述する関数です。外部に格納されます。詳細は、623 ページの「 サブルーチンの作成 」 を参照してください。

内部関数と外部関数の使用方法についての詳細は、45 ページの 「 関数へのアクセスと呼び出し 」 を参照してください。

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。

- **簡略分析関数**　内部マトリックスの複数行を使用する計算を実行します。詳細は、22 ページの 「 簡略分析関数 」 を参照してください。
- **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を使用する文字列関数です。この関数には、出力引数はありません。詳細は、23 ページの 「 簡略文字列関数 」 を参照してください。
- **文字列関数**　文字フィールドまたは文字列を操作します。詳細は、24 ページの 「 文字列関数 」 を参照してください。
- **可変長文字列関数**　AnV フィールドまたは文字列を操作します。詳細は、26 ページの「 可変長文字列関数 」 を参照してください。
- **DBCS コードページの文字列関数**　DBCS コードページで文字フィールドまたは文字列を操作します。詳細は、26 ページの 「 DBCS コードページの文字列関数 」 を参照してください。
- **データソースおよびデコード関数**　データソースからレコードや値を検索または抽出し、値を割り当てます。詳細は、27 ページの 「 データソースおよびデコード関数 」 を参照してください。
- **簡略日付関数および日付時間関数**　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を使用する日付関数および日付時間関数です。この関数には、出力引数はありません。詳細は、27 ページの 「 簡略日付関数および日付時間関数 」 を参照してください。

- **日付関数**　日付を操作します。詳細は、28 ページの 「 日付関数 」 を参照してください。
- **日付時間関数**　日付時間値を操作します。詳細は、30 ページの 「 日付時間関数 」 を参照してください。
- **簡略変換関数**　簡略化されたパラメータリストを使用して、フィールドのフォーマットを変換します。詳細は、31 ページの 「 簡略変換関数 」 を参照してください。
- **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。詳細は、32 ページの 「 フォーマット変換関数 」 を参照してください。
- **簡略数値関数**　簡略化されたパラメータリストを使用して、数値定数と数値フィールドの計算を実行します。詳細は、33 ページの 「 簡略数値関数 」 を参照してください。
- **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。詳細は、33 ページの 「 数値関数 」 を参照してください。
- **簡略統計関数**　統計的計算を実行します。詳細は、35 ページの 「 簡略統計関数 」 を参照してください。
- **機械学習 (Python ベース) 関数**　分類および回帰分析を実行する Python スクリプトを実行します。詳細は、35 ページの 「 機械学習 (Python ベース) 関数 」 を参照してください。
- **簡略システム関数**　簡略化されたパラメータリストを使用し、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ついての情報を取得したり、システムサービスを使用したりすることを可能にします。詳細は、36 ページの 「 簡略システム関数 」 を参照してください。
- **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ついての情報を取得したり、システムサービスを使用したりすることを可能にします。詳細は、36 ページの 「 システム関数 」 を参照してください。
- **簡略地理関数**　さまざまなタイプの位置データに対して位置ベースの演算を実行し、ジオコードされた地点を返します。詳細は、37 ページの 「 簡略地理関数 」 を参照してください。
- **三角関数**　三角関数計算、逆三角関数、角度変換関数を実行します。詳細は、611 ページの 「 三角関数 」 を参照してください。

TIBCO WebFOCUS 固有の関数

提供されている関数のほとんどは WebFOCUS と FOCUS で使用できます。ただし、WebFOCUS のみで使用できる関数もあります。これには次の関数があります。

- SYSTEM

この関数についての詳細は、該当するトピックを参照してください。

簡略分析関数

次の関数は、内部マトリックスの複数行に基づいて計算を実行します。詳細は、65 ページの「 簡略分析関数 」 を参照してください。

FORECAST_MOVAVE
単純移動平均フィールドの計算を実行します。

FORECAST_EXPAVE
単純指数平滑フィールドの計算を実行します。

FORECAST_DOUBLEXP
二重指数平滑フィールドの計算を実行します。

FORECAST_SEASONAL
三重指数平滑フィールドの計算を実行します。

FORECAST_LINEAR
線形重回帰フィールドの計算を実行します。

PARTITION_AGGR
ローリング演算を作成します。

PARTITION_REF
前後のフィールド値を取得します。

INCREASE
パーティション内の現在の行と前の行の値の差を計算します。

PCT_INCREASE
パーティション内の現在の行と前の行の値の差のパーセントを計算します。

PREVIOUS
パーティション内の前の値を取得します。

RUNNING_AVE
パーティション内の行グループの平均を計算します。

RUNNING_MIN
パーティション内の行グループの最小値を計算します。

RUNNING_MAX
パーティション内の行グループの最大値を計算します。

RUNNING_SUM
パーティション内の行グループの合計値を計算します。

簡略文字列関数

次の関数は、文字フィールドまたは文字列を操作します。この関数では、簡略化されたパラメータリストが使用されます。詳細は、121 ページの 「 簡略文字列関数 」 を参照してください。

CHAR_LENGTH
文字列の長さをバイト数で返します。

DIGITS
数値を特定の長さの文字列に変換します。

GET_TOKEN
トークン番号および使用可能な区切り文字を含む文字列に基づいて、トークン (サブ文字列) を抽出します。

INITCAP
文字列の各単語の先頭文字を大文字にし、その他すべての文字を小文字にします。この場合の単語は、文字列の先頭、ブランクの直後または特殊文字の直後の単語です。

LAST_NONBLANK
ブランクとミッシングのどちらでもない最終フィールド値を取得します。先行する値がすべてブランクまたはミッシングの場合はミッシング値を返します。

LOWER
文字列を小文字に変換します。

LPAD
文字列の左側に特定の文字をパディングします。

LTRIM
文字列の左端からブランクをすべて削除します。

PATTERNS
ソース文字列の構造を示すパターンを返します。

POSITION
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。

REGEX
文字列を正規表現で照合し、true (1) または false (0) を返します。

RPAD
文字列の右側に特定の文字をパディングします。

RTRIM
文字列の右端からブランクをすべて削除します。

SUBSTRING
ソース文字列からサブ文字列を抽出します。

TOKEN
トークン番号および区切り文字列に基づいて、トークン (サブ文字列) を抽出します。

TRIM_
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。

UPPER
文字列を大文字に変換します。

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。詳細は、179 ページの 「 文字列関数 」 を参照してください。

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。

ASIS
ダイアログマネージャでブランクと 0 (ゼロ) を区別します。

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。

BITVAL
文字列内の文字列のビット数を評価します。

BYTVAL
文字列を ASCII 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
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。

LCWORD3
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。

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。

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。

SPELLNM
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。

SQUEEZ
文字列内の連続するブランクを 1 つにします。

STRIP
文字列から特定の文字をすべて削除します。

STRREP
特定の文字列をすべて置換します。

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。

TRIM
文字列内のあるパターンの先頭と末尾の文字を削除します。

UPCASE
文字列を大文字に変換します。

可変長文字列関数

次の関数は、可変長文字フィールドまたは文字列を操作します。詳細は、233 ページの 「 可変長文字列関数 」 を参照してください。

LENV
AnV 入力フィールドの実際の長さ、または An フィールドのサイズを返します。

LOCASV
テキストを AnV フィールドの小文字に変換しま す。

POSITV
AnV フィールドのサブ文字列の開始位置を検索します。

SUBSTV
AnV フィールドのソース文字列内の開始位置と長さに基づいてサブ文字列を抽出します。

TRIMV
AnV フィールドの文字列内から先頭と末尾のパターンを削除します。

UPCASV
AnV フィールドの文字列を大文字に変換します。

DBCS コードページの文字列関数

次の関数は、DBCS コードページの文字列を操作します。詳細は、245 ページの 「 DBCS コードページの文字列関数 」 を参照してください。

DCTRAN
1 バイトまたは 2 バイト文字を他の文字に変換します。

DEDIT
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。

DSTRIP
文字列から 1 バイトまたは 2 バイト文字を削除します。

DSUBSTR
サブ文字列の長さおよび位置に基づいて、ソース文字列からサブ文字列を抽出します。

JPTRANS
日本語の文字を変換します。

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内を検索し、レコードまたは値を抽出して値を割り当てます。詳細は、257 ページの 「 データソースおよびデコード関数 」 を参照してください。

COALESCE
ミッシング値以外の引数の先頭値を取得します。

DB_EXPR
リレーショナルデータベースに対するリクエストで生成される SQL に SQL 式を挿入します。

DB_INFILE
ソースファイルの値とターゲットファイルの値を比較します。また、ソースファイルがリレーショナルデータソースの場合、ソースファイルの値と、サブクエリにより取得された値を比較します。

CHECKMD5
入力パラメータの MD5 ハッシュチェック値を計算します。

CHECKSUM
入力パラメータのハッシュサムを計算します。

DB_LOOKUP
参照データソースからデータ値を抽出します。

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。

FIND
入力データ値が FOCUS データソースのインデックスフィールドに存在するかどうかを確認します。

IMPUTE
ミッシング値を集計値で置換します。

LAST
フィールドの前の値を取得します。

NULLIF
2 つの入力パラメータ値が等しい場合にミッシング値を返します。

簡略日付関数および日付時間関数

次の関数は、日付値および日付時間値を操作します。詳細は、287 ページの 「 簡略日付関数および日付時間関数 」 を参照してください。

DT_CURRENT_DATE
現在の日付を返します。

DT_CURRENT_DATETIME
現在の日付時間を返します。

DT_CURRENT_TIME
現在の時刻を返します。

DTADD
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。

DTDIFF
指定された 2 つの日付間の構成要素の差分を返します。

DTIME
日付時間の値から時間構成要素を抽出します。

DTPART
構成要素の値を整数フォーマットで返します。

DTRUNC
日付範囲の開始日を返します。

日付関数

次の関数は、日付を操作します。詳細は、313 ページの 「 日付関数 」 を参照してください。

標準日付関数

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。

DATETRAN
日付を国際フォーマットに変換します。

DPART
日付フィールドから構成要素を抽出し、結果を数値フォーマットで返します。

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。

HMASK
日付時間値から 1 つ以上の日付時間構成要素を抽出し、これらをターゲット日付時間フィールドに移動します。ターゲット日付時間フィールドのその他の日付時間構成要素はすべて保持されます。

TODAY
システムから現在の日付を取得します。

レガシー日付関数

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。

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。

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。

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。

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。

日付時間関数

次の関数は、日付時間値を操作します。詳細は、373 ページの 「 日付時間関数 」 を参照してください。

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。

HEXTR
日付時間値から 1 つ以上の日付時間構成要素を抽出し、これらをターゲット日付時間フィールドに移動します。ターゲット日付時間フィールドのその他の日付時間構成要素はすべて 0 (ゼロ) 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。

HMASK
日付時間値から 1 つ以上の日付時間構成要素を抽出し、これらをターゲット日付時間フィールドに移動します。ターゲット日付時間フィールドのその他の日付時間構成要素はすべて保持されます。

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。

HTMTOTS または TIMETOTS
時間をタイムスタンプに変換します。

簡略変換関数

次の関数は、簡略化されたパラメータリストを使用して、フィールドのフォーマットを変換します。詳細は、413 ページの 「 簡略変換関数 」 を参照してください。

CHAR
数値コードに基づいて文字を取得します。

COMPACTFORMAT
数値を短縮形式で表示する K、M、B、T の文字を使用して、数値を文字値に変換します。

CTRLCHAR
非表示制御文字を取得します。

DT_FORMAT
日付値または日付時間値を文字値に変換します。

FPRINT
数値、日付値、日付時間値を文字列に変換します。

HEXTYPE
入力値の 16 進数表記を取得します。

PHONETIC
音声キーを取得します。

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。詳細は、427 ページの 「 フォーマット変換関数 」 を参照してください。

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。

FPRINT
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 の 10 進コードに対応する文字を取得します。

ITONUM
FOCUS 以外の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
FOCUS 以外の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。

PTOA
パック 10 進数を文字フォーマットに変換します。

TSTOPACK
Microsoft SQL Server または Sybase の TIMESTAMP フィールド (増分カウンタを格納するフィールド) をパック 10 進数に変換します。

UFMT
文字フィールド値の文字列を 16 進数表現に変換します。

XTPACK
最大で有効数字 31 桁のパック 10 進数値を、10 進数のデータを保持したまま文字フィールドに格納します。

簡略数値関数

次の関数は、簡略化されたパラメータリストを使用して、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。詳細は、451 ページの 「 簡略数値関数 」 を参照してください。

CEILING
特定の値以上の最小整数値を返します。

EXPONENT
定数 *e* を指数でべき乗します。

FLOOR
特定の値以下の最大整数値を返します。

MOD
除算の剰余を計算します。

POWER
数値を指数でべき乗します。

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。詳細は、465 ページの 「 数値関数 」 を参照してください。

ABS
数値の絶対値を返します。

ASIS
ダイアログマネージャでブランクと 0 (ゼロ) を区別します。

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。

EXPN
指数 (科学) 表記の数値を評価する演算子です。詳細は、『ibi™ WebFOCUS® Language リファレンス』の「式の使用」を参照してください。

FMLINFO
FML レポートで各行に関連付けられた FOR 値を返します。

FMLLIST
FML リクエスト内の各行のタグの完全なリストを含む文字列を返します。

FMLFOR
FML リクエスト内の各行のタグ値を取得します。

FMLCAP
FML 階層リクエスト内の各行のキャプション値を返します。

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、最小値をそれぞれ返します。

MIRR
定期的キャッシュフローの修正内部利益率を計算します。

NORMSDST および NORMSINV
標準正規分布曲線の計算を実行します。

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。

簡略統計関数

次の関数は、統計関数を実行します。詳細は、497 ページの 「 簡略統計関数 」 を参照してください。

CORRELATION
2 つの独立したデータセット間の相関度を計算します。

KMEANS_CLUSTER
最近傍の平均値に基づいて観測値をクラスタに分割します。

MULTIREGRESS
複数フィールドに基づいて線形重回帰フィールドを計算します。

OUTLIER
1.5 x IQR の原則を使用して数値フィールドの異常値を検出します。

RSERVE
R スクリプトを実行します。

STDDEV
一連のデータ値の標準偏差を計算します。

機械学習 (Python ベース) 関数

次の関数は、分類および回帰分析を実行する Python スクリプトを実行します。詳細は、509 ページの 「 機械学習 (Python ベース) 関数 」 を参照してください。

ANOMALY_IF
アイソレーションフォレストを使用して異常値を検出します。

CLASSIFY_BLR
予測子によって範囲が定められた空間の 2 クラス間の最適な線形分離を導き、クラス割り当て (1 または 0) かクラス 1 の確率のいずれかを返します。

CLASSIFY_KNN
k 最近傍の最頻クラスを割り当てることで、任意のデータポイントにクラスメンバーシップを割り当てます

CLASSIFY_RF
ランダムフォレスト (決定木群) を作成し、個々の予測の平均予測を返します。

CLASSIFY_XGB
ランダムフォレスト (決定木群) を作成し、新しいツリーのそれぞれが前の木群の予測精度の向上を試みます。

REGRESS_KNN
k 最近傍のターゲット値の平均を割り当てることで、任意のデータポイントのターゲット値を予測します。

REGRESS_POLY
予測子フィールドの多項式にターゲットフィールドを適合させます。

REGRESS_RF
ランダムフォレスト (決定木群) を作成し、個々の分類予測の多数決予測を返します。

REGRESS_XGB
ランダムフォレスト (決定木群) を作成し、新しいツリーのそれぞれが前の木群の予測精度の向上を試みます。

RUN_MODEL、RUN_MODEL2
RUN_MODEL 関数は、モデルの実行時と作成時に使用されたデータソースのフィールド名が同一の場合に、保存済みモデルの実行に使用します。RUN_MODEL2 関数は、モデルの実行時と作成時に使用されたデータソースのフィールド名が異なる場合に、保存済みモデルの実行に使用します。

簡略システム関数

次の関数は、簡略化されたパラメータリストを使用し、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ついての情報を取得したり、システムサービスを使用したりすることを可能にします。詳細は、543 ページの 「 簡略システム関数 」 を参照してください。

EDAPRINT
EDAPRINT ログファイルにカスタムメッセージを挿入します。

ENCRYPT
パスワードを暗号化します。

GETENV
環境変数の値を取得します。

PUTENV
環境変数に値を割り当てます。

SLACK
WebFOCUS リクエストから Slack チャンネルにメッセージを投稿します。

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ついての情報を取得したり、システムサービスを使用したりすることを可能にします。詳細は、549 ページの 「 システム関数 」 を参照してください。

CLSDDREC
ファイルを閉じて、開いているファイルの情報を格納しているメモリを解放します。

FEXERR
WebFOCUS エラーメッセージを取得します。

GETCOOKI
ブラウザの Cookie 値を取得します。

GETHEADR
HTTP ヘッダ変数値を取得します。

GETUSER
接続ユーザの ID を取得します。

SLEEP
指定した時間 (秒数) だけ実行を保留します。

SYSTEM
DOS プログラム、DOS バッチプログラム、または Windows アプリケーションを呼び出します。この関数は WebFOCUS のみで使用できます。

利用可能なオペレーティングシステム - Windows

簡略地理関数

次の関数は、さまざまなタイプの位置データに対して位置ベースの演算を実行し、ジオコードされた地点を返します。詳細は、567 ページの 「 簡略地理関数 」 を参照してください。

GIS_DISTANCE
2 地点間の距離を計算します。

GIS_DRIVE_ROUTE
2 地点間の走行経路を計算します。

GIS_POINT
地点を作成します。

GIS_GEOCODE_ADDR
完全な住所をジオコードします。

GIS_GEOCODE_ADDR_CITY
番地、市、州をジオコードします。

GIS_GEOCODE_ADDR_POSTAL
番地、郵便番号をジオコードします。

GIS_GEOMETRY
JSON ジオメトリオブジェクトを作成します。

GIS_IN_POLYGON
複雑な多角形内の地点の有無を特定します。

GIS_LINE
JSON 線を作成します。

GIS_REVERSE_COORDINATE
緯度と経度、およびコンポーネント名から、適切な地理コンポーネントを取得します。

GIS_SERVICE_AREA
特定の地点を囲む領域を計算します。

GIS_SERV_AREA_XY
特定の座標点を囲む領域を計算します。

三角関数

三角関数は、三角関数計算、逆三角関数、角度変換関数を提供します。詳細は、611 ページの 「 三角関数 」 を参照してください。

ACOS
ACOS (アークコサイン) 関数は、指定されたラジアン単位のコサインに基づいて、0 (ゼロ) から pi ラジアンの間の角度を返します。

ASIN
ASIN (アークサイン) 関数は、指定されたラジアン単位のサインに基づいて、-(pi/2) ラジアンと pi/2 ラジアンの間の角度を返します。

ATAN
ATAN (アークタンジェント) 関数は、指定されたラジアン単位のタンジェントに基づいて、-(pi/2) ラジアンと pi/2 ラジアンの間の角度を返します。

ATAN2
ATAN2 (アークタンジェント 2) 関数は、指定されたラジアン単位のタンジェントの座標に基づいて、-pi ラジアンと pi ラジアンの間の角度を返します。

COS
COS 関数は、指定されたラジアン単位の角度に基づいて、その角度のコサインを計算します。

COT
COT 関数は、指定されたラジアン単位の角度に基づいて、その角度のコタンジェントを計算します。

DEGREES
角度 (ラジアン) を角度 (度単位) に変換します。

PI
定数 pi を浮動小数点数として返します。

RADIANS
角度 (度単位) を角度 (ラジアン単位) に変換します。

SIN

SIN 関数は、指定されたラジアン単位の角度に基づいて、その角度のサインを計算します。

TAN

TAN 関数は、指定されたラジアン単位の角度に基づいて、その角度のタンジェントを計算します。

ASCII 文字コード表

下表は、ASCII 文字セットの主要な表示可能文字と、対応するコード番号を示します。拡張 ASCII コード (127 以降) は含まれません。

10 進コード	ASCII	
33	!	感嘆符
34	"	二重引用符
35	#	ナンバー
36	$	ドル
37	%	パーセント
38	&	アンパサンド
39	'	アポストロフィ
40	(	左括弧
41	)	右括弧
42	*	アスタリスク
43	+	プラス符号
44	,	カンマ
45	-	ハイフン
46	.	ピリオド
47	/	スラッシュ
48	0	0

10 進コード	ASCII	
49	1	1
50	2	2
51	3	3
52	4	4
53	5	5
54	6	6
55	7	7
56	8	8
57	9	9
58	:	コロン
59	;	セミコロン
60	<	不等号 (より小さい)
61	=	等号 (等しい)
62	>	不等号 (より大きい)
63	?	疑問符
64	@	アットマーク
65	A	A
66	B	B
67	C	C
68	D	D
69	E	E
70	F	F

10 進コード	ASCII	
71	G	G
72	H	H
73	I	I
74	J	J
75	K	K
76	L	L
77	M	M
78	N	N
79	O	O
80	P	P
81	Q	Q
82	R	R
83	S	S
84	T	T
85	U	U
86	V	V
87	W	W
88	X	X
89	Y	Y
90	Z	Z
91	[	左大括弧
92	¥	円

10 進コード	ASCII	
93	]	右大括弧
94	^	アクサンシルコンフレクス
95	_	アンダースコア
96	`	低アクセント
97	a	a
98	b	b
99	c	c
100	d	d
101	e	e
102	f	f
103	g	g
104	h	h
105	i	i
106	j	j
107	k	k
108	l	l
109	m	m
110	n	n
111	o	o
112	p	p
113	q	q
114	r	r

10 進コード	ASCII	
115	s	s
116	t	t
117	u	u
118	v	v
119	w	w
120	x	x
121	y	y
122	z	z
123	{	左中括弧
124	|	縦線
125	}	右中括弧
126	~	チルダ

3

関数へのアクセスと呼び出し

この章では、関数に引数を指定する際に、考慮すべき事項について説明します。また、関数をコマンド内で使用する方法、および外部に格納されている関数にアクセスする方法についても説明します。

トピックス

関数の呼び出し

COMPUTE、DEFINE、または VALIDATE コマンドから関数を呼び出すことができます。また、ダイアログマネージャコマンド、Financial Modeling Language (FML) コマンドから関数を呼び出すことも可能です。関数を呼び出すには、関数名および引数が必要です。外部関数では出力フィールドも必要です。

外部関数についての詳細は、20 ページの 「 関数カテゴリ 」 を参照してください。

構文　関数の呼び出し

```
function(arg1, arg2, ... [outfield])
```

説明

`function`
　関数の名前です。

`arg1, arg2, ...`
引数です。

`outfield`
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。この引数は、外部関数のみで必要です。

ダイアログマネージャでは、フォーマットを指定する必要があります。

構文　出力のフィールドへの格納

```
COMPUTE field/fmt = function(input1, input2,... [outfield]);
```

または

```
DEFINE FILE file
field/fmt = function(input1, input2,... [outfield]);
```

または

```
-SET &var = function(input1, input2,... [outfield]);
```

説明

`DEFINE`
一時項目 (DEFINE) を作成します。一時項目 (DEFINE) は、リクエスト内でソースフィールドの実データ同様に使用することができます。

`COMPUTE`
リクエスト内の 1 つ以上の一時項目 (COMPUTE) です。このフィールドはすべてのレコードの選択、ソート、集計の後に計算されます。

`field`
数値を含むフィールドです。

`file`
一時項目 (DEFINE) が作成されるファイルです。

`var`
結果を含む変数です。

`fmt`
結果を格納するフィールドのフォーマットです。

`function`
8 バイト以内の関数名です。

`input1, input2,...`
入力引数です。関数処理で使用するデータ値またはフィールドです。引数についての詳細は、47 ページの 「 関数の引数指定 」 を参照してください。

`outfield`
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。この引数は、外部関数のみで必要です。

ダイアログマネージャでは、フォーマットを指定する必要があります。

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、AnV 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
  ```

- ダイアログマネージャ変数 (例、&CODE や &DDNAME など)。
- 一重引用符 (') で囲んだ出力値フォーマット。
- 他の関数。

- ラベル、または R や E などの他の行列参照。FML RECAP コマンド内で関数を呼び出す場合、RECAP 演算名。

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。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。リテラルは、ダイアログマネージャの RUN コマンドをサポートするオペレーティングシステムで指定される場合 (例、MVS RUN) を除き、一重引用符 (') で囲みます。

- **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。

  注意：コンチネンタル 10 進表記 (CDN=ON) では、複数の数値引数は、カンマ (,) とそれに続くブランクで区切る必要があります。

- **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、YRTHRESH、DEFCENT パラメータ設定に基づいて割り当てられます。

引数の長さ

引数は関数へ参照として渡されます。引数のメモリロケーションが渡され、引数の長さを識別するものは与えられません。

文字列には引数の長さを指定する必要があります。関数には入力と出力の長さが必要なもの (例、SUBSTR) があり、それ以外は両方の引数に同一の長さを使用 (例、UPCASE) します。

すべての長さを正しく指定するよう注意が必要です。長さに誤りがあると、正しい結果が得られない原因となります。

- 実際の長さよりも小さい長さを指定すると、文字列の一部が使用されます。たとえば、引数「ABCDEF」に対して「3」という長さを指定すると、関数は文字列「ABC」を処理します。
- 必要以上の長さを指定すると、メモリ上のこの長さまでのものがすべて含められます。たとえば、「ABC」という引数を渡す際に長さ「6」を指定すると、関数は、「ABC」で始まる文字列に加え、メモリ上の次の 3 バイトも処理します。追加される次の 3 バイトは、メモリ使用状況によって異なります。

オペレーティングシステムのルーチンの中には、長さを正しく指定しないと、不適切なフォーマットのメモリ領域に読み込むものがあります。

引数の数と順序

必要な引数の数は、関数ごとに異なります。製品に付属の関数では、最大 6 つの引数が必要です。ユーザが作成するサブルーチンでは、出力引数を含めて最大 200 個の引数が必要です。200 個を超える引数が必要な場合、これらを関数に渡すためには、複数の呼び出しを使用しなければなりません。

引数は各関数の構文内に示す順序で指定する必要があります。順序は関数により異なります。

関数パラメータの検証

USERFCHK の設定は、DEFINE 関数と 付属の関数の引数に適用する検証レベルを制御します。このパラメータは、パラメータの個数の検証には影響しませんが、常に正しい数字を指定する必要があります。

関数は通常、特定のタイプ、または他のパラメータ値に依存する長さのパラメータを期待します。状況によっては、パラメータの長さを切り捨てることでこれらの規則を適用し、結果として実行時のエラー生成を回避することも考えられます。

検証レベルと実行可能な有効フォーマットへの変換は、関数ごとに異なります。通常、次の 2 つの状況では、変換に問題は発生しません。

- 数値パラメータで文字パラメータの最大サイズを指定し、入力される文字列が指定サイズよりも大きい場合、文字列は切り捨てられます。
- 数値リテラルとして入力するパラメータにパラメータの最大サイズよりも大きい値が指定される場合、適切な値に縮小することができます。

構文　パラメータ検証の有効化

パラメータの検証は、DEFINE 関数と製品に付属の関数のみで有効にすることができます。内部的な関数と同じ名前の関数がローカルに書き込まれている場合、USERFNS 設定がどの関数を使用するのかを決定します。

```
SET USERFNS= {SYSTEM|LOCAL}
```

説明

SYSTEM
付属の関数を優先します。デフォルト値は SYSTEM です。パラメータの検証を実行するには、この設定を使用する必要があります。

LOCAL
ローカルで作成した関数を優先します。この設定を有効にした場合、パラメータの検証は実行されません。

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　関数パラメータ検証の制御

FOCPARM、FOCPROF、コマンドライン、プロシジャ、または ON TABLE コマンドで、次のコマンドを発行します。USERFNS=SYSTEM 設定が有効になっている必要があります。

```
SET USERFCHK = setting
```

説明

setting
次のいずれかの値です。

- **ON** - デフォルト値です。リクエスト内のパラメータを検証しますが、マスターファイルの DEFINE で使用している関数のパラメータは検証しません。パラメータの長さが妥当でない場合は、問題の解決を試みます。問題を解決できない場合、エラーメッセージが生成され、影響する式の評価は終了します。

  マスターファイルで指定した関数のパラメータは検証されないため、DEFINE フィールドを使用したリクエストを次に送信するまで、これらの関数のエラーは生成されません。問題が発生した場合は、次のメッセージが生成されます。

  ```
  (FOC003) フィールド名に誤りがあります：
  ```

- **OFF** - 次の場合を除き、パラメータは検証されません。
  - パラメータが長すぎるため、演算用コードが格納されたメモリ領域を上書きする可能性がある場合は、サイズを自動的に小さくします。この場合、メッセージは発行されません。
  - 文字パラメータが短すぎる場合は、ブランクを追加して長さを修正します。

  注意

  - OFF 設定は、将来廃止される予定です。
  - パラメータの検証を無効にすると予期しない問題が発生する可能性があるため、OFF 設定は使用しないでください。
- **FULL** - ON と同一ですが、マスターファイルの DEFINE で使用する関数のパラメータも検証します。
- **ALERT** - リクエスト内のパラメータを検証します。問題が検知されても処理は中断されません。マスターファイルの DEFINE で使用する関数のパラメータは検証しません。パラメータの長さが妥当でない場合は、問題の修正をバックグラウンドで試みます。問題が修正されてもメッセージは表示されません。問題を修正できない場合は、警告メッセージが生成されます。その後、設定が OFF のときと同様に実行は継続されますが、正しい結果は取得できない可能性があります。

  注意

  - 指定されたパラメータのタイプが正しくない場合、検証は失敗し、処理が終了します。
  - サブルーチンの処理中に発生したエラー (システムレベルの致命的なエラーを除く) は、未変更のリターンパラメータ (サブルーチンコールの最終パラメータ) を返すことで、呼び出し元ルーチンに通知されます。文字フィールドの出力では、このエラーは常にブランクとして転送されます。

例　修正可能なエラーのあるパラメータの検証

次のリクエストは、SUBSTR 関数を使用して、TITLE フィールドの文字列から、開始位置が 6、終了位置が 14 であるサブ文字列を抽出します。5 つ目の引数ではサブ文字列の長さ 500 が指定されていますが、これは最大値の 9 を超えています。

```
SET USERFCHK = ON
TABLE FILE MOVIES
PRINT TITLE
COMPUTE
  NEWTITLE/A9  = SUBSTR(39, TITLE, 6  ,14, 500, NEWTITLE);
WHERE CATEGORY EQ 'CHILDREN'
END
```

USERFCHK=ON または USERFCHK=OFF でリクエストを実行するときは、長さが正しくない場合は修正され、リクエストは処理を継続します。

```
TITLE                                          NEWTITLE
-----                                          --------
SMURFS, THE                                    S, THE
SHAGGY DOG, THE                                Y DOG, TH
SCOOBY-DOO-A DOG IN THE RUFF                   Y-DOO-A D
ALICE IN WONDERLAND                             IN WONDE
SESAME STREET-BEDTIME STORIES AND SONGS        E STREET-
ROMPER ROOM-ASK MISS MOLLY                     R ROOM-AS
SLEEPING BEAUTY                                ING BEAUT
BAMBI
```

例　修正不可能なエラーのあるパラメータの検証

次のリクエストは、SUBSTR の最後の引数のデータタイプが正しくありません。このパラメータには、文字フィールドを指定するか、抽出するサブ文字列のフォーマットを指定する必要があります。

```
SET USERFCHK = ON
TABLE FILE MOVIES
PRINT TITLE
COMPUTE
  NEWTITLE/F9  = SUBSTR(39, TITLE, 6  ,14, 500, 'F9');
WHERE CATEGORY EQ 'CHILDREN'
END
```

- USERFCHK=ON でリクエストを実行するときは、メッセージが生成され、リクエストは終了します。

```
エラーのある行       5  プロシジャ名 ADHOCRQ FOCEXEC *
(FOC36355) ユーザ関数 SUBSTR に対する無効な引数タイプ #6 です。
(FOC009) リクエストが完結していません。
コマンドの終わりまで処理をバイパスします
```

- USERFCHK=OFF でリクエストを実行するときは、検証は実行されず、メッセージは生成されません。リクエストは実行され、正しくない結果が生成されます。環境によっては、この種のエラーはアプリケーションの異常終了の原因となる可能性があります。

```
DIRECTOR          TITLE                                      NEWTITLE
--------          -----                                      --------
                  SMURFS, THE                               *********
BARTON C.         SHAGGY DOG, THE                           *********
                  SCOOBY-DOO-A DOG IN THE RUFF              *********
GEROMINI          ALICE IN WONDERLAND                               1
                  SESAME STREET-BEDTIME STORIES AND SONGS    -265774
                  ROMPER ROOM-ASK MISS MOLLY                *********
DISNEY W.         SLEEPING BEAUTY                           *********
DISNEY W.         BAMBI                                             0
```

DEFINE、COMPUTE、VALIDATE コマンドからの関数呼び出し

関数は、DEFINE コマンド、マスターファイル属性、COMPUTE コマンド、または VALIDATE コマンドから呼び出すことができます。

構文　COMPUTE、DEFINE、VALIDATE からの関数の呼び出し

```
DEFINE [FILE filename]
tempfield[/format] = function(input1, input2, input3, ... [outfield]);
COMPUTE
tempfield[/format] = function(input1, input2, input3, ... [outfield]);
VALIDATE
tempfield[/format] = function(input1, input2, input3, ... [outfield]);
```

説明

`filename`
使用するデータソースです。

`tempfield`
DEFINE または COMPUTE で作成した一時項目です。これは *outfield* で指定したフィールドと同一です。関数呼び出しで出力値のフォーマットを *outfield* で指定する場合、一時項目のフォーマットは *outfield* 引数と一致させる必要があります。

`format`
一時項目のフォーマットです。フィールドをはじめて作成する場合、このフォーマットは必須です。それ以外の場合はオプションです。デフォルト値は D12.2 です。

`function`
関数の名前です。

`input1, input2, input3...`
引数です。

`outfield`
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。これは外部関数のみで必須です。

ダイアログマネージャでは、フォーマットを指定する必要があります。

ダイアログマネージャコマンドからの関数の呼び出し

ダイアログマネージャで関数を呼び出すには、次の方法があります。

- -SET コマンドからの呼び出し。関数の結果を変数に格納します。詳細は、54 ページの「 関数結果の変数への割り当て 」 を参照してください。
- -IF コマンドからの呼び出し。詳細は、59 ページの 「 WHERE/IF 条件での関数の呼び出し 」 を参照してください。
- オペレーティングシステムの -RUN コマンドからの呼び出し。詳細は、57 ページの 「 オペレーティングシステム RUN コマンドからの関数の呼び出し 」 を参照してください。

ダイアログマネージャは数値引数を倍精度フォーマットに変換します。これは、引数の値が数値の場合に発生し、関数で使用可能なフォーマットとは無関係です。このため、ダイアログマネージャで関数の引数を指定する場合は、注意が必要です。

文字列を受容する関数に数値文字列を入力すると、倍精度浮動小数点数へ変換されるため、正しくない結果が発生します。この問題を解決するには、文字列の末尾に数値ではない文字を追加します。ただし、この追加文字は引数の長さから除外します。

ダイアログマネージャの日付変数 (例、&YYMD) は、レガシー日付を日付フォーマット (基準日からのオフセット) ではなく、文字フォーマットで返します。レガシー日付の代わりに日付オフセットを必要とする関数の場合、日付変数を引数として使用する前に、DATECVT 関数を使用して日付変数を日付オフセットに変換する必要があります。次に、DATECVT 関数を再度使用して、結果をレガシー日付に変換します。以下はその例です。

```
-SET &TODAY_OFFSET=DATECVT(&YYMD , 'I8YYMD' , 'YYMD');
-SET &BEG_CUR_YR=DATEMOV(&TODAY_OFFSET.EVAL , 'BOY');
-SET &CLOSE_DTBOY=DATECVT(&BEG_CUR_YR.EVAL , 'YYMD' , 'I8YYMD')';
```

関数結果の変数への割り当て

-SET コマンドを使用すると、関数の結果を変数に格納することができます。

ダイアログマネージャ変数は、文字データのみを格納することができます。関数からダイアログマネージャ変数に返される値が数値の場合、値は整数に切り捨てられ、文字フォーマットに変換されてから変数に格納されます。

構文　関数の結果の変数への割り当て

```
-SET &variable = function(arg1, arg2[.LENGTH],..., 'format');
```

説明

`variable`

結果が割り当てられる変数です。

`function`

関数です。

`arg1, arg2`

関数の引数です。

`.LENGTH`

変数の長さを返します。入力引数として文字列の長さをとる関数では、文字列の入力を要求して .LENGTH 接尾語で長さを決定することができます。

`format`

結果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。.EVAL 接尾語を使用しない限り、出力引数にダイアログマネージャ変数を指定することはできません。ただし、変数を入力引数として指定することは可能です。

例　-SET コマンドからの関数の呼び出し

AYMD は、&INDATE の値に 14 日を加えます。&INDATE 変数は、プロシジャにより、6 桁の年月日フォーマットに設定済みです。

```
-SET &OUTDATE = AYMD(&INDATE, 14, 'I6');
```

出力日付のフォーマットは、6 桁の整数 (I6) です。フォーマットは出力が整数であることを示していますが、この値は文字列として &OUTDATE 変数に格納されます。このため、&OUTDATE の値を表示すると、年、月、日の間にスラッシュ (/) はありません。

関数の結果に基づく分岐の設定

ダイアログマネージャの -IF コマンドから関数を呼び出すことにより、関数の結果に基づいて分岐を設定することができます。

分岐コマンドが複数行にわたる場合、先頭フィールドにハイフン (-) を入力することで、次の行に続けることができます。

構文　関数の結果関数の結果に基づく分岐の設定

```
-IF function(args) relation expression GOTO label1 [ELSE GOTO label2];
```

説明

`function`

関数です。

`args`

引数です。

`relation`

EQ や LE など、関数と式との関係を表す演算子です。

`expression`

値、論理式、または関数です。カンマ (,) やブランクが含まれていない限り、リテラルを一重引用符 (') で囲むことはできません。

`label1, label2`

12 バイト以内のユーザ定義の名前です。-QUIT または -EXIT を除くダイアログマネージャコマンド名やブランクを含めることはできません。関数、算術演算子、論理演算子と間違えやすい語句を使用することはできません。

label テキストは、プロシジャ内の -IF 条件の前後に記述することができます。

`ELSE GOTO`

-IF テストに失敗した場合にコントロールを label2 に渡します。

例　関数の結果に基づく分岐の設定

AYMD 関数の結果は -IF テストの条件を指定します。関数の結果に応じて、2 つのリクエストのいずれかが実行されます。

```
   -LOOP
1. -IF &INDATE EQ 0 GOTO EXIT;
2. -SET &WEEKDAY = DOWK(&INDATE, 'A4');
3. -TYPE START DATE IS &WEEKDAY &INDATE
4. -IF AYMD(&INDATE, &DAYS, 'I6YMD') LT 960101 GOTO EARLY;
5. -TYPE LONG PROJECT
   -*EX LONGPROJ
   -RUN
   -GOTO EXIT
6. -EARLY
   -TYPE SHORT PROJECT
   -*EX SHRTPROJ
   -RUN
   -EXIT
```

プロシジャの処理は次のとおりです。

1. 0 (ゼロ) を入力すると、制御は -EXIT に渡され、処理は終了します。
2. DOWK 関数により開始日の曜日が取得されます。
3. -TYPE コマンドにより、プロジェクト開始日の曜日と日付が表示されます。
4. AYMD 関数によりプロジェクト終了日が計算されます。この日付が 1996 年 1 月 1 日以前の場合、-IF コマンドによりラベル「EARLY」への分岐が実行されます。
5. 1996 年 1 月 1 日より後にプロジェクトが終了する場合、TYPE コマンドにより「LONG PROJECT」という語句が表示され、処理は終了します。
6. プロシジャがラベル「EARLY」に分岐する場合、TYPE コマンドにより「SHORT PROJECT」という語句が表示され、処理は終了します。

オペレーティングシステム RUN コマンドからの関数の呼び出し

文字フォーマットの引数のみを含む関数は、ダイアログマネージャコマンドの -TSO RUN または -MVS RUN コマンドで呼び出すことができます。この種の関数は特定の作業を実行しますが、通常、値を返すことはありません。

関数が数値フォーマットの引数をとる場合、ATODBL 関数により、倍精度浮動小数点数フォーマットに変換する必要があります。これは、-SET コマンドとは異なり、オペレーティングシステムの RUN コマンドは、数値引数を倍精度フォーマットに自動的に変換しないためです。

構文　オペレーティングシステム -RUN コマンドからの関数の呼び出し

```
{-TSO|-MVS} RUN function, input1, input2, ... [,&output]
```

説明

`-TSO|-MVS`

オペレーティングシステムです。

`function`

関数の名前です。

`input1, input2,...`

引数です。 関数名と各引数はカンマ (,) で区切ります。文字リテラルは一重引用符 (') で囲まないでください。文字列の長さを引数にとる関数の場合、文字列の入力を要求し、.LENGTH 接尾語で長さをテストすることができます。

`&output`

ダイアログマネージャ変数です。関数が値を返す場合、この引数を含めます。それ以外の場合は、省略します。出力変数を指定する場合、-SET コマンドにより、この長さを定義しておく必要があります。

たとえば、関数が 8 バイト長の値を返す場合は、関数を呼び出す前に、一重引用符 (') で囲んだ 8 バイト長の変数を定義しておきます。

```
-SET &output = '12345678';
```

例　オペレーティングシステム -RUN コマンドからの関数の呼び出し

次の例では、-MVS RUN コマンドから CHGDAT 関数を呼び出します。

```
-SET &RESULT = '12345678901234567';
-MVS RUN CHGDAT, YYMD., MXDYY, &YYMD, &RESULT
-TYPE &RESULT
```

別関数からの関数の呼び出し

関数は、別の関数の引数として使用することもできます。

構文　別関数からの関数の呼び出し

```
field = function([arguments,] function2[arguments2,] arguments);
```

説明

`field`

関数の結果を格納するフィールドです。

`function`

関数です。

`arguments`

function の引数です。

`function2`

function の引数となる関数です。

`arguments2`

function2 の引数です。

例　別関数からの関数の呼び出し

次の例では、AYMD 関数は YMD 関数の引数です。

```
-SET &DIFF = YMD(&YYMD, AYMD(&YYMD, 4, 'I8'));
```

WHERE/IF 条件での関数の呼び出し

WHERE 条件または IF 条件で関数を呼び出すことができます。その際、関数の出力値がテスト値と比較されます。

構文　WHERE 条件での関数の呼び出し

```
WHERE function relation expression
```

説明

`function`

関数です。

`relation`

EQ や LE など、関数と式との関係を表す演算子です。

expression

定数、フィールド、または関数です。リテラルは一重引用符 (') で囲みます。

構文　IF 条件での関数の呼び出し

```
IF function relation value
```

説明

function

関数です。

relation

EQ や LE など、関数と式との関係を表す演算子です。

value

定数です。DEFINE または COMPUTE コマンドでは、値は一重引用符 (') で囲みます。

例　WHERE 条件での関数の呼び出し

SUBSTR 関数は、LAST_NAME の先頭の 2 バイトをサブ文字列として抽出します。このリクエストは、このサブ文字列が MC である従業員の名前と給与を表示します。

```
TABLE FILE EMPLOYEE
PRINT FIRST_NAME LAST_NAME CURR_SAL
WHERE SUBSTR(15, LAST_NAME, 1, 2, 2, 'A2') IS 'MC';
END
```

出力結果は次のとおりです。

```
FIRST_NAME   LAST_NAME          CURR_SAL
----------   ---------          --------
JOHN         MCCOY            $18,480.00
ROGER        MCKNIGHT         $16,100.00
```

複合 IF コマンドまたは演算の使用

演算や複合 IF コマンド内では、出力値のフォーマットを指定する必要があります。これには次の 2 通りの方法があります。

- 別のコマンドでフォーマットを定義しておく。次の例では、AMOUNT フィールドはフォーマット D8.2 として定義済みであり、この関数は AMOUNT 出力フィールドに値を返します。IF コマンドは AMOUNT の値をテストし、結果を一時項目 AMOUNT_FLAG に格納します。

```
COMPUTE
AMOUNT/D8.2 =;
AMOUNT_FLAG/A5 = IF function(input1, input2, AMOUNT) GE 500
   THEN 'LARGE' ELSE 'SMALL';
```

- フォーマットは、関数呼び出し内の最後の引数として指定します。次の例では、このコマンドにより直接、戻り値がテストされます。これは、関数が戻り値のフォーマット D8.2 を定義するため可能になります。

```
DEFINE
AMOUNT_FLAG/A5 = IF function(input1, input2, 'D8.2') GE 500
   THEN 'LARGE' ELSE 'SMALL';
```

WHEN 条件での関数の呼び出し

WHEN 条件では、関数をブール式の一部として呼び出すことができます。

構文　WHEN 条件での関数の呼び出し

```
WHEN({function|value} relation {function|value});
```

または

```
WHEN NOT(function)
```

説明

`function`

関数です。

`value`

値または論理式です。

`relation`

LE や GT など、関数と式との関係を表す演算子です。

例　WHEN 条件での関数の呼び出し

次のリクエストは LAST_NAME 内の値を CHKFMT 関数の結果と比較します。値が一致すると、ソート項目を指定している脚注を表示します。

```
TABLE FILE EMPLOYEE
PRINT DEPARTMENT BY LAST_NAME
ON LAST_NAME SUBFOOT
"*** LAST NAME <LAST_NAME DOES MATCH MASK"
WHEN NOT CHKFMT(15, LAST_NAME, 'SMITH          ', 'I6');
END
```

出力結果は次のとおりです。

```
LAST_NAME         DEPARTMENT
---------         ----------
BANNING           PRODUCTION
BLACKWOOD         MIS
CROSS             MIS
GREENSPAN         MIS
IRVING            PRODUCTION
JONES             MIS
MCCOY             MIS
MCKNIGHT          PRODUCTION
ROMANS            PRODUCTION
SMITH             MIS
                  PRODUCTION
*** LAST NAME SMITH DOES MATCH MASK
STEVENS           PRODUCTION
```

RECAP コマンドからの関数の呼び出し

FML の RECAP コマンドから関数を呼び出すことができます。

構文　RECAP コマンドからの関数の呼び出し

```
RECAP name[(n)|(n,m)|(n,m,i)][/format1] =
function(input1,...,['format2']);
```

説明

`name`

演算名です。

`n`

n で指定した列番号の値を表示します。この列番号を省略すると、すべての列に値が表示されます。

`n,m`

n から *m* で指定した番号すべての列に値を表示します。

`n,m,i`

n から *m* で指定した列番号の範囲において、*i* で指定した間隔で列に値を表示します。たとえば、*n* が 1、*m* が 5、*i* が 2 の場合、値は列 1、列 3、列 5 に表示されます。

`format1`

演算のフォーマットです。デフォルト値はレポートフィールドのフォーマットです。

`function`

関数です。

`input1,...`

入力引数です。数値定数、文字リテラル、行列参照 (R 表記、E 表記、またはラベル)、RECAP 演算名などがあります。

`format2`

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。演算のフォーマットがフィールド幅より大きい場合、値はアスタリスク (*) で表示されます。

例 RECAP コマンドでの関数の呼び出し

次のリクエストは AMOUNT フィールドの account 1010、account 1020、account 1030 をそれぞれラベル CASH、DEMAND、TIME で集計します。MAX 関数はこれらのアカウントの最大値を表示します。

```
TABLE FILE LEDGER
SUM AMOUNT FOR ACCOUNT
1010 AS 'CASH ON HAND'      LABEL CASH   OVER
1020 AS 'DEMAND DEPOSITS'   LABEL DEMAND OVER
1030 AS 'TIME DEPOSITS'     LABEL TIME   OVER
BAR                         OVER
RECAP MAXCASH = MAX(CASH, DEMAND, TIME); AS 'MAX CASH'
END
```

出力結果は次のとおりです。

```
                  AMOUNT
                  ------
CASH ON HAND       8,784
DEMAND DEPOSITS    4,494
TIME DEPOSITS      7,961
                  ------
MAX CASH           8,784
```

外部関数の格納とアクセス

内部関数はビルトイン関数であり、アクセスするために追加の作業は必要ありません。外部関数は、ロードライブラリ内に格納されており、ここから取得する必要があります。これらの外部関数へのアクセス方法は、プラットフォームにより異なります。ここで紹介する手法は、関数へのアクセス時に毎回使用する必要はありません。ロードライブラリへのアクセスは、インストール時に一度だけ設定します。

ユーザ定義のプライベートサブルーチンにアクセスすることもできます。サブルーチンの独自開発やカスタマイズを行い、サブルーチンのプライベートコレクションを所有する場合、これらは関数ライブラリには格納しないでください。別に保存することで、インストールを更新するたびに上書きされることを防止します。サブルーチンの作成についての詳細は、623 ページの 「 サブルーチンの作成 」 を参照してください。

UNIX 上での関数の格納とアクセス

追加の作業はありません。

Windows 上での関数の格納とアクセス

追加の作業はありません。

4

簡略分析関数

簡略分析関数を使用することで、内部マトリックスの複数行を使用した計算および検索を実行することができます。

トピックス

FORECAST_MOVAVE - 単純移動平均の使用

単純移動平均は、フィールドから抽出する値の数を指定して計算した算術平均です。連続したデータ値から新しい平均を計算するには、前回の計算で使用した最初の値を除外し、次のデータ値を計算に追加します。

単純移動平均は、時間とともに変化する株価の傾向分析に使用することもあります。その場合、株価の期間数を指定して平均を計算します。この指標の難点は、先へ進むとともに一番古い値を計算から除外するため、時間とともに過去の記録を失うことです。さらに、この方法では各点に同一の加重が加えられるため、上下の極値により平均値に歪みが生じます。

データ値の範囲外で値を予測する場合は、計算した傾向値を新しいデータ点として扱う移動平均を使用してその予測値が計算されます。

最初の完全な移動平均は、計算に n 個の値が必要になることから、n 番目のデータ点で得られます。これは「ラグ (遅れ)」と呼ばれます。ラグの行に対する移動平均値は次のように計算されます。移動平均フィールドの最初の値は最初のデータ値と等しく、移動平均フィールドの 2 つ目の値は最初の 2 つのデータ値の平均値と等しくするというように、移動平均の計算に使用する値の数が指定した数に達する n 行目までこれを繰り返します。

構文　単純移動平均フィールドの計算

```
FORECAST_MOVAVE(display, infield, interval,
 npredict, npoint1)
```

説明

`display`

キーワード

既存データを表す出力行に表示する値を指定します。有効な値には、次のものがあります。

- **INPUT_FIELD**　既存データを表す行に元のフィールド値を表示します。
- **MODEL_DATA**　既存データを表す行に一時項目 (COMPUTE) を表示します。

注意：同一リクエスト内に 2 つの独立した COMPUTE コマンドを作成し、それぞれ別の表示オプションを使用して、任意のフィールドに対して両方の出力タイプを表示することができます。

`infield`

任意の数値フィールドです。結果フィールドと同一のフィールドにすることも、異なるフィールドにすることもできます。ただし、日付時間フィールドおよび日付表示オプションの数値フィールドにすることはできません。

`interval`

それぞれのソートフィールド値に追加する増加値です。最後のデータ点の後に追加して次の値を作成します。この値は正の整数でなければなりません。降順にソートするには、BY HIGHEST 句を使用します。ソートフィールド値にこの数値を追加した結果が、ソートフィールドと同じフォーマットに変換されます。

日付フィールドでは、フォーマットの最小単位の要素によりその数字の認識方法が決まります。たとえば、フォーマットが YMD、MDY、DMY のいずれかの場合、間隔値の 2 は 2 日と認識され、フォーマットが YM の場合は間隔値の 2 は 2 か月と認識されます。

npredict

計算する FORECAST の予測数です。この値は、0 (ゼロ) 以上の整数でなければなりません。この値の 0 (ゼロ) は予測値が必要ないことを表し、それは非再帰的の FORECAST でのみサポートされます。

npoint1

MOVAVE を使用する場合、平均する値の数です。

例　新しい単純移動平均フィールドの計算

次のリクエストでは、「PERIOD」という名前の整数値を定義し、それを移動平均の独立変数として使用します。ここでは、取得したデータの範囲外に値が発生する 3 期間を予測します。レポート出力の MOVAVE フィールドには、既存のデータポイントについて計算された移動平均値が表示されます。

```
DEFINE FILE GGSALES
SDATE/YYM = DATE;
SYEAR/Y = SDATE;
SMONTH/M = SDATE;
PERIOD/I2 = SMONTH;
END
TABLE FILE GGSALES
SUM UNITS DOLLARS
COMPUTE  MOVAVE/D10.1= FORECAST_MOVAVE(MODEL_DATA, DOLLARS,1,3,3);
BY CATEGORY BY PERIOD
WHERE SYEAR EQ 97 AND CATEGORY NE 'Gifts'
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果は次のとおりです。

Category	PERIOD	Unit Sales	Dollar Sales	MOVAVE
Coffee	1	61666	801123	801,123.0
	2	54870	682340	741,731.5
	3	61608	765078	749,513.7
	4	57050	691274	712,897.3
	5	59229	720444	725,598.7
	6	58466	742457	718,058.3
	7	60771	747253	736,718.0
	8	54633	655896	715,202.0
	9	57829	730317	711,155.3
	10	57012	724412	703,541.7
	11	51110	620264	691,664.3
	12	58981	762328	702,334.7
	13	0	0	694,975.6
	14	0	0	719,879.4
	15	0	0	705,729.9
Food	1	54394	672727	672,727.0
	2	54894	699073	685,900.0
	3	52713	642802	671,534.0
	4	58026	718514	686,796.3
	5	53289	660740	674,018.7
	6	58742	734705	704,653.0
	7	60127	760586	718,677.0
	8	55622	695235	730,175.3
	9	55787	683140	712,987.0
	10	57340	713768	697,381.0
	11	57459	710138	702,348.7
	12	57290	705315	709,740.3
	13	0	0	708,397.8
	14	0	0	707,817.7
	15	0	0	708,651.9

このレポートでは、平均の計算に使用した値の数は 3 で、生成された PERIOD の値に対して UNITS および DOLLARS の値は存在しません。

それぞれの平均 (MOVAVE の値) は、DOLLARS 値が存在する場合はその値を使用して計算されます。移動平均の計算は次のように開始されます。

- 最初の MOVAVE の値 (801,123.0) は、最初の DOLLARS の値に等しくなります。

- 2 つ目の MOVAVE の値 (741,731.5) は、最初の 2 つの DOLLARS 値の平均に等しくなります。その計算は、(801,123 + 682,340) /2 です。
- 3 つ目の MOVAVE の値 (749,513.7) は、最初の 3 つの DOLLARS 値の平均に等しくなります。その計算は、(801,123 + 682,340 + 765,078) / 3 です。
- 4 つ目の MOVAVE の値 (712,897.3) は、2 つ目から 4 つ目までの DOLLARS 値の平均に等しくなります。その計算は、(682,340 + 765,078 + 691,274) /3 です。

指定された値の範囲外で値を予測する場合は、計算された MOVAVE の値が新しいデータ点として使用され、移動平均が引き続き計算されます。MOVAVE の予測値 (PERIOD 13 の 694,975.6 から開始) は、前回の MOVAVE の値を新しいデータ点として使用して計算されます。たとえば、最初の予測値 (694,975.6) は、11 および 12 番目の期間のデータ点 (620,264 および 762,328) と 12 番目の期間の移動平均 (702,334.7) の平均になります。実際の計算は、(620,264 + 762,328 + 702,334.7)/3 = 694,975 のようになります。

例　単純移動平均フィールドでの元のフィールド値の表示

次のリクエストでは、「PERIOD」という名前の整数値を定義し、それを移動平均の独立変数として使用します。ここでは、取得したデータの範囲外に値が発生する 3 期間を予測します。FORECAST パラメータリストの 1 つ目の引数として、INPUT_FIELD キーワードを使用します。傾向値はレポートに表示されません。レポートフィールドの予測値は、DOLLARS の実データ値の次に表示されます。

```
DEFINE FILE GGSALES
SDATE/YYM = DATE;
SYEAR/Y = SDATE;
SMONTH/M = SDATE;
PERIOD/I2 = SMONTH;
END
TABLE FILE GGSALES
SUM UNITS DOLLARS
COMPUTE MOVAVE/D10.1 = FORECAST_MOVAVE(INPUT_FIELD,DOLLARS,1,3,3);
BY CATEGORY BY PERIOD
WHERE SYEAR EQ 97 AND CATEGORY NE 'Gifts'
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果は次のとおりです。

Category	PERIOD	Unit Sales	Dollar Sales	MOVAVE
Coffee	1	61666	801123	801,123.0
	2	54870	682340	682,340.0
	3	61608	765078	765,078.0
	4	57050	691274	691,274.0
	5	59229	720444	720,444.0
	6	58466	742457	742,457.0
	7	60771	747253	747,253.0
	8	54633	655896	655,896.0
	9	57829	730317	730,317.0
	10	57012	724412	724,412.0
	11	51110	620264	620,264.0
	12	58981	762328	762,328.0
	13	0	0	694,975.6
	14	0	0	719,879.4
	15	0	0	705,729.9
Food	1	54394	672727	672,727.0
	2	54894	699073	699,073.0
	3	52713	642802	642,802.0
	4	58026	718514	718,514.0
	5	53289	660740	660,740.0
	6	58742	734705	734,705.0
	7	60127	760586	760,586.0
	8	55622	695235	695,235.0
	9	55787	683140	683,140.0
	10	57340	713768	713,768.0
	11	57459	710138	710,138.0
	12	57290	705315	705,315.0
	13	0	0	708,397.8
	14	0	0	707,817.7
	15	0	0	708,651.9

FORECAST_EXPAVE - 単純指数平滑法の使用

単純指数平滑法では、新しい値と古い値に適用する加重値を選択して平均を計算します。

最新の値に割り当てる加重値は、次の計算式で求められます。

```
k = 2/(1+n)
```

説明

`k`

最新の値に割り当てる加重値です。

n

1 より大きい整数値です。n の値が増加するにつれて、これまでの観察値 (つまり、データインスタンス) に割り当てた加重値の方が最新のものよりも大きくなります。

次の計算式により、指数移動平均 (EMA) の値を得るための計算が導き出されます。

```
EMA = (EMA * (1-k)) + (datavalue * k)
```

この式は、データソースの最新の値に係数 k を乗算し、現在の移動平均に係数 (1-k) を乗算することを表しています。これらの数値を集計して、新しい EMA を算出します。

注意：データ値をすべて使用し終わった場合は、ソートグループの最後のデータ値が次のデータ値として使用されます。

構文　単純指数平滑フィールドの計算

```
FORECAST_EXPAVE(display, infield, interval,
 npredict, npoint1)
```

説明

display

キーワード

既存データを表す出力行に表示する値を指定します。有効な値には、次のものがあります。

- **INPUT_FIELD**　既存データを表す行に元のフィールド値を表示します。
- **MODEL_DATA**　既存データを表す行に一時項目 (COMPUTE) を表示します。

注意：同一リクエスト内に 2 つの独立した COMPUTE コマンドを作成し、それぞれ別の表示オプションを使用して、任意のフィールドに対して両方の出力タイプを表示することができます。

infield

任意の数値フィールドです。結果フィールドと同一のフィールドにすることも、異なるフィールドにすることもできます。ただし、日付時間フィールドおよび日付表示オプションの数値フィールドにすることはできません。

interval

それぞれのソートフィールド値に追加する増加値です。最後のデータ点の後に追加して次の値を作成します。この値は正の整数でなければなりません。降順にソートするには、BY HIGHEST 句を使用します。ソートフィールド値にこの数値を追加した結果が、ソートフィールドと同じフォーマットに変換されます。

日付フィールドでは、フォーマットの最小単位の要素によりその数字の認識方法が決まります。たとえば、フォーマットが YMD、MDY、DMY のいずれかの場合、間隔値の 2 は 2 日と認識され、フォーマットが YM の場合は間隔値の 2 は 2 か月と認識されます。

`npredict`

計算する FORECAST の予測数です。この値は、0 (ゼロ) 以上の整数でなければなりません。この値の 0 (ゼロ) は予測値が必要ないことを表し、それは非再帰的の FORECAST でのみサポートされます。

`npoint1`

EXPAVE では、この数を使用して、平均の各要素に対する加重値を計算します。この値は正の自然数でなければなりません。次の計算式を使用して、加重値 k を計算します。

```
k=2/(1+npoint1)
```

例 単純指数平滑フィールドの計算

次の例では、「PERIOD」という名前の整数値を定義し、それを移動平均の独立変数として使用します。ここでは、取得したデータの範囲外に値が発生する 3 期間を予測します。

```
DEFINE FILE GGSALES
SDATE/YYM = DATE;
SYEAR/Y = SDATE;
SMONTH/M = SDATE;
PERIOD/I2 = SMONTH;
END
TABLE FILE GGSALES
SUM UNITS DOLLARS
COMPUTE EXPAVE/D10.1= FORECAST_EXPAVE(MODEL_DATA,DOLLARS,1,3,3);
BY CATEGORY BY PERIOD
WHERE SYEAR EQ 97 AND CATEGORY NE 'Gifts'
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果を示しています。

```
Category      PERIOD  Unit Sales  Dollar Sales        EXPAVE
--------      ------  ----------  ------------        ------
Coffee             1       61666        801123     801,123.0
                   2       54870        682340     741,731.5
                   3       61608        765078     753,404.8
                   4       57050        691274     722,339.4
                   5       59229        720444     721,391.7
                   6       58466        742457     731,924.3
                   7       60771        747253     739,588.7
                   8       54633        655896     697,742.3
                   9       57829        730317     714,029.7
                  10       57012        724412     719,220.8
                  11       51110        620264     669,742.4
                  12       58981        762328     716,035.2
                  13           0             0     739,181.6
                  14           0             0     750,754.8
                  15           0             0     756,541.4
Food               1       54394        672727     672,727.0
                   2       54894        699073     685,900.0
                   3       52713        642802     664,351.0
                   4       58026        718514     691,432.5
                   5       53289        660740     676,086.3
                   6       58742        734705     705,395.6
                   7       60127        760586     732,990.8
                   8       55622        695235     714,112.9
                   9       55787        683140     698,626.5
                  10       57340        713768     706,197.2
                  11       57459        710138     708,167.6
                  12       57290        705315     706,741.3
                  13           0             0     706,028.2
                  14           0             0     705,671.6
                  15           0             0     705,493.3
```

このレポートでは、CATEGORY 別に 3 つの EXPAVE の予想値が計算されます。データの範囲外の値に対して、前回の期間 (PERIOD) の値に間隔値 (1) が追加されて新しい期間 (PERIOD) の値が生成されます。

それぞれの平均 (EXPAVE の値) は、DOLLARS 値が存在する場合はその値を使用して計算されます。移動平均の計算は次のように開始されます。

- 最初の EXPAVE 値 (801,123.0) は、最初の DOLLARS 値と等しくなります。
- 2 つ目の EXPAVE 値 (741,731.5) は、次のように計算されます。なお、使用する小数の端数処理および小数点以下の桁数により、この例で得られた計算値がレポート出力に表示されるものと多少異なる場合があります。

  ```
  n=3 (number used to calculate weights)

  k = 2/(1+n) = 2/4 = 0.5

  EXPAVE = (EXPAVE*(1-k))+(new-DOLLARS*k) = (801123*0.5) + (682340*0.50) =
  400561.5 + 341170 = 741731.5
  ```

- 3 つ目の EXPAVE 値 (753,404.8) は、次のように計算されます。

```
EXPAVE = (EXPAVE*(1-k))+(new-DOLLARS*k) = (741731.5*0.5)+(765078*0.50) =
370865.75 + 382539 = 753404.75
```

FORECAST_DOUBLEXP - 二重指数平滑法の使用

二重指数平滑法は、繰り返すことなく時間の変化とともに増減するデータの傾向を考慮した指数移動平均です。この計算には、2 つの定数を用いた 2 つの方程式が使用されます。

- 最初の方程式は、現在の期間を考慮した式で、現在のデータ値と前回の平均の加重平均です。追加の要素 (b) は、前回期間の傾向値を表します。k は加重定数です。

```
DOUBLEXP(t) = k * datavalue(t) + (1-k) * ((DOUBLEXP(t-1) + b(t-1))
```

- 2 つ目の方程式は、傾向値を計算する式で、現在と前回の平均差と前期間の傾向値との加重平均です。b(t) は平均の傾向値を表します。g は加重定数です。

```
b(t) = g * (DOUBLEXP(t)-DOUBLEXP(t-1)) + (1 - g) * (b(t-1))
```

これら 2 つの方程式を解いて、平滑化した平均を求めます。平滑化した最初の平均が、最初のデータ値に設定されます。最初の傾向要素は 0 (ゼロ) に設定されます。一般的に、最善の結果を得るには、2 つの定数を選定する際に、データ値と計算した平均値の間の平均平方誤差 (MSE) を最小にする定数を選択するようにします。最適な定数を得るために、非線形の最適化手法を使用する方がよい場合があります。

二重指数平滑法では、次の方程式を使用してデータ点の範囲外で値を予測します。

```
forecast(t+m) = DOUBLEXP(t) + m * b(t)
```

説明

`m`

予測を行う期間数です。

構文　二重指数平滑フィールドの計算

```
FORECAST_DOUBLEXP(display, infield,
interval, npredict, npoint1, npoint2)
```

説明

`display`

キーワード

既存データを表す出力行に表示する値を指定します。有効な値には、次のものがあります。

- **INPUT_FIELD** 既存データを表す行に元のフィールド値を表示します。
- **MODEL_DATA** 既存データを表す行に一時項目 (COMPUTE) を表示します。

注意：同一リクエスト内に 2 つの独立した COMPUTE コマンドを作成し、それぞれ別の表示オプションを使用して、任意のフィールドに対して両方の出力タイプを表示することができます。

`infield`

任意の数値フィールドです。結果フィールドと同一のフィールドにすることも、異なるフィールドにすることもできます。ただし、日付時間フィールドおよび日付表示オプションの数値フィールドにすることはできません。

`interval`

それぞれのソートフィールド値に追加する増加値です。最後のデータ点の後に追加して次の値を作成します。この値は正の整数でなければなりません。降順にソートするには、BY HIGHEST 句を使用します。ソートフィールド値にこの数値を追加した結果が、ソートフィールドと同じフォーマットに変換されます。

日付フィールドでは、フォーマットの最小単位の要素によりその数字の認識方法が決まります。たとえば、フォーマットが YMD、MDY、DMY のいずれかの場合、間隔値の 2 は 2 日と認識され、フォーマットが YM の場合は間隔値の 2 は 2 か月と認識されます。

`npredict`

計算する FORECAST の予測数です。この値は、0 (ゼロ) 以上の整数でなければなりません。この値の 0 (ゼロ) は予測値が必要ないことを表し、それは非再帰的の FORECAST でのみサポートされます。

`npoint1`

DOUBLEXP を使用する場合、この数値を使用して、平均の各要素に対する加重値を計算します。この値は正の自然数でなければなりません。次の計算式を使用して、加重値 k を計算します。

```
k=2/(1+npoint1)
```

```
npoint2
```

DOUBLEXP を使用する場合、この正の自然数を使用して、傾向の各項に対する加重値を計算します。次の計算式を使用して、加重値 g を計算します。

```
g=2/(1+npoint2)
```

例　二重指数平滑フィールドの計算

次のリクエストは、VIDEOTRK データソースの TRANSTOT フィールドを、TRANSDATE ごとに合計し、単純指数および二重指数移動平均を計算します。レポートの各列には、既存のデータポイントに対して計算された値が示されます。

```
TABLE FILE VIDEOTRK
SUM TRANSTOT
COMPUTE EXP/D15.1 = FORECAST_EXPAVE(MODEL_DATA,TRANSTOT,1,0,3);
DOUBLEXP/D15.1 = FORECAST_DOUBLEXP(MODEL_DATA,TRANSTOT,1,0,3,3);
BY TRANSDATE
WHERE TRANSDATE NE '19910617'
ON TABLE SET STYLE *
GRID=OFF,$
END
```

下図は、出力結果を示しています。

TRANSDATE	TRANSTOT	EXP	DOUBLEXP
91/06/18	21.25	21.3	21.3
91/06/19	38.17	29.7	35.0
91/06/20	14.23	22.0	30.7
91/06/21	44.72	33.3	39.7
91/06/24	126.28	79.8	86.2
91/06/25	47.74	63.8	80.2
91/06/26	40.97	52.4	65.7
91/06/27	60.24	56.3	61.9
91/06/28	31.00	43.7	45.0

FORECAST_SEASONAL - 三重指数平滑法の使用

三重指数平滑法は、時間とともに特定の間隔で繰り返される値の傾向を考慮した指数移動平均です。たとえば、現在の売上データが増加していることおよび毎年 12 月には常に 25 パーセントの売り上げが期待されることから、この売上データには傾向と季節性の両方の要因が重なっています。三重指数平滑法は、3 つの定数を用いた 3 つの方程式を使用して、傾向と季節性の両方の要因を考慮します。

三重指数平滑法では、各期間でのデータ点の数 (次の方程式の L) を知る必要があります。季節性を考慮するには、季節インデックスを計算します。データを前回の季節インデックスで除算し、そのデータを使用して平滑化した平均を計算します。

- 最初の方程式は、現在の期間を表した式で、現在のデータ値を季節係数で除算した値と前期間の傾向値で調整された前回平均との加重平均です。k は加重定数です。

  ```
  SEASONAL(t) = k * (datavalue(t)/I(t-L)) + (1-k) * (SEASONAL(t-1) +
  b(t-1))
  ```

- 2 つ目の方程式は、傾向値を計算する式で、現在と前回の平均差と前期間の傾向値との加重平均です。b(t) は平均の傾向値を表します。g は加重定数です。

  ```
  b(t) = g * (SEASONAL(t)-SEASONAL(t-1)) + (1-g) * (b(t-1))
  ```

- 3 つ目の方程式は、季節インデックスを計算する式で、現在のデータを現在の平均で除算した値と前回の季節での季節インデックスとの加重平均です。I(t) は平均季節係数を表します。p は加重値の定数です。

  ```
  I(t) = p * (datavalue(t)/SEASONAL(t)) + (1 - p) * I(t-L)
  ```

これらの方程式を解いて、三重に平滑化された平均を求めます。平滑化した最初の平均が、最初のデータ値に設定されます。季節係数の初期値は、データソースに存在するデータの全期間の最大数に基づいて計算されます。それに対して、初期の傾向値は 2 期間のデータに基づいて計算されます。これらの値は次の手順で計算されます。

1. 次の計算式で初期の傾向係数を計算します。

   ```
   b(0) = (1/L) ((y(L+1)-y(1))/L + (y(L+2)-y(2))/L + ... + (y(2L) -
   y(L))/L )
   ```

2. 初期の季節係数は、各期間のデータ値の平均 A(j) (1<=j<=N) に基づいて計算します。

   ```
   A(j) = ( y((j-1)L+1) + y((j-1)L+2) + ... + y(jL) ) / L
   ```

3. 次の計算式から初期の周期係数が得られます。ここで、N はデータの中で使用可能な全期間数、L は期間ごとのデータ点の数、n は期間内のデータ点 (1<= n <= L) を表します。

   ```
   I(n) = ( y(n)/A(1) + y(L+n)/A(2) + ... + y((N-1)L+n)/A(N) ) / N
   ```

これら 3 つの定数は慎重に選定する必要があります。一般的に、最善の結果を得るには、データ値と計算した平均値の間の平均平方誤差 (MSE) を最小にする定数を選択するようにします。結果は、npoint1 と npoint2 の値の変化に影響され、値によっては精度の高い予測値を得ることがあります。精度の高い予測値を得るには、MSE を最小にする値を特定する必要があります。

三重指数平滑法を使用して最終のデータ点以降を予測するには、次の方程式を使用します。

```
forecast(t+m) = (SEASONAL(t) + m * b(t)) / I(t-L+MOD(m/L))
```

説明

`m`

予測する期間数です。

構文　三重指数平滑フィールドの計算

```
FORECAST_SEASONAL(display, infield,
interval, npredict, nperiod, npoint1, npoint2, npoint3)
```

説明

`display`

キーワード

既存データを表す出力行に表示する値を指定します。有効な値には、次のものがあります。

- **INPUT_FIELD**　既存データを表す行に元のフィールド値を表示します。
- **MODEL_DATA**　既存データを表す行に一時項目 (COMPUTE) を表示します。

注意：同一リクエスト内に 2 つの独立した COMPUTE コマンドを作成し、それぞれ別の表示オプションを使用して、任意のフィールドに対して両方の出力タイプを表示することができます。

`infield`

任意の数値フィールドです。結果フィールドと同一のフィールドにすることも、異なるフィールドにすることもできます。ただし、日付時間フィールドおよび日付表示オプションの数値フィールドにすることはできません。

interval

それぞれのソートフィールド値に追加する増加値です。最後のデータ点の後に追加して次の値を作成します。この値は正の整数でなければなりません。降順にソートするには、BY HIGHEST 句を使用します。ソートフィールド値にこの数値を追加した結果が、ソートフィールドと同じフォーマットに変換されます。

日付フィールドでは、フォーマットの最小単位の要素によりその数字の認識方法が決まります。たとえば、フォーマットが YMD、MDY、DMY のいずれかの場合、間隔値の 2 は 2 日と認識され、フォーマットが YM の場合は間隔値の 2 は 2 か月と認識されます。

npredict

計算する FORECAST の予測数です。この値は、0 (ゼロ) 以上の整数でなければなりません。この値の 0 (ゼロ) は予測値が必要ないことを表し、それは非再帰的の FORECAST でのみサポートされます。SEASONAL を使用した方法では、npredict は計算する periods の数になります。作成する points の数は次のとおりです。

```
nperiod * npredict
```

nperiod

SEASONAL を使用する場合、1 つの期間でのデータ点の数を指定する正の自然数です。

npoint1

SEASONAL では、この数を使用して、平均の各要素に対する加重値を計算します。この値は正の自然数でなければなりません。次の計算式を使用して、加重値 k を計算します。

```
k=2/(1+npoint1)
```

npoint2

SEASONAL では、この正の自然数を使用して、傾向の各項に対する加重値を計算します。次の計算式を使用して、加重値 g を計算します。

```
g=2/(1+npoint2)
```

npoint3

SEASONAL では、この正の自然数を使用して、季節調整の各項に対する加重値を計算します。次の計算式を使用して、加重値 p を計算します。

```
p=2/(1+npoint3)
```

例　三重指数平滑フィールドの計算

次の例では、季節性は考慮するが、傾向値は考慮しないデータを取り扱います。そのため、npoint2 を高 (1000) に設定して、傾向係数を計算内で無視します。

```
TABLE FILE VIDEOTRK
SUM TRANSTOT
COMPUTE SEASONAL/D10.1 = FORECAST_SEASONAL(MODEL_DATA,TRANSTOT,
1,3,3,3,1000,1);
BY TRANSDATE
WHERE TRANSDATE NE '19910617'
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果では、npredict は 3 です。つまり、3 期間分 (9 点 nperiod * npredict) が出力されます。

TRANSDATE	TRANSTOT	SEASONAL
91/06/18	21.25	21.3
91/06/19	38.17	31.0
91/06/20	14.23	34.6
91/06/21	44.72	53.2
91/06/24	126.28	75.3
91/06/25	47.74	82.7
91/06/26	40.97	73.7
91/06/27	60.24	62.9
91/06/28	31.00	66.3
91/06/29		45.7
91/06/30		94.1
91/07/01		53.4
91/07/02		72.3
91/07/03		140.0
91/07/04		75.8
91/07/05		98.9
91/07/06		185.8
91/07/07		98.2

FORECAST_LINEAR - 線形回帰式の使用

線形回帰式では、従属変数 (新しく計算した値) と独立変数 (ソートフィールド値) が直線で表される関数で関係が成り立つと仮定して値を予測します。

```
y = mx + b
```

説明

y

従属変数です。

x

独立変数です。

m

直線の傾きです。

b

y 切片です。

FORECAST_LINEAR は「最小二乗推定法」と呼ばれる方法を使用して *m* および *b* の値を計算し、結果として得られた直線とデータ間の平方差の合計を最小にします。

m および b は、次の計算式で計算されます。

$$m = \frac{(\sum xy - (\sum x \bullet \sum y)/n)}{(\sum x^2 - (\sum x)^2/n)}$$

$$b = (\sum y)/n - (m \bullet (\sum x)/n)$$

説明

n

データ点の数です。

y

データ値 (従属変数) です。

x

ソートフィールド値 (独立変数) です。

傾向値および予測値は、回帰直線の方程式を使用して計算します。

構文　線形回帰フィールドの計算

```
FORECAST_LINEAR(display, infield, interval,
 npredict)
```

説明

`display`

キーワード

既存データを表す出力行に表示する値を指定します。有効な値には、次のものがあります。

- **INPUT_FIELD**　既存データを表す行に元のフィールド値を表示します。
- **MODEL_DATA**　既存データを表す行に一時項目 (COMPUTE) を表示します。

注意：同一リクエスト内に 2 つの独立した COMPUTE コマンドを作成し、それぞれ別の表示オプションを使用して、任意のフィールドに対して両方の出力タイプを表示することができます。

`infield`

任意の数値フィールドです。結果フィールドと同一のフィールドにすることも、異なるフィールドにすることもできます。ただし、日付時間フィールドおよび日付表示オプションの数値フィールドにすることはできません。

`interval`

それぞれのソートフィールド値に追加する増加値です。最後のデータ点の後に追加して次の値を作成します。この値は正の整数でなければなりません。降順にソートするには、BY HIGHEST 句を使用します。ソートフィールド値にこの数値を追加した結果が、ソートフィールドと同じフォーマットに変換されます。

日付フィールドでは、フォーマットの最小単位の要素によりその数字の認識方法が決まります。たとえば、フォーマットが YMD、MDY、DMY のいずれかの場合、間隔値の 2 は 2 日と認識され、フォーマットが YM の場合は間隔値の 2 は 2 か月と認識されます。

`npredict`

計算する FORECAST の予測数です。この値は、0 (ゼロ) 以上の整数でなければなりません。この値の 0 (ゼロ) は予測値が必要ないことを表し、それは非再帰的の FORECAST でのみサポートされます。

例　新しい線形回帰フィールドの計算

次のリクエストは、TRANSDATE を使用して、QUANTITY の VIDEOTRK データソースから回帰直線を計算します。間隔値は 1 日で、3 つの予測値が計算されます。

```
TABLE FILE VIDEOTRK
SUM QUANTITY
COMPUTE FORTOT=FORECAST_LINEAR(MODEL_DATA,QUANTITY,1,3);
BY TRANSDATE
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果を示しています。

TRANSDATE	QUANTITY	FORTOT
06/17/91	12	6.63
06/18/91	2	6.57
06/19/91	5	6.51
06/20/91	3	6.45
06/21/91	7	6.39
06/24/91	12	6.21
06/25/91	8	6.15
06/26/91	2	6.09
06/27/91	9	6.03
06/28/91	3	5.97
06/29/91		5.91
06/30/91		5.85
07/01/91		5.79

注意

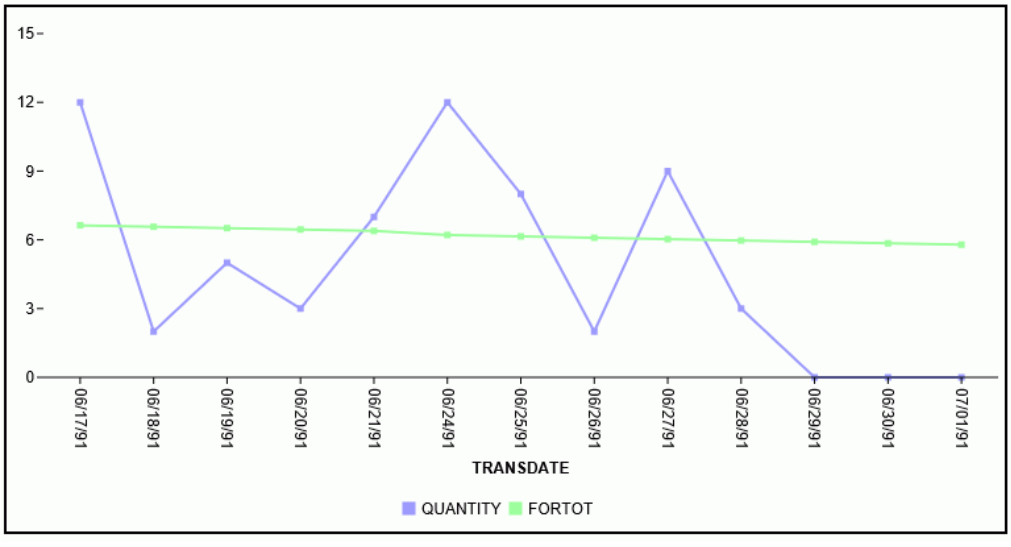
- FORTOT の 3 つの予測値が計算されます。データの範囲外の値に対して、前回の TRANSDATE 値に間隔値 (1) が追加されて新しい TRANSDATE 値が生成されます。
- 生成された FORTOT 値には QUANTITY 値がありません。
- FORTOT のそれぞれの値は、QUANTITY のすべての実データ値から計算された回帰直線を使用して算出されます。

TRANSDATE は独立変数 (x) で、QUANTITY は従属変数 (y) です。この方程式を使用して、QUANTITY FORECAST の傾向値と予測値が計算されます。

次のリクエストは、データ値と回帰直線を表すグラフを作成します。

```
GRAPH FILE VIDEOTRK
SUM QUANTITY
COMPUTE FORTOT=FORECAST_LINEAR(MODEL_DATA,QUANTITY,1,3);
BY TRANSDATE
ON GRAPH PCHOLD FORMAT JSCHART
ON GRAPH SET LOOKGRAPH VLINE
END
```

下図は、出力結果を示しています。



PARTITION_AGGR - ローリング演算の作成

PARTITION_AGGR 関数を使用して、TABLE リクエストの内部マトリックスから、行ブロックに基づくローリング演算を生成することができます。ローリング演算の境界を指定するには、ソートフィールドまたは TABLE 全体に基づいてデータのパーティションを指定します。どちらの境界タイプを使用する場合でも、演算の開始点をパーティションの先頭位置にすることも、現在行から特定の行数を後方に戻った位置や前方に進めた位置にすることもできます。ローリング演算の終了点は、現在行にすることも、開始点より後の任意の行にすることも、パーティションの最終行にすることもできます。

デフォルト設定では、演算に使用されるフィールド値は、リクエストで指定されたメジャーの合計値 (SUM) です。特定の演算接頭語を使用して、内部マトリックスにフィールドを追加し、そのフィールドをローリング演算で使用することができます。ローリング演算に使用可能な演算接頭語は、SUM、AVE、CNT、MIN、MAX、FST、LST です。

構文　PARTITION_AGGR を使用したローリング演算の生成

```
PARTITION_AGGR([prefix.]measure,reset_key,lower,upper,operation)
```

説明

`prefix.`
メジャーに適用する集計演算子を定義します。この演算子がローリング演算に使用されます。有効な演算子には次のものがあります。

- **SUM**　メジャーフィールド値の合計を計算します。デフォルト演算子は SUM です。
- **CNT**　メジャーフィールド値の個数を計算します。
- **AVE**　メジャーフィールド値の平均を計算します。
- **MIN**　メジャーフィールド値の最小値を計算します。
- **MAX**　メジャーフィールド値の最大値を計算します。
- **FST**　メジャーフィールドの最初の値を取得します。
- **LST**　メジャーフィールドの最後の値を取得します。
- **STDP**　母標準偏差を計算します。
- **STDS**　標本標準偏差を計算します。

注意：PCT.、RPCT.、TOT.、MDN.、DST. 演算子はサポートされません。これらの未サポート演算子を参照する COMPUTE もサポートされません。

`measure`
集計の対象となるメジャーフィールドです。メジャーフィールドには、リクエスト内の実フィールドを指定することも、COMPUTE コマンドで生成される一時項目 (COMPUTE) を指定することもできます (COMPUTE コマンドで未サポートの演算接頭語が参照されていない場合)。

`reset_key`
演算を再開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。

- リクエスト内のソートフィールドの名前。

- PRESET - PARTITION_ON パラメータの値を使用します。詳細は、497 ページの 「 簡略統計関数のパーティションサイズの指定 」 を参照してください。
- TABLE - ソートフィールドに区切りが含まれないことを示します。

ソートフィールドに BY HIGHEST を使用して、降順ソートを指定することもできます。ACROSS COLUMNS AND はサポートされます。BY ROWS OVER および FOR はサポートされません。

`lower`
ローリング演算の開始点を指定します。有効な値には、次のものがあります。

- **n, -n** 現在の行から *n* 行前方または後方の位置から演算を開始します。
- **B** 現在のソート区切りの先頭位置から演算を開始します (現在行が属するソート値グループの先頭行)。

`upper`
ローリング演算の終了点を指定します。lower (開始点) の行は、upper (終了点) の行に先行する必要があります。

有効な値には、次のものがあります。

- **C** 内部マトリックスの現在行で演算を終了します。
- **n, -n** 現在の行から *n* 行前方または後方の位置で演算を終了します。
- **E** ソート区切りの最終位置でローリング演算を終了します (現在行が属するソート値グループの最終行)。

注意 : 演算に使用される値は、リクエストで指定したソート順 (昇順または降順) により異なります。日付または時間ディメンションを降順で表示すると、予想とは異なる結果が生じる可能性があることに注意してください。

`operation`
内部マトリックスの値に対して使用するローリング演算を指定します。サポートされる演算子には次のものがあります。

- **SUM** ローリング合計を計算します。
- **AVE** ローリング平均を計算します。
- **CNT** パーティション内の行数を計算します。
- **MIN** パーティション内の最小値を返します。
- **MAX** パーティション内の最大値を返します。

- **MEDIAN** パーティション内の中央値を返します。
- **MODE** パーティション内の最頻値を返します。
- **FST** パーティション内の先頭値を返します。
- **LST** パーティション内の最終値を返します。
- **STDP** パーティション内の母集団の標準偏差を返します。集計の重複を回避するため、PRINT 動詞を使用する必要があります。
- **STDS** パーティション内の標本の標準偏差を返します。集計の重複を回避するため、PRINT 動詞を使用する必要があります。

この演算は、WHERE_GROUPED テストの後、WHERE TOTAL テストの前に実行されます。

例 ローリング平均の計算

次のリクエストは、四半期を境界とする内部マトリックスの現在行の値と 1 行前の値のローリング平均を計算します。

```
TABLE FILE WF_RETAIL_LITE
SUM COGS_US
COMPUTE AVE1/D12.2M = PARTITION_AGGR(COGS_US, TIME_QTR, -1, C, AVE);
BY BUSINESS_REGION
BY TIME_QTR
BY TIME_MTH
WHERE BUSINESS_REGION EQ 'North America' OR 'South America'
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果を示しています。各四半期の 1 行目の平均には、Q1 の値がそのまま使用されます。これは、1 行前の値は境界を越えて別の四半期に属するためです。2 行目の平均は、各四半期の 1 行目と 2 行目から計算され、3 行目の平均は、各四半期の 2 行目と 3 行目から計算されます。

Customer Business Region	Sale Quarter	Sale Month	Cost of Goods	AVE1
North America	1	1	$26,361,956.00	$26,361,956.00
		2	$24,348,729.00	$25,355,342.50
		3	$26,118,420.00	$25,233,574.50
	2	4	$23,776,352.00	$23,776,352.00
		5	$24,717,633.00	$24,246,992.50
		6	$24,284,736.00	$24,501,184.50
	3	7	$25,317,633.00	$25,317,633.00
		8	$25,916,286.00	$25,616,959.50
		9	$24,968,297.00	$25,442,291.50
	4	10	$30,717,478.00	$30,717,478.00
		11	$30,055,782.00	$30,386,630.00
		12	$32,225,143.00	$31,140,462.50
South America	1	1	$3,216,999.00	$3,216,999.00
		2	$2,745,677.00	$2,981,338.00
		3	$3,163,526.00	$2,954,601.50
	2	4	$2,852,809.00	$2,852,809.00
		5	$2,952,020.00	$2,902,414.50
		6	$2,918,017.00	$2,935,018.50
	3	7	$2,961,406.00	$2,961,406.00
		8	$3,077,824.00	$3,019,615.00
		9	$2,895,280.00	$2,986,552.00
	4	10	$3,642,505.00	$3,642,505.00
		11	$3,482,327.00	$3,562,416.00
		12	$3,517,651.00	$3,499,989.00

次のように変更すると、ローリング平均の計算がソート区切りの先頭位置から開始されます。

```
COMPUTE AVE1/D12.2M = PARTITION_AGGR(COGS_US, TIME_QTR ,B, C, AVE);
```

下図は、出力結果を示しています。各四半期の 1 行目の平均には、Q1 の値がそのまま使用されます。これは、1 行前の値は境界を越えて別の四半期に属するためです。2 行目の平均は、各四半期の 1 行目と 2 行目から計算され、3 行目の平均は、各四半期の 1 行目から 3 行目までを使用して計算されます。

Customer Business Region	Sale Quarter	Sale Month	Cost of Goods	AVE1
North America	1	1	$26,361,956.00	$26,361,956.00
		2	$24,348,729.00	$25,355,342.50
		3	$26,118,420.00	$25,609,701.67
	2	4	$23,776,352.00	$23,776,352.00
		5	$24,717,633.00	$24,246,992.50
		6	$24,284,736.00	$24,259,573.67
	3	7	$25,317,633.00	$25,317,633.00
		8	$25,916,286.00	$25,616,959.50
		9	$24,968,297.00	$25,400,738.67
	4	10	$30,717,478.00	$30,717,478.00
		11	$30,055,782.00	$30,386,630.00
		12	$32,225,143.00	$30,999,467.67
South America	1	1	$3,216,999.00	$3,216,999.00
		2	$2,745,677.00	$2,981,338.00
		3	$3,163,526.00	$3,042,067.33
	2	4	$2,852,809.00	$2,852,809.00
		5	$2,952,020.00	$2,902,414.50
		6	$2,918,017.00	$2,907,615.33
	3	7	$2,961,406.00	$2,961,406.00
		8	$3,077,824.00	$3,019,615.00
		9	$2,895,280.00	$2,978,170.00
	4	10	$3,642,505.00	$3,642,505.00
		11	$3,482,327.00	$3,562,416.00
		12	$3,517,651.00	$3,547,494.33

次のコマンドは、パーティション境界として TABLE を使用します。

```
COMPUTE AVE1/D12.2M = PARTITION_AGGR(COGS_US, TABLE, B, C, AVE);
```

下図は、出力結果を示しています。ローリング平均は、ソートフィールドの区切りなしで、各行が順に累加された平均として計算されます。

Customer Business Region	Sale Quarter	Sale Month	Cost of Goods	AVE1
North America	1	1	$26,361,956.00	$26,361,956.00
		2	$24,348,729.00	$25,355,342.50
		3	$26,118,420.00	$25,609,701.67
	2	4	$23,776,352.00	$25,151,364.25
		5	$24,717,633.00	$25,064,618.00
		6	$24,284,736.00	$24,934,637.67
	3	7	$25,317,633.00	$24,989,351.29
		8	$25,916,286.00	$25,105,218.13
		9	$24,968,297.00	$25,090,004.67
	4	10	$30,717,478.00	$25,652,752.00
		11	$30,055,782.00	$26,053,027.45
		12	$32,225,143.00	$26,567,370.42
South America	1	1	$3,216,999.00	$24,771,188.00
		2	$2,745,677.00	$23,197,937.21
		3	$3,163,526.00	$21,862,309.80
	2	4	$2,852,809.00	$20,674,216.00
		5	$2,952,020.00	$19,631,733.88
		6	$2,918,017.00	$18,703,194.06
	3	7	$2,961,406.00	$17,874,678.89
		8	$3,077,824.00	$17,134,836.15
		9	$2,895,280.00	$16,456,762.05
	4	10	$3,642,505.00	$15,874,295.82
		11	$3,482,327.00	$15,335,514.57
		12	$3,517,651.00	$14,843,103.58

参照 PARTITION_AGGR 使用上の注意

- フィールドが PARTITION_AGGR パラメータで参照されているが、リクエストに記述されていない場合、そのフィールドは、内部マトリックスの連続列番号にはカウントされず、HOLD ファイルにも継承されません
- 次の例のように、SUM コマンドで WITHIN 句を使用する方法と、COMPUTE コマンドに PARTITION_AGGR を使用し、WITHIN ソートフィールドの B (ソート区切りの先頭) から E (ソート区切りの末尾) までの範囲に SUM を適用する方法では、同一の結果が得られます。

```
TABLE FILE WF_RETAIL_LITE
SUM COGS_US WITHIN TIME_QTR AS 'WITHIN Qtr'
COMPUTE PART_WITHIN_QTR/D12.2M = PARTITION_AGGR(COGS_US, TIME_QTR, B, E,
SUM);
BY BUSINESS_REGION AS Region
BY TIME_QTR
BY TIME_MTH
WHERE BUSINESS_REGION EQ 'North America' OR 'South America'
ON TABLE SET PAGE NOPAGE
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果を示しています。

Region	Sale Quarter	Sale Month	WITHIN Qtr	PART_WITHIN_QTR
North America	1	1	$76,829,105.00	$76,829,105.00
		2	$76,829,105.00	$76,829,105.00
		3	$76,829,105.00	$76,829,105.00
	2	4	$72,778,721.00	$72,778,721.00
		5	$72,778,721.00	$72,778,721.00
		6	$72,778,721.00	$72,778,721.00
	3	7	$76,202,216.00	$76,202,216.00
		8	$76,202,216.00	$76,202,216.00
		9	$76,202,216.00	$76,202,216.00
	4	10	$92,998,403.00	$92,998,403.00
		11	$92,998,403.00	$92,998,403.00
		12	$92,998,403.00	$92,998,403.00
South America	1	1	$9,126,202.00	$9,126,202.00
		2	$9,126,202.00	$9,126,202.00
		3	$9,126,202.00	$9,126,202.00
	2	4	$8,722,846.00	$8,722,846.00
		5	$8,722,846.00	$8,722,846.00
		6	$8,722,846.00	$8,722,846.00
	3	7	$8,934,510.00	$8,934,510.00
		8	$8,934,510.00	$8,934,510.00
		9	$8,934,510.00	$8,934,510.00
	4	10	$10,642,483.00	$10,642,483.00
		11	$10,642,483.00	$10,642,483.00
		12	$10,642,483.00	$10,642,483.00

その他の演算タイプでは、結果は同一になりません。たとえば、次のリクエストは、WITHIN 句を使用した各四半期の平均と PARTITION_AGGR を使用した各四半期の平均を計算します。

```
TABLE FILE WF_RETAIL_LITE
SUM  COGS_US AS Cost
CNT.COGS_US AS Count AVE.COGS_US WITHIN TIME_QTR AS 'Ave Within'
COMPUTE PART_WITHIN_QTR/D12.2M = PARTITION_AGGR(COGS_US, TIME_QTR, B, E,
AVE);
BY BUSINESS_REGION AS Region
BY TIME_QTR
ON TIME_QTR SUBTOTAL COGS_US CNT.COGS_US
BY TIME_MTH
WHERE BUSINESS_REGION EQ 'North America'
ON TABLE SET PAGE NOPAGE
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果を示しています。WITHIN 句を使用した平均では、四半期の総コストが四半期のインスタンス総数で除算されていますが (例、$76,829,105.00/252850 = $303.85)、PARTITION_AGGR を使用した平均では、四半期の総コストが四半期のレポート行数で除算されています (例、$76,829,105.00/3 = $25,609,701.67)。

Region	Sale Quarter	Sale Month	Cost	Count	Ave Within	PART_WITHIN_QTR
North America	1	1	$26,361,956.00	86369	$303.85	$25,609,701.67
		2	$24,348,729.00	79791	$303.85	$25,609,701.67
		3	$26,118,420.00	86690	$303.85	$25,609,701.67
*TOTAL TIME_QTR	1		$76,829,105.00	252850		
	2	4	$23,776,352.00	79093	$303.40	$24,259,573.67
		5	$24,717,633.00	81317	$303.40	$24,259,573.67
		6	$24,284,736.00	79469	$303.40	$24,259,573.67
*TOTAL TIME_QTR	2		$72,778,721.00	239879		
	3	7	$25,317,633.00	82158	$308.06	$25,400,738.67
		8	$25,916,286.00	83941	$308.06	$25,400,738.67
		9	$24,968,297.00	81262	$308.06	$25,400,738.67
*TOTAL TIME_QTR	3		$76,202,216.00	247361		
	4	10	$30,717,478.00	99572	$309.47	$30,999,467.67
		11	$30,055,782.00	97042	$309.47	$30,999,467.67
		12	$32,225,143.00	103898	$309.47	$30,999,467.67
*TOTAL TIME_QTR	4		$92,998,403.00	300512		
TOTAL			$318,808,445.00	1040602		

- PARTITION_AGGR を使用する際に、オフセットを使用して特定期間の演算を実行する場合 (例、年が異なる四半期の演算)、これらの各四半期が存在する必要があります。一部の年に四半期が存在しない場合、そのオフセットで正しいデータにアクセスすることができません。この場合、各年のすべての四半期が存在する HOLD ファイルを生成し (BY QUARTER ROWS OVER 1 OVER 2 OVER 3 OVER 4 を使用可能)、その HOLD ファイルに対して PARTITION_AGGR を使用します。

PARTITION_REF - 演算での前後のフィールド値の使用

演算に LAST を使用すると、その演算の最終実行時の特定フィールドの最終値が取得されます。一方、PARTITION_REF 関数では、値を取得するために前後に移動する行数と、値の計算が実行されるソート区切りを指定することができます。

構文　　演算で使用する前後のフィールド値を取得

```
PARTITION_REF([prefix.]field, reset_key, offset)
```

説明

`prefix`

オプションです。次の集計演算子のいずれかを指定できます。

- **AVE** 平均
- **MAX**　最大
- **MIN**　最小
- **CNT**　件数
- **SUM**　合計

`field`

取得する値を含むフィールドです。

`reset_key`

検索を再開する区切り位置を指定します。有効な値には、次のものがあります。

- リクエスト内のソートフィールドの名前。
- PRESET - PARTITION_ON パラメータの値を使用します。詳細は、497 ページの 「 簡略統計関数のパーティションサイズの指定 」 を参照してください。
- TABLE - ソートフィールドに区切りが含まれないことを示します。

ソートフィールドに BY HIGHEST を使用して、降順ソートを指定することもできます。ACROSS COLUMNS AND はサポートされます。BY ROWS OVER および FOR はサポートされません。

注意：検索に使用される値は、リクエストで指定したソート順序 (昇順または降順) により異なります。日付または時間ディメンションを降順で表示すると、予想とは異なる結果が生じる可能性があることに注意してください。

`offset`

値を取得するために前方に進むレコード数 (正のオフセット) または後方に戻るレコード数 (負のオフセット) です (いずれも整数)。

オフセットで後方に戻った結果、パーティション境界のソート値を越える場合は、そのフィールドのデフォルト値が返されます。この演算は、WHERE_GROUPED テストの後、WHERE TOTAL テストの前に実行されます。

例　PARTITION_REF による前レコードの取得

次のリクエストは、PRODUCT_CATEGORY ソートフィールド値を境界として前レコードを取得します。

```
TABLE FILE WF_RETAIL_LITE
SUM DAYSDELAYED
COMPUTE NEWDAYS/I5=PARTITION_REF(DAYSDELAYED, PRODUCT_CATEGORY, -1);
BY PRODUCT_CATEGORY
BY PRODUCT_SUBCATEG
ON TABLE SET PAGE NOPAGE
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果を示しています。各ソート区切りの先頭値は 0 (ゼロ) です。これは、取得する前レコードが存在しないためです。

Product Category	Product Subcategory	Days Delayed	NEWDAYS
Accessories	Charger	12,301	0
	Headphones	26,670	12301
	Universal Remote Controls	20,832	26670
Camcorder	Handheld	29,446	0
	Professional	1,531	29446
	Standard	22,248	1531
Computers	Smartphone	24,113	0
	Tablet	21,293	24113
Media Player	Blu Ray	78,989	0
	DVD Players	31	78989
	Streaming	8,153	31
Stereo Systems	Home Theater Systems	47,214	0
	Receivers	17,999	47214
	Speaker Kits	28,468	17999
	iPod Docking Station	37,556	28468
Televisions	Flat Panel TV	10,941	0
Video Production	Video Editing	23,553	0

次のリクエストは、PRODUCT_CATEGORY ソートフィールド値を境界として、現在レコードから 2 レコード前の売上原価平均を取得します。

```
TABLE FILE WF_RETAIL_LITE
SUM COGS_US AVE.COGS_US AS Average
COMPUTE PartitionAve/D12.2M=PARTITION_REF(AVE.COGS_US, PRODUCT_CATEGORY,
-2);
BY PRODUCT_CATEGORY
BY PRODUCT_SUBCATEG
ON TABLE SET PAGE NOPAGE
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果を示しています。

Product Category	Product Subcategory	Cost of Goods	Average	PartitionAve
Accessories	Charger	$2,052,711.00	$27.48	$.00
	Headphones	$51,663,564.00	$319.05	$.00
	Universal Remote Controls	$36,037,623.00	$285.21	$27.48
Camcorder	Handheld	$20,576,916.00	$116.02	$.00
	Professional	$35,218,308.00	$3,897.56	$.00
	Standard	$49,071,633.00	$359.54	$116.02
Computers	Smartphone	$44,035,774.00	$302.01	$.00
	Tablet	$25,771,890.00	$247.89	$.00
Media Player	Blu Ray	$181,112,921.00	$376.11	$.00
	DVD Players	$3,756,254.00	$281.45	$.00
	DVD Players - Portable	$306,576.00	$77.01	$376.11
	Streaming	$5,064,730.00	$104.99	$281.45
Stereo Systems	Boom Box	$840,373.00	$125.67	$.00
	Home Theater Systems	$56,428,589.00	$199.38	$.00
	Receivers	$40,329,668.00	$377.67	$125.67
	Speaker Kits	$81,396,140.00	$471.02	$199.38
	iPod Docking Station	$26,119,093.00	$118.66	$377.67
Televisions	CRT TV	$1,928,416.00	$590.09	$.00
	Flat Panel TV	$59,077,345.00	$900.19	$.00
	Portable TV	$545,348.00	$95.74	$590.09
Video Production	Video Editing	$40,105,657.00	$283.23	$.00

関数コールを次の構文に置き換えると、パーティション境界が TABLE に変更されます。

```
COMPUTE PartitionAve/D12.2M=PARTITION_REF(AVE.COGS_US, TABLE, -2);
```

下図は、出力結果を示しています。

Product Category	Product Subcategory	Cost of Goods	Average	PartitionAve
Accessories	Charger	$2,052,711.00	$27.48	$.00
	Headphones	$51,663,564.00	$319.05	$.00
	Universal Remote Controls	$36,037,623.00	$285.21	$27.48
Camcorder	Handheld	$20,576,916.00	$116.02	$319.05
	Professional	$35,218,308.00	$3,897.56	$285.21
	Standard	$49,071,633.00	$359.54	$116.02
Computers	Smartphone	$44,035,774.00	$302.01	$3,897.56
	Tablet	$25,771,890.00	$247.89	$359.54
Media Player	Blu Ray	$181,112,921.00	$376.11	$302.01
	DVD Players	$3,756,254.00	$281.45	$247.89
	DVD Players - Portable	$306,576.00	$77.01	$376.11
	Streaming	$5,064,730.00	$104.99	$281.45
Stereo Systems	Boom Box	$840,373.00	$125.67	$77.01
	Home Theater Systems	$56,428,589.00	$199.38	$104.99
	Receivers	$40,329,668.00	$377.67	$125.67
	Speaker Kits	$81,396,140.00	$471.02	$199.38
	iPod Docking Station	$26,119,093.00	$118.66	$377.67
Televisions	CRT TV	$1,928,416.00	$590.09	$471.02
	Flat Panel TV	$59,077,345.00	$900.19	$118.66
	Portable TV	$545,348.00	$95.74	$590.09
Video Production	Video Editing	$40,105,657.00	$283.23	$900.19

参照 **PARTITION_REF 使用上の注意**

- フィールドが PARTITION_REF パラメータで参照されているが、リクエストに記述されていない場合、そのフィールドは、内部マトリックスの連続列番号にはカウントされず、HOLD ファイルにも継承されません。

INCREASE - 現在のフィールド値と前のフィールド値の差を計算

INCREASE 関数は、集計された入力フィールドと負のオフセットから、レポート出力のソート区切り内またはテーブル全体で、現在の行と前の 1 行または複数行の値の差を計算します。演算をリセットする位置は、497 ページの 「 簡略統計関数のパーティションサイズの指定 」で説明した PARTITION_ON パラメータの値で指定されます。

注意：演算に使用される値は、リクエストで指定したソート順 (昇順または降順) により異なります。日付または時間ディメンションを降順で表示すると、予想とは異なる結果が生じる可能性があることに注意してください。

構文　現在のフィールド値と前のフィールド値の差を計算するには

```
INCREASE([prefix.]field, offset)
```

説明

`prefix`

次の集計演算子のいずれかを演算に使用する前にフィールドに適用します (オプション)。

- **SUM**　フィールド値の合計を計算します。SUM がデフォルト値です。
- **CNT**　フィールド値の個数を計算します。
- **AVE**　フィールド値の平均を計算します。
- **MIN**　フィールド値の最小値を計算します。
- **MAX**　フィールド値の最大値を計算します。
- **FST**　フィールドの最初の値を取得します。
- **LST**　フィールドの最後の値を取得します。

`field`

数値

演算に使用されるフィールドです。

`offset`

数値

負の数です。演算に使用される、現在の行から後方に戻る行数を示します。

例　現在のフィールド値と前のフィールド値の差の計算

次のリクエストは、SET PARTITION_ON (PENULTIMATE) のデフォルト値を使用して、PRODUCT_CATEGORY ソートフィールド内の現在の行と前の行との増分を計算します。

```
SET PARTITION_ON=PENULTIMATE
TABLE FILE wf_retail_lite
SUM QUANTITY_SOLD
COMPUTE INC = INCREASE(QUANTITY_SOLD,-1);
BY PRODUCT_CATEGORY
BY PRODUCT_SUBCATEG
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果を示しています。INC の 1 つ目の値は、前の値が存在しないため、Accessories カテゴリの販売数量の値になります。INC の 2 つ目の値は、Headphones と Charger の値の差で、3 つ目の値は、Universal Remote Controls と Headphones の値の差です。次に、リセット位置の Camcorder で演算の開始点がリセットされます。

Product Category	Product Subcategory	Quantity Sold	INC
Accessories	Charger	105,257	105,257.00
	Headphones	228,349	123,092.00
	Universal Remote Controls	178,061	-50,288.00
Camcorder	Handheld	250,167	250,167.00
	Professional	12,872	-237,295.00
	Standard	192,205	179,333.00
Computers	Smartphone	205,049	205,049.00
	Tablet	146,728	-58,321.00
Media Player	Blu Ray	679,495	679,495.00
	DVD Players	18,835	-660,660.00
	DVD Players - Portable	5,694	-13,141.00
	Streaming	67,910	62,216.00
Stereo Systems	Boom Box	9,370	9,370.00
	Home Theater Systems	399,092	389,722.00
	Receivers	150,568	-248,524.00
	Speaker Kits	244,199	93,631.00
	iPod Docking Station	311,103	66,904.00
Televisions	CRT TV	4,638	4,638.00
	Flat Panel TV	92,501	87,863.00
	Portable TV	8,049	-84,452.00
Video Production	Video Editing	199,749	199,749.00

PCT_INCREASE - 現在のフィールド値と前のフィールド値の差のパーセントを計算

PCT_INCREASE 関数は、集計された入力フィールドと負のオフセットから、レポート出力のソート区切り内またはテーブル全体で、現在の行の値と前の行の値の差のパーセントを計算します。演算をリセットする位置は、497 ページの 「簡略統計関数のパーティションサイズの指定」 で説明した PARTITION_ON パラメータの値で指定されます。

増加パーセントは、次の式を使用して計算します。

```
(current_value - prior_value) / prior_value
```

注意：演算に使用される値は、リクエストで指定したソート順 (昇順または降順) により異なります。日付または時間ディメンションを降順で表示すると、予想とは異なる結果が生じる可能性があることに注意してください。

構文　現在のフィールド値と前のフィールド値の差のパーセントを計算

```
PCT_INCREASE([prefix.]field, offset)
```

説明

`prefix`

次の集計演算子のいずれかを演算に使用する前にフィールドに適用します (オプション)。

- **SUM**　フィールド値の合計を計算します。SUM がデフォルト値です。
- **CNT**　フィールド値の個数を計算します。
- **AVE**　フィールド値の平均を計算します。
- **MIN**　フィールド値の最小値を計算します。
- **MAX**　フィールド値の最大値を計算します。
- **FST**　フィールドの最初の値を取得します。
- **LST**　フィールドの最後の値を取得します。

`field`

数値

演算に使用されるフィールドです。

`offset`

数値

負の数です。演算に使用される、現在の行から後方に戻る行数を示します。

例 PCT_INCREASE - 現在のフィールド値と前のフィールド値の増加パーセントの計算

次のリクエストは、SET PARTITION_ON (PENULTIMATE) のデフォルト値を使用して、PRODUCT_CATEGORY ソートフィールド内の現在の行と前の行との増分のパーセントを計算します。

```
SET PARTITION_ON=PENULTIMATE
TABLE FILE wf_retail_lite
SUM QUANTITY_SOLD
COMPUTE PCTINC/D8.2p = PCT_INCREASE(QUANTITY_SOLD,-1);
BY PRODUCT_CATEGORY
BY PRODUCT_SUBCATEG
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果を示しています。前に値が存在しないため、PCTINC の 1 つ目の値は、0 (ゼロ) パーセントです。PCINC の 2 つ目の値は、Headphones と Charger の差分パーセントで、3 つ目の値は、Universal Remote Controls と Headphones の差分パーセントです。次に、リセット位置の Camcorder で演算の開始点がリセットされます。

Product Category	Product Subcategory	Quantity Sold	PCTINC
Accessories	Charger	105,257	.00%
	Headphones	228,349	116.94%
	Universal Remote Controls	178,061	-22.02%
Camcorder	Handheld	250,167	.00%
	Professional	12,872	-94.85%
	Standard	192,205	1,393.20%
Computers	Smartphone	205,049	.00%
	Tablet	146,728	-28.44%
Media Player	Blu Ray	679,495	.00%
	DVD Players	18,835	-97.23%
	DVD Players - Portable	5,694	-69.77%
	Streaming	67,910	1,092.66%
Stereo Systems	Boom Box	9,370	.00%
	Home Theater Systems	399,092	4,159.25%
	Receivers	150,568	-62.27%
	Speaker Kits	244,199	62.19%
	iPod Docking Station	311,103	27.40%
Televisions	CRT TV	4,638	.00%
	Flat Panel TV	92,501	1,894.42%
	Portable TV	8,049	-91.30%
Video Production	Video Editing	199,749	.00%

PREVIOUS - フィールドの前の値を取得

集計された入力フィールドと負のオフセットから、PREVIOUS は、ソート区切り内またはテーブル全体で、前の行の値を取得します。演算をリセットする位置は、497 ページの 「 簡略統計関数のパーティションサイズの指定 」 で説明した PARTITION_ON パラメータの値で指定されます。

注意：検索に使用される値は、リクエストで指定したソート順序 (昇順または降順) により異なります。日付または時間ディメンションを降順で表示すると、予想とは異なる結果が生じる可能性があることに注意してください。

構文　フィールドの前の値を取得

```
PREVIOUS([prefix.]field, offset)
```

説明

`prefix`

次の集計演算子のいずれかを演算に使用する前にフィールドに適用します (オプション)。

- **SUM**　フィールド値の合計を計算します。SUM がデフォルト値です。
- **CNT**　フィールド値の個数を計算します。
- **AVE**　フィールド値の平均を計算します。
- **MIN**　フィールド値の最小値を計算します。
- **MAX**　フィールド値の最大値を計算します。
- **FST**　フィールドの最初の値を取得します。
- **LST**　フィールドの最後の値を取得します。

`field`

数値、またはすべての数値が含まれた文字フィールド

演算に使用されるフィールドです。

`offset`

数値

負の数です。検索に使用される、現在の行から後方に戻る行数を示します。

例　フィールドの前の値の取得

次のリクエストは、TABLE に PARTITION_ON を設定し、現在の行から 2 行戻った QUANTITIY_SOLD フィールドの値を取得します。

```
SET PARTITION_ON=TABLE
TABLE FILE wf_retail_lite
SUM QUANTITY_SOLD
COMPUTE PREV = PREVIOUS(QUANTITY_SOLD,-2);
BY PRODUCT_CATEGORY
BY PRODUCT_SUBCATEG
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果を示しています。取得する前の行が存在しないため、PREV の最初の 2 行の値は 0.00 (ゼロ) になっています。3 行目以降、PREV の各値は 2 行前の販売数量の値になっています。ここでは、リセットはされません。

Product Category	Product Subcategory	Quantity Sold	PREV
Accessories	Charger	105,257	.00
	Headphones	228,349	.00
	Universal Remote Controls	178,061	105,257.00
Camcorder	Handheld	250,167	228,349.00
	Professional	12,872	178,061.00
	Standard	192,205	250,167.00
Computers	Smartphone	205,049	12,872.00
	Tablet	146,728	192,205.00
Media Player	Blu Ray	679,495	205,049.00
	DVD Players	18,835	146,728.00
	DVD Players - Portable	5,694	679,495.00
	Streaming	67,910	18,835.00
Stereo Systems	Boom Box	9,370	5,694.00
	Home Theater Systems	399,092	67,910.00
	Receivers	150,568	9,370.00
	Speaker Kits	244,199	399,092.00
	iPod Docking Station	311,103	150,568.00
Televisions	CRT TV	4,638	244,199.00
	Flat Panel TV	92,501	311,103.00
	Portable TV	8,049	4,638.00
Video Production	Video Editing	199,749	92,501.00

RUNNING_AVE - 行グループの平均を計算

RUNNING_AVE 関数は、集計された入力フィールドと負のオフセットから、レポート出力のソート区切り内またはテーブル全体で、現在の行と前の 1 行または複数行の値の平均を計算します。演算をリセットする位置は、特定のソートフィールド、テーブル全体、または 497 ページの 「 簡略統計関数のパーティションサイズの指定 」 で説明した PARTITION_ON パラメータの値で指定されます。

構文　フィールドの現在値と前の値の間の移動平均を計算

```
RUNNING_AVE(field, reset_key, lower)
```

説明

`field`

数値

演算に使用されるフィールドです。

`reset_key`

移動平均の演算を再開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。

- リクエスト内のソートフィールドの名前。
- PRESET - PARTITION_ON パラメータの値を使用します。詳細は、497 ページの 「 簡略統計関数のパーティションサイズの指定 」 を参照してください。
- TABLE - ソートフィールドに区切りが含まれないことを示します。

注意：演算に使用される値は、リクエストで指定したソート順 (昇順または降順) により異なります。日付または時間ディメンションを降順で表示すると、予想とは異なる結果が生じる可能性があることに注意してください。

`lower`

移動平均のパーティションの開始点です。有効な値には、次のものがあります。

- 負の数 - 現在の行からのオフセットを指定します。
- B - ソートグループの開始点を指定します。

例　移動平均の計算

次のリクエストは、PRODUCT_CATEGORY ソートフィールド内の QUANTITY_SOLD の移動平均を計算します。この場合、常にソート区切りが開始点です。

```
TABLE FILE wf_retail_lite
SUM QUANTITY_SOLD
COMPUTE RAVE = RUNNING_AVE(QUANTITY_SOLD,PRODUCT_CATEGORY,B);
BY PRODUCT_CATEGORY
BY PRODUCT_SUBCATEG
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果を示しています。RAVE の 1 つ目の値は、前の値が存在しないため、Accessories カテゴリの販売数量の値になります。RAVE の 2 つ目の値は、Headphones と Charger の値の平均で、3 つ目の値は、Headphones、Charger、Universal Remote Controls の値の平均です。次に、リセット位置の Camcorder で演算の開始点がリセットされます。

Product Category	Product Subcategory	Quantity Sold	RAVE
Accessories	Charger	105,257	105,257.00
	Headphones	228,349	166,803.00
	Universal Remote Controls	178,061	170,555.00
Camcorder	Handheld	250,167	250,167.00
	Professional	12,872	131,519.00
	Standard	192,205	151,748.00
Computers	Smartphone	205,049	205,049.00
	Tablet	146,728	175,888.00
Media Player	Blu Ray	679,495	679,495.00
	DVD Players	18,835	349,165.00
	DVD Players - Portable	5,694	234,674.00
	Streaming	67,910	192,983.00
Stereo Systems	Boom Box	9,370	9,370.00
	Home Theater Systems	399,092	204,231.00
	Receivers	150,568	186,343.00
	Speaker Kits	244,199	200,807.00
	iPod Docking Station	311,103	222,866.00
Televisions	CRT TV	4,638	4,638.00
	Flat Panel TV	92,501	48,569.00
	Portable TV	8,049	35,062.00
Video Production	Video Editing	199,749	199,749.00

RUNNING_MAX - 行グループの最大値を計算

RUNNING_MAX 関数は、集計された入力フィールドと負のオフセットから、レポート出力のソート区切り内またはテーブル全体で、現在の行と前の 1 行または複数行の値の最大値を計算します。演算をリセットする位置は、特定のソートフィールド、テーブル全体、または 497 ページの 「 簡略統計関数のパーティションサイズの指定 」 で説明した PARTITION_ON パラメータの値で指定されます。

構文　フィールドの現在値と前の値の移動最大値を計算

```
RUNNING_MAX(field, reset_key, lower)
```

説明

`field`

数値、またはすべての数値が含まれた文字フィールド

演算に使用されるフィールドです。

`reset_key`

移動最大値の計算を再開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。

- リクエスト内のソートフィールドの名前。
- PRESET - PARTITION_ON パラメータの値を使用します。詳細は、497 ページの 「 簡略統計関数のパーティションサイズの指定 」 を参照してください。
- TABLE - ソートフィールドに区切りが含まれないことを示します。

注意：演算に使用される値は、リクエストで指定したソート順 (昇順または降順) により異なります。日付または時間ディメンションを降順で表示すると、予想とは異なる結果が生じる可能性があることに注意してください。

`lower`

移動最大値を計算するパーティションの開始点です。有効な値には、次のものがあります。

- 負の数 - 現在の行からのオフセットを指定します。
- B - ソートグループの開始点を指定します。

例　移動最大値の計算

次のリクエストは、テーブルの先頭から QUANTITY_SOLD の現在値までの行の移動最大値を計算します。この場合、リセットはされません。

```
TABLE FILE wf_retail_lite
SUM QUANTITY_SOLD
COMPUTE RMAX = RUNNING_MAX(QUANTITY_SOLD,TABLE,B);
BY PRODUCT_CATEGORY
BY PRODUCT_SUBCATEG
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果を示しています。RMAX の 1 つ目の値は、前の値が存在しないため、Accessories カテゴリの販売数量の値になります。RMAX の 2 つ目の値は、Headphones の値です。これが最大値です。RMAX の 3 つ目の値も、Headphones の値です。これは、Headphones の値が 3 行目の販売数量の値より大きいためです。このテーブルの最大値は Blue Ray の値であるため、この値が後続のすべての行で繰り返されます。この場合、リセットはされません。

Product Category	Product Subcategory	Quantity Sold	RMAX
Accessories	Charger	105,257	105,257.00
	Headphones	228,349	228,349.00
	Universal Remote Controls	178,061	228,349.00
Camcorder	Handheld	250,167	250,167.00
	Professional	12,872	250,167.00
	Standard	192,205	250,167.00
Computers	Smartphone	205,049	250,167.00
	Tablet	146,728	250,167.00
Media Player	Blu Ray	679,495	679,495.00
	DVD Players	18,835	679,495.00
	DVD Players - Portable	5,694	679,495.00
	Streaming	67,910	679,495.00
Stereo Systems	Boom Box	9,370	679,495.00
	Home Theater Systems	399,092	679,495.00
	Receivers	150,568	679,495.00
	Speaker Kits	244,199	679,495.00
	iPod Docking Station	311,103	679,495.00
Televisions	CRT TV	4,638	679,495.00
	Flat Panel TV	92,501	679,495.00
	Portable TV	8,049	679,495.00
Video Production	Video Editing	199,749	679,495.00

RUNNING_MIN - 行グループの最小値を計算

RUNNING_MIN 関数は、集計された入力フィールドと負のオフセットから、レポート出力のソート区切り内またはテーブル全体で、現在の行と前の 1 行または複数行の値の最小値を計算します。演算をリセットする位置は、特定のソートフィールド、テーブル全体、または 497 ページの 「 簡略統計関数のパーティションサイズの指定 」 で説明した PARTITION_ON パラメータの値で指定されます。

構文　　フィールドの現在値と前の値の移動最小値を計算

```
RUNNING_MIN(field, reset_key, lower)
```

説明

`field`

数値、またはすべての数値が含まれた文字フィールド

演算に使用されるフィールドです。

`reset_key`

移動最小値の計算を再開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。

- リクエスト内のソートフィールドの名前。
- PRESET - PARTITION_ON パラメータの値を使用します。詳細は、497 ページの 「 簡略統計関数のパーティションサイズの指定 」 を参照してください。
- TABLE - ソートフィールドに区切りが含まれないことを示します。

注意：演算に使用される値は、リクエストで指定したソート順 (昇順または降順) により異なります。日付または時間ディメンションを降順で表示すると、予想とは異なる結果が生じる可能性があることに注意してください。

`lower`

移動最小値を計算するパーティションの開始点です。有効な値には、次のものがあります。

- 負の数 - 現在の行からのオフセットを指定します。
- B - ソートグループの開始点を指定します。

例　移動最小値の計算

次のリクエストは、PRODUCT_CATEGORY ソートフィールド (ソート区切りは SET PARTITION_ON = PENULTIMATE で定義) 内の QUANTITY_SOLD の移動最小値を計算します。この場合、ソート区切りの開始点で常にリセットされます。

```
SET PARTITION_ON=PENULTIMATE
TABLE FILE wf_retail_lite
SUM QUANTITY_SOLD
COMPUTE RMIN = RUNNING_MIN(QUANTITY_SOLD,PRESET,B);
BY PRODUCT_CATEGORY
BY PRODUCT_SUBCATEG
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果を示しています。RMIN の 1 つ目の値は、前の値が存在しないため、Accessories カテゴリの販売数量の値になります。RMIN の 2 つ目の値は、1 行目 (Charger) の値です。この値が 2 行目の値より小さいためです。3 つ目の値も同じ値で、最小値は変わりません。次に、リセット位置の Camcorder で演算の開始点がリセットされます。

Product Category	Product Subcategory	Quantity Sold	RMIN
Accessories	Charger	105,257	105,257.00
	Headphones	228,349	105,257.00
	Universal Remote Controls	178,061	105,257.00
Camcorder	Handheld	250,167	250,167.00
	Professional	12,872	12,872.00
	Standard	192,205	12,872.00
Computers	Smartphone	205,049	205,049.00
	Tablet	146,728	146,728.00
Media Player	Blu Ray	679,495	679,495.00
	DVD Players	18,835	18,835.00
	DVD Players - Portable	5,694	5,694.00
	Streaming	67,910	5,694.00
Stereo Systems	Boom Box	9,370	9,370.00
	Home Theater Systems	399,092	9,370.00
	Receivers	150,568	9,370.00
	Speaker Kits	244,199	9,370.00
	iPod Docking Station	311,103	9,370.00
Televisions	CRT TV	4,638	4,638.00
	Flat Panel TV	92,501	4,638.00
	Portable TV	8,049	4,638.00
Video Production	Video Editing	199,749	199,749.00

RUNNING_SUM - 行グループの合計を計算

RUNNING_SUM 関数は、集計された入力フィールドと負のオフセットから、レポート出力のソート区切り内またはテーブル全体で、現在の行と前の 1 行または複数行の値の合計値を計算します。演算をリセットする位置は、特定のソートフィールド、テーブル全体、または 497 ページの 「 簡略統計関数のパーティションサイズの指定 」 で説明した PARTITION_ON パラメータの値で指定されます。

構文　フィールドの現在値から前の値までの移動合計値を計算

```
RUNNING_SUM(field, reset_key, lower)
```

説明

`field`

数値

演算に使用されるフィールドです。

`reset_key`

移動合計値の計算を再開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。

- リクエスト内のソートフィールドの名前。
- PRESET - PARTITION_ON パラメータの値を使用します。詳細は、497 ページの 「 簡略統計関数のパーティションサイズの指定 」 を参照してください。
- TABLE - ソートフィールドに区切りが含まれないことを示します。

注意：演算に使用される値は、リクエストで指定したソート順 (昇順または降順) により異なります。日付または時間ディメンションを降順で表示すると、予想とは異なる結果が生じる可能性があることに注意してください。

`lower`

移動合計値を計算するパーティションの開始点です。有効な値には、次のものがあります。

- 負の数 - 現在の行からのオフセットを指定します。
- B - ソートグループの開始点を指定します。

例　移動合計値の計算

次のリクエストは、PARTITION_ON パラメータで設定されたリセット位置までの範囲内で、QUANTITY_SOLD の現在の値から前の値までの移動合計を計算します。この場合のリセット位置は、PRODUCT_CATEGORY ソートフィールドに設定されています。

```
SET PARTITION_ON=PENULTIMATE
TABLE FILE wf_retail_lite
SUM QUANTITY_SOLD
COMPUTE RSUM = RUNNING_SUM(QUANTITY_SOLD,PRESET,-1);
BY PRODUCT_CATEGORY
BY PRODUCT_SUBCATEG
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果を示しています。RSUM の 1 つ目の値は、前に値が存在しないため、Accessories カテゴリの販売数量の値になります。RSUM の 2 つ目の値は、Headphones と Charger の値の合計で、3 つ目の値は、Headphone と Universal Remote Controls の値の合計です。次に、リセット位置の Camcorder で演算の開始点がリセットされます。

Product Category	Product Subcategory	Quantity Sold	RSUM
Accessories	Charger	105,257	105,257.00
	Headphones	228,349	333,606.00
	Universal Remote Controls	178,061	406,410.00
Camcorder	Handheld	250,167	250,167.00
	Professional	12,872	263,039.00
	Standard	192,205	205,077.00
Computers	Smartphone	205,049	205,049.00
	Tablet	146,728	351,777.00
Media Player	Blu Ray	679,495	679,495.00
	DVD Players	18,835	698,330.00
	DVD Players - Portable	5,694	24,529.00
	Streaming	67,910	73,604.00
Stereo Systems	Boom Box	9,370	9,370.00
	Home Theater Systems	399,092	408,462.00
	Receivers	150,568	549,660.00
	Speaker Kits	244,199	394,767.00
	iPod Docking Station	311,103	555,302.00
Televisions	CRT TV	4,638	4,638.00
	Flat Panel TV	92,501	97,139.00
	Portable TV	8,049	100,550.00
Video Production	Video Editing	199,749	199,749.00

5

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

- PATTERNS - 入力文字列の構造を表すパターンの取得
- UPPER - 文字列をすべて大文字で取得

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
```

CONCAT - 文字列を連結

CONCAT 関数は、2 つの文字列を連結します。出力は、可変長文字として返されます。

構文　　文字列を連結

```
CONCAT(string1, string2)
```

説明

`string2`

文字

連結する文字列です。

`string1`

文字

連結する文字列です。

例　文字列を連結

次のリクエストは、都市名に州名を連結します。都市名および州名は、連結前に固定長文字フィールドに変換されます。

```
DEFINE FILE WF_RETAIL_LITE
CITY/A50 = CITY_NAME;
STATE/A50 = STATE_PROV_NAME;
CONCAT_CS/A100 = CONCAT(CITY,STATE);
END

TABLE FILE WF_RETAIL_LITE
SUM CITY AS City STATE AS State CONCAT_CS AS Concatenation
BY STATE_PROV_NAME NOPRINT
WHERE COUNTRY_NAME EQ 'United States'
WHERE STATE LE 'Louisiana'
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果を示しています。

City	State	Concatenation
Montgomery	Alabama	Montgomery Alabama
Anchorage	Alaska	Anchorage Alaska
Phoenix	Arizona	Phoenix Arizona
Little Rock	Arkansas	Little Rock Arkansas
Saratoga	California	Saratoga California
Colorado Springs	Colorado	Colorado Springs Colorado
Old Lyme	Connecticut	Old Lyme Connecticut
Wyoming	Delaware	Wyoming Delaware
Washington	District of Columbia	Washington District of Columbia
Orlando	Florida	Orlando Florida
Atlanta	Georgia	Atlanta Georgia
Honolulu	Hawaii	Honolulu Hawaii
Boise	Idaho	Boise Idaho
Chicago	Illinois	Chicago Illinois
Indianapolis	Indiana	Indianapolis Indiana
Dubuque	Iowa	Dubuque Iowa
Wichita	Kansas	Wichita Kansas
Lexington	Kentucky	Lexington Kentucky
New Orleans	Louisiana	New Orleans Louisiana

DIFFERENCE - 文字列間の音声的類似度を計測

DIFFERENCE 関数は、2 つの文字式について、SOUNDEX または METAPHONE の値間の差異を計測する整数値を返します。

構文　文字列間の音声的類似度を計測

```
DIFFERENCE(chrexp1, chrexp2)
```

説明

`chrexp1, chrexp2`

文字

比較する文字列です。

0 (ゼロ) は、類似性が最も低いことを示します。SOUNDEX の場合、4 が最も高い類似性を示し、METAPHONE の場合、16 が最も高い類似性を示します。

SOUNDEX または METAPHONE のいずれを使用するかは、PHONETIC_ALGORITHM 設定に基づきます。デフォルト設定のアルゴリズムは、METAPHONE です。

例　文字列間の音声的類似度を測定

次のリクエストは、DIFFERENCE 関数でデフォルト設定の音声アルゴリズム (METAPHONE) を使用して、データソースの名前 (ファーストネーム) を JOHN および MARY と比較します。

```
TABLE FILE VIDEOTRK
PRINT FIRSTNAME
COMPUTE
JOHN_DIFF/I5 = DIFFERENCE(FIRSTNAME,'JOHN') ;
MARY_DIFF/I5 = DIFFERENCE(FIRSTNAME, 'MARY');
BY LASTNAME NOPRINT
WHERE RECORDLIMIT EQ 30
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果を示しています。JOANN および JOHN では JOHN との一致度が最も高く、MARCIA、MICHAEL、MARTHA では MARY との一致度が最も高くなっています。

FIRSTNAME	JOHN_DIFF	MARY_DIFF
NATALIA	3	5
MARCIA	3	10
IVY	0	0
JASON	6	6
JANET	10	6
JOANN	16	4
JOHN	16	4
WESTON	0	3
GEORGIA	6	6
EVAN	0	0
JESSICA	5	5
MICHAEL	3	10
JAMES	6	6
CHERYL	3	10
DAVID	3	6
JOSHUA	8	8
JOHN	16	4
CATHERINE	2	4
PATRICK	3	3
DONALD	5	5
GLENDA	0	0
RICHARD	3	5
MICHAEL	3	10
LESLIE	3	3
TOM	5	4
MICHAEL	3	10
PATRICIA	2	2
KENNETH	6	6
KELLY	4	8
MARTHA	3	10

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。

GET_TOKEN - 複数区切り文字の文字列に基づいてトークンを取得

GET_TOKEN 関数は、複数の文字 (それぞれが単一区切り文字を表す) が含まれた文字列に基づいてトークン (サブ文字列) を抽出します。

構文　　複数区切り文字の文字列に基づいてトークンを取得

```
GET_TOKEN(string, delimiter_string, occurrence)
```

説明

`string`

文字

トークンの取得元となる入力文字列です。文字フィールドまたは定数を指定することができます。

`delimiter_string`

文字定数

区切り文字のリストが含まれた文字列です。たとえば、文字列「'; ,'」には、セミコロン (;)、ブランク、カンマ (,) の 3 つの区切り文字が含まれています。

`occurrence`

整数定数

抽出されるトークンを指定する正の整数です。構文では負の整数は受容されますが、トークンは取得されません。0 (ゼロ) はサポートされません。

例　複数区切り文字の文字列に基づいてトークンを取得

次のリクエストでは、入力文字列を定義し、区切り文字のリスト (カンマ (,)、セミコロン (;)、スラッシュ (/)) に基づいて 2 つのトークンを抽出します。

```
DEFINE FILE EMPLOYEE
InputString/A20 = 'ABC,DEF;GHI/JKL';
FirstToken/A20 WITH DEPARTMENT = GET_TOKEN(InputString, ',;/', 1);
FourthToken/A20 WITH DEPARTMENT = GET_TOKEN(InputString, ',;/', 4);
END
TABLE FILE EMPLOYEE
PRINT InputString FirstToken FourthToken
WHERE READLIMIT EQ 1
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID = OFF,$
END
```

下図は、出力結果を示しています。1 つ目のトークンは、カンマ (,) を区切り文字として抽出されています。4 つ目のトークンは、スラッシュ (/) を区切り文字として抽出されています。

InputString	FirstToken	FourthToken
ABC,DEF;GHI/JKL	ABC	JKL

INITCAP - 文字列の各単語を先頭大文字に変換

INITCAP 関数は、入力文字列の各単語の先頭文字を大文字にし、その他すべての文字を小文字にします。単語は、ブランクの直後または特殊文字の直後の先頭文字から始まります。

構文　文字列の各単語を先頭大文字に変換

```
INITCAP(input_string)
```

説明

`input_string`

文字

先頭大文字にする文字列です。

例　文字列の各単語を先頭大文字に変換

次のリクエストは、EMPLOYEE データソースの LAST_NAME フィールドを先頭大文字に変換し、NewName フィールドのブランクの直後または特殊文字の直後の先頭文字を大文字にします。

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
Caps1/A30 = INITCAP(LAST_NAME);
NewName/A30 = 'abc,def!ghi'jKL MNO';
Caps2/A30 = INITCAP(NewName);
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果を示しています。

LAST_NAME	Caps1	NewName	Caps2
STEVENS	Stevens	abc,def!ghi'jKL MNO	Abc,Def!Ghi'Jkl Mno
SMITH	Smith	abc,def!ghi'jKL MNO	Abc,Def!Ghi'Jkl Mno
JONES	Jones	abc,def!ghi'jKL MNO	Abc,Def!Ghi'Jkl Mno
SMITH	Smith	abc,def!ghi'jKL MNO	Abc,Def!Ghi'Jkl Mno
BANNING	Banning	abc,def!ghi'jKL MNO	Abc,Def!Ghi'Jkl Mno
IRVING	Irving	abc,def!ghi'jKL MNO	Abc,Def!Ghi'Jkl Mno
ROMANS	Romans	abc,def!ghi'jKL MNO	Abc,Def!Ghi'Jkl Mno
MCCOY	Mccoy	abc,def!ghi'jKL MNO	Abc,Def!Ghi'Jkl Mno
BLACKWOOD	Blackwood	abc,def!ghi'jKL MNO	Abc,Def!Ghi'Jkl Mno
MCKNIGHT	Mcknight	abc,def!ghi'jKL MNO	Abc,Def!Ghi'Jkl Mno
GREENSPAN	Greenspan	abc,def!ghi'jKL MNO	Abc,Def!Ghi'Jkl Mno
CROSS	Cross	abc,def!ghi'jKL MNO	Abc,Def!Ghi'Jkl Mno

LAST_NONBLANK - ブランク/ミッシング以外の最終フィールド値の取得

LAST_NONBLANK 関数は、ブランクとミッシングのどちらでもない最終フィールド値を取得します。先行する値がすべてブランクまたはミッシングの場合、LAST_NONBLANK 関数はミッシング値を返します。

構文　ブランク/ミッシング以外の最終フィールド値の取得

```
LAST_NONBLANK(field)
```

説明

`field`

ブランク/ミッシング以外の最終値を取得するフィールド名です。現在の値がブランク/ミッシング以外の値の場合、現在の値が返されます。

注意：LAST_NONBLANK 関数は、複合式で使用することはできません (例、IF 条件の一部として使用)。

例　ブランク/ミッシング以外の最終値の取得

次の区切りファイルについて考察します。このファイル名は input1.csv で、FIELD_1 および FIELD_2 と名付けられた 2 つのフィールドが含まれます。

```
,
A,
,
 ,
B,
C,
```

Input1 マスターファイルは次のとおりです。

```
FILENAME=INPUT1, SUFFIX=DFIX    ,
 DATASET=baseapp/input1.csv(LRECL 15 RECFM V, BV_NAMESPACE=OFF, $
  SEGMENT=INPUT1, SEGTYPE=S0, $
    FIELDNAME=FIELD_1, ALIAS=E01, USAGE=A1V, ACTUAL=A1V,
      MISSING=ON, $
    FIELDNAME=FIELD_2, ALIAS=E02, USAGE=A1V, ACTUAL=A1V,
      MISSING=ON, $
```

Input1 アクセスファイルは次のとおりです。

```
SEGNAME=INPUT1,
  DELIMITER=',',
  HEADER=NO,
  PRESERVESPACE=NO,
  CDN=COMMAS_DOT,
  CONNECTION=<local>, $
```

次のリクエストは、FIELD_1 の値を表示し、各 FIELD_1 のブランク以外の最終フィールド値を計算します。

```
TABLE FILE baseapp/INPUT1
PRINT FIELD_1 AS Input
COMPUTE
Last_NonBlank/A1 MISSING ON = LAST_NONBLANK(FIELD_1);
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果を示しています。

Input	Last_NonBlank
.	.
A	A
.	A
	A
B	B
C	C

LEFT - 文字列の左側から文字を取得

LEFT 関数は、指定されたソース文字列 (または可変長文字に変換可能な式) および整数値に基づいて、文字列の左側からその数だけの文字を取得します。

構文　文字列の左側から文字を取得

```
LEFT(chr_exp, int_exp)
```

説明

`chr_exp`

文字、または可変長文字に変換可能な式です。

ソース文字列です。

`int_exp`

整数

取得する文字数です。

例　文字列の左側から文字を取得

次のリクエストは、FULLNAME フィールドの名前 (ファーストネーム) の長さを計算し、その文字数を FIRST に返します。

```
TABLE FILE WF_RETAIL_EMPLOYEE
PRINT FULLNAME AND
COMPUTE LEN/I5 = ARGLEN(54, GET_TOKEN(FULLNAME, ' ', 1), LEN); NOPRINT
COMPUTE FIRST/A20 = LEFT(FULLNAME, LEN);
WHERE RECORDLIMIT EQ 20
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果は次のとおりです。

Full Name	FIRST
Steven Wagoner	Steven
Adan Geoghegan	Adan
Candace Aguilar	Candace
Dianna Turpin	Dianna
John Blankinship	John
John Chang	John
John Mackey	John
Elaine Duran	Elaine
Douglas Sanders	Douglas
Linda Whitlow	Linda
Phyllis Carey	Phyllis
Alfred Amerson	Alfred
Jeremy Maness	Jeremy
David Christopher	David
Alice Flemming	Alice
Delia Tennison	Delia
Diane Eads	Diane
Wilfredo Delacruz	Wilfredo
Dorothy Newman	Dorothy
Delia Tennison	Delia

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

`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.
```

OVERLAY - 文字列内の文字を置換

OVERLAY 関数は、指定された開始位置、長さ、ソース文字列、挿入文字列に基づいて、ソース文字列の length で定義された文字数を、開始位置から挿入文字列で置換します。

構文　　文字列内の文字を置換

```
OVERLAY(src, ins, start, len)
```

説明

src

文字

置換する文字を含む文字列です。

ins

文字

置換文字列を含む挿入文字列です。

start

数値

ソース文字列内の置換の開始位置です。

len

数値

ソース文字列を挿入文字列全体で置換する文字数です。

例 文字列内の文字を置換

次のリクエストは、姓 (ラストネーム) の最初の 3 文字を名前 (ファーストネーム) の 4 文字で置換します。

```
TABLE FILE EMPLOYEE
PRINT
COMPUTE FIRST4/A4 = LEFT(FIRST_NAME,4);
NEWNAME/A20 = OVERLAY(LAST_NAME, FIRST4, 1, 3);
BY LAST_NAME
BY FIRST_NAME
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果は次のとおりです。

LAST_NAME	FIRST_NAME	FIRST4	NEWNAME
BANNING	JOHN	JOHN	JOHNNING
BLACKWOOD	ROSEMARIE	ROSE	ROSECKWOOD
CROSS	BARBARA	BARB	BARBSS
GREENSPAN	MARY	MARY	MARYENSPAN
IRVING	JOAN	JOAN	JOANING
JONES	DIANE	DIAN	DIANES
MCCOY	JOHN	JOHN	JOHNOY
MCKNIGHT	ROGER	ROGE	ROGENIGHT
ROMANS	ANTHONY	ANTH	ANTHANS
SMITH	MARY	MARY	MARYTH
	RICHARD	RICH	RICHTH
STEVENS	ALFRED	ALFR	ALFRVENS

PATTERNS - 入力文字列の構造を表すパターンの取得

PATTERNS 関数は、入力引数の構造を表す文字列を返します。返されるパターンは、以下の文字で構成されます。

- **A** 入力文字列の任意の位置にある文字が大文字の場合に返されます。
- **a** 入力文字列の任意の位置にある文字が小文字の場合に返されます。
- **9** 入力文字列の任意の位置にある文字が数字の場合に返されます。

特殊文字 (例、+-/=%) は、入力文字列の元の表記どおりに返されます。

出力は、可変長文字として返されます。

構文　　入力引数のパターンプロファイルを表す文字列の取得

```
PATTERNS(string)
```

説明

`string`

文字

パターンを取得する文字列です。

例　入力文字列の構造を表すパターンの取得

次のリクエストは、顧客住所の構造を表すパターンを返します。

```
DEFINE FILE WF_RETAIL_LITE
Address_Pattern/A40V = PATTERNS(ADDRESS_LINE_1);
END

TABLE FILE WF_RETAIL_LITE
PRINT FST.ADDRESS_LINE_1 OVER
Address_Pattern
BY ADDRESS_LINE_1 NOPRINT SKIP-LINE
WHERE COUNTRY_NAME EQ 'United States'
WHERE CITY_NAME EQ 'Houston' OR 'Indianapolis' OR 'Chapel Hill' OR 'Bronx'
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果の一部を示しています。住所に含まれる特殊記号 (#-,) は、パターンでも元の表記で示されています。

FST Customer Address Line 1	1010 Milam St # Ifp-2352
Address_Pattern	9999 Aaaaa Aa # Aaa-9999
FST Customer Address Line 1	10700 Richmond Ave
Address_Pattern	99999 Aaaaaaaa Aaa
FST Customer Address Line 1	10777 North Fwy
Address_Pattern	99999 Aaaaa Aaa
FST Customer Address Line 1	11 E Greenway Plz Ste 100
Address_Pattern	99 A Aaaaaaaa Aaa Aaa 999
FST Customer Address Line 1	111 Monument Cir
Address_Pattern	999 Aaaaaaaa Aaa
FST Customer Address Line 1	111 Monument Circle - Ste 2100
Address_Pattern	999 Aaaaaaaa Aaaaaa - Aaa 9999
FST Customer Address Line 1	1205 Dart St, Rm 219
Address_Pattern	9999 Aaaa Aa, Aa 999

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
```

POSITION - ソース文字列内のサブ文字列の開始位置を取得

POSITION 関数は、指定された検索文字列、ソース文字列、開始位置に基づいて、ソース文字列内の検索文字列の位置を取得します。検索は、指定された開始位置から開始され、左から右へ検索されます。POSITION は、文字列が検出されない場合、0 (ゼロ) を返します。検索では大文字と小文字が区別されます。

構文　ソース文字列内のサブ文字列の開始位置を取得

```
POSITION(search, source, start)
```

説明

`search`

文字

検索文字列です。

`source`

文字

ソース文字列です。

`start`

数値

ソース文字列の検索開始位置です。

例　ソース文字列内のサブ文字列の開始位置を取得

次のリクエストでは、POSITION 関数が 2 回使用されています。1 回目の POSITION で FULLNAME の開始位置 3 以降から最初の文字「A」を検索し、2 回目の POSITION で FULLNAME の開始位置 3 以降の最初の「a」を検索します。

```
TABLE FILE WF_RETAIL_CUSTOMER
PRINT FULLNAME
COMPUTE POS1/I5 = POSITION('A', FULLNAME, 3) ;
POS2/I5 = POSITION('a', FULLNAME, 3) ;
WHERE RECORDLIMIT EQ 5
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果は次のとおりです。

Full Name	POS1	POS2
Joshua Hines	0	6
Sandra Arzola	8	6
Rebecca Smith	0	7
John Nichols	0	0
Hector Briganti	0	12

正規表現関数

正規表現は、検索パターンを作成するために一連のメタ文字とリテラル文字を組み合わせたものです。

注意：正規表現パターンの作成に使用される記号についての詳細は、オンラインで検索できます。たとえば、次のウィキペディアサイトにも記載されています。

「 https://en.wikipedia.org/wiki/Regular_expression 」

以下は、正規表現で使用する一般的なメタ文字を示しています。

- . - 任意の単一文字を表します。
- * - 0 (ゼロ) 個以上の任意の文字を表します。
- * - 1 個以上の任意の文字を表します。
- ? - 0 (ゼロ) 個または 1 個の任意の文字を表します。
- ^ - 行の先頭を表します。
- $ - 行の末尾を表します。
- [] - 大括弧で囲まれた文字セットのいずれか 1 文字を表します。
- [^] - 大括弧で囲まれた文字セット以外の任意の 1 文字を表します。
- | - OR 演算子を表します。
- ¥ - エスケープ特殊文字です。

- () - 文字シーケンスを含みます。

たとえば、正規表現の「^Ste(v|ph)en$」は、「Ste」で始まり、「ph」または「v」が続き、「en」で終わる値に一致します。

REGEX - 文字列を正規表現で照合

REGEX 関数は、文字列を正規表現で照合し、一致する場合は true (1)、一致しない場合は false (0) を返します。

正規表現は、検索パターンを作成するために一連の特殊文字とリテラル文字を組み合わせたものです。

Web 上には、正規表現に関する参考情報が多く提供されています。

概要については、『ibi™ WebFOCUS® サーバ管理者ガイド』の「セキュリティ」の「正規表現の概要」を参照してください。

構文　　文字列を正規表現で照合

```
REGEX(string, regular_expression)
```

説明

`string`
文字

照合する文字列です。

`regular_expression`
文字

リテラルとメタ文字を使用して作成される正規表現です。正規表現は、一重引用符 (') で囲みます。サポートされるメタ文字には次のものがあります。

- . - 任意の単一文字を表します。
- * - 0 (ゼロ) 個以上の任意の文字を表します。
- * - 1 個以上の任意の文字を表します。
- ? - 0 (ゼロ) 個または 1 個の任意の文字を表します。
- ^ - 行の先頭を表します。
- $ - 行の末尾を表します。

- [] - 大括弧で囲まれた文字セットのいずれか 1 文字を表します。
- [^] - 大括弧で囲まれた文字セット以外の任意の 1 文字を表します。
- | - OR 演算子を表します。
- ¥ - エスケープ特殊文字です。
- () - 文字シーケンスを含みます。

たとえば、正規表現の「^Ste(v|ph)en$」は、「Ste」で始まり、「ph」または「v」が続き、「en」で終わる値に一致します。

注意：出力値は数値です。

例　文字列を正規表現で照合

次のリクエストは、FIRSTNAME フィールドを正規表現の「'PATRIC[(I?)K]'」で照合します。この正規表現は、「PATRICIA」または「PATRICK」と一致します。

```
DEFINE FILE VIDEOTRK
PNAME/I5=REGEX(FIRSTNAME,'PATRIC[(I?)K]');
END
TABLE FILE VIDEOTRK
PRINT FIRSTNAME PNAME
BY LASTNAME
WHERE LASTNAME GE 'M'
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果は次のとおりです。

LASTNAME	FIRSTNAME	PNAME
MCMAHON	JOHN	0
MONROE	CATHERINE	0
	PATRICK	1
NON-MEMBER		0
O'BRIEN	DONALD	0
PARKER	GLENDA	0
	RICHARD	0
RATHER	MICHAEL	0
RIESLER	LESLIE	0
SPIVEY	TOM	0
STANDLER	MICHAEL	0
STEWART	MAUDE	0
WHITE	PATRICIA	1
WILLIAMS	KENNETH	0
WILSON	KELLY	0
WU	MARTHA	0

REGEXP_COUNT - 文字列内のパターン一致個数のカウント

REGEXP_COUNT は、ソース文字列内で特定の正規表現パターンに一致する文字列の個数を整数で返します。

構文　　文字列内のパターン一致個数のカウント

```
REGEXP_COUNT(string, pattern)
```

説明

`string`
　文字

　検索対象となる入力文字列です。

`pattern`
　文字

リテラルとメタ文字を使用して作成される正規表現です。正規表現は、一重引用符 (') で囲みます。サポートされるメタ文字には次のものがあります。

- . - 任意の単一文字を表します。
- * - 0 (ゼロ) 個以上の任意の文字を表します。
- * - 1 個以上の任意の文字を表します。
- ? - 0 (ゼロ) 個または 1 個の任意の文字を表します。
- ^ - 行の先頭を表します。
- $ - 行の末尾を表します。
- [] - 大括弧で囲まれた文字セットのいずれか 1 文字を表します。
- [^] - 大括弧で囲まれた文字セット以外の任意の 1 文字を表します。
- | - OR 演算子を表します。
- ¥ - エスケープ特殊文字です。
- () - 文字シーケンスを含みます。

例　文字列内のパターン一致個数のカウント

以下の例では、次の正規表現記号を使用します。

- $ - 文字列の末尾にある特定の表現を検索します。
- ^ - 文字列の先頭にある特定の表現を検索します。
- ¥s* - 任意の数のブランク文字 (連続するブランク文字など) に一致します。
- [T,t] - 文字「T」および「t」に一致します。

次のリクエストで、REG1 は PRODUCT フィールドで後続する任意の個数のブランク文字を含めた「iscotti」の出現回数を表します。REG2 は、PRODUCT フィールドでの文字「T」および「t」の出現回数を表します。

```
TABLE FILE GGSALES
SUM DOLLARS AND COMPUTE
REG1/I5 = REGEXP_COUNT(PRODUCT, 'iscotti¥s*$');
REG2/I5 = REGEXP_COUNT(PRODUCT, '[T,t]');
BY PRODUCT
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果は次のとおりです。

Product	Dollar Sales	REG1	REG2
Biscotti	5263317	1	2
Capuccino	2381590	0	0
Coffee Grinder	2337567	0	0
Coffee Pot	2449585	0	1
Croissant	7749902	0	1
Espresso	3906243	0	0
Latte	10943622	0	2
Mug	4522521	0	0
Scone	4216114	0	0
Thermos	2385829	0	1

例　Windows 上での REGEXP_COUNT の使用

次のリクエストでは、Windows 上で REGEXP_COUNT 関数を使用して、各製品名での母音と子音の数を取得します。VowelCnt は母音、ConsonantCnt は母音以外の数を表します。

```
DEFINE FILE GGSALES
VowelCnt/I5=REGEXP_COUNT(PRODUCT,'[AEIOUaeiou]');
ConsonantCnt/I5=REGEXP_COUNT(PRODUCT,'[^AEIOUaeiou]');
END
TABLE FILE GGSALES
SUM MAX.VowelCnt AS 'Vowels'
    MAX.ConsonantCnt AS 'Consonants'
BY PRODUCT
END
```

注意

- 正規表現パターンに一致する文字リストを指定するには、大括弧 ([]) を使用します。
- 文字リストを大括弧 ([]) で囲んでアクサンシルコンフレクス (^) を前置した場合、この正規表現はリストに存在しない任意の文字と一致します。

出力結果は次のとおりです。

Product	Vowels	Consonants
Biscotti	3	5
Capuccino	4	5
Coffee Grinder	5	8
Coffee Pot	4	5
Croissant	3	6
Espresso	3	5
Latte	2	3
Mug	1	2
Scone	2	3
Thermos	2	5

REGEXP_INSTR - 文字列内の 1 つ目のパターンの位置を取得

REGEXP_INSTR は、ソース文字列内で特定の正規表現パターンに一致する 1 つ目の文字列の位置を整数で返します。文字列内の 1 つ目の文字位置が、値 1 で示されます。ソース文字列内に一致する文字列がない場合、値 0 が返されます。

構文　文字列内のパターンの位置の取得

```
REGEXP_INSTR(string, pattern)
```

説明

`string`
　文字

　検索対象となる入力文字列です。

`pattern`
　文字

リテラルとメタ文字を使用して作成される正規表現です。正規表現は、一重引用符 (') で囲みます。サポートされるメタ文字には次のものがあります。

- . - 任意の単一文字を表します。
- * - 0 (ゼロ) 個以上の任意の文字を表します。
- * - 1 個以上の任意の文字を表します。
- ? - 0 (ゼロ) 個または 1 個の任意の文字を表します。
- ^ - 行の先頭を表します。
- $ - 行の末尾を表します。
- [] - 大括弧で囲まれた文字セットのいずれか 1 文字を表します。
- [^] - 大括弧で囲まれた文字セット以外の任意の 1 文字を表します。
- | - OR 演算子を表します。
- ¥ - エスケープ特殊文字です。
- () - 文字シーケンスを含みます。

例　文字列内のパターンの位置の検索

以下の例では、次の正規表現記号を使用します。

- $ - 文字列の末尾にある特定の表現を検索します。
- ^ - 文字列の先頭にある特定の表現を検索します。
- ¥s* - 任意の数のブランク文字 (連続するブランク文字など) に一致します。
- [B,C,S] - 大文字の「B」、「C」、および「S」に一致します。

次のリクエストで、REG1 は PRODUCT フィールドの末尾に出現する後続する任意の個数のブランク文字を含めた「iscotti」の位置を表します。REG2 は、PRODUCT フィールド値の先頭に出現する文字「B」、「C」、または「S」の位置を表します。

```
TABLE FILE GGSALES
SUM DOLLARS AND COMPUTE
REG1/I5 = REGEXP_INSTR(PRODUCT, 'iscotti¥s*$');
REG2/I5 = REGEXP_INSTR(PRODUCT, '^[B,C,S]');
BY PRODUCT
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果は次のとおりです。

Product	Dollar Sales	REG1	REG2
Biscotti	5263317	2	1
Capuccino	2381590	0	1
Coffee Grinder	2337567	0	1
Coffee Pot	2449585	0	1
Croissant	7749902	0	1
Espresso	3906243	0	0
Latte	10943622	0	0
Mug	4522521	0	0
Scone	4216114	0	1
Thermos	2385829	0	0

REGEXP_REPLACE - 文字列内のすべてのパターン一致の置換

REGEXP_REPLACE は、ソース文字列内の正規表現パターンに一致するすべての文字列を特定の置換文字列で置き換えることで生成される文字列を返します。置換文字列は、NULL 文字列にすることもできます。

構文　文字列内のパターン一致の置換

```
REGEXP_REPLACE(string, pattern, replacement)
```

説明

`string`
文字

検索対象となる入力文字列です。

`pattern`
文字

リテラルとメタ文字を使用して作成される正規表現です。正規表現は、一重引用符 (') で囲みます。サポートされるメタ文字には次のものがあります。

- . - 任意の単一文字を表します。
- * - 0 (ゼロ) 個以上の任意の文字を表します。
- * - 1 個以上の任意の文字を表します。
- ? - 0 (ゼロ) 個または 1 個の任意の文字を表します。
- ^ - 行の先頭を表します。
- $ - 行の末尾を表します。
- [] - 大括弧で囲まれた文字セットのいずれか 1 文字を表します。
- [^] - 大括弧で囲まれた文字セット以外の任意の 1 文字を表します。
- | - OR 演算子を表します。
- ¥ - エスケープ特殊文字です。
- () - 文字シーケンスを含みます。

`replacement`
文字

置換文字列です。

例　文字列内のパターン一致の置換

以下の例では、次の正規表現記号を使用します。

- ^ - 文字列の先頭にある特定の表現を検索します。

次のリクエストでは、REG1 は REGION フィールド値の先頭にある文字列「North」を文字列「South」に置き換え、REG2 は REGION フィールド値の先頭にある文字列「Mid」を NULL 文字列に置き換えます。

```
TABLE FILE GGSALES
SUM DOLLARS NOPRINT AND COMPUTE
REG1/A25 = REGEXP_REPLACE(REGION, '^North', 'South');
REG2/A25 = REGEXP_REPLACE(REGION, '^Mid', '');
BY REGION
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果は次のとおりです。

Region	REG1	REG2
Midwest	Midwest	west
Northeast	Southeast	Northeast
Southeast	Southeast	Southeast
West	West	West

REGEXP_SUBSTR - 文字列内の 1 つ目のパターン一致の取得

REGEXP_SUBSTR は、ソース文字列内で特定の正規表現パターンに一致する 1 つ目の文字列を含む文字列を取得します。ソース文字列内に一致する文字列がない場合、NULL 文字列が返されます。

構文　文字列内の 1 つ目のパターン一致の取得

```
REGEXP_SUBSTR(string, pattern)
```

説明

`string`
　文字

　検索対象となる入力文字列です。

`pattern`
　文字

リテラルとメタ文字を使用して作成される正規表現です。正規表現は、一重引用符 (') で囲みます。サポートされるメタ文字には次のものがあります。

- . - 任意の単一文字を表します。
- * - 0 (ゼロ) 個以上の任意の文字を表します。
- * - 1 個以上の任意の文字を表します。
- ? - 0 (ゼロ) 個または 1 個の任意の文字を表します。
- ^ - 行の先頭を表します。
- $ - 行の末尾を表します。
- [] - 大括弧で囲まれた文字セットのいずれか 1 文字を表します。
- [^] - 大括弧で囲まれた文字セット以外の任意の 1 文字を表します。
- | - OR 演算子を表します。
- ¥ - エスケープ特殊文字です。
- () - 文字シーケンスを含みます。

例　文字列内の 1 つ目のパターン一致の取得

以下の例では、次の正規表現記号を使用します。

- [A-Z] - すべての大文字に一致します。
- [a-z] - すべての小文字に一致します。

次のリクエストでは、REG1 には、REGION フィールド値内で、先頭が大文字、次に任意の数の小文字、その後に「west」という文字が続く文字列の最初のインスタンスが含まれています。REG2 には、REGION フィールド値内で、先頭が大文字、次に任意の数の小文字、その後に「east」という文字列が続く最初のインスタンスが含まれています。

```
TABLE FILE GGSALES
SUM DOLLARS NOPRINT AND COMPUTE
REG1/A25 = REGEXP_SUBSTR(REGION, '[A-Z][a-z]*west');
REG2/A25 = REGEXP_SUBSTR(REGION, '[A-Z][a-z]*east');
BY REGION
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果は次のとおりです。

Region	REG1	REG2
Midwest	Midwest	
Northeast		Northeast
Southeast		Southeast
West		

REPEAT - 文字列の指定回数の繰り返し

REPEAT 関数は、指定されたソース文字列および整数値に基づいて、ソース文字列を指定回数繰り返した文字列を取得します。繰り返しを格納する文字列は、結果となる繰り返し文字列を格納するために十分なサイズにする必要があります。サイズが十分でない場合、値は切り捨てられます。

構文　文字列の指定回数の繰り返し

```
REPEAT(source_str, number)
```

説明

`source_str`

文字

繰り返すソース文字列です。source_str がフィールドの場合、ブランクを含めたフィールド全体が繰り返しの対象になります。

`number`

数値

ソース文字列の繰り返し回数です。

例　文字列の指定回数の繰り返し

次のリクエストは、FIRST_NAME を 3 回繰り返した文字列を返します。

```
TABLE FILE EMPLOYEE
PRINT FIRST_NAME
COMPUTE REPEAT3/A25 = REPEAT(FIRST_NAME,3);
ON TABLE SET PAGE NOLEAD
ON TABLE PCHOLD FORMAT PDF
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、PDF の出力結果を示しています。

FIRST_NAME	REPEAT3		
ALFRED	ALFRED	ALFRED	ALFRE
MARY	MARY	MARY	MARY
DIANE	DIANE	DIANE	DIANE
RICHARD	RICHARD	RICHARD	RICHA
JOHN	JOHN	JOHN	JOHN
JOAN	JOAN	JOAN	JOAN
ANTHONY	ANTHONY	ANTHONY	ANTHO
JOHN	JOHN	JOHN	JOHN
ROSEMARIE	ROSEMARIE	ROSEMARIE	ROSEM
ROGER	ROGER	ROGER	ROGER
MARY	MARY	MARY	MARY
BARBARA	BARBARA	BARBARA	BARBA

REPLACE - 文字列の置換

REPLACE 関数は、入力文字列に存在する検索文字列のインスタンスすべてを特定の文字列に置換します。出力は常に可変長文字フォーマットで返され、長さは入力パラメータに基づいて決定されます。

構文　文字列のすべてのインスタンスを置換

```
REPLACE(input_string , search_string , replacement)
```

説明

`input_string`

文字またはテキスト (An、AnV、TX)

入力文字列です。

`search_string`

文字またはテキスト (An、AnV、TX)

入力文字列内で検索する文字列です。

`replacement`

文字またはテキスト (An、AnV、TX)

検索文字列を置換する文字列です。NULL 文字列 ('') を指定することもできます。

例　文字列の置換

次の REPLACE 関数は、国名の「SOUTH」という文字列を「S.」に置換します。

```
SET TRACEUSER = ON
SET TRACEON = STMTRACE//CLIENT
SET TRACESTAMP=OFF
DEFINE FILE WF_RETAIL_LITE
NEWNAME/A20 = REPLACE(COUNTRY_NAME, 'SOUTH', 'S.');
END
TABLE FILE WF_RETAIL_LITE
SUM COUNTRY_NAME
BY NEWNAME AS 'New,Name'
WHERE COUNTRY_NAME LIKE 'S%'
ON TABLE SET PAGE NOLEAD
END
```

下図は、出力結果を示しています。

New Name	Customer Country
S. Africa	South Africa
S. Korea	South Korea
Singapore	Singapore
Spain	Spain
Sweden	Sweden
Switzerland	Switzerland

例　　文字列のすべてのインスタンスを置換

次のリクエストでは、DAY1 で文字列が定義され、「DAY」という文字列のインスタンスがすべて「day」という文字列に置換された上で、一時項目 (DEFINE) の DAYNAME1 に格納されます。一時項目 (DEFINE) の DAYNAME2 には、「DAY」という文字列のインスタンスがすべて削除された後の文字列が格納されます。

```
DEFINE FILE WF_RETAIL
DAY1/A30 = 'SUNDAY MONDAY TUESDAY';
DAYNAME1/A30 = REPLACE(DAY1, 'DAY', 'day' );
DAYNAME2/A30 = REPLACE(DAY1, 'DAY', '' );
END
TABLE FILE WF_RETAIL
PRINT DAY1 OVER
DAYNAME1 OVER
DAYNAME2
WHERE EMPLOYEE_NUMBER EQ 'AH118'
ON TABLE SET PAGE NOPAGE
END
```

出力結果は次のとおりです。

```
DAY1       SUNDAY MONDAY TUESDAY
DAYNAME1   SUNday MONday TUESday
DAYNAME2   SUN MON TUES
```

RIGHT - 文字列の右側から文字を取得

RIGHT 関数は、指定されたソース文字列、(または可変長文字に変換可能な式) および整数値に基づいて、文字列の右側からその数だけの文字を取得します。

構文　文字列の右側から文字を取得

```
RIGHT(chr_exp, int_exp)
```

説明

`chr_exp`

文字、または可変長文字に変換可能な式です。

ソース文字列です。

`int_exp`

整数

取得する文字数です。

例　文字列の右側から文字を取得

次のリクエストは、FULLNAME フィールドの姓 (ラストネーム) の長さを計算し、その文字数を LAST に返します。

```
TABLE FILE WF_RETAIL_EMPLOYEE
PRINT FULLNAME AND
COMPUTE LEN/I5 = ARGLEN(54, GET_TOKEN(FULLNAME, ' ', 2), LEN); NOPRINT
COMPUTE LAST/A20 = RIGHT(FULLNAME, LEN);
WHERE RECORDLIMIT EQ 20
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果は次のとおりです。

Full Name	LAST
Steven Wagoner	Wagoner
Adan Geoghegan	Geoghegan
Candace Aguilar	Aguilar
Dianna Turpin	Turpin
John Blankinship	Blankinship
John Chang	Chang
John Mackey	Mackey
Elaine Duran	Duran
Douglas Sanders	Sanders
Linda Whitlow	Whitlow
Phyllis Carey	Carey
Alfred Amerson	Amerson
Jeremy Maness	Maness
David Christopher	Christopher
Alice Flemming	Flemming
Delia Tennison	Tennison
Diane Eads	Eads
Wilfredo Delacruz	Delacruz
Dorothy Newman	Newman
Delia Tennison	Tennison

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./
```

SPACE - 指定された数のブランクを含む文字列の取得

SPACE 関数は、整数 (count) から、この整数個分のブランクで構成される文字列を取得します。

注意：HTML レポート出力でブランクを保持するには、SHOWBLANKS パラメータを ON に設定する必要があります。

構文　　指定された数のブランクを含む文字列の取得

```
SPACE(count)
```

説明

`count`

数値

追加するブランクの数です。

例　　指定した数のブランクを含む文字列の取得

次のリクエストは、文字値に変換された DOLLARS 値および UNITS 値の間に、20 個のブランクを挿入します。等幅フォントの Courier が使用されているため、20 個のブランクがプロポーショナルにならずに表示されています。

```
SET SHOWBLANKS = ON
TABLE FILE GGSALES
SUM DOLLARS NOPRINT  UNITS  NOPRINT AND
COMPUTE ALPHADOLL/A8 = EDIT(DOLLARS); NOPRINT
COMPUTE ALPHAUNIT/A8 = EDIT(UNITS); NOPRINT
COMPUTE Dollars_And_Units_With_Spaces/A60 = ALPHADOLL | SPACE(20) |
ALPHAUNIT;
BY CATEGORY
ON TABLE SET PAGE NOLEAD
ON TABLE PCHOLD FORMAT PDF
ON TABLE SET STYLE *
GRID=OFF, FONT=COURIER,$
ENDSTYLE
END
```

出力結果は次のとおりです。

```
Category  Dollars_And_Units_With_Spaces
Coffee    17231455                    01376266
Food      17229333                    01384845
Gifts     11695502                    00927880
```

SPLIT - 文字列から要素を抽出

SPLIT 関数は、文字列から特定タイプの要素を返します。出力は、可変長文字として返されます。

構文　文字列から要素を抽出

```
SPLIT(element, string)
```

説明

`element`

次のキーワードのいずれかを指定できます。

- **EMAIL_DOMAIN**　文字列内の Email アドレスのドメイン名です。
- **EMAIL_USERID**　文字列内の Email アドレスのユーザ ID です。
- **URL_PROTOCOL**　文字列内の URL プロトコルです。
- **URL_HOST**　文字列内の URL のホスト名です。
- **URL_PORT**　文字列内の URL のポート番号です。
- **URL_PATH**　文字列内の URL パスです。
- **NAME_FIRST**　文字列内の最初のトークン (連続文字のグループ) です。各トークンはブランクで区切られます。
- **NAME_LAST**　文字列内の最後のトークン (連続文字のグループ) です。各トークンはブランクで区切られます。

`string`

文字

要素の抽出元の文字列です。

例　文字列から要素を抽出

次のリクエストは、文字列を定義し、その文字列から要素を抽出します。

```
DEFINE FILE WF_RETAIL_LITE
STRING1/A50 WITH COUNTRY_NAME= 'http://www.informationbuilders.com';
STRING2/A20 = 'user1@ibi.com';
STRING3/A20 = 'Louisa May Alcott';
Protocol/A20 = SPLIT(URL_PROTOCOL, STRING1);
Path/A50 = SPLIT(URL_PATH, STRING1);
Domain/A20 = SPLIT(EMAIL_DOMAIN, STRING2);
User/A20 =  SPLIT(EMAIL_USERID, STRING2);
First/A10 = SPLIT(NAME_FIRST, STRING3);
Last/A10 = SPLIT(NAME_LAST, STRING3);
END
TABLE FILE WF_RETAIL_LITE
SUM Protocol Path User Domain First Last
ON TABLE SET PAGE NOLEAD
END
```

下図は、出力結果を示しています。

Protocol	Path	User	Domain	First	Last
http	http://www.informationbuilders.com	user1	ibi.com	Louisa	Alcott

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、1 つまたは複数の文字で構成された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

`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

1 つまたは複数の文字で構成される区切り文字です。

TOKEN は、区切り文字が単一の文字で構成される場合に最適化されます。

`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.
```

例　末尾のブランクの削除

次のリクエストは、ディレクタ名から末尾のピリオド (.) を削除します。DIRECTOR フィールドのフォーマットは A17 のため、このフィールドのほとんどのインスタンスに末尾のブランクが含まれています。末尾のブランクを含まないフィールド (DIRECTORV) を作成するために、SQUEEZ 関数を使用して DIRECTOR フィールドの末尾のブランクすべてを単一ブランクに変換し、TRIMV 関数を使用して残りの末尾のブランクを削除した上で、結果を A17V フォーマットで格納します。これにより、実際の文字の長さが明確になります。次に DIRECTOR および DIRECTORV フィールドに対して TRIM_ 関数が呼び出され、TRIMDIR および TRIMDIRV フィールドが作成されます。TRIMDIR には DIRECTOR から末尾の文字が削除された値、TRIMDIRV には DIRECTORV から末尾の文字が削除された値が格納されます。

```
DEFINE FILE MOVIES
DIRECTORV/A17V = TRIMV('T', SQUEEZ(17, DIRECTOR, 'A17V'), 17, ' ', 1,
DIRECTORV) ;
TRIMDIR/A17 = TRIM_(TRAILING, '.', DIRECTOR);
TRIMDIRV/A17V = TRIM_(TRAILING, '.', DIRECTORV);
END
TABLE FILE MOVIES
PRINT DIRECTOR TRIMDIR DIRECTORV TRIMDIRV
ON TABLE SET PAGE NOPAGE
END
```

以下は、出力結果の一部を示しています。TRIMDIR (DIRECTOR から末尾の文字が削除された値) には末尾のピリオド (.) が残っています。これは、ピリオド (.) がフィールドの末尾の文字でないためです。TRIMDIRV (DIRECTORV から末尾の文字が削除された値) では、末尾のピリオド (.) が削除されています。

```
DIRECTOR        TRIMDIR         DIRECTORV       TRIMDIRV
--------        -------         ---------       --------
SPIELBERG S.    SPIELBERG S.    SPIELBERG S.    SPIELBERG S
KAZAN E.        KAZAN E.        KAZAN E.        KAZAN E
WELLES O.       WELLES O.       WELLES O.       WELLES O
LUMET S.        LUMET S.        LUMET S.        LUMET S
```

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
```

6

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

文字列関数の注意

多くの関数では、*output* 引数にフィールド名またはフォーマットを指定することができます。フォーマットを指定する場合、一重引用符 (') で囲みます。ただし、関数がダイアログマネージャコマンドから呼び出される場合、この引数には常にフォーマットを指定する必要があります。関数の呼び出しおよび引数の指定についての詳細は、45 ページの「 関数へのアクセスと呼び出し 」 を参照してください。

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。

ダイアログマネージャでは、接尾語「.LENGTH」を使用することにより、入力された文字列の長さを取得することができます。

構文　　文字列の長さを取得

```
ARGLEN(length, source_string, output)
```

説明

`length`
　整数

　文字列を含むフィールドの長さです。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

　文字列を含むフィールド名です。

`output`
　整数

　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。

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。

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
NAME_LEN/I3 = ARGLEN(15, LAST_NAME, NAME_LEN);
WHERE DEPARTMENT EQ 'MIS';
END
```

出力結果は次のとおりです。

```
LAST_NAME       NAME_LEN
---------       --------
SMITH                  5
JONES                  5
MCCOY                  5
BLACKWOOD              9
GREENSPAN              9
CROSS                  5
```

ASIS - ブランクと 0 (ゼロ) を区別

ASIS 関数は、ダイアログマネージャ内のブランクと 0 (ゼロ) を区別します。ASIS 関数は、数値文字列、数値文字列として定義された定数または変数 (一重引用符 (') で囲まれた数値)、数値として定義されたフィールドを区別します。変数は数値に変換されるのではなく、入力されたとおりの値として評価されます。これは、ダイアログマネージャの等価式のみに使用されます。

構文　ブランクと 0 (ゼロ) を区別

```
ASIS(argument)
```

説明

`argument`
文字

評価される値です。実際の値、値を含むフィールド名、値を返す式のいずれかを指定します。式は関数を呼び出すことができます。

文字リテラルを指定する場合、文字リテラルは一重引用符 (') で囲みます。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　ブランクと 0 (ゼロ) を区別

次のリクエストは ASIS 関数を使用しません。このため、ブランクおよび 0 (ゼロ) として定義された変数は区別されません。

```
-SET &VAR1 = ' ';
-SET &VAR2 = 0;
-IF &VAR2 EQ &VAR1 GOTO ONE;
-TYPE VAR1 &VAR1 EQ VAR2 &VAR2 NOT TRUE
-QUIT
-ONE
-TYPE VAR1 &VAR1 EQ VAR2 &VAR2 TRUE
```

出力結果は次のとおりです。

```
VAR1 EQ VAR2 0 TRUE
```

次のリクエストは ASIS 関数を使用しているため、2 つの変数は区別されます。

```
-SET &VAR1 = ' ';
-SET &VAR2 = 0;
-IF &VAR2 EQ ASIS(&VAR1) GOTO ONE;
-TYPE VAR1 &VAR1 EQ VAR2 &VAR2 NOT TRUE
-QUIT
-ONE
-TYPE VAR1 &VAR1 EQ VAR2 &VAR2 TRUE
```

出力結果は次のとおりです。

```
VAR1 EQ VAR2 0 NOT TRUE
```

参照　ASIS 使用上の注意

一般にダイアログマネージャ変数は文字値として処理されます。ただし、ダイアログマネージャ変数の値がピリオド (.) の場合、使用されるコンテキストに応じて、その変数値が文字値 (.) または数値 (0) として処理される場合があります。

- ダイアログマネージャ変数のピリオド (.) が演算式で使用される場合、その値は数値として処理されます。たとえば、次のリクエストでは、&DMVAR1 変数が演算式で使用され、その値が 0 (ゼロ) として評価されます。

```
-SET &DMVAR1='.';
-SET &DMVAR2=10 + &DMVAR1;
-TYPE DMVAR2 = &DMVAR2
```

出力結果は次のとおりです。

```
DMVAR2 = 10
```

- 次の例のように、ダイアログマネージャ変数のピリオド (.) が IF テストで使用され、比較される値がブランク、0 (ゼロ)、またはピリオド (.) の場合、ASIS 関数が使用されている場合でも、結果は TRUE になります。次の IF テストはすべて、TRUE として評価されます。

```
-SET &DMVAR1='.';
-SET &DMVAR2=IF &DMVAR1 EQ ' ' THEN 'TRUE' ELSE 'FALSE';
-SET &DMVAR3=IF &DMVAR1 EQ '.' THEN 'TRUE' ELSE 'FALSE';
-SET &DMVAR4=IF &DMVAR1 EQ '0' THEN 'TRUE' ELSE 'FALSE';
```

- 次の例のように、ダイアログマネージャ変数が ASIS 関数とともに使用されている場合、ASIS 関数の結果は常に文字値と見なされ、ブランク、0 (ゼロ)、ピリオド (.) がそれぞれ区別されます。次の IF テストはすべて、TRUE として評価されます。

```
-SET &DMVAR2=IF ASIS('.') EQ '.' THEN 'TRUE' ELSE 'FALSE';
-SET &DMVAR3=IF ASIS(' ') EQ ' ' THEN 'TRUE' ELSE 'FALSE';
-SET &DMVAR4=IF ASIS('0') EQ '0' THEN 'TRUE' ELSE 'FALSE';
```

- ASIS('0') とブランクの比較、ASIS(' ') と '0' の比較は、常に FALSE として評価されます。

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, source_string, output)
```

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。

`source_string`
文字

評価する文字列です。文字列は一重引用符 (') で囲むことも、文字列を含むフィールドまたは変数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。

`output`
整数

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。

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。

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
BIT_24/I1 = BITSON(24, LAST_NAME, BIT_24);
WHERE DEPARTMENT EQ 'MIS';
END
```

出力結果は次のとおりです。

```
LAST_NAME         BIT_24
---------         ------
SMITH                  1
JONES                  1
MCCOY                  1
BLACKWOOD              1
GREENSPAN              1
CROSS                  0
```

BITVAL - ビット列を整数として評価

BITVAL 関数は、文字列内のビット列を評価します。ビット列は文字列内の任意のビットグループであり、バイトおよびワード境界をまたがることができます。この関数は、文字列内のビットのサブセットを整数値として評価します。

ビット数に応じて、次の値が返されます。

- ビット数が 1 未満の場合、返される値は 0 (ゼロ) です。
- ビット数が 1 から 31 までの場合 (推奨される範囲)、返される値は 0 (ゼロ)、または指定したビット数 (合計が 32 ビットになるまで上位桁に 0 (ゼロ) を加えたビット数) を表す正の数です。
- ビット数が 32 の場合、返される値は、指定した 32 ビットを表す正の値、0 (ゼロ)、または負の 2 の補数値です。
- ビット数が 33 以上の場合、返される値は、指定したビット数の右端 32 ビットを表す正の値、0 (ゼロ)、または負の 2 の補数値です。

構文　ビット列を評価

```
BITVAL(source_string, startbit, number, output)
```

説明

`source_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または変数を指定することもできます。

`startbit`
整数

ビット列の左から数えて 1 ビット目の数値です。この引数が 0 (ゼロ) 以下の場合、0 (ゼロ) が返されます。

`number`
整数

評価対象となるビット列内のビット数です。この引数が 0 (ゼロ) 以下の場合、0 (ゼロ) が返されます。

`output`
整数

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。

例　　ビット列を評価

BITVAL 関数は、LAST_NAME の 12 ビット目から 20 ビット目までを評価し、結果を I5 フォーマットのフィールドに格納します。

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
STRING_VAL/I5 = BITVAL(LAST_NAME, 12, 9, 'I5');
WHERE DEPARTMENT EQ 'MIS';
END
```

出力結果は次のとおりです。

```
LAST_NAME         STRING_VAL
---------         ----------
SMITH                    332
JONES                    365
MCCOY                     60
BLACKWOOD                316
GREENSPAN                412
CROSS                    413
```

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 または Unicode の 10 進数に変換します。

構文　　文字を変換

```
BYTVAL(character, output)
```

説明

`character`
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。

`output`
整数

対応する 10 進数の値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。

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 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。

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND
COMPUTE LAST_INIT_CODE/I3 = BYTVAL(LAST_NAME, 'I3');
WHERE DEPARTMENT EQ 'MIS';
END
```

ASCII プラットフォームの出力結果は次のとおりです。

```
LAST_NAME   LAST_INIT_CODE
---------   --------------
SMITH                   83
JONES                   74
MCCOY                   77
BLACKWOOD               66
GREENSPAN               71
CROSS                   67
```

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, source_string, 'mask', output)
```

説明

`numchar`
整数

mask と比較する文字のバイト数です。

`string`
文字

確認する文字列です。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または変数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。

`output`
整数

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。

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。

```
TABLE FILE EMPLOYEE
PRINT EMP_ID AND LAST_NAME AND
COMPUTE CHK_ID/I3 = CHKFMT(9, EMP_ID, '119999999', CHK_ID);
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

出力結果は次のとおりです。

```
EMP_ID     LAST_NAME      CHK_ID
------     ---------      ------
071382660  STEVENS             1
119265415  SMITH               0
119329144  BANNING             0
123764317  IRVING              2
126724188  ROMANS              2
451123478  MCKNIGHT            1
```

CHKNUM - 文字列の数値フォーマットの確認

CHKNUM 関数は、文字列の数値フォーマットを確認します。文字列の数値フォーマットが有効な場合、CHKNUM 関数は 1 を返します。文字列の数値フォーマットに無効な文字が含まれている場合、CHKNUM 関数は 0 (ゼロ) を返します。

構文　文字列のフォーマットを確認

```
CHKNUM(numchar, source_string, output)
```

説明

`numchar`

整数

文字列のバイト数です。

`string`

文字

確認する文字列です。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または変数を指定することもできます。

output

数値

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。

例 文字列の数値フォーマットの確認

CHKNUM 関数は、STR1、STR2、STR3 文字列の数値フォーマットを検証します。

```
DEFINE FILE WF_RETAIL_LITE
STR1/A8 = '12345E01';
STR2/A8 = 'ABCDEFG';
STR3/A8 = '1234.567';
CHK1/I1= CHKNUM(8,STR1,CHK1);
CHK2/I1= CHKNUM(8,STR2,CHK2);
CHK3/I1= CHKNUM(8,STR3,CHK3);
END
TABLE FILE WF_RETAIL_LITE
PRINT STR1 IN 20 CHK1 STR2 CHK2 STR3 CHK3
BY PRODUCT_CATEGORY
WHERE PRODUCT_CATEGORY EQ 'Video Production'
ON  TABLE SET PAGE NOPAGE
ON TABLE PCHOLD FORMAT WP
END
```

出力結果は次のとおりです。

```
Product

  Category           STR1      CHK1  STR2      CHK2  STR3
CHK3
  --------           ----      ----  ----      ----  ----
----
  Video Production   12345E01     1  ABCDEFG      0  1234.567     1
                     12345E01     1  ABCDEFG      0  1234.567     1
                     12345E01     1  ABCDEFG      0  1234.567     1
                     12345E01     1  ABCDEFG      0  1234.567     1
                     12345E01     1  ABCDEFG      0  1234.567     1
                     12345E01     1  ABCDEFG      0  1234.567     1
```

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。カンマ (,) やアポストロフィ (') など、ダイアログマネージャ - PROMPT コマンドに応答する際に、入力が難しい文字の入力に使用することもできます。入力文字を一重引用符 (') で囲む必要もなくなります。

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。文字コード変換表はプラットフォームに依存するため、使用されるエンコード (ASCII または Unicode) は、プラットフォームおよび構成オプションによって異なります。

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

構文　文字を他の文字に変換

```
CTRAN(length, source_string, decimal, decvalue, output)
```

説明

`length`
整数

ソース文字列のバイト数です。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または変数を指定することもできます。

`decimal`
整数

変換する文字の ASCII 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 の 10 進コードです。

`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。

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

CTRAN 関数は、ADDRESS_LN3 のブランク (ASCII 10 進数は 32) をアンダースコア (ASCII 10 進数は 95) に変換し、ALT_ADDR に格納します。

```
TABLE FILE EMPLOYEE
PRINT ADDRESS_LN3 AND COMPUTE
ALT_ADDR/A20 = CTRAN(20, ADDRESS_LN3, 32, 95, ALT_ADDR);
BY EMP_ID
WHERE TYPE EQ 'HSM';
END
```

出力結果は次のとおりです。

```
EMP_ID     ADDRESS_LN3           ALT_ADDR
------     -----------           --------
117593129  RUTHERFORD NJ 07073   RUTHERFORD_NJ_07073_
119265415  NEW YORK NY 10039     NEW_YORK_NY_10039___
119329144  FREEPORT NY 11520     FREEPORT_NY_11520___
123764317  NEW YORK NY 10001     NEW_YORK_NY_10001___
126724188  FREEPORT NY 11520     FREEPORT_NY_11520___
451123478  ROSELAND NJ 07068     ROSELAND_NJ_07068___
543729165  JERSEY CITY NJ 07300  JERSEY_CITY_NJ_07300
818692173  FLUSHING NY 11354     FLUSHING_NY_11354
```

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(source_string, length, output)
```

説明

`source_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または変数を指定することもできます。

`length`
整数

source_string および output の長さをバイト数で指定します。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。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。

`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。

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。

```
SET STYLE=OFF
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
CENTER_NAME/A12 = CTRFLD(LAST_NAME, 12, 'A12');
WHERE DEPARTMENT EQ 'MIS'
END
```

出力結果は次のとおりです。

```
LAST_NAME         CENTER_NAME
---------         -----------
SMITH                SMITH
JONES                JONES
MCCOY                MCCOY
BLACKWOOD          BLACKWOOD
GREENSPAN          GREENSPAN
CROSS                CROSS
```

EDIT - 文字を抽出または追加

EDIT 関数は、指定したマスクに基づいてソース文字列から文字を抽出したり、文字を出力文字列に追加したりします。この関数は、ソース文字列の任意の部分からサブ文字列を抽出することができます。また、ソース文字列から抽出した文字を出力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、マスク内の文字とソース文字列内の文字を比較します。マスク内に「9」が見つかると、EDIT はソース文字列から対応する文字を出力文字列にコピーします。マスク内に「$」が見つかると、EDIT はソース文字列内の対応する文字を無視します。マスク内でその他の文字が見つかると、EDIT は出力文字列内の対応する位置に、その文字をコピーします。マスク文字すべての処理が完了した時点で、この処理は終了します。

注意

- EDIT 関数では、結果は文字フォーマットに限定され、長さはマスク値に基づいて決定されるため、output 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(source_string, 'mask');
```

説明

`source_string`
文字

文字の抽出元となる文字列です。マスク内の「9」は、それぞれ 1 桁を表します。そのため、source_string の長さは、マスク内の「9」の個数以上である必要があります。

`mask`
文字

マスク文字列です。文字列は一重引用符 (') で囲みます。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。出力フィールドの長さは、マスクの長さ (9 と $ を除く) に基づいて決定されます。

例　文字の抽出および追加

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。名前のイニシャルを抽出するための mask は、「MASK1」という名前の一時項目 (DEFINE) に保存されます。

```
DEFINE FILE EMPLOYEE
MASK1/A10 = '9$$$$$$$$$'
END
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
FIRST_INIT/A1 = EDIT(FIRST_NAME, MASK1);
EMPIDEDIT/A11 = EDIT(EMP_ID, '999-99-9999');
WHERE DEPARTMENT EQ 'MIS';
END
```

出力結果は次のとおりです。

```
LAST_NAME       FIRST_INIT  EMPIDEDIT
---------       ----------  ---------
SMITH           M           112-84-7612
JONES           D           117-59-3129
MCCOY           J           219-98-4371
BLACKWOOD       R           326-17-9357
GREENSPAN       M           543-72-9165
CROSS           B           818-69-2173
```

GETTOK - サブ文字列 (トークン) を抽出

GETTOK 関数は、ソース文字列を「トークン」と呼ばれるサブ文字列に分割します。データには「区切り文字」と呼ばれる特殊文字が含まれている必要があります。文字列は、この区切り文字に基づいてトークンに分割されます。GETTOK 関数は、token_number 引数で指定されたトークンを返します。GETTOK 関数は、ソース文字列内の先頭と末尾のブランクを無視します。

たとえば、複数の単語で構成された文から 4 つ目の単語を抽出する場合を想定します。この場合、区切り文字としてブランクを使用し、トークン番号 (token_number) に 4 を設定します。GETTOK は、この区切り文字を使用して文を単語に分割し、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　　サブ文字列 (トークン) を抽出

```
GETTOK(source_string, inlen, token_number, 'delim', outlen, output)
```

説明

`source_string`
　文字

　トークンを抽出するソース文字列です。

`inlen`
　整数

　source_string 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

　ソース文字列内の区切り文字です。区切り文字は一重引用符 (') 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。

　注意：ダイアログマネージャでは、区切り文字のブランク (' ') が倍精度 0 (ゼロ) に変換されないように、ブランクの後ろに数値以外の文字 (例、%) を追加してください。GETTOK 関数は、先頭文字 (ブランク) のみを区切り文字として使用します。追加された文字 (%) は倍精度への変換を防止するために使用されます。

`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。

output
文字

トークン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。

抽出されたトークンに、区切り文字は含まれません。

例　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。

以下のように、区切り文字にはブランクを使用します。

```
TABLE FILE EMPLOYEE
PRINT ADDRESS_LN3 AND COMPUTE
LAST_TOKEN/A10 = GETTOK(ADDRESS_LN3, 20, -1, ' ', 10, LAST_TOKEN);
AS 'LAST TOKEN,(ZIP CODE)'
WHERE TYPE EQ 'HSM';
END
```

出力結果は次のとおりです。

```
                        LAST TOKEN
ADDRESS_LN3             (ZIP CODE)
------------            ----------
RUTHERFORD NJ 07073     07073
NEW YORK NY 10039       10039
FREEPORT NY 11520       11520
NEW YORK NY 10001       10001
FREEPORT NY 11520       11520
ROSELAND NJ 07068       07068
JERSEY CITY NJ 07300    07300
FLUSHING NY 11354       11354
```

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。この関数は、すべてのアルファベットを小文字に変換します。ただし、各単語の先頭文字、および一重引用符 (') と二重引用符 (") の後の先頭文字を除きます。これらの文字は大文字に変換されます。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 は、ソース文字列内の数値および特殊文字をスキップし、それに続くアルファベットの変換を続行します。LCWORD の結果、すべての単語の先頭文字が大文字、それ以外が小文字になります。

構文　文字列を先頭大文字に変換

```
LCWORD(length, source_string, output)
```

説明

`length`
整数

source_string および output のバイト数です。

`string`
文字

変換する文字列です。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または変数を指定することもできます。

`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。この長さは、length で指定した値以上にする必要があります。

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。

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
MIXED_CASE/A15 = LCWORD(15, LAST_NAME, MIXED_CASE);
WHERE DEPARTMENT EQ 'PRODUCTION'
END
```

出力結果は次のとおりです。

```
LAST_NAME       MIXED_CASE
---------       ----------
STEVENS         Stevens
SMITH           Smith
BANNING         Banning
IRVING          Irving
ROMANS          Romans
MCKNIGHT        Mcknight
```

LCWORD2 - 文字列を先頭大文字に変換

LCWORD2 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、二重引用符 (") またはブランクは、その直後の文字が大文字に変換されることを示します。

たとえば、「SMITH」は「Smith」に、「JACK S」は「Jack S」に変換されます。

構文　文字列を先頭大文字に変換

```
LCWORD2(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

変換する文字列です。文字列を含む一時項目を指定することもできます。

`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。この長さは、length で指定した値以上にする必要があります。

例　文字列を先頭大文字に変換

LCWORD2 は、文字列「O'CONNOR's」を先頭大文字に変換します。

```
DEFINE FILE EMPLOYEE
MYVAL1/A10='O'CONNOR'S';
LC2/A10 = LCWORD2(10, MYVAL1, 'A10');
END
TABLE FILE EMPLOYEE
SUM LAST_NAME NOPRINT MYVAL1 LC2
END
```

出力結果は次のとおりです。

```
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 (') に続く文字も大文字に変換されますが、その文字の後にブランクが続く場合、またはその文字が入力文字列の最終文字の場合は大文字に変換されません。

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。この長さは、length で指定した値以上にする必要があります。

例　LCWORD3 を使用して文字列を先頭大文字に変換

次の LCWORD3 は、「O'CONNOR's」および「o'connor's」を先頭大文字の文字列に変換します。

```
DEFINE FILE EMPLOYEE
MYVAL1/A10='O'CONNOR'S';
MYVAL2/A10='o'connor's';
LC1/A10 = LCWORD3(10, MYVAL1, 'A10');
LC2/A10 = LCWORD3(10, MYVAL2, 'A10');
END
TABLE FILE EMPLOYEE
SUM LAST_NAME NOPRINT MYVAL1 LC1 MYVAL2 LC2
END
```

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、文字列を左揃えにします。文字列は、フィールド内で左揃えになります。先頭のブランクはすべて、末尾のブランクに変更されます。

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, source_string, output)
```

説明

`length`
整数

source_string および output の長さをバイト数で指定します。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

左揃えにする文字列です。文字列を含むフィールドまたは変数を指定することもできます。

`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。

例　文字列を左揃え

次のリクエストは、XNAME フィールドを作成します。このフィールドには、左揃えされていない LAST_NAME が含まれています。その後、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。

```
SET STYLE=OFF
DEFINE FILE EMPLOYEE
XNAME/A25=IF LAST_NAME EQ 'BLACKWOOD' THEN '    '|LAST_NAME ELSE
''|LAST_NAME;
YNAME/A25=LJUST(15, XNAME, 'A25');
END

TABLE FILE EMPLOYEE
PRINT LAST_NAME XNAME YNAME
END
```

出力結果は次のとおりです。

```
LAST_NAME         XNAME                YNAME
---------         -----                -----
STEVENS            STEVENS             STEVENS
SMITH              SMITH               SMITH
JONES              JONES               JONES
SMITH              SMITH               SMITH
BANNING            BANNING             BANNING
IRVING             IRVING              IRVING
ROMANS             ROMANS              ROMANS
MCCOY              MCCOY               MCCOY
BLACKWOOD             BLACKWOOD        BLACKWOOD
MCKNIGHT           MCKNIGHT            MCKNIGHT
GREENSPAN          GREENSPAN           GREENSPAN
CROSS              CROSS               CROSS
```

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, source_string, output)
```

説明

`length`
　整数

　source_string および output の長さをバイト数で指定します。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。長さは 1 以上にします。また、両方の引数の値を一致させる必要があります。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。

source_string
文字

変換する文字列です。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または変数を指定することもできます。

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。フィールド名は、source_string で指定したフィールド名と同一にすることもできます。

例　文字列を小文字に変換

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。

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
LOWER_NAME/A15 = LOCASE(15, LAST_NAME, LOWER_NAME);
WHERE DEPARTMENT EQ 'MIS';
END
```

出力結果は次のとおりです。

```
LAST_NAME       LOWER_NAME
---------       ----------
SMITH           smith
JONES           jones
MCCOY           mccoy
BLACKWOOD       blackwood
GREENSPAN       greenspan
CROSS           cross
```

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。この関数により、文字フィールドのすべてを置換することなく、一部分を編集することができます。

構文　文字列を上書き

```
OVRLAY(source_string, length, substring, sublen, position, output)
```

説明

source_string
文字

ソース文字列です。

length
整数

source_string および output の長さをバイト数で指定します。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。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。

substring
文字

source_string で指定した文字列を上書きするサブ文字列です。

sublen
整数

substring で指定したサブ文字列のバイト数です。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。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

source_string で指定した文字列の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 sublen で指定した値よりも大きい場合、この関数はソース文字列を返します。

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。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。

上書き文字列が出力フィールドよりも長い場合、文字列は出力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。

```
TABLE FILE EMPLOYEE
PRINT EMP_ID AND CURR_JOBCODE AND COMPUTE
NEW_ID/A9 = OVRLAY(EMP_ID, 9, CURR_JOBCODE, 3, 7, NEW_ID);
BY LAST_NAME BY FIRST_NAME
WHERE DEPARTMENT EQ 'MIS';
END
```

出力結果は次のとおりです。

```
LAST_NAME       FIRST_NAME  EMP_ID     CURR_JOBCODE  NEW_ID
---------       ----------  ------     ------------  ------
BLACKWOOD       ROSEMARIE   326179357  B04           326179B04
CROSS           BARBARA     818692173  A17           818692A17
GREENSPAN       MARY        543729165  A07           543729A07
JONES           DIANE       117593129  B03           117593B03
MCCOY           JOHN        219984371  B02           219984B02
SMITH           MARY        112847612  B14           112847B14
```

PARAG - テキストを行に分割

PARAG 関数は、区切り文字を使用して文字列を複数のサブ文字列に分割します。文字列の先頭から特定のバイト数をスキャンし、グループ内の末尾のブランクを区切り文字で置換することにより、最初のサブ文字列を作成します。このサブ文字列はトークンとも呼ばれます。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。

区切り文字で各トークンに分割した後、GETTOK 関数を使用して、これらのトークンをそれぞれ異なるフィールドに配置することができます。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。したがって、文字グループには少なくとも 1 つのブランクが含まれている必要があります。スキャンされた文字数は、最大トークンサイズとして指定されます。

たとえば、「subtitle」というフィールドにブランクで区切られた長いテキストが存在する場合、最大トークンサイズを指定することにより、このフィールドをほぼ均等な長さのサブ文字列に分割することができます。このフィールドのサイズが 350 バイトの場合、最大トークンサイズとして 120 バイトを指定することにより、3 つのサブ文字列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。

ヒント：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。この行は、60 バイトよりも長くなります。

構文　テキストを行に分割

```
PARAG(length, source_string, 'delimiter', max_token_size, output)
```

説明

length
整数

source_string および output の長さをバイト数で指定します。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

トークンに分割する文字列です。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または変数を指定することもできます。

delimiter
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。

max_token_size
整数

各トークンのサイズの上限値です。

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。

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。

```
TABLE FILE EMPLOYEE
PRINT ADDRESS_LN2 AND COMPUTE
PARA_ADDR/A20 = PARAG(20, ADDRESS_LN2, ',', 10, PARA_ADDR);
BY LAST_NAME
WHERE TYPE EQ 'HSM';
END
```

出力結果は次のとおりです。

```
LAST_NAME       ADDRESS_LN2           PARA_ADDR
---------       -----------           ---------
BANNING         APT 4C                APT 4C    ,
CROSS           147-15 NORTHERN BLD   147-15,NORTHERN,BLD
GREENSPAN       13 LINDEN AVE.        13 LINDEN,AVE.
IRVING          123 E 32 ST.          123 E 32,ST.      ,
JONES           235 MURRAY HIL PKWY   235 MURRAY,HIL PKWY
MCKNIGHT        117 HARRISON AVE.     117,HARRISON,AVE.
ROMANS          271 PRESIDENT ST.     271,PRESIDENT,ST.
SMITH           136 E 161 ST.         136 E 161,ST.
```

PATTERN - 文字列からパターンを生成

PATTERN 関数は、ソース文字列を分析し、ソース文字列の数字、小文字、大文字の配列を表すパターンを作成します。この関数は、データを分析して標準パターンに適合したデータであることを確認する場合に役立ちます。

出力パターンは次のようになります。

- 1 バイトの入力文字は「9」で表されます。
- 大文字は「A」、小文字は「a」で表されます。欧州 NLS モード (西ヨーロッパ、中央ヨーロッパ) では、「A」および「a」はアクセント記号付きアルファベットにも適用されます。
- 日本語の場合、2 バイト文字と半角カタカナは大文字の「C」で表示されます。2 バイト文字には、ひらがな、カタカナ、漢字、全角英数字、および全角記号が含まれます。中国語や韓国語などのすべての 2 バイト文字も「C」で表示されます。
- 特殊文字はそのまま表示されます。
- 表示不可の文字は「X」で表示されます。

構文　入力文字列からパターンを作成

```
PATTERN (length, source_string,  output)
```

説明

`length`

数値

source_string の長さです。

`source_string`

文字

ソース文字列です。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。

`output`

文字

結果を格納するフィールド名、または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 (') で囲みます。

例　文字データからパターンを作成

TESTFILE という名前の固定フォーマットシーケンシャルファイル (LRECL 14) には、次の 19 件のレコードが格納されています。

```
212-736-6250
212 736 4433
123-45-6789
800-969-INFO
10121-2898
10121
2 Penn Plaza
917-339-6380
917-339-4350
(212) 736-6250
(212) 736-4433
212-736-6250
212-736-6250
212-736-6250
(212) 736 5533
(212) 736 5533
(212) 736 5533
10121 Æ
800-969-INFO
```

マスターファイルは次のとおりです。

```
FILENAME=TESTFILE, SUFFIX=FIX     ,
  SEGMENT=TESTFILE, SEGTYPE=S0, $
    FIELDNAME=TESTFLD, USAGE=A14, ACTUAL=A14, $
```

次のリクエストは、TESTFLD の各インスタンスのパターンを生成し、生成されたパターン別にインスタンスを表示します。また、このリクエストは各パターンの個数を集計し、総数に対するパーセントを表示します。PRINT コマンドは、生成された各パターンの TESTFLD の値を表示します。

```
FILEDEF TESTFILE DISK testfile.ftmDEFINE FILE TESTFILE
   PATTERN/A14 = PATTERN (14, TESTFLD, 'A14' ) ;
END
TABLE FILE TESTFILE
  SUM CNT.PATTERN AS 'COUNT' PCT.CNT.PATTERN AS 'PERCENT'
   BY PATTERN
 PRINT TESTFLD
   BY PATTERN
ON TABLE COLUMN-TOTAL
END
```

最後から 2 つ目の行を見ると、入力文字列に表示不可の文字が含まれていたため、生成されたパターンではその文字が「X」で表示されています。それ以外の出力文字列では、数字は「9」、大文字は「A」、小文字は「a」で表示されています。出力結果は次のとおりです。

```
PATTERN         COUNT  PERCENT  TESTFLD
-------         -----  -------  -------
(999) 999 9999      3    15.79  (212) 736 5533
                                (212) 736 5533
                                (212) 736 5533
(999) 999-9999      2    10.53  (212) 736-6250
                                (212) 736-4433
9 Aaaa Aaaaa        1     5.26  2 Penn Plaza
999 999 9999        1     5.26  212 736 4433
999-99-9999         1     5.26  123-45-6789
999-999-AAAA        2    10.53  800-969-INFO
                                800-969-INFO
999-999-9999        6    31.58  212-736-6250
                                917-339-6380
                                917-339-4350
                                212-736-6250
                                212-736-6250
                                212-736-6250
99999               1     5.26  10121
99999 X             1     5.26  10121 Æ
99999-9999          1     5.26  10121-2898
TOTAL              19   100.00
```

POSIT - サブ文字列の開始位置を検索

POSIT 関数は、ソース文字列内のサブ文字列の開始位置を検索します。たとえば、文字列 PRODUCTION 内のサブ文字列 DUCT の開始位置は 4 です。指定したサブ文字列がソース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(source_string, length, substring, sublength, output)
```

説明

`source_string`
文字

解析する文字列です。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または変数を指定することもできます。

`length`
整数

source_string で指定した文字列のバイト数です。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。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

位置を検索するサブ文字列です。一重引用符 (') で囲まれたサブ文字列、またはサブ文字列を含むフィールドです。

`sublength`
整数

substring のバイト数です。この引数が 0 (ゼロ) 以下の場合、または length で指定した値よりも大きい場合、0 (ゼロ) が返されます。

`output`
整数

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。

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。

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
I_IN_NAME/I2 = POSIT(LAST_NAME, 15, 'I', 1, 'I2');
WHERE DEPARTMENT EQ 'PRODUCTION'
END
```

出力結果は次のとおりです。

```
LAST_NAME         I_IN_NAME
---------         ---------
STEVENS                   0
SMITH                     3
BANNING                   5
IRVING                    1
ROMANS                    0
MCKNIGHT                  5
```

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。末尾のブランクも対象になり、入れ替えを行うと先頭のブランクになります。ただし、SET SHOWBLANKS=OFF (デフォルト値) の HTML レポートでは、先頭のブランクは表示されません。

構文　文字列の順序を入れ替え

```
REVERSE(length, source_string, output)
```

説明

`length`

整数

source_string および output の長さをバイト数で指定します。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

入れ替える文字列です。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。

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。

例 文字列の順序を入れ替え

EMPLOYEE データソースに対する以下のリクエストでは、REVERSE 関数を使用して LAST_NAME フィールドの文字を入れ替え、REVERSE_LAST という名前のフィールドを作成します。このフィールドでは、LAST_NAME フィールドにある末尾のブランクが、先頭のブランクになっています。TRIM 関数を使用して REVERSE_LAST から先頭のブランクを削除し、TRIM_REVERSE という名前のフィールドを作成します。

```
DEFINE FILE EMPLOYEE
REVERSE_LAST/A15 = REVERSE(15, LAST_NAME, REVERSE_LAST);
TRIM_REVERSE/A15 = TRIM('L', REVERSE_LAST, 15, ' ', 1, 'A15');
END
TABLE FILE EMPLOYEE
PRINT REVERSE_LAST TRIM_REVERSE
BY LAST_NAME
END
```

出力結果は次のとおりです。

```
LAST_NAME       REVERSE_LAST     TRIM_REVERSE
---------       ------------     ------------
BANNING                GNINNAB  GNINNAB
BLACKWOOD            DOOWKCALB  DOOWKCALB
CROSS                    SSORC  SSORC
GREENSPAN            NAPSNEERG  NAPSNEERG
IRVING                  GNIVRI  GNIVRI
JONES                    SENOJ  SENOJ
MCCOY                    YOCCM  YOCCM
MCKNIGHT               THGINKCM  THGINKCM
ROMANS                   SNAMOR  SNAMOR
SMITH                     HTIMS  HTIMS
                          HTIMS  HTIMS
STEVENS                 SNEVETS  SNEVETS
```

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, source_string, output)
```

説明

`length`
整数

source_string および output の長さをバイト数で指定します。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。位置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。

`source_string`
文字

右揃えにする文字列です。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または変数を指定することもできます。

`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。

例　文字列を右揃え

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。

```
SET STYLE=OFF
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
RIGHT_NAME/A15 = RJUST(15, LAST_NAME, RIGHT_NAME);
WHERE DEPARTMENT EQ 'MIS';
END
```

出力結果は次のとおりです。

```
LAST_NAME         RIGHT_NAME
---------         ----------
SMITH                  SMITH
JONES                  JONES
MCCOY                  MCCOY
BLACKWOOD          BLACKWOOD
GREENSPAN          GREENSPAN
CROSS                  CROSS
```

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(length, source_string, output)
```

説明

`length`
文字

source_string で指定した文字列のバイト数です。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。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。

`source_string`
文字

分析する文字列です。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または変数を指定することもできます。

`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。

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 には、音声コードが一致する場合は YES が、一致しない場合は NO が含まれます。

WHERE 条件は、想定した音声コードに一致する LAST_NAME を選択します。

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

TABLE FILE EMPLOYEE
PRINT LAST_NAME
IF PHON_MATCH IS 'YES'
END
```

出力結果は次のとおりです。

```
LAST_NAME
---------
MCCOY
```

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, output)
```

説明

`outlength`
整数

output で指定した文字列のバイト数です。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。

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`
文字または数値

文字に変換する数字です。この値は小数点以下 2 桁を含む必要があります。

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。

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。

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL AND COMPUTE
AMT_IN_WORDS/A82 = SPELLNM(82, CURR_SAL, AMT_IN_WORDS);
WHERE DEPARTMENT EQ 'MIS'
END
```

出力結果は次のとおりです。

```
CURR_SAL     AMT_IN_WORDS
--------     ------------

$13,200.00  THIRTEEN THOUSAND TWO HUNDRED DOLLARS AND NO CENTS
$18,480.00  EIGHTEEN THOUSAND FOUR HUNDRED EIGHTY DOLLARS AND NO CENTS
$18,480.00  EIGHTEEN THOUSAND FOUR HUNDRED EIGHTY DOLLARS AND NO CENTS
$21,780.00  TWENTY-ONE THOUSAND SEVEN HUNDRED EIGHTY DOLLARS AND NO CENTS
$9,000.00   NINE THOUSAND DOLLARS AND NO CENTS
$27,062.00  TWENTY-SEVEN THOUSAND SIXTY-TWO DOLLARS AND NO CENTS
```

SQUEEZ - 複数のブランクを 1 つに変換

SQUEEZ 関数は、文字列内の連続するブランクを 1 つにします。結果の文字列の長さは元の文字列と同一になりますが、右側にブランクが追加されます。

構文　複数のブランクを 1 つに変換

```
SQUEEZ(length, source_string, output)
```

説明

length
整数

source_string および output の長さをバイト数で指定します。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

複数のブランクを 1 つに変換する文字列です。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または変数を指定することもできます。

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。

例　複数のブランクを 1 つに変換

SQUEEZ 関数は、NAME フィールド内の複数のブランクを 1 つのブランクに変換し、結果を A30 フォーマットでフィールドに格納します。

```
DEFINE FILE EMPLOYEE
NAME/A30 = FIRST_NAME | LAST_NAME;
END
TABLE FILE EMPLOYEE
PRINT NAME AND COMPUTE
SQNAME/A30 = SQUEEZ(30, NAME, 'A30');
WHERE DEPARTMENT EQ 'MIS';
END
```

出力結果は次のとおりです。

```
NAME                          SQNAME
----                          ------
MARY        SMITH             MARY SMITH
DIANE       JONES             DIANE JONES
JOHN        MCCOY             JOHN MCCOY
ROSEMARIE   BLACKWOOD         ROSEMARIE BLACKWOOD
MARY        GREENSPAN         MARY GREENSPAN
BARBARA     CROSS             BARBARA CROSS
```

STRIP - 文字列から文字を削除

STRIP 関数は、文字列から特定の文字をすべて削除します。結果の文字列の長さは元の文字列と同一になりますが、右側にブランクが追加されます。

構文　文字列から文字を削除

```
STRIP(length, source_string, char, output)
```

説明

`length`
整数

source_string および output で指定した文字列のバイト数です。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

文字の削除元となる文字列です。文字列を含むフィールドを指定することもできます。

`char`
文字

文字列から削除する文字です。一重引用符 (') で囲まれた文字リテラル、またはその文字を含むフィールドを指定することもできます。2 文字以上を指定した場合、最も左側の文字が削除文字として使用されます。

注意：一重引用符 (') を削除するには、2 つの一重引用符 ('') を使用します。 さらに、この文字の組み合わせを一重引用符 (') で囲む必要があります。

`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。

例　文字列から文字を削除

STRIP 関数は、DIRECTOR フィールドのピリオド (.) をすべて削除し、結果を A17 フォーマットでフィールドに格納します。

```
TABLE FILE MOVIES
PRINT DIRECTOR AND COMPUTE
SDIR/A17 = STRIP(17, DIRECTOR, '.', 'A17');
WHERE CATEGORY EQ 'COMEDY'
END
```

出力結果は次のとおりです。

```
DIRECTORS       SDIR
---------       ----
ZEMECKIS R.     ZEMECKIS R
ABRAHAMS J.     ABRAHAMS J
ALLEN W.        ALLEN W
HALLSTROM L.    HALLSTROM L
MARSHALL P.     MARSHALL P
BROOKS J.L.     BROOKS JL
```

例　文字列から一重引用符を削除

STRIP 関数は、TITLE フィールドから一重引用符 (') をすべて削除し、結果を A39 フォーマットでフィールドに格納します。

```
TABLE FILE MOVIES
PRINT TITLE AND COMPUTE
STITLE/A39 = STRIP(39, TITLE, '''', 'A39');
WHERE TITLE CONTAINS ''''
END
```

出力結果は次のとおりです。

```
TITLE                           STITLE
-----                           ------
BABETTE'S FEAST                 BABETTES FEAST
JANE FONDA'S COMPLETE WORKOUT   JANE FONDAS COMPLETE WORKOUT
JANE FONDA'S NEW WORKOUT        JANE FONDAS NEW WORKOUT
MICKEY MANTLE'S BASEBALLTIPS    MICKEY MANTLES BASEBALL TIPS
```

STRREP - 文字列を置換

STRREP 関数は、ソース文字列内の特定の文字列のインスタンスをすべて置換します。NULL 文字列による置換も可能です。

構文　文字列を置換

```
STRREP (inlength, instring, searchlength, searchstring, replength,
repstring, outlength, output)
```

説明

`inlength`

数値

source_string のバイト数です。

`instring`

文字

ソース文字列です。

`searchlength`

数値

置換される文字列のバイト数です。

`searchstring`

文字

置換される文字列です。

`replength`

数値

置換文字列のバイト数です。この値は、0 (ゼロ) 以上である必要があります。

`repstring`

文字

置換文字列 (英数文字) です。replength が 0 (ゼロ) の場合は無視されます。

`outlength`

数値

結果として出力される文字列のバイト数です。この値は 1 以上である必要があります。

output

文字

すべての置換およびパディングが行われた後に出力される文字列です。

参照 STRREP 関数使用上の注意

長さの最大値は 4095 です。

例 カンマとドル記号を置換

次の例では、STRREP 関数は、CS_ALPHA フィールド内のカンマ (,) とドル記号 ($) を検索して置換します。置換作業では、まずカンマ (,) を NULL 文字列で置き換えて CS_NOCOMMAS (カンマの削除) を生成し、次に右側の CURR_SAL フィールドのドル記号 ($) を (USD) に置き換えます。

```
TABLE FILE EMPLOYEE
SUM CURR_SAL NOPRINT
COMPUTE CS_ALPHA/A15=FTOA(CURR_SAL,'(D12.2M)',CS_ALPHA);
        CS_NOCOMMAS/A14=STRREP(15,CS_ALPHA,1,',',0,'X',14,CS_NOCOMMAS);
        CS_USD/A17=STRREP(14,CS_NOCOMMAS,1,'$',4,'USD ',17,CS_USD);
        NOPRINT
        CS_USD/R AS CURR_SAL
BY LAST_NAME
END
```

出力結果は次のとおりです。

```
LAST_NAME    CS_ALPHA             CS_NOCOMMAS                 CURR_SAL
---------    --------             -----------         -----------------
BANNING           $29,700.00           $29700.00          USD 29700.00
BLACKWOOD         $21,780.00           $21780.00          USD 21780.00
CROSS             $27,062.00           $27062.00          USD 27062.00
GREENSPAN          $9,000.00            $9000.00          USD  9000.00
IRVING            $26,862.00           $26862.00          USD 26862.00
JONES             $18,480.00           $18480.00          USD 18480.00
MCCOY             $18,480.00           $18480.00          USD 18480.00
MCKNIGHT          $16,100.00           $16100.00          USD 16100.00
ROMANS            $21,120.00           $21120.00          USD 21120.00
SMITH             $22,700.00           $22700.00          USD 22700.00
STEVENS           $11,000.00           $11000.00          USD 11000.00
```

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(length, source_string, start, end, sublength, output)
```

説明

`length`
整数

source_string で指定した文字列のバイト数です。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

サブ文字列の抽出元となる文字列です。文字列は一重引用符 (') で囲みます。ソース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

ソース文字列内のサブ文字列の開始位置です。start の値が 1 より小さい場合、または length で指定した値より大きい場合、この関数はブランクを返します。

`end`
整数

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

`sublength`
整数

サブ文字列の長さです。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 よりも大きい場合はサブ文字列の末尾にブランクが追加され、小さい場合はサブ文字列の末尾が切り捨てられます。この値は、output で宣言した長さに一致させる必要があります。sublength の長さ分の文字のみが処理されます。

`output`
文字

結果が返されるフィールド、または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。

```
TABLE FILE EMPLOYEE
PRINT
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, I_SUBSTR);
BY LAST_NAME
WHERE DEPARTMENT EQ 'PRODUCTION'
END
```

出力結果は次のとおりです。

```
LAST_NAME      I_IN_NAME  I_SUBSTR
---------      ---------  --------
BANNING                5  ING
IRVING                 1  IRV
MCKNIGHT               5  IGH
ROMANS                 0
SMITH                  3  ITH
STEVENS                0
```

ROMANS と STEVENS には「I」が含まれていないため、SUBSTR 関数はブランクの文字列を抽出します。

TRIM - 先頭と末尾の文字を削除

TRIM 関数は、文字列内のあるパターンの先頭と末尾の文字を削除します。

構文　先頭と末尾の文字を削除

```
TRIM(trim_where, source_string, length, pattern, sublength, output)
```

説明

`trim_where`
文字

次のオプションのいずれかを選択して、削除するパターンの位置を指定します。

L - 先頭の文字を削除します。

T - 末尾の文字を削除します。

B - 先頭と末尾の両方の文字を削除します。

source_string
文字

先頭または末尾の文字を削除する文字列です。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。

pattern
文字

削除する文字列パターンです。文字列は一重引用符 (') で囲みます。

sublength
整数

pattern のバイト数です。

output
文字

結果が返されるフィールド、または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　先頭文字を削除

TRIM 関数は、DIRECTOR の先頭の BR を削除し、結果を A17 フォーマットのフィールドに格納します。

```
TABLE FILE MOVIES
PRINT  DIRECTOR AND
COMPUTE
  TRIMDIR/A17 = TRIM('L', DIRECTOR, 17, 'BR', 2, 'A17');
  WHERE DIRECTOR CONTAINS 'BR'
END
```

出力結果は次のとおりです。

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

例　末尾文字を削除

TRIM 関数は、TITLE から末尾文字「ER」を削除します。末尾のブランク以外の文字を適切に削除するためには、まず末尾のブランクを削除する必要があります。TITLE フィールドの末尾にはブランクが含まれているため、TRIM 関数は TRIMT フィールドの作成時に文字列「ER」を削除しません。一方、SHORT フィールドの末尾にはブランクが含まれていないため、TRIM 関数は TRIMS フィールドの作成時に末尾文字列「ER」を削除することができます。

```
DEFINE FILE MOVIES
SHORT/A19 = SUBSTR(19, TITLE, 1, 19, 19, SHORT);
END
TABLE FILE MOVIES
PRINT  TITLE  IN 1  AS 'TITLE: '
       SHORT  IN 40 AS 'SHORT: ' OVER
COMPUTE
  TRIMT/A39 = TRIM('T', TITLE, 39, 'ER', 2, 'A39'); IN 1 AS 'TRIMT: '
COMPUTE
  TRIMS/A19 = TRIM('T', SHORT, 19, 'ER', 2, 'A19'); IN 40 AS 'TRIMS: '
WHERE TITLE LIKE '%ER'
END
```

出力結果は次のとおりです。

```
TITLE:   LEARN TO SKI BETTER          SHORT:   LEARN TO SKI BETTER
TRIMT:   LEARN TO SKI BETTER          TRIMS:   LEARN TO SKI BETT
TITLE:   FANNY AND ALEXANDER          SHORT:   FANNY AND ALEXANDER
TRIMT:   FANNY AND ALEXANDER          TRIMS:   FANNY AND ALEXAND
```

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。

構文　テキストを大文字に変換

```
UPCASE(length, source_string, output)
```

説明

`length`
　整数

　source_string および output のバイト数です。

source_string
文字

変換する文字列です。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。

output
文字 (A*n* または A*n*V)

結果が返されるフィールド、または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文字列を大文字に変換

UPCASE 関数は、LAST_NAME_MIXED フィールドを大文字に変換します。

```
DEFINE FILE EMPLOYEE
LAST_NAME_MIXED/A15=IF DEPARTMENT EQ 'MIS' THEN LAST_NAME ELSE
  LCWORD(15, LAST_NAME, 'A15');
LAST_NAME_UPPER/A15=UPCASE(15, LAST_NAME_MIXED, 'A15') ;
END

TABLE FILE EMPLOYEE
PRINT LAST_NAME_MIXED AND FIRST_NAME BY LAST_NAME_UPPER
WHERE CURR_JOBCODE EQ 'B02' OR 'A17' OR 'B04';
END
```

このリクエストを実行すると、名前は正しくソートされます。

出力結果は次のとおりです。

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

値がすべて大文字のフィールドを非表示にするには、そのフィールドに対して NOPRINT を指定します。

XMLDECOD - XML エンコード文字のデコード

XMLDECOD 関数は、次の 5 つの標準 XML エンコード文字が文字列内に存在する場合に、これらの文字をデコードします。

文字名	文字	XML エンコード表記
アンパサンド	&	&amp;
不等号 (より大きい)	>	&gt;
不等号 (より小さい)	<	&lt;
二重引用符	"	&quot;
一重引用符 (アポストロフィ)	'	&apos;

構文

XML エンコード文字のデコード

```
XMLDECOD(inlength, source_string, outlength,  output)
```

説明

`inlength`

整数

ソース文字列が格納されているフィールドの長さです。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

ソース文字列が格納されているフィールドの名前、または一重引用符 (') で囲まれた文字列です。

`outlength`

整数

出力文字列の長さです。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。

`output`

整数

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。

例 XML エンコード文字のデコード

XMLFUNCS ファイルは、エンコードされていない文字と XML エンコード文字の両方が含まれた .csv ファイルです。マスターファイルは次のとおりです。

```
FILE = XMLFUNCS, SUFFIX=COM,$
SEGNAME = SEG01, SEGTYPE=S1,$
FIELD=INSTRING, ALIAS=CHARS, USAGE=A30,ACTUAL=A30,$
```

このファイルのコンテンツは次のとおりです。

```
CHARS: &  < >  ,$
ENCODED: &amp; &gt;  ,$
ENCODED: &quot; &apos;   ,$
MIXED:  &amp;  < &gt;   ,$
```

XMLDECOD 関数は、サポートされている XML エンコード文字をデコードします。一部のビューアではエンコードされた値が自動的にデコードされて表示されるため、出力がテキストフォーマット (FORMAT WP) で生成されます。

```
FILEDEF XMLFUNCS DISK xmlfuncs.csv
DEFINE FILE XMLFUNCS
OUTSTRING/A30=XMLDECOD(30,INSTRING,30,'A30');
END
TABLE FILE XMLFUNCS
PRINT INSTRING OUTSTRING
ON TABLE PCHOLD FORMAT WP
ON TABLE SET PAGE NOPAGE
```

出力文字列では、XML エンコード文字がデコードされ、エンコードされていない元の文字は入力文字列のとおりに表示されます。

```
INSTRING                        OUTSTRING
--------                        ---------
CHARS: &  < >                   CHARS: &  < >
ENCODED: &amp; &gt;             ENCODED: & >
ENCODED: &quot; &apos;          ENCODED: " '
MIXED:  &amp;  < &gt;           MIXED:  &  < >
```

XMLENCOD - 文字の XML エンコード

XMLENCOD 関数は、次の 5 つの標準文字が文字列内に存在する場合に、これらの文字をエンコードします。

文字名	文字	エンコード表記
アンパサンド	&	&amp;

文字名	文字	エンコード表記
不等号 (より大きい)	>	>
不等号 (より小さい)	<	<
二重引用符	"	"
一重引用符 (アポストロフィ)	'	'

構文　文字の XML エンコード

```
XMLENCOD(inlength, source_string, option, outlength,  output)
```

説明

`inlength`

整数

ソース文字列が格納されているフィールドの長さです。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

ソース文字列が格納されているフィールドの名前、または一重引用符 (') で囲まれた文字列です。

`option`

整数

XML エンコード文字が含まれた文字列を処理するかどうかを指定するコードです。有効な値には、次のものがあります。

- 0 (デフォルト) - 1 つ以上の XML エンコード文字が含まれた文字列を処理しません。
- 1 - XML エンコード文字が含まれた文字列を処理します。

`outlength`

整数

出力文字列の長さです。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。

注意：出力文字列の長さは、最大の場合でも、入力文字列の長さの 6 倍になります。

output

整数

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。

例 文字の XML エンコード

XMLFUNCS ファイルは、エンコードされていない文字と XML エンコード文字の両方が含まれた .csv ファイルです。マスターファイルは次のとおりです。

```
FILE = XMLFUNCS, SUFFIX=COM,$
SEGNAME = SEG01, SEGTYPE=S1,$
FIELD=INSTRING, ALIAS=CHARS, USAGE=A30,ACTUAL=A30,$
```

このファイルのコンテンツは次のとおりです。

```
CHARS: &  < >  ,$
ENCODED: &amp; &gt;  ,$
ENCODED: &quot; &apos;   ,$
MIXED:  &amp;  < &gt;   ,$
```

XMLENCOD 関数は、サポートされている文字のすべてを XML でエンコードし、OUTSTRING1 を生成します。OUTSTRING1 では、入力文字列に XML エンコード文字が含まれているかどうかに関係なく、すべての入力文字列が処理されます。OUTSTRING2 では、XML エンコード文字が 1 つも含まれていない入力文字列のみがエンコードされます。一部のビューアではエンコードされた値が自動的にデコードされて表示されるため、出力がテキストフォーマット (FORMAT WP) で生成されます。

```
FILEDEF XMLFUNCS DISK xmlfuncs.csv
DEFINE FILE XMLFUNCS
OUTSTRING1/A30=XMLENCOD(30,INSTRING,1,30,'A30');
OUTSTRING2/A30=XMLENCOD(30,INSTRING,0,30,'A30');
END
TABLE FILE XMLFUNCS
PRINT INSTRING OUTSTRING1 IN 24 OUTSTRING2 IN 48
ON TABLE SET PAGE NOPAGE
ON TABLE PCHOLD FORMAT WP
END
```

OUTSTRING1 では、サポートされている文字が XML でエンコードされ、入力文字列にエンコード文字が含まれている場合でも、出力が生成されます。OUTSTRING2 では、入力文字列に XML エンコード文字が 1 つも含まれていない場合にのみ出力が生成されます。

```
INSTRING                OUTSTRING1                 OUTSTRING2
--------                ----------                 ----------
CHARS: &  < >           CHARS: &amp;  &lt; &gt; CHARS: &amp;  &lt; &gt;
ENCODED: &amp; &gt;     ENCODED: &amp; &gt;
ENCODED: &quot; &apos; ENCODED: &quot; &apos;
MIXED:  &amp;  < &gt;   MIXED:  &amp;  &lt; &gt;
```

7

可変長文字列関数

文字フォーマット AnV は、FOCUS、XFOCUS、およびリレーショナルデータソースのシノニムでサポートされます。このフォーマットは、リレーショナルデータベース管理システムでサポートされる VARCHAR (可変長文字) データタイプを表すために使用します。

トピックス

概要

リレーショナルデータソースでは、AnV は VARCHAR フィールドの実際の長さの情報を保持します。この情報は、異なる RDBMS 内の VARCHAR フィールドに値を入力するときに特に役立ちます。この情報は、文字列連結時に末尾のブランクを保持するかどうかに影響します。また Oracle の場合は、文字列の比較に影響します。なお、Oracle 以外のリレーショナルエンジンでは、文字列連結の末尾のブランクは無視されます。

ibi™ FOCUS® および XFOCUS データソースでは、AnV は、実際の可変長文字サポートを提供するわけではありません。固定長文字列フィールドの先頭に 2 バイトを追加し、フィールドに格納されるデータの実際の長さを保持します。この長さは、2 バイトを占有する短整数値として格納されます。2 バイトのオーバーヘッドと、これを除去するための追加処理が必要になるため、非リレーショナルデータでの AnV フォーマットの使用は推奨されません。

AnV フィールドは、引数に文字をとるすべての関数の引数として使用することができます。AnV 入力パラメータは An パラメータとして扱われ、宣言されたサイズ (n) になるようブランクが追加されます。最後のパラメータが AnV フォーマットの場合、関数の結果は、AnV タイプに変換され、実際の長さはこのサイズに設定されます。

ここで説明する関数は、AnV データタイプのパラメータのみをとります。

参照　関数での AnV フィールド使用上の注意

関数で AnV フィールドを使用するにあたり、次のことに注意してください。

- 関数で AnV 引数を使用する際、入力パラメータは An パラメータとして扱われ、宣言されたサイズ (n) になるようブランクが追加されます。最後のパラメータが AnV フォーマットの場合、関数の結果は、AnV タイプに変換され、実際の長さはこのサイズに設定されます。
- 多くの関数は、入力引数として文字列とその長さが両方必要です。指定された文字列が AnV フィールドに格納される場合でも、関数の要件を満たすためには文字列の長さ引数を指定する必要があります。ただし、関数の計算に使用されるのは、AnV フィールドの先頭の 2 バイトとして格納された実際の長さです。
- 入力引数には、フィールドとリテラルのいずれも使用できます。ほとんどの場合、これらの関数には数値の入力引数がリテラルとして指定され、これらは整数値として指定されます。ただし、指定される値が整数ではない場合、フィールドかリテラルかに関係なく、数値は切り捨てられて整数値になります。

LENV - 文字フィールドの長さを取得

LENV 関数は、AnV 入力フィールドの実際の長さ、または An フィールドのサイズを返します。

構文　文字フィールドの長さを取得

```
LENV(source_string, output)
```

説明

`source_string`
文字 (An または AnV)

ソース文字列またはフィールドです。An フォーマットのフィールドの場合、この関数はフィールドのサイズ n を返します。一重引用符で囲んだ文字列、または変数の場合、その文字列または変数のサイズが返されます。AnV フォーマットのフィールドの場合は、フィールドのバイト長が返されます。

output
整数

結果が返されるフィールド、または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　AnV フィールドの長さを検索

TRIMV 関数は、TITLE 値から末尾のブランクを削除することにより、「TITLEV」という名前の AnV フィールドを作成します。その後、LENV 関数は TITLEV の各インスタンスの実際の長さを ALEN フィールドに返します。

```
TABLE FILE MOVIES
PRINT
COMPUTE TITLEV/A39V = TRIMV('T', TITLE, 39, ' ', 1, TITLEV);
        ALEN/I2 = LENV(TITLEV,ALEN);
BY CATEGORY NOPRINT
WHERE CATEGORY EQ 'CHILDREN'
END
```

出力結果は次のとおりです。

```
TITLEV                                    ALEN
------                                    ----
SMURFS, THE                                 11
SHAGGY DOG, THE                             15
SCOOBY-DOO-A DOG IN THE RUFF                28
ALICE IN WONDERLAND                         19
SESAME STREET-BEDTIME STORIES AND SONGS     39
ROMPER ROOM-ASK MISS MOLLY                  26
SLEEPING BEAUTY                             15
BAMBI                                        5
```

LOCASV - 可変長小文字列を作成

LOCASV 関数は、LOCASE 関数と同様、ソース文字列内の文字を小文字に変換します。LOCASV は、AnV ソース文字列と入力パラメータの upper_limit 値のうち、実際の長さよりも小さい AnV 出力を返します。

構文　可変長小文字列を作成

```
LOCASV(upper_limit, source_string, output)
```

説明

upper_limit
整数

ソース文字列の長さ制限値です。

`source_string`
文字 (A*n* または A*n*V)

小文字に変換する文字列です。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または変数を指定することもできます。フィールドを指定した場合、フォーマットには A*n* または A*n*V を使用します。A*n*V タイプのフィールドの場合、長さはフィールドのバイト長から取得されます。upper_limit で指定した長さが実際の長さよりも小さい場合、ソース文字列はこの長さ制限値で切り捨てられます。

`output`
文字 (A*n* または A*n*V)

結果を格納するフィールド名、または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。この値は、AnV または An フォーマットのフィールドに使用することができます。

出力フォーマットが A*n*V の場合、返される実際の長さは、ソース文字列長または制限値のいずれか小さい方になります。

例　可変長小文字列を作成

この例では、LOCASV は LAST_NAME フィールドを小文字に変換し、長さを 5 バイトに制限します。その結果は LOWCV_NAME フィールドに格納されます。

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
LOWCV_NAME/A15V = LOCASV(5, LAST_NAME, LOWCV_NAME);
WHERE DEPARTMENT EQ 'MIS';
END
```

出力結果は次のとおりです。

```
LAST_NAME       LOWCV_NAME
---------       ----------
SMITH           smith
JONES           jones
MCCOY           mccoy
BLACKWOOD       black
GREENSPAN       green
CROSS           cross
```

POSITV - 可変長サブ文字列の開始位置を検索

POSITV 関数は、文字列内のサブ文字列の開始位置を検索します。たとえば、文字列 PRODUCTION 内のサブ文字列 DUCT の開始位置は 4 です。指定したサブ文字列がソース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。 POSITV は POSIT に類似の関数です。だたし、A*n*V パラメータの長さは、サイズを指定する他の 2 つのパラメータとの比較による、これらのパラメータの実際の長さを基準としています。

構文　　可変長サブ文字列の開始位置を検索

```
POSITV(source_string, upper_limit, substring, sub_limit, output)
```

説明

`source_string`
文字 (A*n* または A*n*V)

ソース文字列です。このソース文字列内のサブ文字列の位置が検索されます。文字列は一重引用符 (') で囲みます。ソース文字列を含むフィールドまたは変数を指定することもできます。AnV フォーマットのフィールドの場合、長さはフィールドのバイト長から取得されます。upper_limit で指定した長さが実際の長さよりも小さい場合、ソース文字列はこの長さ制限値で切り捨てられます。

`upper_limit`
整数

ソース文字列の長さ制限値です。

`substring`
文字 (A*n* または A*n*V)

位置を検索するサブ文字列です。サブ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または変数を指定することもできます。フィールドを指定した場合、フォーマットには A*n* または A*n*V を使用します。A*n*V タイプのフィールドの場合、長さはフィールドのバイト長から取得されます。 sub_limit で指定した長さが実際の長さよりも小さい場合、サブ文字列はこの長さ制限値で切り捨てられます。

`sub_limit`
整数

サブ文字列の長さ制限値です。

output
整数

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。

例 可変長パターンの開始位置を検索

POSITV 関数は、映画の題名末尾の定冠詞や不定冠詞 (例、「SMURFS, THE」の「, THE」) の開始位置を検索します。まず、TRIMV は題名から末尾のブランクを削除します。これにより、冠詞が末尾パターンとなります。

```
DEFINE FILE MOVIES
  TITLEV/A39V = TRIMV('T',TITLE, 39,' ', 1, TITLEV);
  PSTART/I4 = POSITV(TITLEV,LENV(TITLEV,'I4'), ',', 1,'I4');
  PLEN/I4 = IF PSTART NE 0 THEN LENV(TITLEV,'I4') - PSTART +1
                    ELSE 0;
END
TABLE FILE MOVIES
  PRINT TITLE
   PSTART AS 'Pattern,Start' IN 25
   PLEN AS 'Pattern,Length'
BY CATEGORY  NOPRINT
WHERE PLEN NE 0
END
```

出力結果は次のとおりです。

```
                          Pattern  Pattern
TITLE                       Start   Length
-----                     -------  -------
SMURFS, THE                     7        5
SHAGGY DOG, THE                11        5
MALTESE FALCON, THE            15        5
PHILADELPHIA STORY, THE        19        5
TIN DRUM, THE                   9        5
FAMILY, THE                     7        5
CHORUS LINE, A                 12        3
MORNING AFTER, THE             14        5
BIRDS, THE                      6        5
BOY AND HIS DOG, A             16        3
```

SUBSTV - 可変長サブ文字列を抽出

SUBSTV 関数は、SUBSTR 関数と同様に、文字列からサブ文字列を抽出します。ただし、この関数では、サブ文字列の終了位置は、サブ文字列の開始位置と文字列の長さから計算されます。このため、パラメータ数は SUBSTR よりも少なくなります。また、出力フィールドが A*n*V フォーマットの場合、フィールドの実際の長さは、サブ文字列の長さに基づいて決定されます。

構文　可変長サブ文字列を抽出

```
SUBSTV(upper_limit, source_string, start, sub_limit, output)
```

説明

`upper_limit`
整数

ソース文字列の長さ制限値です。

`source_string`
文字 (A*n* または A*n*V)

抽出するサブ文字列を含む文字列です。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。フィールドを指定した場合、フォーマットには A*n* または A*n*V を使用します。A*n*V タイプのフィールドの場合、長さはフィールドのバイト長から取得されます。 upper_limit で指定した長さが実際の長さよりも小さい場合、ソース文字列はこの長さ制限値で切り捨てられます。この比較により決定される長さの最終値は *p_length* です。詳細は、*outfield* パラメータの説明を参照してください。

`start`
整数

ソース文字列内のサブ文字列の開始位置です。開始位置がソース文字列の長さを超えることは可能ですが、その場合はブランクが返されます。

`sub_limit`
整数

サブ文字列の長さをバイト数で指定します。 start および sublength で指定した値によっては、終了位置がソース文字列の長さを超えることがあります。

`output`
文字 (A*n* または A*n*V)

結果が返されるフィールド、または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。 このフィールドのフォーマットには、An または AnV を使用できます。

output のフォーマットが AnV で、end がサブ文字列の終了位置とした場合、実際の長さ outlen は、end、start、p_length の値から、次のように計算されます。詳細は、source_string パラメータの説明を参照してください。

end > p_length または end < start ならば outlen = 0。それ以外の場合、outlen = end - start + 1。

例　　可変長サブ文字列を抽出

次のリクエストは、映画の題名末尾の定冠詞または不定冠詞 (例、「SMURFS, THE」の「, THE」) を抽出します。まず、末尾のブランクが削除されます。これにより、冠詞が末尾パターンとなります。次に、パターンの開始位置と長さが検索されます。その後、SUBSTV 関数はパターンを抽出し、TRIMV 関数は題名からパターンを削除します。

```
DEFINE FILE MOVIES
  TITLEV/A39V = TRIMV('T',TITLE, 39,' ', 1, TITLEV);
  PSTART/I4 = POSITV(TITLEV,LENV(TITLEV,'I4'), ',', 1,'I4');
  PLEN/I4 = IF PSTART NE 0 THEN LENV(TITLEV,'I4') - PSTART +1
                    ELSE 0;
  PATTERN/A20V= SUBSTV(39, TITLEV, PSTART, PLEN, PATTERN);
  NEWTIT/A39V = TRIMV('T',TITLEV,39,PATTERN,LENV(PATTERN,'I4'), NEWTIT);
END
TABLE FILE MOVIES
  PRINT TITLE
   PSTART AS 'Pattern,Start' IN 25
   PLEN AS 'Pattern,Length'
  NEWTIT AS 'Trimmed,Title' IN 55
BY CATEGORY  NOPRINT
WHERE PLEN NE 0
END
```

出力結果は次のとおりです。

```
                         Pattern Pattern  Trimmed
TITLE                      Start  Length  Title
-----                     ------ -------  -------
SMURFS, THE                    7       5  SMURFS
SHAGGY DOG, THE               11       5  SHAGGY DOG
MALTESE FALCON, THE           15       5  MALTESE FALCON
PHILADELPHIA STORY, THE       19       5  PHILADELPHIA STORY
TIN DRUM, THE                  9       5  TIN DRUM
FAMILY, THE                    7       5  FAMILY
CHORUS LINE, A                12       3  CHORUS LINE
MORNING AFTER, THE            14       5  MORNING AFTER
BIRDS, THE                     6       5  BIRDS
BOY AND HIS DOG, A            16       3  BOY AND HIS DOG
```

TRIMV - 文字列から文字を削除

TRIMV 関数は、文字列内のあるパターンの先頭と末尾の文字を削除します。TRIMV は、TRIM に類似しています。ただし、TRIMV 関数では、ソース文字列と削除するパターンに AnV フォーマットを使用することができます。

TRIMV 関数は A*n* フィールドを A*n*V フィールド (ブランク以外の末尾の文字までの、データの実際のバイト長) に変換するときに役立ちます。

構文　文字列から文字を削除

```
TRIMV(trim_where, source_string, upper_limit, pattern, pattern_limit,
output)
```

説明

`trim_where`
文字

次のオプションのいずれかを選択して、削除するパターンの位置を指定します。

L - 先頭の文字を削除します。

T - 末尾の文字を削除します。

B - 先頭と末尾の両方の文字を削除します。

`source_string`
文字 (A*n* または A*n*V)

先頭または末尾の文字を削除するソース文字列です。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。 フィールドを指定した場合、フォーマットには A*n* または A*n*V を使用します。A*n*V タイプのフィールドの場合、長さはフィールドのバイト長から取得されます。upper_limit で指定した長さが実際の長さよりも小さい場合、ソース文字列はこの長さ制限値で切り捨てられます。

`upper_limit`
整数

ソース文字列の長さ上限値です。

`pattern`
文字 (A*n* または A*n*V)

削除するパターンです。パターン文字列は一重引用符 (') で囲みます。フィールドを指定した場合、フォーマットには A*n* または A*n*V を使用します。A*n*V タイプのフィールドの場合、長さはフィールドのバイト長から取得されます。 pattern_limit で指定した長さが実際の長さよりも小さい場合、パターンはこの長さ制限値で切り捨てられます。

`pattern_limit`
整数

パターンの長さ制限値です。

output
文字 (A*n* または A*n*V)

結果が返されるフィールド、または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。 フィールドのフォーマットには、AnV または An を使用することができます。

出力フォーマットが A*n*V の場合、実際の長さは、削除後のバイト数になります。

例 末尾のブランクを削除し A*n*V フィールドを作成

TRIMV 関数は、TITLE 値から末尾のブランクを削除することにより、TITLEV という名前の A*n*V フィールドを作成します。

```
TABLE FILE MOVIES
PRINT DIRECTOR
COMPUTE TITLEV/A39V = TRIMV('T', TITLE, 39, ' ', 1, TITLEV);
BY CATEGORY
END
```

出力結果の最初の 10 行は次のとおりです。

```
CATEGORY  DIRECTOR         TITLEV
--------  --------         ------
ACTION    SPIELBERG S.     JAWS
          VERHOVEN P.      ROBOCOP
          VERHOVEN P.      TOTAL RECALL
          SCOTT T.         TOP GUN
          MCDONALD P.      RAMBO III
CHILDREN                   SMURFS, THE
          BARTON C.        SHAGGY DOG, THE
                           SCOOBY-DOO-A DOG IN THE RUFF
          GEROMINI         ALICE IN WONDERLAND
                           SESAME STREET-BEDTIME STORIES AND SONGS
```

UPCASV - 可変長大文字列を作成

UPCASV 関数は、UPCASE に類似した関数で、アルファベットを大文字に変換します。ただし、UPCASV 関数は、A*n*V ソース文字列の実際の長さ、または制限値を指定する入力パラメータのいずれか小さい方を実際の長さとする A*n*V 出力を返すことができます。

構文　可変長大文字列を作成

```
UPCASV(upper_limit, source_string, output)
```

説明

`upper_limit`
整数

ソース文字列の長さ制限値です。 正の定数またはフィールドです。この値の整数部分が長さ制限値を表します。

`source_string`
文字 (A*n* または A*n*V)

大文字に変換する文字列です。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。フィールドを指定した場合、フォーマットには A*n* または A*n*V を使用します。A*n*V タイプのフィールドの場合、長さはフィールドのバイト長から取得されます。 upper_limit で指定した長さが実際の長さよりも小さい場合、ソース文字列はこの長さ制限値で切り捨てられます。

`output`
文字 (A*n* または A*n*V)

結果が返されるフィールド、または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。 AnV または An フォーマットのフィールドを指定することもできます。

出力フォーマットが A*n*V の場合、返される長さは、ソース文字列長または制限値のいずれか小さい方になります。

例　可変長大文字列を作成

大文字のみの文字列と大文字と小文字が混在する文字列を含むフィールドをソートすることを想定します。次のリクエストは、大文字のみの値と大文字と小文字が混在する値を含む「LAST_NAME_MIXED」という名前のフィールドを定義します。

```
DEFINE FILE EMPLOYEE
LAST_NAME_MIXED/A15=IF DEPARTMENT EQ 'MIS' THEN LAST_NAME ELSE
LCWORD(15, LAST_NAME, 'A15');
LAST_NAME_UPCASV/A15V=UPCASV(5, LAST_NAME_MIXED, 'A15') ;
END
```

このフィールドを基準にソートするリクエストを実行したことを想定します。

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME_MIXED AND FIRST_NAME BY LAST_NAME_UPCASV
WHERE CURR_JOBCODE EQ 'B02' OR 'A17' OR 'B04';
END
```

出力結果は次のとおりです。

```
LAST_NAME_UPCASV  LAST_NAME_MIXED  FIRST_NAME
----------------  ---------------  ----------
BANNI             Banning          JOHN
BLACK             BLACKWOOD        ROSEMARIE
CROSS             CROSS            BARBARA
MCCOY             MCCOY            JOHN
MCKNI             Mcknight         ROGER
ROMAN             Romans           ANTHONY
```

8

DBCS コードページの文字列関数

ここでは、DBCS コードページを使用するよう構成されている場合に、DBCS および SBCS 文字の文字列を操作する関数について説明します。

トピックス

DCTRAN - 1 バイトまたは 2 バイト文字を他の文字に変換

DCTRAN 関数は、文字列内の 1 バイトまたは 2 バイトの文字を 10 進数に基づいて他の文字に変換します。

DCTRAN 関数は、1 バイト文字を 2 バイト文字に、2 バイト文字を 1 バイト文字に変換する以外に、1 バイト文字を 1 バイト文字に、2 バイト文字を 2 バイト文字に変換することもできます。

構文　1 バイトまたは 2 バイト文字を他の文字に変換

```
DCTRAN(length, source_string, indecimal, outdecimal, output)
```

説明

`length`
整数

source_string の長さをバイト数で指定します。

`source_string`
文字

変換前の文字列です。

`indecimal`
整数

変換する文字の ASCII の 10 進コードです。

`outdecimal`
整数

indecimal の代わりに使用する文字の ASCII 10 進コードです。

`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。

例　DCTRAN による 2 バイト文字の変換

次のように使用します。

```
DCTRAN(8,'AｱA本B語',177,70,A8)

For AｱA本B語, the result is AFA本B語.
```

DEDIT - 文字を抽出または追加

ユーザの構成で DBCS コードページを使用する場合に、DEDIT 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ができます。

DEDIT 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、DEDIT 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、DEDIT 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、DEDIT 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。

構文　DBCS または SBCS 文字を抽出または追加

```
DEDIT(inlength, source_string, mask_length, mask, output)
```

説明

`inlength`

整数

source_string で指定した文字列のバイト数です。文字列には DBCS 文字と SBCS 文字が混在する場合があるため、このバイト数には、ソース文字列として格納可能な最大バイト数を指定します。

`source_string`

文字

編集する文字列です。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。

`mask_length`

整数

mask 内の文字数です。

`mask`

文字

mask 文字列です。

mask 内の「9」は、新しいフィールドにコピーされる、ソースフィールド内の対応する文字を表します。

mask 内のドル記号 ($) は、無視されるソースフィールド内の対応する文字を表します。

mask 内のその他すべての文字が新しいフィールドにコピーされます。

`output`

文字

結果が返されるフィールド、または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　　DBCS 文字の追加と抽出

次の例では、ソース文字列の文字を新しいフィールドへ 1 つおきにコピーします。コピーはソース文字列の 1 文字目から開始し、抽出された文字の後に複数の新しい文字を追加します。

```
DEDIT( 15, 'あaいiうuえeおo', 16, '9$9$9$9$9$-かきくけこ', 'A30')
The result is あいうえお-かきくけこ.
```

次の例では、ソース文字列の文字を新しいフィールドへ 1 つおきにコピーします。コピーはソース文字列の 2 文字目から開始し、抽出された文字の後に複数の新しい文字を追加します。

```
DEDIT( 15, 'あaいiうuえeおo', 16, '$9$9$9$9$9-ABCDE', 'A20')
The result is aiueo-ABCDE.
```

DSTRIP - 文字列から 1 バイトまたは 2 バイト文字を削除

DSTRIP 関数は、文字列から特定の 1 バイトまたは 2 バイトの文字をすべて削除します。結果の文字列の長さは元の文字列と同一になりますが、右側にブランクが追加されます。

構文　　文字列から 1 バイトまたは 2 バイト文字を削除

```
DSTRIP(length, source_string, char, output)
```

説明

`length`

整数

source_string および outfield の入力バイト数です。

`source_string`

文字

文字を削除する文字列です。

`char`

文字

文字列から削除する文字です。2 文字以上を指定した場合、最も左側の文字が削除文字として使用されます。

注意：一重引用符 (') を削除するには、2 つの一重引用符 ('') を使用します。 さらに、この文字の組み合わせを一重引用符 (') で囲む必要があります。

`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。

例　文字列から 2 バイト文字を削除

次のように使用します。

```
DSTRIP(9, 'A日A本B語', '日', A9)
```

For A日A本B語, the result is AA本B語.

DSUBSTR - サブ文字列を抽出

DBCS コードページを使用するよう構成されている場合、DSUBSTR 関数を使用して、サブ文字列の長さと位置に基づいてソース文字列からサブ文字列を抽出することができます。

構文　サブ文字列を抽出

```
DSUBSTR(inlength, source_string, start, end, sublength, output)
```

説明

`inlength`

整数

ソース文字列の長さをバイト数で指定します。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。文字列には DBCS 文字と SBCS 文字が混在する場合があるため、このバイト数には、ソース文字列として格納可能な最大バイト数を指定します。

`source_string`

文字

サブ文字列の抽出元となる文字列です。文字列は一重引用符 (') で囲みます。ソース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

ソース文字列から抽出するサブ文字列の開始位置を文字数で指定します。この引数が 1 より小さい、または *end* より大きい場合、この関数はブランクを返します。

`end`

整数

サブ文字列の終了位置を文字数で指定します。この引数が *start* より小さい、または *sublength* より大きい場合、この関数はブランクを返します。

`sublength`

整数

サブ文字列の長さを文字数で指定します。通常、この長さは end - start + 1 (end と start の差に 1 を加算した値) です。*sublength* が *end* - *start* +1 よりも長い場合、サブ文字列の末尾にブランクが追加されます。短い場合、サブ文字列の末尾が切り捨てられます。この値は、output で宣言した長さに一致させる必要があります。 sublength の長さ分の文字のみが処理されます。

`output`

文字

結果が返されるフィールド、または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　サブ文字列を抽出

次の例では、15 バイト長の文字列から、4 文字目から 6 文字目までの 3 文字のサブ文字列を抽出します。

```
DSUBSTR( 15, 'あaいiうuえeおo', 4, 6, 3, 'A10')
The result is iうu.
```

JPTRANS - 日本語の文字を変換

JPTRANS 関数は、日本語の文字を変換します。

構文　日本語の文字を変換

```
JPTRANS ('type_of_conversion', length, source_string, 'output_format')
```

説明

`type_of_conversion`
次のオプションのいずれかで、日本語の文字に適用する変換タイプを指定します。下表の入力コンポーネントは 1 つです。

変換タイプ	説明
`'UPCASE'`	全角 (2 バイト) アルファベットを全角大文字に変換します。
`'LOCASE'`	全角アルファベットを全角小文字に変換します。
`'HNZNALPHA'`	半角英数 (1 バイト) 文字を全角に変換します。
`'HNZNSIGN'`	半角 ASCII 記号を全角に変換します。
`'HNZNKANA'`	半角カタカナを全角に変換します。
`'HNZNSPACE'`	半角ブランク (スペース) を全角に変換します。
`'ZNHNALPHA'`	全角英数文字を半角に変換します。
`'ZNHNSIGN'`	全角 ASCII 記号を半角に変換します。
`'ZNHNKANA'`	全角カタカナを半角に変換します。
`'ZNHNSPACE'`	全角ブランク (スペース) を半角に変換します。
`'HIRAKATA'`	ひらがなを全角カタカナに変換します。
`'KATAHIRA'`	全角カタカナをひらがなに変換します。

`length`
整数

source_string のバイト数です。

`source_string`
文字

変換する文字列です。

`output_format`
文字

出力を格納するフィールド名、またはフォーマットです。フォーマットは一重引用符 (') で囲みます。

例　JPTRANS 関数の使用

```
JPTRANS('UPCASE', 20, Alpha_DBCS_Field, 'A20')
```

For ａｂｃ, the result is ＡＢＣ.

```
JPTRANS('LOCASE', 20, Alpha_DBCS_Field, 'A20')
```

For ＡＢＣ, the result is ａｂｃ.

```
JPTRANS('HNZNALPHA', 20, Alpha_SBCS_Field, 'A20')
```

For AaBbCc123, the result is Ａａ Ｂｂ Ｃｃ １２３.

```
JPTRANS('HNZNSIGN', 20, Symbol_SBCS_Field, 'A20')
```

For !@$%,.?, the result is ！＠＄％、。？

```
JPTRANS('HNZNKANA', 20, Hankaku_Katakana_Field, 'A20')
```

For「ﾍﾞｰｽﾎﾞｰﾙ。」, the result is 「ベースボール。」

```
JPTRANS('HNZNSPACE', 20, Hankaku_Katakana_Field, 'A20')
```

For ｱｲｳ, the result is ア　イ　ウ

```
JPTRANS('ZNHNALPHA', 20, Alpha_DBCS_Field, 'A20')
```

For Ａａ Ｂｂ Ｃｃ １２３, the result is AaBbCc123.

```
JPTRANS('ZNHNSIGN', 20, Symbol_DBCS_Field, 'A20')
```

For ！＠＄％、。？, the result is !@$%,.?

```
JPTRANS('ZNHNKANA', 20, Zenkaku_Katakana_Field, 'A20')
```

For 「ベースボール。」, the result is｢ﾍﾞｰｽﾎﾞｰﾙ。｣

```
JPTRANS('ZNHNSPACE', 20, Zenkaku_Katakana_Field, 'A20')
```

For ア　イ　ウ, the result is ア イ ウ

```
JPTRANS('HIRAKATA', 20, Hiragana_Field, 'A20')
```

For あいう, the result is アイウ

```
JPTRANS('KATAHIRA', 20, Zenkaku_Katakana_Field, 'A20')
```

For アイウ, the result is あいう

参照　JPTRANS 関数使用上の注意

- HNZNSIGN および ZNHNSIGN は、記号の変換に使用します。

  多くの記号において、日本語全角文字と ASCII 文字との間には 1 対 1 の関係が成立します。ただし、1 対 n の関係を持つ記号がいくつかあります。たとえば、日本語の読点 (U+3001) および全角カンマ (U+FF0C) は、同一の半角カンマ (U+002C) に変換されます。このような特殊な場合については、次の規則が追加されています。

  HNZNSIGN

  - 半角二重引用符「"」(U+0022) は、全角右二重引用符「”」(U+201D) に変換される。
  - 半角一重引用符「'」(U+0027) は、全角右一重引用符「’」(U+2019) に変換される。
  - 半角カンマ「,」(U+002C) は、全角読点「、」(U+3001) に変換される。
  - 半角ピリオド「.」(U+002E) は、全角句点「。」(U+3002) に変換される。
  - 半角円記号「¥」(U+005C) は、全角バックスラッシュ「＼」(U+FF3C) に変換される。
  - 半角左かぎ括弧 (「) (U+FF62) は、全角左かぎ括弧 (「) (U+300C) に変換される。

- 半角右かぎ括弧 (」) (U+FF63) は、全角右かぎ括弧 (」) (U+300D) に変換される。
- 半角中点「･」(U+FF65) は、全角中点「・」(U+30FB) に変換される。

ZNHNSIGN

- 全角右二重引用符「”」(U+201D) は、半角二重引用符「"」(U+0022) に変換される。
- 全角左二重引用符「“」(U+201C) は、半角二重引用符「"」(U+0022) に変換される。
- 全角引用符「 " 」(U+FF02) は、半角二重引用符「"」(U+0022) に変換される。
- 全角右一重引用符「’」(U+2019) は、半角一重引用符「'」(U+0027) に変換される。
- 全角左一重引用符「‘」(U+2018) は、半角一重引用符「'」(U+0027) に変換される。
- 全角一重引用符「 ' 」(U+FF07) は、半角一重引用符「'」(U+0027) に変換される。
- 全角読点 「、」(U+3001) は、半角カンマ「,」(U+002C) に変換される。
- 全角カンマ「, 」(U+FF0C) は、半角カンマ「,」(U+002C) に変換される。
- 全角句点「。」(U+3002) は、半角ピリオド「.」(U+002E) に変換される。
- 全角ピリオド「. 」(U+FF0E) は、半角ピリオド「.」(U+002E) に変換される。
- 全角円記号「￥」(U+FFE5)は、半角円記号「¥」(U+00A5) に変換される。
- 全角バックスラッシュ「＼」(U+FF3C) は、半角円記号「¥」(U+005C) に変換される。
- 全角左かぎ括弧 (「) (U+300C) は、半角左かぎ括弧 (「) (U+FF62) に変換される。
- 全角右かぎ括弧 (」) (U+300D) は、半角右かぎ括弧 (」) (U+FF63) に変換される。
- 全角中点「・」(U+30FB) は、半角中点「･」(U+FF65) に変換される。

- HNZNKANA および ZNHNKANA は、カタカナの変換に使用します。

  これらは文字だけでなく、次の句読点記号変換も行います。

  - 全角読点「、」(U+3001) と半角読点「､」(U+FF64) （双方向変換）

- 全角句点「。」(U+3002) と半角句点「。」(U+FF61) （双方向変換）
- 全角左かぎ括弧 (「) (U+300C) と半角左かぎ括弧 (「) (U+FF62) (双方向変換)。
- 全角右かぎ括弧 (」) (U+300D) と半角右かぎ括弧 (」) (U+FF63)（双方向変換）
- 全角中点「・」(U+30FB) と半角中点「･」(U+FF65) （双方向変換）
- 全角長音記号「ー」(U+30FC) と半角長音記号「ｰ」(U+FF70) （双方向変換）

- JPTRANS をネスト化し、複数の変換を実行することができます。

  たとえば、テキストデータには全角の数字と全角の記号が含まれている場合があります。これらの数字と記号を ASCII に変換したい状況が考えられます。

  For バンゴウ＃ １ ２ ３ , the result is バンゴウ#123

```
JPTRANS('ZNHNALPHA', 20, JPTRANS('ZNHNSIGN', 20, Symbol_DBCS_Field,
'A20'), 'A20')
```

- HNZNSPACE および ZNHNSPACE は、ブランク (スペース文字) の変換に使用します。

  現在、U+0020 と U+3000 の間の変換のみがサポートされています。

KKFCUT - 文字列の末尾の切り捨て

DBCS コードページを使用するよう構成されている場合、KKFCUT 関数を使用して文字列の末尾を切り捨てることができます。

構文　文字列の末尾の切り捨て

```
KKFCUT(length, source_string, output)
```

説明

`length`
整数

ソース文字列の長さをバイト数で指定します。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。文字列には DBCS 文字と SBCS 文字が混在する場合があるため、このバイト数には、ソース文字列として格納可能な最大バイト数を指定します。

`source_string`
文字

末尾を切り捨てる文字列です。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。

`output`
文字

結果が返されるフィールド、または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。

文字列は、出力フィールドのバイト数にまで末尾が切り捨てられます。

例　文字列の末尾の切り捨て

次のリクエストでは、KKFCUT 関数を使用して、COUNTRY フィールド (最大 10 バイト長) を A4 フォーマットの長さにまで末尾を切り捨てます。

```
COUNTRY_CUT/A4 = KKFCUT(10, COUNTRY, 'A4');
```

下図は、ASCII 環境の出力結果を示しています。

国名	COUNTRY_CUT
イギリス	イギ
日本	日本
イタリア	イタ
ドイツ	ドイ
フランス	フラ

9

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、データソースレコードの検索、レコードまたは値の抽出、入力フィールドの値に基づく値の割り当てを実行します。

データソース関数の結果は、フィールドに格納する必要があります。結果はダイアログマネージャ変数には格納できません。

多くの関数では、*output* 引数にフィールド名またはフォーマットを指定することができます。フォーマットを指定する場合、一重引用符 (') で囲みます。 ただし、関数がダイアログマネージャコマンドから呼び出される場合、常にフォーマットとして指定する必要があります。関数の呼び出しおよび引数の指定についての詳細は、45 ページの 「 関数へのアクセスと呼び出し 」 を参照してください。

トピックス

CHECKMD5 - MD5 ハッシュチェック値の計算

CHECKMD5 関数は、MD5 ハッシュ関数を使用して、数値入力値を取得し、128 ビット値を固定長文字で返します。ハッシュ関数は、任意サイズのデータを固定サイズのデータにマッピング可能な任意の関数です。ハッシュ関数から返される値は、ハッシュ値と呼ばれます。これらの関数は、送信されたデータの整合性を検証するために使用することができます。

構文　MD5 ハッシュチェック値の計算

```
CHECKMD5(buffer)
```

説明

`buffer`

ハッシュ値を計算するデータバッファです。An、AnV、TXn データタイプフォーマットのいずれかで単一フィールドまたはグループフィールドとして表される、さまざまなタイプのデータセットにすることができます。

例　MD5 ハッシュチェック値の計算

次のリクエストは、MD5 ハッシュチェック値を計算し、その値を文字 16 進数値に変換して表示します。

```
DEFINE FILE WF_RETAIL_LITE
MD5/A32 = HEXTYPE(CHECKMD5(PRODUCT_CATEGORY));
END
TABLE FILE WF_RETAIL_LITE
SUM MD5
BY PRODUCT_CATEGORY
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
TYPE=REPORT,FONT=COURIER,$
 ENDSTYLE
END
```

下図は、出力結果を示しています。これらの入力値の長さは異なりますが、等幅フォントで出力されているため、すべての出力値が固定長であることが分かります。

Product Category	MD5
Accessories	98EDB85B00D9527AD5ACEBE451B3FAE6
Camcorder	612A923BDD05C2231F81991B8D12A3A1
Computers	45888A4DA062F16A099A7F7C6CC15EE0
Media Player	D34BEA29F24AF9FDE2E10B3E1D857CF9
Stereo Systems	3AA9FFE9806E269A7EB066A84092F0A3
Televisions	A3B5BC99DD2B42627EF64A4FCAAAB0B2
Video Production	60913E95848330A2C4A5D921E7C8BB14

CHECKSUM - ハッシュサムの計算

CHECKSUM 関数は、入力パラメータのハッシュサム (チェックサムと呼ばれる) を I11 フォーマットの整数として計算します。この計算は、フィールドの等価検索に使用することができます。チェックサムは、ファイルが特定のストレージデバイスから別のストレージデバイスに移動された後にファイルの完全性を確保するために使用するハッシュサムです。

構文　CHECKSUM ハッシュ値の計算

```
CHECKSUM(buffer)
```

説明

`buffer`

ハッシュインデックスを計算するデータバッファです。An、AnV、TXn データタイプフォーマットのいずれかで単一フィールドとして表される、さまざまなタイプのデータセットにすることができます。

例　CHECKSUM ハッシュ値の計算

次のリクエストは、チェックサムハッシュ値を計算します。

```
DEFINE FILE WF_RETAIL_LITE
CHKSUM/I11 = (CHECKSUM(PRODUCT_CATEGORY));
END
TABLE FILE WF_RETAIL_LITE
PRINT CHKSUM
BY PRODUCT_CATEGORY
WHERE PRODUCT_CATEGORY NE LAST PRODUCT_CATEGORY
ON TABLE SET PAGE NOLEAD
END
```

下図は、出力結果を示しています。

Product Category	CHKSUM
Accessories	-830549649
Camcorder	-912058982
Computers	-469201037
Media Player	-1760917009
Stereo Systems	-1853215244
Televisions	810407163
Video Production	275494446

COALESCE - ミッシング値以外の先頭値の取得

COALESCE 関数は、入力された引数リストから、ミッシング値以外の最初の引数値を返します。すべての引数値がミッシングの場合、ミッシング値が返されます (MISSING が ON に設定されている場合)。それ以外の場合、デフォルト値 (0 (ゼロ) またはブランク) が返されます。

構文　ミッシング値以外の先頭値の取得

```
COALESCE(arg1, arg2, ...)
```

説明

`arg1, arg2, ...`

任意のフィールド、式、定数のいずれかです。これらの引数は、すべて数値であるか、すべて文字である必要があります。

これらの入力パラメータのミッシング値がテストされます。

出力データタイプは入力データタイプと同一です。

例　ミッシング値以外の先頭値の取得

この例では、ミッシング値が追加された SALE データソースを使用します。これらのミッシング値は、次の SALEMISS というプロシジャによって追加されます。

```
MODIFY FILE SALES
 FIXFORM STORE/4 DATE/5 PROD/4
 FIXFORM UNIT/3 RETAIL/5 DELIVER/3
 FIXFORM OPEN/3 RETURNS/C2 DAMAGED/C2
 MATCH STORE
    ON NOMATCH REJECT
    ON MATCH CONTINUE
 MATCH DATE
    ON NOMATCH REJECT
    ON MATCH CONTINUE
 MATCH PROD_CODE
    ON NOMATCH INCLUDE
    ON MATCH REJECT
DATA
14Z 1017 C13 15 1.99 35 30   6
14Z 1017 C14 18 2.05 30 25 4
14Z 1017 E2  33 0.99 45 40
END
```

次のリクエストは、COALESCE 関数を使用して、ミッシング値以外の先頭値を返します。

```
TABLE FILE SALES
PRINT DAMAGED RETURNS RETAIL_PRICE
COMPUTE
COAL1/D12.2 MISSING ON = COALESCE(DAMAGED, RETURNS, RETAIL_PRICE);
BY STORE_CODE
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果を示しています。DAMAGED の値がミッシングでない場合、その DAMAGED 値が返されます。DAMAGED の値がミッシングで、RETURNS の値がミッシングでない場合、その RETURNS 値が返されます。両方の値がミッシングの場合、RETAIL_PRICE の値が返されます。

STORE_CODE	DAMAGED	RETURNS	RETAIL_PRICE	COAL1
14B	6	10	$.95	6.00
	3	3	$1.29	3.00
	1	2	$1.89	1.00
	0	3	$1.99	.00
	4	5	$2.39	4.00
	0	0	$2.19	.00
	4	9	$.99	4.00
	9	8	$1.09	9.00
14Z	3	2	$.85	3.00
	1	2	$1.89	1.00
	1	0	$1.99	1.00
	6	.	$1.99	6.00
	.	4	$2.05	4.00
	0	0	$2.09	.00
	2	3	$2.09	2.00
	7	4	$.89	7.00
	.	.	$.99	.99
	2	4	$1.09	2.00
77F	1	1	$2.09	1.00
	0	0	$2.49	.00
K1	0	1	$1.49	.00
	1	1	$.99	1.00

DB_EXPR - リクエストへの SQL 式の挿入

DB_EXPR 関数は、FOCUS 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、マスターファイル内の DEFINE、WHERE 句、FILTER FILE コマンド、マスターファイル内のフィルタ、SQL ステートメント内で使用できます。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。WITH 句が指定された DEFINE でこの関数を使用する場合を除いて、SQL 文字列では、各フィールド参照を二重引用符 (") で囲む必要があります。

参照　DB_EXPR 関数使用上の注意

- この式は、単一値を返す必要があります。
- 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。

例　TABLE リクエストへの DB2 BIGINT および CHAR 関数の挿入

次の TABLE リクエストは、WF_RETAIL データソースに対して実行され、COMPUTE コマンドに 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。

```
TABLE FILE WF_RETAIL
SUM REVENUE NOPRINT
AND COMPUTE BIGREV/A31 = DB_EXPR(CHAR(BIGINT("REVENUE" * "REVENUE") ) ) ;
AS 'Alpha Square Revenue'
BY REGION
ON TABLE SET PAGE NOPAGE
END
```

WF_RETAIL は、ユーザが作成可能なサンプルデータソースです。このデータソースを作成するには、Reporting Server ブラウザインターフェースでアプリケーションを右クリックし、コンテキストメニューから [新規]、[チュートリアル] を順に選択します。

トレースには、DB_EXPR 関数で指定された式が、DB2 SELECT ステートメントに挿入されたことが示されています。

```
  SELECT
  T11."REGION",
   SUM(T1."Revenue"),
   ((CHAR(BIGINT( SUM(T1."Revenue") *  SUM(T1."Revenue")) ) ))
   FROM
  wrd_fact_sales T1,
  wrd_dim_customer T5,
  wrd_dim_geography T11
   WHERE
  (T5."ID_CUSTOMER" = T1."ID_CUSTOMER") AND
  (T11."ID_GEOGRAPHY" = T5."ID_GEOGRAPHY")
   GROUP BY
  T11."REGION  "
   ORDER BY
  T11."REGION  "
   FOR FETCH ONLY;
END
```

出力結果は次のとおりです。

Region	Alpha Square Revenue
Central	459024717717929
MidEast	61720506151994
NorthEast	247772056471221
NorthWest	42335175855351
SouthEast	205820846242532
SouthWest	9449541537794
West	164356565757257

DB_INFILE - ファイルまたは SQL サブクエリに対する値のテスト

DB_INFILE 関数は、ソースファイル内の 1 つまたは複数のフィールド値をターゲットファイル内の値と比較します。この比較は、1 つまたは複数のフィールド値に基づいて行えます。DB_INFILE 関数は、ソースフィールド値のセットがターゲットファイル内の値のセットと一致する場合、1 (TRUE) を返します。一致しない場合、0 (FALSE) を返します。DB_INFILE は、WebFOCUS リクエスト内で関数を使用可能な任意の位置 (例、DEFINE、WHERE 句) で使用することができます。

ターゲットファイルは、WebFOCUS で読み取り可能な任意のデータソースにすることができます。アクセスするデータソースおよびリクエスト内のコンポーネントに応じて、WebFOCUS または RDBMS のいずれかが値の比較を処理します。

WebFOCUS が比較を処理する場合、ターゲットデータソースが動的に読み取られ、ターゲットデータ値が格納されたシーケンシャルファイルと、データファイルを記述したシノニムが作成されます。次に、ソース値とターゲット値の組み合わせがすべて含まれた IF または WHERE 構造がメモリ内に作成されます。WebFOCUS でワイルドカードと見なされる文字がターゲットデータに含まれている場合、SET EQTEST = EXACT コマンドが有効になってい場合を除いて、ワイルドカードとして処理されます。

ソースファイルがリレーショナルデータソースの場合、次のような状況があります。

- **ターゲット値が同一の RDBMS および接続のリレーショナルデータソース内に存在する。**

  この場合、DB_INFILE で参照するターゲットファイルには、次のファイルを指定することができます。

  - ターゲット値を取得するためのサブクエリが含まれた SQL ファイル。ターゲット SQL ファイルを記述したシノニムが存在する必要があります。アクセスファイルでターゲットファイルの CONNECTION および DATASET を指定する必要があります。

    サブクエリから生成された SELECT ステートメントが RDBMS でサポートされている場合、リレーショナルアダプタは、生成された SQL の WHERE 述語内にそのサブクエリを挿入します。

    サブクエリから生成された SELECT ステートメントが RDBMS で有効でない場合、リレーショナルアダプタは、ターゲット値を取得します。次に、ソースおよびターゲットのフィールド値のすべての組み合わせが含まれた WHERE 述語を生成します。

    HOLD FORMAT SQL_SCRIPT コマンドを使用して、SQL サブクエリが含まれた SQL ファイルおよび対応するシノニムを作成することができます。詳細は、『ibi™ WebFOCUS® Language リファレンス』を参照してください。

- リレーショナルデータソース。ターゲットデータソースを記述したシノニムが存在する必要があります。

  ターゲットフィールドとして DB_INFILE で参照されるフィールドのみがデータソースに格納されている場合、リレーショナルアダプタは、ターゲット値を取得するためのサブクエリを作成します。サブクエリから生成された SELECT ステートメントが RDBMS でサポートされている場合、リレーショナルアダプタは、生成された SQL の WHERE 述語内にそのサブクエリを挿入します。

  サブクエリから生成された SELECT ステートメントが RDBMS で有効でない場合、リレーショナルアダプタは、ターゲット値の一意のリストを取得します。次に、ソースおよびターゲットのフィールド値のすべての組み合わせが含まれた WHERE 述語を生成します。

- **ターゲット値が別の RDBMS または接続の非リレーショナルデータソースまたはリレーショナルデータソースに存在する。**

  この場合、ターゲット値が取得され、WebFOCUS に渡されて処理されます。

構文　DB_INFILE によるソースおよびターゲットのフィールド値の比較

```
DB_INFILE(target_file, s1, t1, ... sn, tn)
```

説明

`target_file`
　ターゲットファイルのシノニムです。

`s1, ..., sn`
　ソースファイルのフィールドです。

`t1, ..., tn`
　ターゲットファイルのフィールドです。

この関数は、ターゲット値のセットとソース値のセットが一致する場合、1 を返します。一致しない場合、0 (ゼロ) を返します。

参照　DB_INFILE 関数使用上の注意

- ソースとターゲット両方のデータソースで比較フィールドに MISSING=ON が指定されている場合、両方のファイルのミッシング値は「等しい」と見なされます。いずれかのファイルまたは両方のファイルで MISSING=OFF が指定されている場合、これらのファイルのミッシング値は「等しくない」と見なされます。

- 日付値および日付時間値を比較する場合を除いて、比較時にブランクが値に追加されたり、値の末尾が切り取られたりすることはありません。
  - ソースフィールドが日付フィールド、ターゲットフィールドが日付時間フィールドの場合、比較前に時間構成要素が除外されます。
  - ソースフィールドが日付時間フィールド、ターゲットフィールドが日付フィールドの場合、比較前に 0 (ゼロ) 時間構成要素がターゲット値に追加されます。
- 文字フィールドを数値フィールドと比較する場合、比較前に文字値から数値への変換が試みられます。
- WebFOCUS が比較を処理し、WebFOCUS でワイルドカードとして見なされる文字がターゲットデータに含まれている場合、SET EQTEST = EXACT コマンドが有効になっている場合を除いて、ワイルドカードとして処理されます。

例 SQL サブクエリファイルによるソース値とターゲット値の比較

この例では、WF_RETAIL DB2 データソースを使用します。

WF_RETAIL は、ユーザが作成可能なサンプルデータソースです。このデータソースを作成するには、Reporting Server ブラウザインターフェースでアプリケーションを右クリックし、コンテキストメニューから [チュートリアル] を選択します。

retail_subquery.sql という SQL ファイルには、Central および NorthEast 地域の特定の州コードを取得するサブクエリが含まれています。

```
SELECT  MAX(T11.REGION), MAX(T11.STATECODE)  FROM wrd_dim_geography T11
WHERE (T11.STATECODE IN('AR', 'IA', 'KS', 'KY', 'WY', 'CT', 'MA', 'NJ',
'NY', 'RI')) AND (T11.REGION IN('Central', 'NorthEast'))  GROUP BY
T11.REGION, T11.STATECODE
```

retail_subquery.mas マスターファイルは次のとおりです。

```
FILENAME=RETAIL_SUBQUERY, SUFFIX=DB2     , $
  SEGMENT=RETAIL_SUBQUERY, SEGTYPE=S0, $
    FIELDNAME=REGION, ALIAS=E01, USAGE=A15V, ACTUAL=A15V,
      MISSING=ON, $
    FIELDNAME=STATECODE, ALIAS=E02, USAGE=A2, ACTUAL=A2,
      MISSING=ON, $
```

retail_subquery.acx アクセスファイルは次のとおりです。

```
SEGNAME=RETAIL_SUBQUERY, CONNECTION=CON1, DATASET=RETAIL_SUBQUERY.SQL, $
```

注意：HOLD FORMAT SQL_SCRIPT コマンドを使用すると、SQL サブクエリファイルおよび対応するシノニムを作成することができます。詳細は、『ibi™ WebFOCUS® Language リファレンス』を参照してください。

次のリクエストは、DB_INFILE 関数を使用して、地域名および州コードを、サブクエリから取得された名前と比較します。

```
TABLE FILE WF_RETAIL
SUM REVENUE
BY REGION
BY STATECODE
WHERE DB_INFILE(RETAIL_SUBQUERY, REGION, REGION, STATECODE, STATECODE)
ON TABLE SET PAGE NOPAGE
END
```

トレースには、生成された SQL の WHERE 述語にサブクエリが挿入されたことが示されています。

```
 SELECT
  T11."REGION",
  T11."STATECODE",
   SUM(T1."Revenue")
   FROM
  wrd_fact_sales T1,
  wrd_dim_customer T5,
  wrd_dim_geography T11
   WHERE
  (T5."ID_CUSTOMER" = T1."ID_CUSTOMER") AND
  (T11."ID_GEOGRAPHY" = T5."ID_GEOGRAPHY") AND
  ((T11."REGION", T11."STATECODE") IN (SELECT  MAX(T11.REGION),
  MAX(T11.STATECODE)  FROM wrd_dim_geography T11 WHERE
  (T11.STATECODE IN('AR', 'IA', 'KS', 'KY', 'WY', 'CT', 'MA',
  'NJ', 'NY', 'RI')) AND (T11.REGION IN('Central', 'NorthEast'))
  GROUP BY T11.REGION, T11.STATECODE))
   GROUP BY
  T11."REGION",
  T11."STATECODE  "
   ORDER BY
  T11."REGION",
  T11."STATECODE  "
   FOR FETCH ONLY;
END
```

出力結果は次のとおりです。

Region	State Code	Revenue
Central	AR	839,075.22
	IA	1,197,171.09
	KS	1,014,388.99
	KY	1,014,825.22
	WY	182,808.08
NorthEast	CT	1,146,626.05
	MA	2,070,919.74
	NJ	2,148,955.56
	NY	6,360,267.52
	RI	342,972.30

例　シーケンシャルファイルによるソース値とターゲット値の比較

empvalues.ftm シーケンシャルファイルには、MIS 部門の従業員の姓および名が格納されています。

```
SMITH           MARY        JONES           DIANE       MCCOY
JOHN         BLACKWOOD      ROSEMARIE   GREENSPAN       MARY
CROSS           BARBARA
```

empvalues.mas マスターファイルには、empvalues.ftm ファイル内のデータが記述されています。

```
FILENAME=EMPVALUES, SUFFIX=FIX     , IOTYPE=BINARY, $
  SEGMENT=EMPVALUE, SEGTYPE=S0, $
    FIELDNAME=LN, ALIAS=E01, USAGE=A15, ACTUAL=A16, $
    FIELDNAME=FN, ALIAS=E02, USAGE=A10, ACTUAL=A12, $
```

注意：HOLD FORMAT SQL_SCRIPT コマンドを使用して、シーケンシャルファイルおよび対応するシノニムを作成することができます。詳細は、『ibi™ WebFOCUS® Language リファレンス』を参照してください。

次のリクエストは、FOCUS EMPLOYEE データソースに対して実行され、DB_INFILE 関数を使用して、従業員の名前を empvalues.ftm ファイルに格納されている名前と比較します。

```
FILEDEF EMPVALUES DISK baseapp/empvalues.ftm
TABLE FILE EMPLOYEE
SUM CURR_SAL
BY LAST_NAME BY FIRST_NAME
WHERE DB_INFILE(EMPVALUES, LAST_NAME, LN, FIRST_NAME, FN)
ON TABLE SET PAGE NOPAGE
END
```

出力結果は次のとおりです。

LAST_NAME	FIRST_NAME	CURR_SAL
BLACKWOOD	ROSEMARIE	$21,780.00
CROSS	BARBARA	$27,062.00
GREENSPAN	MARY	$9,000.00
JONES	DIANE	$18,480.00
MCCOY	JOHN	$18,480.00
SMITH	MARY	$13,200.00

構文　DB_INFILE 最適化の制御

DB_INFILE 式を最適化するかどうかを制御するには、次のコマンドを発行します。

```
SET DB_INFILE = {DEFAULT|EXPAND_ALWAYS|EXPAND_NEVER}
```

TABLE リクエストでは、次のコマンドを発行します。

```
ON TABLE SET DB_INFILE  {DEFAULT|EXPAND_ALWAYS|EXPAND_NEVER}
```

説明

DEFAULT
: サブクエリの作成が可能と分析された場合に、DB_INFILE によるサブクエリの作成を有効にします。これがデフォルト値です。

EXPAND_ALWAYS
: DB_INFILE によるサブクエリの作成を無効にします。代わりに、式がメモリ内の IF および WHERE 句に拡張されます。

EXPAND_NEVER
DB_INFILE により、式がメモリ内の IF および WHERE 句に拡張されないようにします。代わりに、サブクエリの作成を試みます。サブクエリを作成できない場合、FOC32585 メッセージが生成され、処理が停止します。

DB_LOOKUP - データソースの値を抽出

DB_LOOKUP 関数を使用すると、1 つのデータソースに対するリクエストの実行中に、JOIN や COMBINE により結合することなく、別のデータソースから値を抽出することができます。

DB_LOOKUP はデータソースと参照データのフィールドの組を比較し、一致するレコードを検索後、リクエストに返す値を抽出します。抽出する値を含むレコードを検索するために、任意の数の組を指定することができます。フィールドリストの組から検索される参照レコードが一意ではない場合、最初に一致する参照レコードが使用されます。

DB_LOOKUP は DEFINE コマンド、TABLE COMPUTE コマンド、または ibi™ Data Migrator フローから呼び出すことができます。

ソースファイルには制限はありません。参照ファイルには、クラスタ JOIN のクロスリファレンスファイルとしてサポートされる、FOCUS 以外のすべてのデータソースを指定することができます。一致レコードを検索するために使用する参照フィールドには、参照データソースのクロスリファレンス JOIN フィールドに関する規則が適用されます。固定フォーマットのシーケンシャルファイルは、ソート順序がソースファイルと同一の場合に、参照ファイルとして使用することができます。

構文　参照データソースから値を抽出

```
DB_LOOKUP(look_mf, srcfld1, lookfld1, srcfld2, lookfld2, ..., returnfld);
```

説明

look_mf
参照マスターファイルです。

srcfld1, srcfld2 ...
参照ファイルの一致レコードを検索するために使用する、ソースファイルのフィールドです。

lookfld1, lookfld2 ...
ソースフィールドと値を共有する参照ファイルのフィールドです。テーブルおよびファイルのフィールドのみを使用することができます。DEFINE で作成されたフィールドを使用することはできません。複数セグメントのシノニムの場合、最上位セグメントのフィールドのみを使用することができます。

`returnfld`
一致する参照レコードから値を返す、参照ファイルのフィールド名です。テーブルおよびファイルのフィールドのみを使用することができます。DEFINE で作成されたフィールドを使用することはできません。

参照　DB_LOOKUP 使用上の注意

- レコードの照合に使用可能な組み合わせの最大数は 63 です。
- 参照ファイルが固定フォーマットのシーケンシャルファイルの場合、ソースファイルと同一の順序でソートした上で、抽出する必要があります。ただし、ENGINE INT SET CACHE ON コマンドが有効な場合を除きます。この設定を有効にすると、同一の値が 2 回以上検索される場合に、パフォーマンスが向上するという利点もあります。シーケンシャルファイルのキーフィールドを、DB_LOOKUP リクエストで指定された最初の参照フィールドにする必要があります。それ以外の場合は、一致レコードを検索することはできません。

  また、シーケンシャルファイルへの DB_LOOKUP リクエストを DEFINE FILE コマンド内で発行する場合、これを参照する TABLE リクエストの末尾で DEFINE FILE コマンドをクリアする必要があります。クリアしないと、参照ファイルは開いたままになります。 参照ファイルは閉じるまで再使用することができず、終了時に問題が発生する場合があります。その他の参照ファイルは、DEFINE をクリアしなくても再使用できます。これらは一時項目 (DEFINE) をクリアするとにきに自動的にクリアされます。
- 参照ファイルのマスターファイル内に MISSING=ON 属性が記述され、DEFINE または COMPUTE コマンドで MISSING ON が指定された場合は、参照フィールドが存在しないときにはこのミッシング値が返されます。MISSING ON が両方に指定されていない場合、ミッシング値はデフォルト値 (文字フィールドではブランク、数値フィールドでは 0 (ゼロ)) に変換されます。
- 参照ファイル内に一致するレコードがない場合も、レポート出力にソースレコードが表示されます。
- 参照フィールドおよび返されるフィールドとしては、参照マスターファイルの実フィールドのみが有効です。
- 参照フィールドと同一のソースフィールドを持つ参照テーブルに複数の行が存在する場合、返されるフィールドの最初の値が返されます。

例 TABLE リクエストによる固定フォーマットシーケンシャルファイルからの値の抽出

次のプロシジャは、GGSALES データソースから「GSALE」と呼ばれる固定フォーマットシーケンシャルファイルを作成します。ファイル内のフィールドは、PRODUCT、CATEGORY、PCD です。ファイルは PCD フィールドでソートされます。

```
SET ASNAMES = ON
TABLE FILE GGSALES
SUM PRODUCT CATEGORY
BY PCD
ON TABLE HOLD AS GSALE FORMAT ALPHA
END
```

次のマスターファイルは、HOLD コマンドの結果として作成されます。

```
FILENAME=GSALE, SUFFIX=FIX     , $
  SEGMENT=GSALE, SEGTYPE=S1, $
    FIELDNAME=PCD, ALIAS=E01, USAGE=A04, ACTUAL=A04, $
    FIELDNAME=PRODUCT, ALIAS=E02, USAGE=A16, ACTUAL=A16, $
    FIELDNAME=CATEGORY, ALIAS=E03, USAGE=A11, ACTUAL=A11, $
```

次の TABLE リクエストは、GGPRODS データソースに対して発行され、参照ファイル内のキーフィールドと一致するフィールドでレポートをソートします。PCD および PRODUCT フィールドとの一致により、GSALE 参照ファイルから CATEGORY フィールドの値を検索します。DEFINE FILE コマンドは、リクエストが終了するときにクリアされます。

```
DEFINE FILE GGPRODS
PCAT/A11 MISSING ON = DB_LOOKUP(GSALE,  PRODUCT_ID, PCD,
         PRODUCT_DESCRIPTION, PRODUCT, CATEGORY);
END
TABLE FILE GGPRODS
PRINT PRODUCT_DESCRIPTION PCAT
BY PRODUCT_ID
END
DEFINE FILE GGPRODS CLEAR
END
```

GSALE マスターファイルに CATEGORY の MISSING=ON 属性が定義されていないため、参照ファイル内に一致するレコードがない PCAT フィールドの行にブランクが表示されます。

```
Product     Product          PCAT
Code

-------     -------          ----

B141        Hazelnut

B142        French Roast
```

```
B144        Kona
F101        Scone             Food
F102        Biscotti          Food
F103        Croissant         Food
G100        Mug               Gifts
G104        Thermos           Gifts
G110        Coffee Grinder    Gifts
G121        Coffee Pot        Gifts
```

GSALE マスターファイル内の CATEGORY フィールドに MISSING=ON を追加すると、参照ファイル内に一致するレコードがない行にミッシングデータ文字が表示されます。

```
Product
Code     Product           PCAT
-------  -------           ----
B141     Hazelnut          .
B142     French Roast      .
B144     Kona              .
F101     Scone             Food
F102     Biscotti          Food
F103     Croissant         Food
G100     Mug               Gifts
G104     Thermos           Gifts
G110     Coffee Grinder    Gifts
G121     Coffee Pot        Gifts
```

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 ]);
DECODE fieldname(filename ...[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 で比較する、コード化された値です。この値と、fieldname で指定したフィールドの現在値が比較されます。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、その文字を一重引用符 (') で囲む必要があります。DECODE が特定の値を見つけると、その値に対応する結果が返されます。コードをフィールド名の値と比較するためには、コードとフィールド名が同一フォーマットである必要があります。

`result`
文字または数値

コードに対応する返される値です。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。二重引用符 (") は使用できません。

結果が文字フォーマットの場合、ブランクではない (非 NULL) 文字列を指定します。結果のフォーマットは式のデータタイプと一致させる必要があります。

`default`
文字または数値

不一致コードの結果として返される値です。このフォーマットは、result のフォーマットと一致させる必要があります。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。

`filename`
文字

コードと結果の組み合わせが格納されるファイルの名前です。ファイル内のすべてのレコードに、この組み合わせが含まれている必要があります。

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。

注意：DECODE には output 引数はありません。

例　DECODE 関数を使用した値の指定

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。

```
TABLE FILE EMPLOYEE
PRINT CURR_JOBCODE AND COMPUTE
DEPX_CODE/A1 = EDIT(CURR_JOBCODE, '9$$'); NOPRINT AND COMPUTE
JOB_CATEGORY/A15 = DECODE DEPX_CODE(A 'ADMINISTRATIVE'
                B 'DATA PROCESSING');
BY LAST_NAME
WHERE DEPARTMENT EQ 'MIS';
END
```

出力結果は次のとおりです。

```
LAST_NAME     CURR_JOBCODE     JOB_CATEGORY
---------     ------------     ------------
BLACKWOOD     B04              DATA PROCESSING
CROSS         A17              ADMINISTRATIVE
GREENSPAN     A07              ADMINISTRATIVE
JONES         B03              DATA PROCESSING
MCCOY         B02              DATA PROCESSING
SMITH         B14              DATA PROCESSING
```

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。
- データは、UNIX および Windows では ASCII フォーマットとして解釈され、DECODE の組の USAGE フォーマットに変換されます。
- 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　ファイルからの DECODE 値の読み込み

次の例は、2 つの部分で構成されています。最初の部分は、ID の一覧を含むファイルを作成し、EDUCFILE データソースを読み込みます。次の部分は、EMPLOYEE データソースを読み込み、クラスを受講した従業員には 0 (ゼロ) を、受講していない従業員には 1 を割り当てます。HOLD ファイルに含まれる値は 1 列のみです。このため、DECODE 関数は、EMP_ID がファイルに含まれる従業員には 0 (ゼロ) を、含まれない従業員には 1 を割り当てます。

```
TABLE FILE EDUCFILE
PRINT EMP_ID
ON TABLE HOLD
END

TABLE FILE EMPLOYEE
PRINT EMP_ID AND LAST_NAME AND FIRST_NAME AND COMPUTE
NOT_IN_LIST/I1 = DECODE EMP_ID(HOLD ELSE 1);
WHERE DEPARTMENT EQ 'MIS';
END
```

出力結果は次のとおりです。

```
EMP_ID     LAST_NAME        FIRST_NAME  NOT_IN_LIST
------     ---------        ----------  -----------
112847612  SMITH            MARY                  0
117593129  JONES            DIANE                 0
219984371  MCCOY            JOHN                  1
326179357  BLACKWOOD        ROSEMARIE             0
543729165  GREENSPAN        MARY                  1
818692173  CROSS            BARBARA               0
```

IMPUTE - 集計値でのミッシング値の置換

IMPUTE は、パーティション内のレポート出力のミッシング値 (数値データ) を置換する値を計算します。

ミッシング値を含むデータレコードを分析から除外する代わりに、IMPUTE を使用して、さまざまな推定値でミッシング値を置換することができます。これらの推定値には、平均値、中央値、最頻値、または数値定数があり、すべてリセットキーで指定されたデータパーティション内で計算されます。この関数は、詳細レベルのレポート (PRINT または LIST コマンド) および一時項目 (COMPUTE) (COMPUTE コマンドで作成されるフィールド) で使用されるよう設計されています。

構文　集計値でミッシング値を置換

```
IMPUTE(field, reset_key, replacement)
```

説明

`field`

MISSING ON で定義された数値入力フィールドの名前です。

`reset_key`

計算範囲のパーティションを定義します。有効な値には、次のものがあります。

- ソートフィールド名
- PRESET - SET PARTITION_ON コマンドで定義された分割を使用します。
- TABLE - テーブル全体で計算を実行します。

`replacement`

数値定数または次のいずれかです。

- MEAN
- MEDIAN
- MODE

例 IMPUTE - 集計値でのミッシング値の置換

このサンプルを実行するには、FOCUS データソースの SALEMISS を作成する必要があります。SALEMISS は、RETURNS および DAMAGED フィールドにミッシング値がいくつか含まれた SALES データソースです。以下は、SALEMISS のマスターファイルで、IBISAMP アプリケーションに含まれています。

```
FILENAME=KSALES, SUFFIX=FOC, REMARKS='Legacy Metadata Sample: sales',$

SEGNAME=STOR_SEG, SEGTYPE=S1,
   FIELDNAME=STORE_CODE,  ALIAS=SNO,  FORMAT=A3,   $
   FIELDNAME=CITY,        ALIAS=CTY,  FORMAT=A15,  $
   FIELDNAME=AREA,        ALIAS=LOC,  FORMAT=A1,   $

SEGNAME=DATE_SEG, PARENT=STOR_SEG, SEGTYPE=SH1,
   FIELDNAME=DATE,        ALIAS=DTE,  FORMAT=A4MD, $

SEGNAME=PRODUCT, PARENT=DATE_SEG, SEGTYPE=S1,
   FIELDNAME=PROD_CODE,     ALIAS=PCODE,   FORMAT=A3,    FIELDTYPE=I, $
   FIELDNAME=UNIT_SOLD,     ALIAS=SOLD,    FORMAT=I5,    $
   FIELDNAME=RETAIL_PRICE,  ALIAS=RP,      FORMAT=D5.2M, $
   FIELDNAME=DELIVER_AMT,   ALIAS=SHIP,    FORMAT=I5,    $
   FIELDNAME=OPENING_AMT,   ALIAS=INV,     FORMAT=I5,    $
   FIELDNAME=RETURNS,       ALIAS=RTN,     FORMAT=I3,    MISSING=ON, $
   FIELDNAME=DAMAGED,       ALIAS=BAD,     FORMAT=I3,    MISSING=ON, $
```

次のプロシジャは、SALEMISS データソースを作成後、RETURNS および DAMAGED フィールドにミッシング値を追加します。

```
CREATE FILE ibisamp/SALEMISS
MODIFY FILE ibisamp/SALEMISS
FIXFORM STORE_CODE/3 CITY/15 AREA/1 DATE/4 PROD_CODE/3
FIXFORM UNIT_SOLD/5 RETAIL_PRICE/5 DELIVER_AMT/5
FIXFORM OPENING_AMT/5 RETURNS/3 DAMAGED/3
MATCH STORE_CODE
ON NOMATCH INCLUDE
ON MATCH CONTINUE
MATCH DATE
ON NOMATCH INCLUDE
ON MATCH CONTINUE
MATCH PROD_CODE
ON NOMATCH INCLUDE
ON MATCH REJECT
DATA
14BSTAMFORD       S1212B10   60  .95   80   65 10  6
14BSTAMFORD       S1212B12   40 1.29   20   50  3  3
14BSTAMFORD       S1212B17   29 1.89   30   30  2  1
14BSTAMFORD       S1212C13   25 1.99   30   40  3  0
14BSTAMFORD       S1212C7    45 2.39   50   49  5  4
14BSTAMFORD       S1212D12   27 2.19   40   35  0  0
14BSTAMFORD       S1212E2    80  .99  100  100  9  4
14BSTAMFORD       S1212E3    70 1.09   80   90  8  9
14ZNEW YORK       U1017B10   30  .85   30   10  2  3
14ZNEW YORK       U1017B17   20 1.89   40   25  2  1
14ZNEW YORK       U1017B20   15 1.99   30    5  0  1
14ZNEW YORK       U1017C17   12 2.09   10   15  0  0
14ZNEW YORK       U1017D12   20 2.09   30   10  3  2
14ZNEW YORK       U1017E1    30  .89   25   45  4  7
14ZNEW YORK       U1017E3    35 1.09   25   45  4  2
77FUNIONDALE      R1018B20   25 2.09   40   25  1  1
77FUNIONDALE      R1018C7    40 2.49   40   40  0  0
K1 NEWARK         U1019B12   29 1.49   30   30  1  0
K1 NEWARK         U1018B10   13  .99   30   15  1  1
END
-RUN
```

```
MODIFY FILE ibisamp/SALEMISS
FIXFORM STORE_CODE/3 DATE/5 PROD_CODE/4
FIXFORM UNIT/3 RETAIL/5 DELIVER/3
FIXFORM OPEN/3 RETURNS/C3 DAMAGED/C3
MATCH STORE_CODE
ON NOMATCH INCLUDE
ON MATCH CONTINUE
MATCH DATE
ON NOMATCH INCLUDE
ON MATCH CONTINUE
MATCH PROD_CODE
ON NOMATCH INCLUDE
ON MATCH REJECT
DATA
14Z1017 C13 15 1.99 35 30    6
14Z1017 C14 18 2.05 30 25 4
14Z1017 E2  33 0.99 45 40
END
-RUN
```

SALEMISS データソースに対する次のリクエストは、同一店舗内の値のみを使用して、RETURNS フィールドのミッシング値を置換する値を生成します。

```
SET PARTITION_ON=FIRST
TABLE FILE SALEMISS
PRINT RETURNS
COMPUTE MEDIAN1 = IMPUTE(RETURNS, PRESET, MEDIAN);
COMPUTE MEAN1 = IMPUTE(RETURNS, PRESET, MEAN);
COMPUTE MODE1 = IMPUTE(RETURNS, PRESET, MODE);
BY STORE_CODE
ON TABLE SET PAGE NOPAGE
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果を示しています。ミッシング値は、店舗 14Z で発生しており、PARTITION_ON が FIRST に設定されているため、この店舗の RETURNS の値のみを使用して置換値が計算されます。

STORE_CODE	RETURNS	MEDIAN1	MEAN1	MODE1
14B	10	10.00	10.00	10.00
	3	3.00	3.00	3.00
	2	2.00	2.00	2.00
	3	3.00	3.00	3.00
	5	5.00	5.00	5.00
	0	.00	.00	.00
	9	9.00	9.00	9.00
	8	8.00	8.00	8.00
14Z	2	2.00	2.00	2.00
	2	2.00	2.00	2.00
	0	.00	.00	.00
	.	2.00	2.00	4.00
	4	4.00	4.00	4.00
	0	.00	.00	.00
	3	3.00	3.00	3.00
	4	4.00	4.00	4.00
	.	2.00	2.00	4.00
	4	4.00	4.00	4.00
77F	1	1.00	1.00	1.00
	0	.00	.00	.00
K1	1	1.00	1.00	1.00
	1	1.00	1.00	1.00

PARTITION_ON の設定を TABLE に変更すると、次の出力が生成されます。この場合、置換値はテーブル内のすべての行を使用して計算されます。

STORE_CODE	RETURNS	MEDIAN1	MEAN1	MODE1
14B	10	10.00	10.00	10.00
	3	3.00	3.00	3.00
	2	2.00	2.00	2.00
	3	3.00	3.00	3.00
	5	5.00	5.00	5.00
	0	.00	.00	.00
	9	9.00	9.00	9.00
	8	8.00	8.00	8.00
14Z	2	2.00	2.00	2.00
	2	2.00	2.00	2.00
	0	.00	.00	.00
	.	2.00	3.00	.00
	4	4.00	4.00	4.00
	0	.00	.00	.00
	3	3.00	3.00	3.00
	4	4.00	4.00	4.00
	.	2.00	3.00	.00
	4	4.00	4.00	4.00
77F	1	1.00	1.00	1.00
	0	.00	.00	.00
K1	1	1.00	1.00	1.00
	1	1.00	1.00	1.00

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。

LAST は、ダイアログマネージャの -SET コマンドとともに使用しないでください。

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。

注意：LAST では、output 引数は使用されません。

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。

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME CURR_SAL AND COMPUTE
RUN_TOT/D12.2M = IF DEPARTMENT EQ LAST DEPARTMENT THEN
                 (RUN_TOT + CURR_SAL) ELSE CURR_SAL ;
AS 'RUNNING,TOTAL,SALARY'
BY DEPARTMENT SKIP-LINE
END
```

出力結果は次のとおりです。

```
                                                    RUNNING
                                                      TOTAL
DEPARTMENT  LAST_NAME            CURR_SAL            SALARY
----------  ---------            --------            -------
MIS         SMITH              $13,200.00        $13,200.00
            JONES              $18,480.00        $31,680.00
            MCCOY              $18,480.00        $50,160.00
            BLACKWOOD          $21,780.00        $71,940.00
            GREENSPAN           $9,000.00        $80,940.00
            CROSS              $27,062.00       $108,002.00
PRODUCTION  STEVENS            $11,000.00        $11,000.00
            SMITH               $9,500.00        $20,500.00
            BANNING            $29,700.00        $50,200.00
            IRVING             $26,862.00        $77,062.00
            ROMANS             $21,120.00        $98,182.00
            MCKNIGHT           $16,100.00       $114,282.00
```

NULLIF - 2 つのパラメータが等しい場合の Null 値の取得

NULLIF 関数は、2 つの入力パラメータが等しい場合に Null (ミッシング) 値を返します。これらの入力パラメータが等しくない場合、1 つ目の値が返されます。値が返されるフィールドには、MISSING ON を設定しておく必要があります。

構文　2 つのパラメータが等しい場合の Null 値の取得

```
NULLIF(arg1,arg2)
```

説明

`arg1, arg2`

任意のフィールドタイプ、定数、式のいずれかです。

これらの入力パラメータの等価性がテストされます。これらの入力パラメータは、両方とも数値であるか、両方とも文字である必要があります。

出力データタイプは入力データタイプと同一です。

例　2 つのパラメータの等価性テスト

次のリクエストでは、NULLIF 関数を使用して、DAMAGED と RETURNS のフィールド値の等価性をテストします。

```
DEFINE FILE SALES
NULL1/I4 MISSING ON = NULLIF(DAMAGED, RETURNS);
END
TABLE FILE SALES
PRINT DAMAGED RETURNS NULL1
BY STORE_CODE
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
END
```

下図は、出力結果を示しています。

STORE_CODE	DAMAGED	RETURNS	NULL1
14B	6	10	6
	3	3	.
	1	2	1
	0	3	0
	4	5	4
	0	0	.
	4	9	4
	9	8	9
14Z	3	2	3
	1	2	1
	1	0	1
	0	0	.
	2	3	2
	7	4	7
	2	4	2
77F	1	1	.
	0	0	.
K1	0	1	0
	1	1	.

10

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。日付および日付時間パラメータは、完全な日付構成要素を入力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

DAYNAME - 日付式から曜日名を取得

DAYNAME 関数は、日付式の曜日部分をデータソース固有の曜日名を含む文字列として取得します。

構文　日付式から曜日名を取得

```
DAYNAME(date_exp)
```

説明

`date_exp`

日付または日付時間式です。

例　日付式から曜日名を取得

次のリクエストは、TIME_DATE フィールドから曜日名を取得します。

```
TABLE FILE WF_RETAIL_TIME
PRINT TIME_DATE
COMPUTE DAYNAME1/A12 = DAYNAME(TIME_DATE);
WHERE RECORDLIMIT EQ 5
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果は次のとおりです。

Date	DAYNAME1
2009/01/01	Thursday
2009/01/02	Friday
2009/01/03	Saturday
2009/01/04	Sunday
2009/01/05	Monday

DT_CURRENT_DATE - 現在日付の取得

DT_CURRENT_DATE 関数は、実行中のオペレーティング環境から提供される現在日付時間を日付時間フォーマットで返します。日付時間の時間部分は 0 (ゼロ) に設定されます。

構文　現在日付の取得

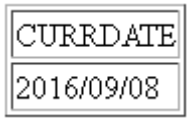
```
DT_CURRENT_DATE()
```

例　現在日付の取得

次のリクエストは、現在の日付を返します。

```
DEFINE FILE WF_RETAIL_LITE
CURRDATE/YYMD WITH COUNTRY_NAME = DT_CURRENT_DATE();
END
TABLE FILE WF_RETAIL_LITE
SUM CURRDATE
ON TABLE SET PAGE NOPAGE
END
```

下図は、出力結果を示しています。



DT_CURRENT_DATETIME - 現在日付時間の取得

DT_CURRENT_DATETIME 関数は、実行中のオペレーティング環境から日付時間フォーマットで提供される現在日付時間を取得し、指定された時間精度で返します。

構文　現在日付時間の取得

```
DT_CURRENT_DATETIME(component)
```

説明

`component`

次の時間精度のいずれかです。

- SECOND
- MILLISECOND

- MICROSECOND

注意：値が返されるフィールドは、指定された時間精度をサポートするフォーマットにする必要があります。

例　現在日付時間の取得

次のリクエストは、現在の日付時間を、指定された時間精度 (マイクロ秒) で返します。

```
DEFINE FILE WF_RETAIL_LITE
CURRDATE/HYYMDm WITH COUNTRY_NAME = DT_CURRENT_DATETIME(MICROSECOND);
END
TABLE FILE WF_RETAIL_LITE
SUM CURRDATE
ON TABLE SET PAGE NOPAGE
END
```

下図は、出力結果を示しています。

CURRDATE
2016/09/08 17:10:31.605718

DT_CURRENT_TIME - 現在時間の取得

DT_CURRENT_TIME 関数は、実行中のオペレーティング環境から日付時間フォーマットで提供される現在時間を、指定された時間精度で返します。返される日付時間値の日付部分は 0 (ゼロ) に設定されます。

構文　現在時間の取得

```
DT_CURRENT_TIME(component)
```

説明

`component`

次の時間精度のいずれかです。

- SECOND
- MILLISECOND
- MICROSECOND

注意：値が返されるフィールドは、指定された時間精度をサポートするフォーマットにする必要があります。

例　現在時間の取得

次のリクエストは、ミリ秒に設定された時間精度で現在時間を返します。

```
DEFINE FILE WF_RETAIL_LITE
CURRTIME/HHISs WITH COUNTRY_NAME = DT_CURRENT_TIME(MILLISECOND);
END
TABLE FILE WF_RETAIL_LITE
SUM CURRTIME
ON TABLE SET PAGE NOPAGE
END
```

下図は、出力結果を示しています。

CURRTIME
17:23:13.098

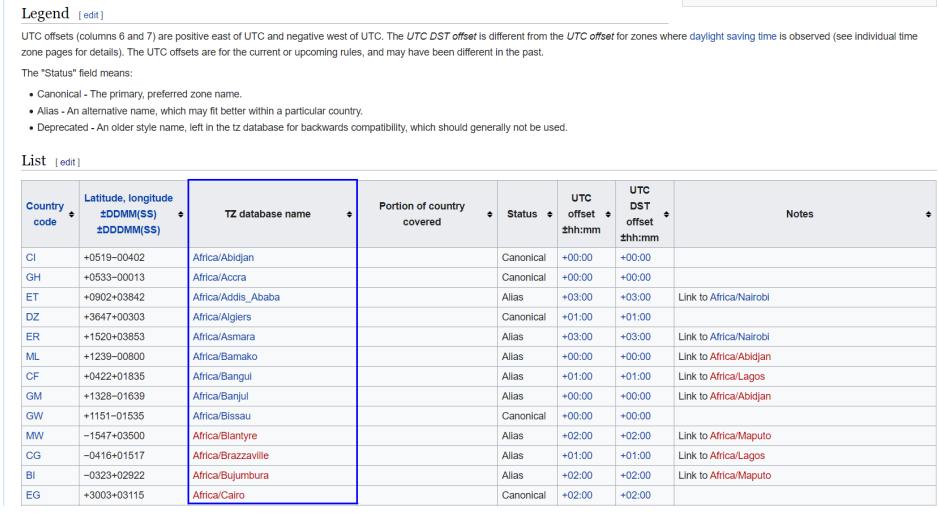
DT_TOLOCAL - UTC からローカルタイムへの変換

協定世界時 (UTC) は、世界共通に使用されている標準時間です。UTC をローカルタイムに変換するには、各地域とグリニッジ標準時 (GMT) 間のタイムゾーン数に応じて、特定の時間数を UTC に加算したり、UTC から減算したりする必要があります。

DT_TOLOCAL 関数は、UTC をローカルタイムに変換します。

各地域のタイムスタンプ値を共通の標準時間に変換することで、イベントを実際のイベント発生順序でソートすることが可能になります。

この関数では、IANA (Internet Assigned Numbers Authority) が管理する TZ データベース ('Area/Location' で表現) をパラメータとして使用する必要があります。IANA TZ データベース名についての詳細は、下図のように、ウィキペディア (https://en.wikipedia.org/wiki/List_of_tz_database_time_zones) を参照してください。



Legend [edit]

UTC offsets (columns 6 and 7) are positive east of UTC and negative west of UTC. The *UTC DST offset* is different from the *UTC offset* for zones where daylight saving time is observed (see individual time zone pages for details). The UTC offsets are for the current or upcoming rules, and may have been different in the past.

The "Status" field means:

- Canonical - The primary, preferred zone name.
- Alias - An alternative name, which may fit better within a particular country.
- Deprecated - An older style name, left in the tz database for backwards compatibility, which should generally not be used.

List [edit]

Country code	Latitude, longitude ±DDMM(SS) ±DDDMM(SS)	TZ database name	Portion of country covered	Status	UTC offset ±hh:mm	UTC DST offset ±hh:mm	Notes
CI	+0519−00402	Africa/Abidjan		Canonical	+00:00	+00:00	
GH	+0533−00013	Africa/Accra		Canonical	+00:00	+00:00	
ET	+0902+03842	Africa/Addis_Ababa		Alias	+03:00	+03:00	Link to Africa/Nairobi
DZ	+3647+00303	Africa/Algiers		Canonical	+01:00	+01:00	
ER	+1520+03853	Africa/Asmara		Alias	+03:00	+03:00	Link to Africa/Nairobi
ML	+1239−00800	Africa/Bamako		Alias	+00:00	+00:00	Link to Africa/Abidjan
CF	+0422+01835	Africa/Bangui		Alias	+01:00	+01:00	Link to Africa/Lagos
GM	+1328−01639	Africa/Banjul		Alias	+00:00	+00:00	Link to Africa/Abidjan
GW	+1151−01535	Africa/Bissau		Canonical	+00:00	+00:00	
MW	−1547+03500	Africa/Blantyre		Alias	+02:00	+02:00	Link to Africa/Maputo
CG	−0416+01517	Africa/Brazzaville		Alias	+01:00	+01:00	Link to Africa/Lagos
BI	−0323+02922	Africa/Bujumbura		Alias	+02:00	+02:00	Link to Africa/Maputo
EG	+3003+03115	Africa/Cairo		Canonical	+02:00	+02:00	

該当するタイムゾーンのエリアおよびロケーションは不明だが、GMT との時差または従来のタイムゾーン名 (例、EST) が分かっている場合は、この表を下方向へスクロールします。これらのタイムゾーン識別子に対応する TZ データベース名が、下図のように記載されています。

		EST		Deprecated	−05:00	−05:00	Choose a zone that currently *observes* EST without daylight saving time, such as America/Cancun.
		EST5EDT		Deprecated	−05:00	−04:00	Choose a zone that *observes* EST with United States daylight saving time rules, such as America/New_York.
		Etc/GMT		Canonical	+00:00	+00:00	
		Etc/GMT+0		Alias	+00:00	+00:00	Link to Etc/GMT
		Etc/GMT+1		Canonical	−01:00	−01:00	Sign is intentionally inverted. See the Etc area description.
		Etc/GMT+10		Canonical	−10:00	−10:00	Sign is intentionally inverted. See the Etc area description.
		Etc/GMT+11		Canonical	−11:00	−11:00	Sign is intentionally inverted. See the Etc area description.
		Etc/GMT+12		Canonical	−12:00	−12:00	Sign is intentionally inverted. See the Etc area description.
		Etc/GMT+2		Canonical	−02:00	−02:00	Sign is intentionally inverted. See the Etc area description.
		Etc/GMT+3		Canonical	−03:00	−03:00	Sign is intentionally inverted. See the Etc area description.
		Etc/GMT+4		Canonical	−04:00	−04:00	Sign is intentionally inverted. See the Etc area description.
		Etc/GMT+5		Canonical	−05:00	−05:00	Sign is intentionally inverted. See the Etc area description.

注意：標準の IANA タイムゾーンデータベース名を 'Area/Location' (例、'America/New_York') 形式で使用する場合は、夏時間の調整が自動的に行われます。GMT との時差に対応する名前または従来のタイムゾーン名を使用する場合は、夏時間の調整をユーザが行う必要があります。

構文　UTC のローカルタイムへの変換

```
DT_TOLOCAL(datetime, timezone)
```

説明

`datetime`

日付時間

UTC を date-time 形式で表します。日付および時間構成要素が含まれます。

`timezone`

文字

ローカルタイムの IANA タイムゾーン名を含む文字式です。'Area/Location' (例、'America/New_York') 形式で記述します。

例　UTC のローカルタイムへの変換

次のリクエストは、現在の日付時間値を UTC から 'America/New_York' タイムゾーンのローカルタイムに変換します。

```
TABLE FILE GGSALES
SUM DOLLARS NOPRINT
COMPUTE UTC1/HYYMDS  = DT_CURRENT_DATETIME(SECOND);
COMPUTE LOCAL1/HYYMDS = DT_TOLOCAL(UTC1, 'America/New_York');
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果は次のとおりです。

UTC1	LOCAL1
2020/09/04 15:00:26	2020/09/04 11:00:26

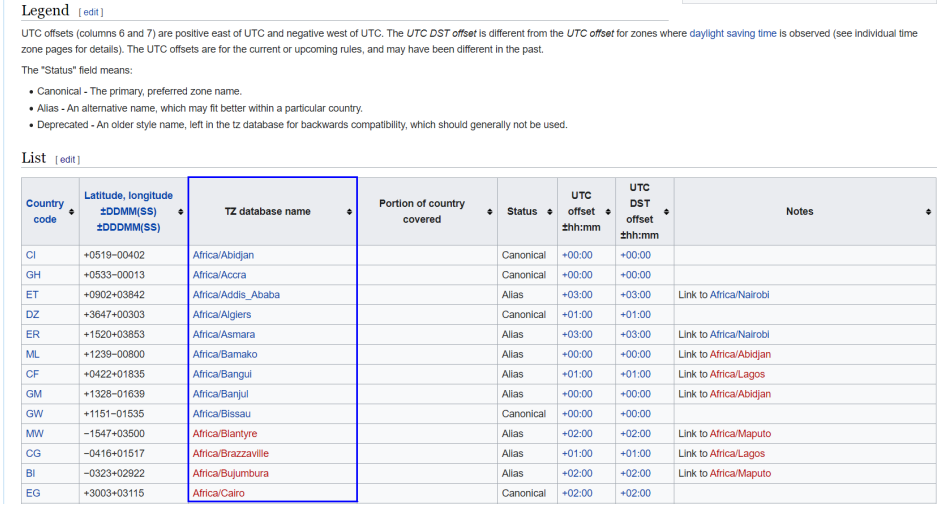
DT_TOUTC - ローカルタイムから UTC への変換

協定世界時 (UTC) は、世界共通に使用されている標準時間です。UTC をローカルタイムに変換するには、各地域とグリニッジ標準時 (GMT) 間のタイムゾーン数に応じて、特定の時間数を UTC に加算したり、UTC から減算したりする必要があります。

DT_TOUTC 関数は、ローカルタイムを UTC に変換します。

各地域のタイムスタンプ値を共通の標準時間に変換することで、イベントを実際のイベント発生順序でソートすることが可能になります。

この関数では、IANA (Internet Assigned Numbers Authority) が管理する TZ データベース ('Area/Location' で表現) をパラメータとして使用する必要があります。IANA TZ データベース名についての詳細は、下図のように、ウィキペディア (https://en.wikipedia.org/wiki/List_of_tz_database_time_zones) を参照してください。



Legend [edit]

UTC offsets (columns 6 and 7) are positive east of UTC and negative west of UTC. The *UTC DST offset* is different from the *UTC offset* for zones where daylight saving time is observed (see individual time zone pages for details). The UTC offsets are for the current or upcoming rules, and may have been different in the past.

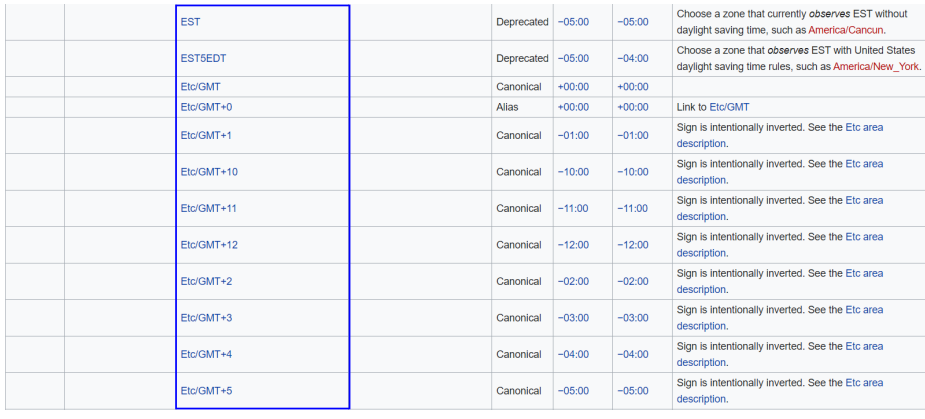
The "Status" field means:

- Canonical - The primary, preferred zone name.
- Alias - An alternative name, which may fit better within a particular country.
- Deprecated - An older style name, left in the tz database for backwards compatibility, which should generally not be used.

List [edit]

Country code	Latitude, longitude ±DDMM(SS) ±DDDMM(SS)	TZ database name	Portion of country covered	Status	UTC offset ±hh:mm	UTC DST offset ±hh:mm	Notes
CI	+0519−00402	Africa/Abidjan		Canonical	+00:00	+00:00	
GH	+0533−00013	Africa/Accra		Canonical	+00:00	+00:00	
ET	+0902+03842	Africa/Addis_Ababa		Alias	+03:00	+03:00	Link to Africa/Nairobi
DZ	+3647+00303	Africa/Algiers		Canonical	+01:00	+01:00	
ER	+1520+03853	Africa/Asmara		Alias	+03:00	+03:00	Link to Africa/Nairobi
ML	+1239−00800	Africa/Bamako		Alias	+00:00	+00:00	Link to Africa/Abidjan
CF	+0422+01835	Africa/Bangui		Alias	+01:00	+01:00	Link to Africa/Lagos
GM	+1328−01639	Africa/Banjul		Alias	+00:00	+00:00	Link to Africa/Abidjan
GW	+1151−01535	Africa/Bissau		Canonical	+00:00	+00:00	
MW	−1547+03500	Africa/Blantyre		Alias	+02:00	+02:00	Link to Africa/Maputo
CG	−0416+01517	Africa/Brazzaville		Alias	+01:00	+01:00	Link to Africa/Lagos
BI	−0323+02922	Africa/Bujumbura		Alias	+02:00	+02:00	Link to Africa/Maputo
EG	+3003+03115	Africa/Cairo		Canonical	+02:00	+02:00	

該当するタイムゾーンのエリアおよびロケーションは不明だが、GMT との時差または従来のタイムゾーン名 (例、EST) が分かっている場合は、この表を下方向へスクロールします。これらのタイムゾーン識別子に対応する TZ データベース名が、下図のように記載されています。



		EST		Deprecated	−05:00	−05:00	Choose a zone that currently *observes* EST without daylight saving time, such as America/Cancun.
		EST5EDT		Deprecated	−05:00	−04:00	Choose a zone that *observes* EST with United States daylight saving time rules, such as America/New_York.
		Etc/GMT		Canonical	+00:00	+00:00	
		Etc/GMT+0		Alias	+00:00	+00:00	Link to Etc/GMT
		Etc/GMT+1		Canonical	−01:00	−01:00	Sign is intentionally inverted. See the Etc area description.
		Etc/GMT+10		Canonical	−10:00	−10:00	Sign is intentionally inverted. See the Etc area description.
		Etc/GMT+11		Canonical	−11:00	−11:00	Sign is intentionally inverted. See the Etc area description.
		Etc/GMT+12		Canonical	−12:00	−12:00	Sign is intentionally inverted. See the Etc area description.
		Etc/GMT+2		Canonical	−02:00	−02:00	Sign is intentionally inverted. See the Etc area description.
		Etc/GMT+3		Canonical	−03:00	−03:00	Sign is intentionally inverted. See the Etc area description.
		Etc/GMT+4		Canonical	−04:00	−04:00	Sign is intentionally inverted. See the Etc area description.
		Etc/GMT+5		Canonical	−05:00	−05:00	Sign is intentionally inverted. See the Etc area description.

注意：標準の IANA タイムゾーンデータベース名を 'Area/Location' (例、'America/New_York') 形式で使用する場合は、夏時間の調整が自動的に行われます。GMT との時差に対応する名前または従来のタイムゾーン名を使用する場合は、夏時間の調整をユーザが行う必要があります。

構文　ローカルタイムの UTC への変換

```
DT_TOUTC(datetime, timezone)
```

説明

`datetime`

日付時間

ローカルタイムを date-time 形式で表します。日付および時間構成要素が含まれます。

`timezone`

文字

ローカルタイムの IANA タイムゾーン名を含む文字式です。'Area/Location' (例、'America/New_York') 形式で記述します。

例　ローカルタイムの UTC への変換

次のリクエストは、America/New_York タイムゾーンの現在のローカル日付時間値を UTC に変換します。

```
TABLE FILE GGSALES
SUM DOLLARS NOPRINT
COMPUTE LOCAL1/HYYMDS  = DT_CURRENT_DATETIME(SECOND);
COMPUTE UTC1/HYYMDS = DT_TOUTC(LOCAL1, 'America/New_York');
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果は次のとおりです。

LOCAL1	UTC1
2020/09/04 14:49:41	2020/09/04 18:49:41

例　UTC によるソート

次のリクエストは、現在の日付時間を LOCALT1 フィールドに取得し、TIMEZONE フィールドを IANA タイムゾーンデータベース名に設定します。次に、DT_TOUTC を使用して、このローカルタイムをさまざまなタイムゾーンごとに対応する UTC に変換し、生成された UTC に基づいて出力をソートします。

```
DEFINE FILE GGSALES
LOCALT1/HYYMDS=DT_CURRENT_DATETIME(SECOND);
TIMEZONE/A30=IF LAST TIMEZONE EQ ' ' THEN 'AMERICA/NEW_YORK'
 ELSE IF LAST TIMEZONE EQ 'AMERICA/NEW_YORK' THEN 'AMERICA/CHICAGO'
 ELSE IF LAST TIMEZONE EQ 'AMERICA/CHICAGO' THEN  'AMERICA/DENVER'
 ELSE IF LAST TIMEZONE EQ 'AMERICA/DENVER' THEN 'ASIA/TOKYO'
 ELSE IF LAST TIMEZONE EQ 'ASIA/TOKYO' THEN 'EUROPE/LONDON'
 ELSE IF LAST TIMEZONE EQ 'EUROPE/LONDON' THEN 'AMERICA/NEW_YORK';
UTCTIME/HYYMDS=DT_TOUTC(LOCALT1,TIMEZONE);
END
TABLE FILE GGSALES
PRINT TIMEZONE LOCALT1 DOLLARS NOPRINT
BY UTCTIME
WHERE PRODUCT EQ 'Thermos'
IF RECORDLIMIT EQ 20
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果は次のとおりです。

UTCTIME	TIMEZONE	LOCALT1
2020/10/02 06:45:59	ASIA/TOKYO	2020/10/02 15:45:59
	ASIA/TOKYO	2020/10/02 15:45:59
	ASIA/TOKYO	2020/10/02 15:45:59
	ASIA/TOKYO	2020/10/02 15:45:59
2020/10/02 14:45:59	EUROPE/LONDON	2020/10/02 15:45:59
	EUROPE/LONDON	2020/10/02 15:45:59
	EUROPE/LONDON	2020/10/02 15:45:59
	EUROPE/LONDON	2020/10/02 15:45:59
2020/10/02 19:45:59	AMERICA/NEW_YORK	2020/10/02 15:45:59
	AMERICA/NEW_YORK	2020/10/02 15:45:59
	AMERICA/NEW_YORK	2020/10/02 15:45:59
	AMERICA/NEW_YORK	2020/10/02 15:45:59
2020/10/02 20:45:59	AMERICA/CHICAGO	2020/10/02 15:45:59
	AMERICA/CHICAGO	2020/10/02 15:45:59
	AMERICA/CHICAGO	2020/10/02 15:45:59
	AMERICA/CHICAGO	2020/10/02 15:45:59
2020/10/02 21:45:59	AMERICA/DENVER	2020/10/02 15:45:59
	AMERICA/DENVER	2020/10/02 15:45:59
	AMERICA/DENVER	2020/10/02 15:45:59
	AMERICA/DENVER	2020/10/02 15:45:59

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。完全な構成要素にする必要があります。

`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の完全構成要素です。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の完全構成要素です。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

DTIME - 日付時間値からの時間構成要素の抽出

DTIME 関数は、指定された日付時間値および時間構成要素キーワードから、要求した構成要素までの (その構成要素を含む) 時間構成要素のすべての値を返します。残りの時間構成要素は 0 (ゼロ) に設定されます。時間構成要素が返されるフィールドには、その構成要素をサポートする時間フォーマットを定義しておく必要があります。

構文　日付時間値からの時間構成要素の抽出

```
DTIME(datetime, component)
```

説明

`datetime`

日付時間

時間構成要素を抽出する日付時間値です。フィールド名にすることも日付時間のリテラルにすることもできます。完全構成要素にする必要があります。

`component`

キーワード

有効な値には、次のものがあります。

- TIME　時間部分が完全形式で返されます。最小構成要素は、入力日付時間フォーマットによって異なります。ナノ秒はサポートされず、取得されません。
- HOUR　時間単位までを含む時間構成要素が抽出されます。
- MINUTE　分単位までを含む時間構成要素が抽出されます。
- SECOND　秒単位までを含む時間構成要素が抽出されます。
- MILLISECOND　ミリ秒単位までを含む時間構成要素が抽出されます。
- MICROSECOND　マイクロ秒単位までを含む時間構成要素が抽出されます。

例　時間構成要素の抽出

次のリクエストは、2 つの日付時間フィールドを定義します。

- TRANSTIME は、TRANSDATE から抽出された時間要素を分単位までを含みます。
- TRANSTIME2 は、リテラルの日付時間値 (2018/01/17 05:45:22.777888) から時間構成要素をすべて抽出します。

```
DEFINE FILE VIDEOTR2
TRANSTIME/HHISsm = DTIME(TRANSDATE, MINUTE);
TRANSTIME2/HHISsm = DTIME(DT(2018/01/17 05:45:22.777888), TIME);
END
TABLE FILE VIDEOTR2
SUM  TRANSTIME TRANSTIME2
BY MOVIECODE
BY  TRANSDATE
WHERE MOVIECODE CONTAINS 'MGM'
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果を示しています。

MOVIECODE	TRANSDATE	TRANSTIME	TRANSTIME2
145MGM	1999/11/06 02:12	02:12:00.000000	05:45:22.777888
243MGM	1991/06/19 04:11	04:11:00.000000	05:45:22.777888
259MGM	1991/06/19 07:18	07:18:00.000000	05:45:22.777888
284MGM	1999/06/18 03:30	03:30:00.000000	05:45:22.777888
505MGM	1996/06/21 01:16	01:16:00.000000	05:45:22.777888
518MGM	1991/06/24 04:43	04:43:00.000000	05:45:22.777888
	1998/10/03 02:41	02:41:00.000000	05:45:22.777888
	1999/11/18 10:27	10:27:00.000000	05:45:22.777888
688MGM	1998/03/19 07:23	07:23:00.000000	05:45:22.777888
	1999/04/22 06:19	06:19:00.000000	05:45:22.777888
	1999/10/22 06:25	06:25:00.000000	05:45:22.777888
	1999/10/30 06:29	06:29:00.000000	05:45:22.777888
	1999/11/19 10:26	10:26:00.000000	05:45:22.777888

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完全構成要素です。

`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, extend)
```

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。完全な構成要素にする必要があります。

`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。

`extend`

オプション。取得される日付範囲に含める特定の日付構成要素の数を示す数値です。

すべての時間単位は同一サイズにする必要があるため、extend 引数は、各日付範囲で次の値に制限されます。

- YEAR - 制限なし。
- QUARTER - 1 と 2 のみ。
- MONTH - 1、2、3、4、および 6 のみ。
- HOUR1、2、3、4、6、および 12 のみ。
- MINUTE1、2、3、4、5、6、10、15、20、および 30 のみ。
- SECOND1、2、3、4、5、6、10、15、20、および 30 のみ。

例　日付範囲の開始日を取得

次のリクエストでは、DTRUNC 関数は、指定された従業員の開始日に基づいて、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

例　DTRUNC 関数での週開始日パラメータの使用

次のリクエストは、特定の従業員の勤務開始日から、その勤務開始日が属する週の開始日の日付を返します。

```
DEFINE FILE WF_RETAIL
DAY1/WT = DTRUNC(START_DATE, DAY);
WKSTART/YYMD = DTRUNC(START_DATE, WEEK);
DAY2/WT = DTRUNC(WKSTART, DAY);
END
TABLE FILE WF_RETAIL
PRINT START_DATE
DAY1 AS 'DOW 1'
WKSTART AS 'Start,of Week'
DAY2 AS 'DOW 2'
BY EMPLOYEE_NUMBER
WHERE START_DATE GT '20130101'
WHERE EMPLOYEE_NUMBER CONTAINS 'AH'
ON TABLE SET PAGE NOPAGE
END
```

出力結果は次のとおりです。

```
Employee      Start              Start
Number        Date        DOW 1  of Week     DOW 2
--------      -----       -----  -------     -----
AH118         2013/01/15  TUE    2013/01/13  SUN
AH2272        2013/01/17  THU    2013/01/13  SUN
AH288         2013/11/11  MON    2013/11/10  SUN
AH3520        2013/09/23  MON    2013/09/22  SUN
AH3591        2013/09/22  SUN    2013/09/22  SUN
AH5177        2013/07/21  SUN    2013/07/21  SUN
```

例　週の開始曜日および終了曜日の日付を取得

次のリクエストは、指定された日付が属する週の開始曜日および終了曜日に対応する日付を返します。

```
DEFINE FILE WF_RETAIL
WEEKSTART/YYMD = DTRUNC(START_DATE, WEEK);
WEEKEND/YYMD = DTRUNC(START_DATE, WEEK_END);
END
TABLE FILE WF_RETAIL
PRINT START_DATE WEEKSTART AS 'Start,of Week'
WEEKEND AS 'End,of Week'
BY EMPLOYEE_NUMBER
WHERE EMPLOYEE_NUMBER CONTAINS 'AH1'
ON TABLE SET PAGE NOPAGE
END
```

下図は、出力結果を示しています。

Employee Number	Start Date	Start of Week	End of Week
AH118	2013/01/15	2013/01/13	2013/01/19
AH1348	2009/11/19	2009/11/15	2009/11/21
AH1398	2009/11/11	2009/11/08	2009/11/14
AH1994	2006/01/01	2006/01/01	2006/01/07

例　extend 引数を使用した日付の取得

次のリクエストでは、指定された各従業員の生年月日に基づいて、DTRUNC 関数の extend 引数により、従業員が生まれた 10 年の開始日が取得されます。

```
DEFINE FILE WF_RETAIL
BIRTH_DECADE/YYMD = DTRUNC(DATE_OF_BIRTH, YEAR, 10);
END
TABLE FILE WF_RETAIL
PRINT DATE_OF_BIRTH BIRTH_DECADE AS 'Start,of Decade'
BY EMPLOYEE_NUMBER
ON TABLE SET PAGE NOPAGE
END
```

出力結果は次のとおりです。

Employee Number	Date of Birth	Start of Decade
AD1804	1975/04/21	1970/01/01
AG5105	1971/02/28	1970/01/01
AT1871	1983/08/04	1980/01/01
BD3005	1975/08/22	1970/01/01
BM1802	1988/12/04	1980/01/01
DW5139	1979/04/02	1970/01/01
HV3086	1977/02/08	1970/01/01
IA1888	1989/08/15	1980/01/01
JF99999	1975/07/03	1970/01/01
JH5164	1970/08/01	1970/01/01
KV5101	1976/12/23	1970/01/01
LE3001	1982/11/05	1980/01/01
MS5102	1986/03/24	1980/01/01
PM5104	1979/05/02	1970/01/01
RA1801	1974/11/14	1970/01/01
RB3033	1977/02/22	1970/01/01
SV3002	1988/09/14	1980/01/01
YS3004	1976/09/13	1970/01/01
ZC1870	1974/05/10	1970/01/01

MONTHNAME - 日付式から月名を取得

MONTHNAME 関数は、日付式の月の部分について、データソース固有の月名を含む文字列を返します。

構文　日付式から月名を取得

```
MONTHNAME(date_exp)
```

説明

`date_exp`

日付または日付時間式です。

例　日付式から月名を取得

次のリクエストは、TRANSDATE フィールドから月名を取得します。

```
TABLE FILE EMPLOYEE
PRINT HIRE_DATE
COMPUTE TRANSDATE/YYMD= HIRE_DATE; NOPRINT
COMPUTE MONTHNAME1/A12 = MONTHNAME(TRANSDATE);
BY  TRANSDATE
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果は次のとおりです。

TRANSDATE	HIRE_DATE	MONTHNAME1
1980/06/02	80/06/02	June
1981/07/01	81/07/01	July
	81/07/01	July
1981/11/02	81/11/02	November
1982/01/04	82/01/04	January
	82/01/04	January
1982/02/02	82/02/02	February
1982/04/01	82/04/01	April
	82/04/01	April
1982/05/01	82/05/01	May
1982/07/01	82/07/01	July
1982/08/01	82/08/01	August

11

日付関数

日付関数は、日付値を操作します。日付関数には、次の 2 種類があります。

- 標準日付関数。レガシー日付以外の日付で使用します。
- レガシー日付関数。レガシー日付のみで使用します。

  日付が日付表示オプション (例、I6YMD) を含む文字または数値フィールドの場合は、レガシー日付関数を使用する必要があります。

トピックス

日付関数の概要

ここでは、2 種類の日付関数の相違点について説明します。

- **標準日付関数**　標準日付フォーマット (通常の日付フォーマット) で使用します。日付フォーマットは、世紀、年、四半期、月、日などの「日付構成要素」を保持することが可能な内部格納データです。時間構成要素は含まれません。シノニムでは、日付フォーマットの内部データタイプや長さは指定せず、D (日)、M (月)、Q (四半期)、Y (2 桁の年)、YY (4 桁の年) などの日付表示要素を指定します。たとえば、MDYY フォーマットは 3 つの日付構成要素が含まれた日付フォーマットで、シノニムの USAGE 属性で使用することができます。このフォーマットにより記述される 2004 年 3 月 9 日などの実際の日付値は、デフォルト設定では、03/09/2004 と表示されます。日付フォーマットは、日付構成要素のすべてまたは一部で構成されます。完全構成要素を持つフォーマットは、3 つの文字すべて (例、D、M、Y) を含みます。ユリウス暦を示す JUL を含めることもできます。これ以外の日付フォーマットは、すべて構成要素の一部で構成されます。日付関数には、引数に日付フィールドの完全構成要素を指定するものと、完全構成要素または構成要素の一部を指定するものがあります。日付フォーマットは、以前は「SmartDate」と呼ばれていました。
- **レガシー日付関数**　レガシー日付のみで使用します。レガシー日付は、I6YMD、A6MDY、I8YYMD、A8MDYY などの日付編集オプションを含むフォーマットです。たとえば、A6MDY は 6 バイトの文字列であり、接尾語 MDY は、日付構成要素がフィールドに保存される順序を示します。さらに、接頭語 I または A は、数値または文字形式であることを示します。たとえば、「030599」という値は A6MDY フォーマットのフィールドに割り当てることができ、この表示は 03/05/99 になります。

日付フォーマットは、数値と文字のいずれかのフォーマットの内部表現を持ちます。たとえば、A6MDY は文字フォーマットに一致し、YYMD および I6DMY は数値フォーマットに一致します。関数の出力が output フォーマットで指定された日付の場合、その出力を同一フォーマットの別のフィールドに割り当てることができます。また、フォーマットが一致する別のデータ操作に使用することもできます。異なる日付フォーマットを別のフィールドへ割り当てると、無作為な結果が発生します。

多くの関数では、output 引数にフィールド名またはフォーマットを指定することができます。フォーマットを指定する場合、一重引用符 (') で囲みます。 ただし、関数がダイアログマネージャコマンドから呼び出される場合、この引数には常にフォーマットを指定する必要があります。関数の呼び出しおよび引数の指定についての詳細は、45 ページの「 関数へのアクセスと呼び出し 」 を参照してください。

標準日付関数の使用

標準日付関数を使用するには、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。

日付関数の動作は、次の方法で変更することができます。

- 営業日を定義する。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。詳細は、315 ページの 「 営業日の指定 」 を参照してください。
- ダイアログマネージャの日付関数が日付を返すときに、先頭の 0 (ゼロ) を表示するかを決定する。詳細は、318 ページの 「 ダイアログマネージャの日付時間関数による先頭 0 (ゼロ) の有効化 」 を参照してください。

各標準日付関数についての詳細は、次のトピックを参照してください。

319 ページの 「 DATEADD - 日付単位数を日付に加算または日付から減算 」

323 ページの 「 DATECVT - 日付フォーマットを変換 」

325 ページの 「 DATEDIF - 2 つの日付の差を計算 」

327 ページの 「 DATEMOV - 日付を有効な位置に移動 」

333 ページの 「 DATETRAN - 日付を国際フォーマットに変換 」

348 ページの 「 DPART - 日付から構成要素を抽出 」

352 ページの 「 FIYR - 会計年度の取得 」

349 ページの 「 FIQTR - 会計四半期の取得 」

354 ページの 「 FIYYQ - カレンダー日付を会計日付に変換 」

357 ページの 「 TODAY - 現在の日付を取得 」

営業日の指定

営業日を決定することができます。営業日の設定は、DATEADD、DATEDIF、および DATEMOV 関数に影響します。営業日を平日または祝日として識別します。

営業日の指定

営業日は通常月曜日から金曜日ですが、そうではない場合もあります。たとえば、営業日が日曜日、火曜日、水曜日、金曜日、土曜日の場合は、営業日を編集してスケジュールに反映することができます。

構文　営業日の設定

```
SET BUSDAYS = smtwtfs
```

説明

`smtwtfs`
営業日を表す 7 バイトの曜日リストです。リストには、日曜日から土曜日までの各曜日の位置が含まれています。

- ある曜日を営業日として識別するには、その曜日の頭文字を所定の位置に入力します。
- 営業日ではない日として識別するには、その曜日の所定の位置にアンダースコア (_) を入力します。

文字が正しい位置に入力されない場合、または営業日ではない曜日にアンダースコア (_) 以外の値を指定した場合は、エラーメッセージが表示されます。

例　営業日の設定と反映

次の例では、営業日として日曜日、火曜日、水曜日、金曜日、土曜日が指定されています。

```
SET BUSDAYS = S_TW_FS
```

構文　現在の営業日の設定の表示

```
? SET BUSDAYS
```

祝日の指定

会社の祝日として定める日付のリストを指定することができます。これらの日付は、営業日に基づいた計算を実行する関数の使用時に除外されます。たとえば、ある週の木曜日が祝日として指定されている場合、その週の水曜日の次の営業日は金曜日になります。

祝日のリストを定義するには、次の手順を実行する必要があります。

1. 標準テキストエディタを使用して祝日ファイルを作成します。
2. SET コマンドを HDAY パラメータ付きで発行し、祝日ファイルを選択します。

参照　祝日ファイルの作成規則

- 日付フォーマットには YYMD フォーマットを使用します。
- 日付の順序には昇順を使用します。

- 日付は 1 行に 1 つのみ指定します。
- データが存在する年のそれぞれが祝日ファイルに含まれている必要があります。データが存在する年が祝日ファイルに含まれていない場合、祝日ファイルが無効であると見なされます。祝日ファイルの範囲外の日付値で日付関数を呼び出すと、営業日のリクエストには 0 (ゼロ) が返されます。

  たとえば、2005 年の 2 つの日付の差を計算する際に、祝日ファイルの最終日が 20041231 に指定されていると、この計算は実行されません。祝日ファイルを常に有効にする方法の 1 つとして、作成する祝日ファイルのすべてに遠い将来の日付 (例、29991231) を含めます。その結果、この祝日ファイルが常に有効であると見なされます。
- 必要に応じて、祝日の説明を入力することができます。説明は、日付の後にブランクを挿入し、その後に入力します。

デフォルト設定では、祝日ファイルは HDAYxxxx.err という形式のファイル名で、指定されたパス上に格納されます。プロシジャまたはリクエストで、SET HDAY=xxxx コマンドを発行してファイル名を識別する必要があります。また、祝日ファイルに任意の名前を定義し、任意の場所に格納することもできます。

手順　祝日ファイルを作成するには

1. テキストエディタを使用して、祝日として定義する日付リストを作成します。詳細は、316 ページの 「 祝日ファイルの作成規則 」 を参照してください。
2. ファイルを保存します。

   デフォルトの名前規則を使用する場合は、次の手順を使用します。

   Windows および UNIX　ファイル名には「HDAYxxxx.ERR」と入力します。

   説明

   xxxx
   　4 バイトの文字列です。

構文　祝日ファイルの選択

```
SET HDAY = xxxx
```

説明

xxxx
　祝日ファイル名の一部で、HDAY の後ろに指定します。この文字列は、4 バイトで指定する必要があります。

例　祝日ファイルの作成および選択

以下は、祝日を設定する HDAYTEST ファイルです。

```
19910325 TEST HOLIDAY
19911225 CHRISTMAS
```

次のコマンドで、HDAYTEST が祝日ファイルに設定されます。

```
SET BUSDAYS = SMTWTFS
SET HDAY = TEST
```

このリクエストは、HDAYTEST を使用して計算を実行します。

```
TABLE FILE MOVIES
PRINT TITLE RELDATE
COMPUTE NEXTDATE/YMD = DATEADD(RELDATE, 'BD', 1);
WHERE RELDATE GE '19910101';
END
```

出力結果は次のとおりです。

```
TITLE                                    RELDATE   NEXTDATE
-----                                    -------   --------
TOTAL RECALL                             91/03/24  91/03/26
```

ダイアログマネージャの日付時間関数による先頭 0 (ゼロ) の有効化

ダイアログマネージャで整数フォーマットを返す日付時間関数を使用する場合、ダイアログマネージャは先頭の 0 (ゼロ) を切り捨てます。たとえば、関数が値 000101 (2000 年 1 月 1 日) を返す場合、ダイアログマネージャは先頭の 0 (ゼロ) を切り捨てるため、101 という誤った日付が生成されます。この問題を回避するには、LEADZERO パラメータを使用します。

LEADZERO は、関数を直接呼び出す式のみをサポートします。ネスティングやその他の数学関数を持つ式では、常に先頭の 0 (ゼロ) は切り捨てられます。以下はその例です。

```
-SET &OUT = AYM(&IN, 1, 'I4')/100;
```

上記の例では、LEADZERO パラメータの設定に関わらず、先頭の 0 (ゼロ) が切り捨てられます。

構文　先頭文字 0 (ゼロ) の表示設定

```
SET LEADZERO = {ON|OFF}
```

説明

ON
　先頭の 0 (ゼロ) が存在する場合は表示します。

OFF
　先頭の 0 (ゼロ) を切り捨てます。デフォルト値は OFF です。

例　先頭文字 0 (ゼロ) の表示

AYM 関数は、入力された日付 (1999 年 12 月) に 1 か月加算します。

```
-SET &IN = '9912';
-RUN
-SET &OUT = AYM(&IN, 1, 'I4');
-TYPE &OUT
```

LEADZERO のデフォルト値を使用すると、次の値が生成されます。

```
1
```

これは、2000 年 1 月を不正確に表示した値です。LEADZERO パラメータをリクエストに次のように設定します。

```
SET LEADZERO = ON
-SET &IN = '9912';
-SET &OUT = AYM(&IN, 1, 'I4');
-TYPE &OUT
```

その結果、次の値が生成されます。

```
0001
```

これは、2000 年 1 月を正確に表示した値です。

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、完全な日付構成要素に特定の日付単位数 (例、日数、月数、年数) を加算または減算します。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 か月を加算すると、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 日を追加すると、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。

DATEADD 関数が次の営業日または前の営業日を計算する場合、常に営業日を基準に計算を開始します。そのため、実際の日が土曜日または日曜日の場合に、リクエストで次の営業日を計算すると、この関数は土曜日または日曜日の代わりに月曜日を開始日として使用し、次の営業日として火曜日を返します。同様に、前の営業日を計算する際は、開始日として金曜日を使用し、前の営業日として木曜日を返します。DATEADD 関数を使用する前に、DATEMOV 関数を使用することで、日付を正しい営業日に移動することができます。

DATEADD 関数で使用する日付は、日付フォーマットである必要があります。ダイアログマネージャでは日付が文字または数値として解釈され、また DATEADD 関数では基準日からのオフセットとして格納された標準日付を使用する必要があることから、ダイアログマネージャでは DATEADD 関数を使用しないでください。ただし、入力日付として使用する変数を、基準日からのオフセットに事前に変換した場合を除きます。

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, 'component', increment)
```

説明

`date`
日付

年月日を含む完全な日付形式です。

component
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。

increment
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。

注意：DATEADD では output 引数は使用されません。この関数の結果には、date 引数のフォーマットが使用されます。 結果が完全構成要素の日付である限り、完全構成要素の日付フィールドまたは整数フィールドのみに割り当てることができます。

例　DATEADD による切り捨て

DATEADD に渡される単位数は、常に整数です。以下はその例です。

```
DATEADD(DATE, 'M', 1.999)
```

上記の例では、単位数が 2 未満であるため、1 か月加算されます。

例　平日単位の使用

平日単位を使用しており、土曜日または日曜日が入力日付である場合、DATEADD は入力日付を月曜日に変更します。関数は以下のようになります。

```
DATEADD('910623', 'WD', 1)
```

上記の関数では、土曜日または日曜日である DATE は火曜日になります。土曜日と日曜日は平日ではないため、DATEADD は平日の開始日である月曜日に 1 を加算します。

最初の引数を一重引用符で囲むと (‘910623’)、自然言語の日付リテラルとして処理されます。

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。

```
TABLE FILE EMPLOYEE
PRINT FIRST_NAME AND HIRE_DATE AND COMPUTE
NEW_DATE/YYMD = HIRE_DATE;
HIRE_DATE_PLUS_THREE/YYMD = DATEADD(NEW_DATE, 'WD', 3);
BY LAST_NAME
WHERE DEPARTMENT EQ 'MIS';
END
```

出力結果は次のとおりです。

```
LAST_NAME      FIRST_NAME   HIRE_DATE   NEW_DATE    HIRE_DATE_PLUS_THREE
---------      ----------   ---------   --------    --------------------
BLACKWOOD      ROSEMARIE     82/04/01   1982/04/01  1982/04/06
CROSS          BARBARA       81/11/02   1981/11/02  1981/11/05
GREENSPAN      MARY          82/04/01   1982/04/01  1982/04/06
JONES          DIANE         82/05/01   1982/05/01  1982/05/06
MCCOY          JOHN          81/07/01   1981/07/01  1981/07/06
SMITH          MARY          81/07/01   1981/07/01  1981/07/06
```

例　営業日を決定

DATEADD は、営業日単位を使用して、TRANSDATE フィールドに 0 (ゼロ) 日加算することにより、TRANSDATE フィールドの営業日ではない値を決定します。TRANSDATE が営業日ではない場合、DATEADD 関数は DATEX に次の営業日を返します。TRANSDATE が DATEX と比較され、これら 2 つのフィールド間で一致しないすべての日付の曜日が出力されます。その結果、営業日ではない日のリストが生成されます。

```
DEFINE FILE VIDEOTRK
DATEX/YMD  = DATEADD(TRANSDATE, 'BD', 0);
DATEINT/I8YYMD = DATECVT(TRANSDATE, 'YMD','I8YYMD');
END
TABLE FILE VIDEOTRK
SUM TRANSDATE NOPRINT
COMPUTE DAYNAME/A8 = DOWKL(DATEINT, DAYNAME); AS 'Day of Week'
BY TRANSDATE AS 'Date'
WHERE TRANSDATE NE DATEX
END
```

出力結果は次のとおりです。

```
Date      Day of Week
----      -----------
91/06/22  SATURDAY
91/06/23  SUNDAY
91/06/30  SUNDAY
```

DATECVT - 日付フォーマットを変換

DATECVT 関数は、任意の標準日付またはレガシー日付フォーマットのフィールド値を、任意の標準日付またはレガシー日付フォーマットの日付 (基準日からのオフセット) に変換します。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。

DATECVT は、最適化およびコンパイルを無効にします。

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, 'in_format', output)
```

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。

`in_format`
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) からのオフセットとして in_formart で指定した非日付フォーマットです。

output
文字

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。フォーマットが定義されたフィールドを指定することもできます。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)。このフォーマットタイプを使用すると、DATECVT は日付を完全構成要素の日付に変換し、指定されたフォーマットの整数値として返します。

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。

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

または

```
ONV_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。

```
TABLE FILE EMPLOYEE
PRINT FIRST_NAME AND HIRE_DATE AND COMPUTE
NEW_HIRE_DATE/YYMD = DATECVT(HIRE_DATE, 'I6YMD', 'YYMD');
BY LAST_NAME
WHERE DEPARTMENT EQ 'MIS';
END
```

出力結果は次のとおりです。

```
LAST_NAME      FIRST_NAME    HIRE_DATE   NEW_HIRE_DATE
---------      ----------    ---------   -------------
BLACKWOOD      ROSEMARIE     82/04/01    1982/04/01
CROSS          BARBARA       81/11/02    1981/11/02
GREENSPAN      MARY          82/04/01    1982/04/01
JONES          DIANE         82/05/01    1982/05/01
MCCOY          JOHN          81/07/01    1981/07/01
SMITH          MARY          81/07/01    1981/07/01
```

DATEDIF - 2 つの日付の差を計算

DATEDIF 関数は、指定した構成要素単位で、2 つの完全構成要素の標準日付の差を返します。構成要素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 で月構成要素を使用すると、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。
- **日付**
- **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。

ダイアログマネージャでは日付が文字または数値として解釈され、また DATEDIF 関数では基準日からのオフセットとして格納された標準日付を使用する必要があることから、ダイアログマネージャでは DATEDIF 関数を使用しないでください。ただし、入力日付として使用する変数を、基準日からのオフセットに事前に変換した場合を除きます。

DATEDIF は、年月日を含む完全な日付形式でのみ有効です。

構文　2 つの日付の差を計算

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

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。

`component`
　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。

注意：DATEDIF では、結果のフォーマットとして「I8」を使用するため、output 引数は使用されません。

例　DATEDIF による切り捨て

DATEDIF 関数は、1996 年 3 月 2 日と 1997 年 3 月 1 日の差を計算します。この場合、その差が 1 年未満のために 0 (ゼロ) を返します。

```
DATEDIF('19960302', '19970301', 'Y')
```

例　月計算の使用

次の式は、結果としてマイナス 1 か月を返します。

```
DATEDIF('19990228', '19990128', 'M')
DATEDIF('19990228', '19990129', 'M')
DATEDIF('19990228', '19990130', 'M')
DATEDIF('19990228', '19990131', 'M')
```

次の例も参照してください。

`DATEDIF( 'March 31 2001', 'May 31 2001', 'M')` は、2 を返します。

`DATEDIF( 'March 31 2001', 'May 30 2001', 'M')` は、1 を返します (May 30 は月末の日付でないため)。

`DATEDIF( 'March 31 2001', 'April 30 2001', 'M')` は、1 を返します。

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。

```
TABLE FILE EMPLOYEE
PRINT FIRST_NAME AND
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');
BY LAST_NAME
IF WDAYS_HIRED NE 0
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

出力結果は次のとおりです。

```
LAST_NAME      FIRST_NAME  NEW_HIRE_DATE  NEW_DAT_INC  WDAYS_HIRED
---------      ----------  -------------  -----------  -----------
IRVING         JOAN        1982/01/04     1982/05/14   94
MCKNIGHT       ROGER       1982/02/02     1982/05/14   73
SMITH          RICHARD     1982/01/04     1982/05/14   94
STEVENS        ALFRED      1980/06/02     1982/01/01   414
               ALFRED      1980/06/02     1981/01/01   153
```

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。

注意：週のはじめ (BOW) を使用すると、常に月曜日が返され、週の終わり (EOW) を使用すると、常に金曜日が返されます。また、DATEMOV 関数で使用される日付が土曜日または日曜日になる場合、この関数で実際に使用される日付は翌月曜日に移動します。日付を土曜日または日曜日から月曜日に移動した演算を回避したい場合、または BOW を日曜日、EOW を土曜日にする場合は、DTRUNC 関数を使用することができます。

ダイアログマネージャでは日付が文字または数値として解釈され、また DATEMOV 関数では基準日からのオフセットとして格納された標準日付を使用する必要があることから、ダイアログマネージャでは DATEMOV 関数を使用しないでください。ただし、入力日付として使用する変数を、基準日からのオフセットに事前に変換した場合を除きます。たとえば、次のように関数を実行すると、整数のレガシー日付 20050131 が日付フォーマット (SmartDate) に変換され、1 か月が加算された上で、結果が文字のレガシー日付に変換されます。

```
-SET &STRT=DATECVT(20050131,'I8YYMD', 'YYMD');
-SET &NMT=DATEADD(&STRT,'M',1);
-SET &NMTA=DATECVT(&NMT,'YYMD','A8MTDYY');
-TYPE A MONTH FROM 20050131 IS &NMTA
```

出力結果では、DATEADD 関数により、1 月末日から翌月の末日までの日数として、2 月の実際の日数が追加されています。

```
A MONTH FROM 20050131 IS 02282005
```

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

移動する日付です。完全な構成要素フォーマットの日付である必要があります (例、MDYY、YYJUL)。

`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。

DATEADD 関数が次の営業日または前の営業日を計算する場合、常に営業日を基準に計算を開始します。そのため、実際の日が土曜日または日曜日の場合に、リクエストで次の営業日を計算すると、この関数は土曜日または日曜日の代わりに月曜日を開始日として使用し、次の営業日として火曜日を返します。同様に、前の営業日を計算する際は、開始日として金曜日を使用し、前の営業日として木曜日を返します。

営業日のスキップ (開始日の繰り上げまたは繰り下げ) を回避するには、DATEMOV を使用します。次の営業日を返すには、最初に BD- または WD- を使用して、前の営業日に移動します (実際の日がすでに営業日の場合、その日は移動されません)。次に DATEADD を使用して、次の営業日に移動します。前の営業日を返すには、最初に BD+ または WD+ を使用して、次の営業日に移動します (実際の日がすでに営業日の場合、その日は移動されません)。次に DATEADD を使用して、前の営業日に移動します。

注意：DATEMOV では output 引数は使用されません。この関数の結果には、date 引数のフォーマットが使用されます。結果が完全構成要素の日付である限り、完全構成要素の日付フィールドまたは整数フィールドのみに割り当てることができます。

例 次の営業日を取得

この例では、正しい結果を取得するために、DATEMOV 関数を使用する必要性について説明します。

次のリクエストは、GGSALES データソースに対して実行され、DATE フィールドに BD (営業日) 移動点を使用します。最初に DATE が日付フォーマット (SmartDate) に変換され、次に BD 移動点が指定された DATEADD が呼び出されます。

```
DEFINE FILE GGSALES
DT1/WMDYY=DATE;
DT2/WMDYY = DATEADD(DT1 ,'BD',1);
DAY/Dt = DT1;
 END

TABLE FILE GGSALES
SUM  DT1
DT2
BY DT1 NOPRINT
WHERE RECORDLIMIT EQ 10
 END
```

日付が土曜日または日曜日の場合、次の営業日は火曜日として返されます。これは、計算の実行前に元の日付が営業日に移動されたためです。

```
  DT1              DT2
  ---              ---
  SUN, 09/01/1996  TUE, 09/03/1996
  FRI, 11/01/1996  MON, 11/04/1996
  SUN, 12/01/1996  TUE, 12/03/1996
  SAT, 03/01/1997  TUE, 03/04/1997
  TUE, 04/01/1997  WED, 04/02/1997
  THU, 05/01/1997  FRI, 05/02/1997
  SUN, 06/01/1997  TUE, 06/03/1997
  MON, 09/01/1997  TUE, 09/02/1997
  WED, 10/01/1997  THU, 10/02/1997
```

次のリクエストでは、DATEMOV が呼び出され、開始日が営業日に設定されます。この呼び出しで指定された移動点は BD- です。この場合、元の日付が営業日でない場合にのみ、日付が前の営業日に移動されます。次に DATEADD が呼び出され、BD 移動点を使用して次の営業日を返します。

```
DEFINE FILE GGSALES
DT1/WMDYY=DATE;
DT1A/WMDYY=DATEMOV(DT1, 'BD-');
DT2/WMDYY = DATEADD(DT1A,'BD',1);
DAY/Dt = DT1;
 END

TABLE FILE GGSALES
SUM  DT1 DT1A DT2
BY DT1 NOPRINT
WHERE RECORDLIMIT EQ 10
 END
```

出力結果では、土曜日または日曜日の次の営業日として月曜日が返されています。

```
DT1              DT1A             DT2
  ---              ----             ---
  SUN, 09/01/1996  FRI, 08/30/1996  MON, 09/02/1996
  FRI, 11/01/1996  FRI, 11/01/1996  MON, 11/04/1996
  SUN, 12/01/1996  FRI, 11/29/1996  MON, 12/02/1996
  SAT, 03/01/1997  FRI, 02/28/1997  MON, 03/03/1997
  TUE, 04/01/1997  TUE, 04/01/1997  WED, 04/02/1997
  THU, 05/01/1997  THU, 05/01/1997  FRI, 05/02/1997
  SUN, 06/01/1997  FRI, 05/30/1997  MON, 06/02/1997
  MON, 09/01/1997  MON, 09/01/1997  TUE, 09/02/1997
  WED, 10/01/1997  WED, 10/01/1997  THU, 10/02/1997
```

例　DEFINE FUNCTION を使用して週のはじめに日付を移動

次の BOWK という名前の DEFINE FUNCTION は、特定の日付および週のはじめと認識される曜日名を取得し、週のはじめに該当する日付を返します。

```
DEFINE FUNCTION BOWK(THEDATE/MDYY,WEEKSTART/A10)
DAYOFWEEK/W=THEDATE;
DAYNO/I1=IF DAYOFWEEK EQ 7 THEN 0 ELSE DAYOFWEEK;
FIRSTOFWK/I1=DECODE WEEKSTART('SUNDAY' 0 'MONDAY' 1 'TUESDAY' 2
'WEDNESDAY' 3 'THURSDAY' 4 'FRIDAY' 5 'SATURDAY' 6
'SUN' 0 'MON' 1 'TUE' 2 'WED' 3 'THU' 4 'FRI' 5 'SAT' 6);
BOWK/MDYY=IF DAYNO GE FIRSTOFWK THEN THEDATE-DAYNO+FIRSTOFWK
ELSE THEDATE-7-DAYNO+FIRSTOFWK;
END
```

次のリクエストは、BOWK 関数を使用して、DT1 フィールドのそれぞれの値に対して、週のはじめに該当する日付 (DT2) を返します。

```
DEFINE FILE GGSALES
DT1/WMDYY=DATE;
DT2/WMDYY = BOWK(DT1 ,'SUN');
 END

TABLE FILE GGSALES
SUM  DT1
DT2
BY DT1 NOPRINT
WHERE RECORDLIMIT EQ 10
ON TABLE SET PAGE NOLEAD
END
```

下図は、出力結果を示しています。

DT1	DT2
SUN, 09/01/1996	SUN, 09/01/1996
FRI, 11/01/1996	SUN, 10/27/1996
SUN, 12/01/1996	SUN, 12/01/1996
SAT, 03/01/1997	SUN, 02/23/1997
TUE, 04/01/1997	SUN, 03/30/1997
THU, 05/01/1997	SUN, 04/27/1997
SUN, 06/01/1997	SUN, 06/01/1997
MON, 09/01/1997	SUN, 08/31/1997
WED, 10/01/1997	SUN, 09/28/1997

例　　日付を有効な位置に移動

BUSDAYS パラメータは、営業日を月曜日、火曜日、水曜日、および木曜日に設定します。DATECVT 関数は、レガシー日付の HIRE_DATE を日付フォーマット YYMD に変換し、日付表示オプションを指定します。DATEMOV 関数は、HIRE_DATE の有効な移動先を決定します。

```
SET BUSDAY = _MTWT__
TABLE FILE EMPLOYEE
PRINT
COMPUTE NEW_DATE/YYMD = DATECVT(HIRE_DATE, 'I6YMD', 'YYMD'); AND
COMPUTE NEW_DATE/WT = DATECVT(HIRE_DATE, 'I6YMD', 'WT'); AS 'DOW' AND
COMPUTE NWD/WT = DATEMOV(NEW_DATE, 'NWD'); AND
COMPUTE PWD/WT = DATEMOV(NEW_DATE, 'PWD'); AND
COMPUTE WDP/WT = DATEMOV(NEW_DATE, 'WD+'); AS 'WD+' AND
COMPUTE WDM/WT = DATEMOV(NEW_DATE, 'WD-'); AS 'WD-' AND
COMPUTE NBD/WT = DATEMOV(NEW_DATE, 'NBD'); AND
COMPUTE PBD/WT = DATEMOV(NEW_DATE, 'PBD'); AND
COMPUTE WBP/WT = DATEMOV(NEW_DATE, 'BD+'); AS 'BD+' AND
COMPUTE WBM/WT = DATEMOV(NEW_DATE, 'BD-'); AS 'BD-' BY LAST_NAME NOPRINT
HEADING
"Examples of DATEMOV"
"Business days are Monday, Tuesday, Wednesday, + Thursday "
" "
"START DATE.. | MOVE POINTS........................."
WHERE DEPARTMENT EQ 'MIS';
END
```

出力結果は次のとおりです。

```
Examples of DATEMOV
Business days are Monday, Tuesday, Wednesday, + Thursday
START DATE.. | MOVE POINTS.........................
NEW_DATE    DOW  NWD  PWD  WD+  WD-  NBD  PBD  BD+  BD-
--------    ---  ---  ---  ---  ---  ---  ---  ---  ---
1982/04/01  THU  FRI  WED  THU  THU  MON  WED  THU  THU
1981/11/02  MON  TUE  FRI  MON  MON  TUE  THU  MON  MON
1982/04/01  THU  FRI  WED  THU  THU  MON  WED  THU  THU
1982/05/01  SAT  TUE  THU  MON  FRI  TUE  WED  MON  THU
1981/07/01  WED  THU  TUE  WED  WED  THU  TUE  WED  WED
1981/07/01  WED  THU  TUE  WED  WED  THU  TUE  WED  WED
```

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。

```
TABLE FILE EMPLOYEE
PRINT FIRST_NAME AND
COMPUTE NEW_DATE/YYMDWT = DATECVT(HIRE_DATE, 'I6YMD', 'YYMDWT'); AND
COMPUTE EOW/YYMDWT = DATEMOV(NEW_DATE, 'EOW');
BY LAST_NAME
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

出力結果は次のとおりです。

```
LAST_NAME      FIRST_NAME  NEW_DATE           EOW
---------      ----------  --------           ---
BANNING        JOHN        1982 AUG  1, SUN   1982 AUG  6, FRI
IRVING         JOAN        1982 JAN  4, MON   1982 JAN  8, FRI
MCKNIGHT       ROGER       1982 FEB  2, TUE   1982 FEB  5, FRI
ROMANS         ANTHONY     1982 JUL  1, THU   1982 JUL  2, FRI
SMITH          RICHARD     1982 JAN  4, MON   1982 JAN  8, FRI
STEVENS        ALFRED      1980 JUN  2, MON   1980 JUN  6, FRI
```

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, output)
```

説明

`indate`
フォーマットを変換する入力日付 (日付フォーマット) 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 (レガシー日付フォーマット) 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、括弧と一重引用符 (') で囲みます。

下表は、入力構成要素が 1 つの場合を示します。

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)

下表は、入力構成要素が 2 つの場合を示します。

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

下表は、入力構成要素が 3 つの場合を示します。

3 つの入力構成要素	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

3 つの入力構成要素	説明
'(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。フォーマットオプションは、次のいずれかになります。

- 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。

  注意：0 (ゼロ) を非表示にすると、先頭の 0 (ゼロ) がブランクに置換されます。

- 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。

下表は、月または日の数値の先頭の 0 (ゼロ) を非表示にする有効なオプションを示しています。先頭の 0 (ゼロ) はブランクで置換されます。

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。

フォーマットオプション	説明
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。

下表は、有効な月名および曜日名の変換オプションを示しています。

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)。
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。

フォーマットオプション	説明
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。

フォーマットオプション	説明
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)。
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)。

下表は、有効な日付区切り文字オプションを示しています。

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。

フォーマットオプション	説明
|	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。
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
- '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
- ❏ 'NL' オランダ語
- ❏ '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。

`output`
文字

変換後の日付を含むフィールド名、またはフォーマットです。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

- 出力フィールドはタイプ AnV ではなくタイプ A である必要がありますが、結果が可変長タイプを含む可能性もあります。月名および曜日名の長さは可変であり、さらにこれらを数字で表し、0 (ゼロ) を省略するオプションを指定すれば、長さは 1 バイトまたは 2 バイトになります。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。
- これらの NLS 文字を正しく表示するには、WebFOCUS Reporting Server および TIBCO WebFOCUS Client がともに正しいコードページで構成されている必要があります。
- DATETRAN 関数は、ダイアログマネージャでは使用できません。

例　DATETRAN 関数の使用

次のリクエストは、曜日を出力します。大文字と小文字は、指定の言語のデフォルト設定に従います。

```
DEFINE FILE VIDEOTRK
TRANS1/YYMD=20050104;
TRANS2/YYMD=20051003;

DATEW/W=TRANS1     ;
DATEW2/W=TRANS2    ;
DATEYYMD/YYMDW=TRANS1    ;
DATEYYMD2/YYMDW=TRANS2   ;

OUT1A/A8=DATETRAN(DATEW, '(W)', '(wr)', 'EN', 8 , 'A8') ;
OUT1B/A8=DATETRAN(DATEW2, '(W)', '(wr)', 'EN', 8 , 'A8') ;
OUT1C/A8=DATETRAN(DATEW, '(W)', '(wr)', 'ES', 8 , 'A8') ;
OUT1D/A8=DATETRAN(DATEW2, '(W)', '(wr)', 'ES', 8 , 'A8') ;
OUT1E/A8=DATETRAN(DATEW, '(W)', '(wr)', 'FR', 8 , 'A8') ;
OUT1F/A8=DATETRAN(DATEW2, '(W)', '(wr)', 'FR', 8 , 'A8') ;
OUT1G/A8=DATETRAN(DATEW, '(W)', '(wr)', 'DE', 8 , 'A8') ;
OUT1H/A8=DATETRAN(DATEW2, '(W)', '(wr)', 'DE', 8 , 'A8') ;
END

TABLE FILE VIDEOTRK
HEADING
"FORMAT wr"
""
"Full day of week name at beginning of date, default case (wr)"
"English / Spanish / French / German"
""
SUM OUT1A AS '' OUT1B AS '' TRANSDATE NOPRINT
OVER OUT1C AS '' OUT1D AS ''
OVER OUT1E AS '' OUT1F AS ''
OVER OUT1G AS '' OUT1H AS ''
ON TABLE SET PAGE-NUM OFF
ON TABLE SET STYLE *
GRID=OFF, $
END
```

出力結果は次のとおりです。

FORMAT wr

Full day of week name at beginning of date, default case (wr)
English / Spanish / French / German

Tuesday	Monday
martes	lunes
mardi	lundi
Dienstag	Montag

次のリクエストは、ブランクで区切られ、月名が英語の略名で表される日付を出力します。日付先頭の 0 (ゼロ) は非表示になっており、数字の後に接尾語が追加されています。

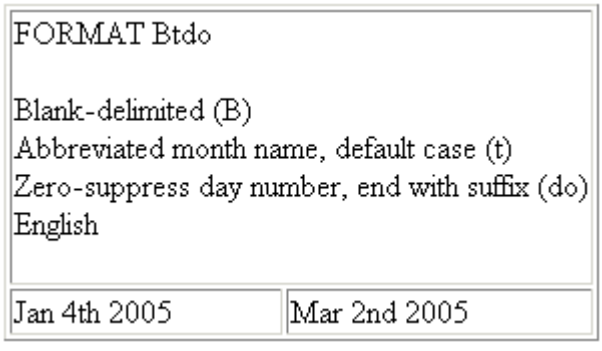
```
DEFINE FILE VIDEOTRK
TRANS1/YYMD=20050104;
TRANS2/YYMD=20050302;

DATEW/W=TRANS1     ;
DATEW2/W=TRANS2    ;
DATEYYMD/YYMDW=TRANS1    ;
DATEYYMD2/YYMDW=TRANS2   ;

OUT2A/A15=DATETRAN(DATEYYMD,  '(MDYY)', '(Btdo)', 'EN', 15, 'A15') ;
OUT2B/A15=DATETRAN(DATEYYMD2, '(MDYY)', '(Btdo)', 'EN', 15, 'A15') ;
END

TABLE FILE VIDEOTRK
HEADING
"FORMAT Btdo"
""
"Blank-delimited (B)"
"Abbreviated month name, default case (t)"
"Zero-suppress day number, end with suffix (do)"
"English"
""
SUM OUT2A AS '' OUT2B AS '' TRANSDATE NOPRINT
ON TABLE SET PAGE-NUM OFF
END
```

出力結果は次のとおりです。



次のリクエストは、ブランクで区切られ、月名がドイツ語の略名で表される日付を出力します。日付先頭の 0 (ゼロ) は非表示になっており、数字の後にピリオド (.) が追加されています。

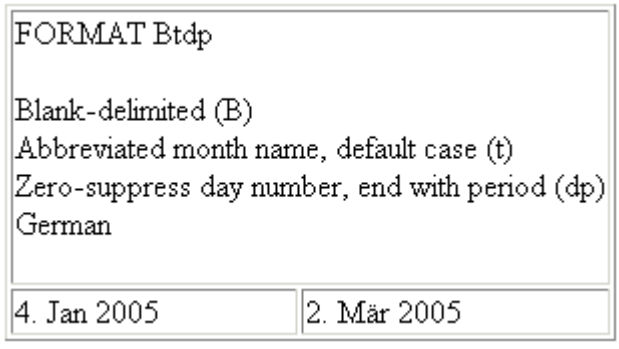
```
DEFINE FILE VIDEOTRK
TRANS1/YYMD=20050104;
TRANS2/YYMD=20050302;

DATEW/W=TRANS1     ;
DATEW2/W=TRANS2    ;
DATEYYMD/YYMDW=TRANS1    ;
DATEYYMD2/YYMDW=TRANS2   ;

OUT3A/A12=DATETRAN(DATEYYMD,  '(DMYY)', '(Btdp)', 'DE', 12, 'A12');
OUT3B/A12=DATETRAN(DATEYYMD2, '(DMYY)', '(Btdp)', 'DE', 12, 'A12');
END

TABLE FILE VIDEOTRK
HEADING
"FORMAT Btdp"
""
"Blank-delimited (B)"
"Abbreviated month name, default case (t)"
"Zero-suppress day number, end with period (dp)"
"German"
""
SUM OUT3A AS '' OUT3B AS '' TRANSDATE NOPRINT
ON TABLE SET PAGE-NUM OFF
END
```

出力結果は次のとおりです。



FORMAT Btdp Blank-delimited (B) Abbreviated month name, default case (t) Zero-suppress day number, end with period (dp) German	
4. Jan 2005	2. Mär 2005

次のリクエストはブランクを区切り文字とした日付を、フランス語の完全な曜日名および月名から (フランス語のデフォルトである) 小文字で表示します。

```
DEFINE FILE VIDEOTRK
TRANS1/YYMD=20050104;
TRANS2/YYMD=20050302;

DATEW/W=TRANS1     ;
DATEW2/W=TRANS2    ;
DATEYYMD/YYMDW=TRANS1    ;
DATEYYMD2/YYMDW=TRANS2   ;

OUT4A/A30 = DATETRAN(DATEYYMD,  '(DMYY)', '(Bwrtr)', 'FR', 30, 'A30');
OUT4B/A30 = DATETRAN(DATEYYMD2, '(DMYY)', '(Bwrtr)', 'FR', 30, 'A30');
END

TABLE FILE VIDEOTRK
HEADING
"FORMAT Bwrtr"
""
"Blank-delimited (B)"
"Full day of week name at beginning of date, default case (wr)"
"Full month name, default case (tr)"
"English"
""
SUM OUT4A AS '' OUT4B AS '' TRANSDATE NOPRINT
ON TABLE SET PAGE-NUM OFF
END
```

出力結果は次のとおりです。

FORMAT Bwrtr Blank-delimited (B) Full day of week name at beginning of date, default case (wr) Full month name, default case (tr) English	
mardi 04 janvier 2005	mercredi 02 mars 2005

次のリクエストは、ブランクで区切られたスペイン語の日付を出力します。日付は小文字の完全な曜日名とカンマ (,) で開始し、日の数値と月名、月名と西暦年の間に「de」が挿入されます。

```
DEFINE FILE VIDEOTRK
TRANS1/YYMD=20050104;
TRANS2/YYMD=20050302;

DATEW/W=TRANS1     ;
DATEW2/W=TRANS2    ;
DATEYYMD/YYMDW=TRANS1    ;
DATEYYMD2/YYMDW=TRANS2   ;

OUT5A/A30=DATETRAN(DATEYYMD,  '(DMYY)', '(Bwrctrde)', 'ES', 30, 'A30');
OUT5B/A30=DATETRAN(DATEYYMD2, '(DMYY)', '(Bwrctrde)', 'ES', 30, 'A30');
END

TABLE FILE VIDEOTRK
HEADING
"FORMAT Bwrctrde"
""
"Blank-delimited (B)"
"Full day of week name at beginning of date, default case (wr)"
"Comma after day name (c)"
"Full month name, default case (tr)"
"Zero-suppress day number (d)"
"de between day and month and between month and year (e)"
"Spanish"
""
SUM OUT5A AS '' OUT5B AS '' TRANSDATE NOPRINT
ON TABLE SET PAGE-NUM OFF
END
```

出力結果は次のとおりです。

FORMAT Bwrctrde	
Blank-delimited (B) Full day of week name at beginning of date, default case (wr) Comma after day name (c) Full month name, default case (tr) Zero-suppress day number (d) de between day and month and between month and year (e) Spanish	
martes, 4 de enero de 2005	miércoles, 2 de marzo de 2005

次のリクエストは日付を日本語の完全な月名をデフォルトの表記で表示します。0 (ゼロ) は非表示です。

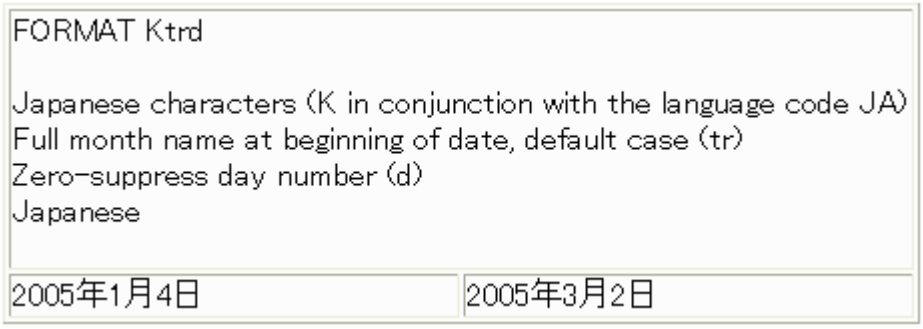
```
DEFINE FILE VIDEOTRK
TRANS1/YYMD=20050104;
TRANS2/YYMD=20050302;

DATEW/W=TRANS1     ;
DATEW2/W=TRANS2    ;
DATEYYMD/YYMDW=TRANS1    ;
DATEYYMD2/YYMDW=TRANS2   ;

OUT6A/A30=DATETRAN(DATEYYMD , '(YYMD)', '(Ktrd)', 'JA', 30, 'A30');
OUT6B/A30=DATETRAN(DATEYYMD2, '(YYMD)', '(Ktrd)', 'JA', 30, 'A30');
END

TABLE FILE VIDEOTRK
HEADING
"FORMAT Ktrd"
""
"Japanese characters (K in conjunction with the language code JA)"
"Full month name at beginning of date, default case (tr)"
"Zero-suppress day number (d)"
"Japanese"
""
SUM OUT6A AS '' OUT6B AS '' TRANSDATE NOPRINT
ON TABLE SET PAGE-NUM OFF
END
```

出力結果は次のとおりです。



DPART - 日付から構成要素を抽出

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。

ダイアログマネージャでは日付が文字または数値として解釈され、また DPART 関数では基準日からのオフセットとして格納された標準日付を使用する必要があることから、ダイアログマネージャでは DPART 関数を使用しないでください。ただし、入力日付として使用する変数を、基準日からのオフセットに事前に変換した場合を除きます。

構文　　日付構成要素を抽出して整数を取得

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

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、DD

平日 - WEEKDAY、WW

四半期 - QUARTER、QQ

`output`
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

次のリクエストでは、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、日の構成要素を抽出します。

```
DEFINE FILE
 VIDEOTRK
 YEAR/I4 = DPART(TRANSDATE, 'YEAR', 'I11');
 MONTH/I4 = DPART(TRANSDATE, 'MM', 'I11');
 DAY/I4 = DPART(TRANSDATE, 'DAY', 'I11');
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
```

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。

ダイアログマネージャでは日付が文字または数値として解釈され、また FIQTR 関数では基準日からのオフセットとして格納された標準日付を使用する必要があることから、ダイアログマネージャでは FIQTR 関数を使用しないでください。ただし、入力日付として使用する変数を、基準日からのオフセットに事前に変換した場合を除きます。

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

次のリクエストは、CENTHR データソースに対して実行され、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。

```
DEFINE FILE CENTHR
FISCALQ/Q=FIQTR(START_DATE,'D',10,1,'FYE',FISCALQ);
FISCALI/I1=FIQTR(START_DATE,'D',10,1,'FYE',FISCALI);
END
TABLE FILE CENTHR
PRINT START_DATE FISCALQ FISCALI
BY LNAME BY FNAME
WHERE LNAME LIKE 'C%'
END
```

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
```

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。

ダイアログマネージャでは日付が文字または数値として解釈され、また FIYR 関数では基準日からのオフセットとして格納された標準日付を使用する必要があることから、ダイアログマネージャでは FIYR 関数を使用しないでください。ただし、入力日付として使用する変数を、基準日からのオフセットに事前に変換した場合を除きます。

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

次のリクエストは、CENTSTMT データソースに対して実行され、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。

```
DEFINE FILE CENTSTMT
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
TABLE FILE CENTSTMT
PRINT PERIOD FISCALYY FISCALY FISCALI
BY GL_ACCOUNT
WHERE GL_ACCOUNT LT '2100'
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
```

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。

ダイアログマネージャでは日付が文字または数値として解釈され、また FIYYQ 関数では基準日からのオフセットとして格納された標準日付を使用する必要があることから、ダイアログマネージャでは FIYYQ 関数を使用しないでください。ただし、入力日付として使用する変数を、基準日からのオフセットに事前に変換した場合を除きます。

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

次のリクエストは、CENTHR データソースに対して実行され、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。

```
DEFINE FILE CENTHR
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);
END
TABLE FILE CENTHR
PRINT START_DATE FISYQ FISYYQ FISQY FISQYY
BY LNAME BY FNAME
WHERE LNAME LIKE 'C%'
END
```

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
```

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。

ダイアログマネージャのシステム変数 &DATE を使用しても日付を同一フォーマット (スラッシュで区切られたフォーマット) で取得することができます。システム変数 &YMD、&MDY、&DMY を使用すると、スラッシュ記号 (/) のない日付を取得することができます。システム変数 &DATE*fmt* は、指定したフォーマットの日付を取得します。

構文　現在の日付を取得

```
TODAY(output)
```

説明

`output`
文字 (A8 以上)

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。

以下のように適用します。

- 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。

- フォーマットが A10 以上の場合、TODAY は 4 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。次のリクエストは、日付をページの見出しに表示します。

```
DEFINE FILE EMPLOYEE
DATE/A10 WITH EMP_ID = TODAY(DATE);
END


TABLE FILE EMPLOYEE
SUM CURR_SAL BY DEPARTMENT
HEADING
"PAGE <TABPAGENO  "
"SALARY REPORT RUN ON <DATE  "
END
```

出力結果は次のとおりです。

```
SALARY REPORT RUN ON 12/13/2006
DEPARTMENT        CURR_SAL
----------      --------
MIS            $108,002.00
PRODUCTION     $114,282.00
```

レガシー日付関数

レガシー日付関数は、整数、パック 10 進数、または文字フォーマットの日付とともに使用します。

各レガシー日付関数についての詳細は、次のトピックを参照してください。

360 ページの 「 AYM - 月数の加算または減算 」

361 ページの 「 AYMD - 日数の加算または減算 」

363 ページの 「 CHGDAT - 日付文字列の表示を変更 」

366 ページの 「 DMY、MDY、YMD - 2 つの日付の差を計算 」

367 ページの 「 DOWK および DOWKL - 曜日を検索 」

368 ページの 「 GREGDT - ユリウス暦から太陽暦フォーマットに変換 」

370 ページの 「 JULDAT - 太陽暦からユリウス暦フォーマットに変換 」

371 ページの 「 YM - 経過月数を計算 」

旧バージョンのレガシー日付関数

ここではレガシー日付関数について説明します。これらは整数または文字フォーマットの日付とともに使用します。日付操作のために使用することはお勧めしません。日付操作には、標準の日付関数および日付時間関数を使用してください。

レガシー日付関数はすべて、2000 年以降の日付をサポートします。

年が 2 桁または 4 桁の日付の使用

レガシー日付関数は、2 桁の年と 4 桁の年のいずれの日付にも対応します。2000 や 1900 など、世紀を表示する 4 桁の年は、フォーマットに I8YYMD、P8YYMD、D8YYMD、F8YYMD、A8YYMD が指定されている場合に使用できます。フィールドの長さが 6 バイトの場合、2 桁の年は、DEFCENT および YRTHRESH パラメータを使用して、世紀の値を割り当てることができます。これらのパラメータについての詳細は、『ibi™ WebFOCUS® アプリケーション作成ガイド』の「環境のカスタマイズ」を参照してください。

例　4 桁の年の使用

EDIT 関数は、年が 4 桁の日付を作成します。次に、JULDAT および GREGDT 関数が、これらの日付をユリウス暦および太陽暦フォーマットに変換します。

```
DEFINE FILE EMPLOYEE
DATE/I8YYMD = EDIT('19'|EDIT(HIRE_DATE));
JDATE/I7 = JULDAT(DATE, 'I7');
GDATE/I8 = GREGDT(JDATE, 'I8');
END
TABLE FILE EMPLOYEE
PRINT DATE JDATE GDATE
END
```

出力結果は次のとおりです。

```
      DATE     JDATE     GDATE
      ----     -----     -----
1980/06/02  1980154  19800602
1981/07/01  1981182  19810701
1982/05/01  1982121  19820501
1982/01/04  1982004  19820104
1982/08/01  1982213  19820801
1982/01/04  1982004  19820104
1982/07/01  1982182  19820701
1981/07/01  1981182  19810701
1982/04/01  1982091  19820401
1982/02/02  1982033  19820202
1982/04/01  1982091  19820401
1981/11/02  1981306  19811102
1982/04/01  1982091  19820401
1982/05/15  1982135  19820515
```

例　2 桁の年の使用

AYMD 関数は、入力引数のレガシー日付フォーマットが 6 桁の場合、8 桁の日付を返します。DEFCENT は 19 で、YRTHRESH が 83 であるため、83 から 99 までの年値は 1983 から 1999 まで、00 から 82 までの年値は 2000 から 2082 までとして解釈されます。

```
SET DEFCENT=19, YRTHRESH=83

DEFINE FILE EMPLOYEE
NEW_DATE/I8YYMD = AYMD(EFFECT_DATE, 30, 'I8');
END

TABLE FILE EMPLOYEE
PRINT EFFECT_DATE NEW_DATE BY EMP_ID
END
```

出力結果は次のとおりです。

```
EMP_ID      EFFECT_DATE    NEW_DATE
------      -----------    --------
071382660
112847612
117593129      82/11/01   2082/12/01
119265415
119329144      83/01/01   1983/01/31
123764317      83/03/01   1983/03/31
126724188
219984371
326179357      82/12/01   2082/12/31
451123478      84/09/01   1984/10/01
543729165
818692173      83/05/01   1983/05/31
```

AYM - 月数の加算または減算

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, output)
```

説明

`indate`

I4、I4YM、I6、または I6YYM

年月フォーマットのレガシー日付です。日付を含むフィールド名、または日付を返す式を指定することもできます。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。

`output`
I4YM または I6YYM

結果のレガシー日付です。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。

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。その結果、日付が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。

```
TABLE FILE EMPLOYEE
PRINT HIRE_DATE AND COMPUTE
HIRE_MONTH/I4YM = HIRE_DATE/100 ;
AFTER6MONTHS/I4YM = AYM(HIRE_MONTH, 6, AFTER6MONTHS);
BY LAST_NAME BY FIRST_NAME
WHERE DEPARTMENT EQ 'MIS';
END
```

出力結果は次のとおりです。

```
LAST_NAME       FIRST_NAME  HIRE_DATE  HIRE_MONTH  AFTER6MONTHS
---------       ----------  ---------  ----------  ------------
BLACKWOOD       ROSEMARIE    82/04/01       82/04         82/10
CROSS           BARBARA      81/11/02       81/11         82/05
GREENSPAN       MARY         82/04/01       82/04         82/10
JONES           DIANE        82/05/01       82/05         82/11
MCCOY           JOHN         81/07/01       81/07         82/01
SMITH           MARY         81/07/01       81/07         82/01
```

AYMD - 日数の加算または減算

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。

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, output)
```

説明

`indate`
I6、I6YMD、I8、I8YYMD

年月日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。

`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。

`output`
I6、I6YMD、I8、I8YYMD

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。indate にフィールドを指定した場合、output にも同一のフォーマットを指定する必要があります。

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。

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。

```
TABLE FILE EMPLOYEE
PRINT HIRE_DATE AND COMPUTE
AFTER35DAYS/I6YMD = AYMD(HIRE_DATE, 35, AFTER35DAYS);
BY LAST_NAME BY FIRST_NAME
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

出力結果は次のとおりです。

```
LAST_NAME       FIRST_NAME  HIRE_DATE  AFTER35DAYS
---------       ----------  ---------  -----------
BANNING         JOHN         82/08/01     82/09/05
IRVING          JOAN         82/01/04     82/02/08
MCKNIGHT        ROGER        82/02/02     82/03/09
ROMANS          ANTHONY      82/07/01     82/08/05
SMITH           RICHARD      82/01/04     82/02/08
STEVENS         ALFRED       80/06/02     80/07/07
```

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。入力および出力の日付文字列を記述するには、日付文字列内の日付構成要素 (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、04 または 2004) を指定する表示オプションを使用します。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。標準日付とレガシー日付のどちらを入力する場合も、CHGDAT を適用する前に、EDIT 関数や DATECVT 関数などを使用して、日付文字列に変換します。

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)	世紀は DEFCENT および YRTHRESH で定義する 100 年で決定されます。 DEFCENT および YRTHRESH についての詳細は、『ibi™ WebFOCUS® アプリケーション作成ガイド』の「環境のカスタマイズ」を参照してください。

構文　日付文字列の表示を変更

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

説明

`in_display_options`
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`
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`
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。

`output`
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。

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。

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。

```
TABLE FILE EMPLOYEE
PRINT HIRE_DATE AND COMPUTE
ALPHA_HIRE/A17 = EDIT(HIRE_DATE); NOPRINT AND COMPUTE
HIRE_MDY/A17 = CHGDAT('YMD', 'MDYYX', ALPHA_HIRE, 'A17');
BY LAST_NAME BY FIRST_NAME
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

出力結果は次のとおりです。

```
LAST_NAME      FIRST_NAME  HIRE_DATE  HIRE_MDY
---------      ----------  ---------  --------
BANNING        JOHN         82/08/01  AUGUST 01 1982
IRVING         JOAN         82/01/04  JANUARY 04 1982
MCKNIGHT       ROGER        82/02/02  FEBRUARY 02 1982
ROMANS         ANTHONY      82/07/01  JULY 01 1982
SMITH          RICHARD      82/01/04  JANUARY 04 1982
STEVENS        ALFRED       80/06/02  JUNE 02 1980
```

DMY、MDY、YMD - 2 つの日付の差を計算

DMY、MDY、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 つのレガシー日付の差を計算します。

構文　2 つの日付の差を計算

```
function(from_date, to_date)
```

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。

`from_date`
日付表示オプションを含む I、P、A フォーマットです。

レガシー日付の開始日です。日付を含むフィールド名を指定することもできます。

to_date
日付表示オプションを含む I、P、A フォーマットです。I6xxx または I8xxx です。xxx は、指定した関数のフォーマット (DMY、YMD、MDY) に対応します。

終了日です。日付を含むフィールドの名前を指定することもできます。

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。

```
TABLE FILE EMPLOYEE
SUM HIRE_DATE FST.DAT_INC AS 'FIRST PAY,INCREASE' AND COMPUTE
DIFF/I4 = YMD(HIRE_DATE, FST.DAT_INC); AS 'DAYS,BETWEEN'
BY LAST_NAME BY FIRST_NAME
WHERE DEPARTMENT EQ 'MIS';
END
```

出力結果は次のとおりです。

```
LAST_NAME       FIRST_NAME      HIRE_DATE       FIRST           DAYS
                                                PAY INCREASE    BETWEEN

BLACKWOOD       ROSEMARIE       82/04/01        82/04/01        0
CROSS           BARBARA         81/11/02        82/04/09        158
GREENSPAN       MARY            82/04/01        82/06/11        71
JONES           DIANE           82/05/01        82/06/01        31
MCCOY           JOHN            81/07/01        82/01/01        184
SMITH           MARY            81/07/01        82/01/01        184
```

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, output)
```

説明

indate
I6YMD または I8YYMD

年月日フォーマットのレガシー日付です。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。

output
DOWK - A4DOWKL - A12

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。

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。

```
TABLE FILE EMPLOYEE
PRINT EMP_ID AND HIRE_DATE AND COMPUTE
DATED/A4 = DOWK(HIRE_DATE, DATED);
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

出力結果は次のとおりです。

```
EMP_ID      HIRE_DATE  DATED
------      ---------  -----
071382660    80/06/02  MON
119265415    82/01/04  MON
119329144    82/08/01  SUN
123764317    82/01/04  MON
126724188    82/07/01  THU
451123478    82/02/02  TUE
```

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 (年日) 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 または 1999001 であり、2004 年 6 月 21 日は 2004173 です。

参照 GREGDT のフォーマットオプション

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。

- フォーマットが I6 または I7 の場合、GREGDT は YMD フォーマットで日付を返します。
- フォーマットが I8 以上の場合、GREGDT は YYMD フォーマットで日付を返します。

構文　ユリウス暦から太陽暦フォーマットに変換

```
GREGDT(indate, output)
```

説明

`indate`
I5 または I7

ユリウス暦の日付です。変換前に日付の末尾が切り捨てられて整数になります。切り捨て後の値が 5 桁または 7 桁の数値になる必要があります。日付が有効でない場合、この関数は 0 (ゼロ) を返します。

`output`
I6、I8、I6YMD、I8YYMD

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。

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。世紀は、デフォルトの DEFCENT および YRTHRESH パラメータ設定により決定されます。

```
TABLE FILE EMPLOYEE
PRINT HIRE_DATE AND
COMPUTE JULIAN/I5 = JULDAT(HIRE_DATE, JULIAN); AND
COMPUTE GREG_DATE/I8 = GREGDT(JULIAN, 'I8');
BY LAST_NAME BY FIRST_NAME
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

出力結果は次のとおりです。

```
LAST_NAME      FIRST_NAME  HIRE_DATE  JULIAN  GREG_DATE
---------      ----------  ---------  ------  ---------
BANNING        JOHN         82/08/01   82213   19820801
IRVING         JOAN         82/01/04   82004   19820104
MCKNIGHT       ROGER        82/02/02   82033   19820202
ROMANS         ANTHONY      82/07/01   82182   19820701
SMITH          RICHARD      82/01/04   82004   19820104
STEVENS        ALFRED       80/06/02   80154   19800602
```

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 のフォーマット設定

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。

- フォーマットが I6 の場合、JULDAT は YYNNN フォーマットで日付を返します。
- フォーマットが I7 以上の場合、JULDAT は YYYYNNN フォーマットで日付を返します。

構文　太陽暦からユリウス暦フォーマットに変換

```
JULDAT(indate, output)
```

説明

`indate`

I6、I8、I6YMD、I8YYMD

変換するレガシー日付です。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を指定することもできます。

`output`

I5 または I7

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。

例　太陽暦からユリウス暦フォーマットに変換

JULDAT 関数は、HIRE_DATE フィールドをユリウス暦フォーマットに変換します。世紀は、デフォルトの DEFCENT および YRTHRESH パラメータの設定により決定されます。

```
TABLE FILE EMPLOYEE
PRINT HIRE_DATE AND COMPUTE
JULIAN/I7 = JULDAT(HIRE_DATE, JULIAN);
BY LAST_NAME BY FIRST_NAME
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

出力結果は次のとおりです。

```
LAST_NAME      FIRST_NAME  HIRE_DATE    JULIAN
---------      ----------  ---------    ------
BANNING        JOHN         82/08/01  1982213
IRVING         JOAN         82/01/04  1982004
MCKNIGHT       ROGER        82/02/02  1982033
ROMANS         ANTHONY      82/07/01  1982182
SMITH          RICHARD      82/01/04  1982004
STEVENS        ALFRED       80/06/02  1980154
```

YM - 経過月数を計算

YM 関数は、2 つの日付間の月数を計算します。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, output)
```

説明

`fromdate`
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。

`to_date`
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。

output
整数

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。

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。

```
TABLE FILE EMPLOYEE
PRINT DAT_INC AS 'RAISE DATE' AND COMPUTE
HIRE_MONTH/I4YM = HIRE_DATE/100; NOPRINT AND COMPUTE
MONTH_INC/I4YM = DAT_INC/100; NOPRINT AND COMPUTE
MONTHS_HIRED/I3 = YM(HIRE_MONTH, MONTH_INC, 'I3');
BY LAST_NAME BY FIRST_NAME BY HIRE_DATE
IF MONTHS_HIRED NE 0
WHERE DEPARTMENT EQ 'MIS';
END
```

出力結果は次のとおりです。

```
LAST_NAME     FIRST_NAME  HIRE_DATE  RAISE DATE  MONTHS_HIRED
---------     ----------  ---------  ----------  ------------
CROSS         BARBARA      81/11/02    82/04/09             5
GREENSPAN     MARY         82/04/01    82/06/11             2
JONES         DIANE        82/05/01    82/06/01             1
MCCOY         JOHN         81/07/01    82/01/01             6
SMITH         MARY         81/07/01    82/01/01             6
```

12

日付時間関数

日付時間関数は、「H フォーマット」とも呼ばれる日付時間フォーマットのタイムスタンプを操作します。タイムスタンプ値は、最小でナノ秒までの日付時間構成要素を保持することが可能な内部格納データです。

トピックス

日付時間関数の使用

ここでは、日付時間フォーマット (H フォーマットとも呼ばれる) のフィールドを操作する関数について説明します。

日付時間パラメータ

DATEFORMAT パラメータは、特定のタイプの日付時間値での、日付構成要素の順序を指定します。WEEKFIRST パラメータは、週の開始日を指定します。DTSTRICT パラメータは、日付時間値の有効性を確認するために適用する制限を指定します。

日付構成要素の順序を指定

DATEFORMAT パラメータは、日付時間値がフォーマット済み文字列または変換済み文字列フォーマットで入力された場合の、日付構成要素 (月/日/年) の順序を指定します。これらのフォーマットについての詳細は、379 ページの 「 日付時間フォーマットの使用 」 を参照してください。このパラメータを使用すると、値の入力フォーマットが、その値に適用される変数のフォーマットに依存しなくなります。

構文　日付時間フィールドの日付構成要素の順序を指定

```
SET DATEFORMAT = option
```

説明

`option`

MDY、DMY、YMD、MYD のいずれかを指定することができます。U.S. English フォーマットでは、デフォルト値は MDY です。

例　DATEFORMAT パラメータの使用

次のリクエストは、HINPUT 関数への入力として、不明確な数値の日付構成要素 (APR 04 05) を用いる自然言語の日付リテラルを使用します。

```
SET DATEFORMAT = MYD
DEFINE FILE EMPLOYEE
DTFLDYYMD/HYYMDI =  HINPUT(9,'APR 04 05', 8, DTFLDYYMD);
END
TABLE FILE EMPLOYEE
SUM   CURR_SAL NOPRINT DTFLDYYMD
END
```

DATEFORMAT を MYD に設定すると、この値は「April 1904,5」と解釈されます。

```
DTFLDYYMD
---------
1904-04-05 00:00
```

日付時間関数で使用する週の開始日の指定

WEEKFIRST パラメータは、曜日のいずれかを週の開始日に指定します。この機能は、HADD、HDIFF、HNAME、HPART、HYYWD 関数による週の演算で使用します。この機能は、DTADD、DTDIFF、DTRUNC、DTPART 関数でも使用されます。これらの関数では、デフォルト値が関数ごとに異なります。詳細は、375 ページの 「 週の開始日となる日付の設定 」 を参照してください。WEEKFIRST パラメータは、週の開始日を指定しますが、それぞれの曜日に対応する日付は変更されません。

HPART、DTPART、HYYWD、HNAME サブルーチンは、日付時間値から週番号を抽出することができます。これらのサブルーチンでは、週番号を特定するために、さまざまな定義を使用することができます。たとえば、ISO 8601 標準の週番号では、年の第 1 週は、4 日以上の日付がある 1 月の最初の週として定義されています。1 月の最初の週の前に日付がある場合、前年の第 52 週または第 53 週に属することになります。また、ISO 標準では月曜日を週の最初の曜日と定めています。

使用する週番号タイプを指定するには、WEEKFIRST パラメータを設定します。詳細は、375 ページの 「 週の開始日となる日付の設定 」 を参照してください。

HNAME、DTPART、HPART 関数によって返される週番号は現在の年またはその前後の年である可能性があるため、週番号自体は役に立たない場合があります。HYYWD 関数は、指定した日付時間値から年と週番号の両方を返します。

構文　週の開始日となる日付の設定

```
SET WEEKFIRST = value
```

説明

`value`

次のいずれかの値を指定します。

- **1 から 7**　標準外の週番号を使用して、日曜日から土曜日を表します。

  これらの値を使用する週番号では、7 日分の日付がある 1 月の最初の週が週番号 1 に設定されます。1 月の最初の週の前に日付がある場合、それらの日付は前年の最後の週に属します。すべての週には 7 日分の日付があります。

- **ISO1 から ISO7**　ISO 標準の週番号を使用して、日曜日から土曜日を表します。

  注意：ISO は ISO2 と同義です。

これらの値を使用する週番号では、4 日以上の日付がある 1 月の最初の週が週番号 1 に設定されます。1 月の最初の週の前に日付がある場合、それらの日付は前年の最後の週に属します。すべての週には 7 日分の日付があります。

- **STD1 から STD7**　数字の 1 (日曜日) から 7 (土曜日) を使用して、週の開始日を指定します。

  注意：数字を含めずに「STD」と指定すると、STD1 と同等になります。

  これらの値を使用する週番号では、週番号 1 は 1 月 1 日に始まり、翌週の開始日の前日に終了します。たとえば、STD1 の場合、第 1 週は年の最初の土曜日に終了します。最初の週および最後の週は、日数が 7 日未満になる場合があります。

- **SIMPLE**　1 月 1 日を第 1 週の開始日、1 月 8 日を第 2 週の開始日 (以降同様) に設定します。週の開始日は、年の開始日に一致します。最後の週 (第 53 週) の日数は、1 日または 2 日になります。

- **0 (ゼロ)**　ユーザが明示的な WEEKFIRST 設定を発行する前の WEEKFIRST 設定値です。日付時間関数の HPART、HNAME、HYYWD、HADD、HDIFF では、WEEKFIRST 設定が 0 (ゼロ) の場合、週の開始日として土曜日が使用されます。簡略関数の DTADD、DTDIFF、DTRUNC、DTPART のほか、週数に切り捨てられた日付の出力や、週番号を含む日付定数文字列の認識では、WEEKFIRST 設定が 0 (ゼロ) の場合、デフォルト値として日曜日が使用されます。ユーザが WEEKFIRST を明示的に別の値に設定した場合、その値がすべての関数で使用されます。

例　週の開始日に日曜日を設定

次の設定は、標準外の週番号を使用して、週の開始日を日曜日に指定します。

```
SET WEEKFIRST = 1
```

構文　現在の WEEKFIRST 設定の表示

```
? SET WEEKFIRST
```

このコマンドを使用すると、週番号アルゴリズムと週の開始日を示す値が返されます。たとえば、整数の 1 は、標準外の週番号で日曜日を表します。

日付時間値の処理の制御

厳密処理は、エンドユーザの入力、トランザクションからの読み取り、表示、サブルーチンからの取得時に、日付時間値をチェックし、これらが有効な日付時間であることを確認します。たとえば、数字の月は 1 から 12 の間で、日は特定の月の日数内でなければなりません。

構文　日付時間値の厳密処理の有効化

```
SET DTSTRICT = {ON|OFF}
```

説明

ON

厳密処理を呼び出します。デフォルト値は ON です。

厳密処理は、エンドユーザの入力、トランザクションからの読み取り、表示、サブルーチンからの取得時に、日付時間値をチェックし、これらが有効な日付時間であることを確認します。たとえば、数字の月は 1 から 12 の間で、日は特定の月の日数内でなければなりません。

DTSTRICT が ON に設定され、結果の日付時間値が無効な場合、関数は値 0 (ゼロ) を返します。

OFF

厳密処理を呼び出しません。日付時間構成要素には、フィールドで使用可能な桁数の制限値内であれば、任意の値を指定することができます。たとえば、フィールドが 2 桁の月の場合、値 12 および 99 は有効ですが、115 は無効です。

日付時間関数の引数の指定

日付時間関数を使用して、日付時間値の構成要素を操作することができます。ここでは、これらの関数で使用可能な有効な構成要素名および短縮名について説明します。

参照　日付時間関数での引数の使用

次の構成要素名、有効な略名、値は、引数を必要とする日付時間関数の引数としてサポートされます。

構成要素名	Abbreviation	有効値
year	yy	0001 から 9999

構成要素名	Abbreviation	有効値
quarter	qq	1 から 4
month	mm	1 から 12、または月名。関数により異なります。
day-of-year	dy	1 から 366
day or day-of-month	dd	1 から 31 (2 つの構成要素名は同等)
week	wk	1 から 53
weekday	dw	1 から 7 (日曜から土曜)
hour	hh	0 から 23
minute	mi	0 から 59
second	ss	0 から 59
millisecond	ms	0 から 999
microsecond	mc	0 から 999999
nanosecond	ns	0 から 999999999

注意

- 8 バイト長、10 バイト長または 12 バイト長を指定する引数では、8 バイトを指定するとミリ秒、10 バイトを指定するとマイクロ秒、12 バイトを指定するとナノ秒が戻り値に含められます。
- 最後の引数は、常に USAGE フォーマットです。これは関数により返されるデータタイプを示します。A (文字)、I (整数)、D (倍精度浮動小数点数)、H (日付時間)、標準日付フォーマット (YYMD など) のいずれかです。

日付時間フォーマットの使用

有効な日付時間フォーマットには、数値文字列フォーマット、フォーマット済み文字列フォーマット、および変換済み文字列フォーマットがあります。これらのフォーマットでは、2 桁の西暦は DEFCENT および YRTHRESH パラメータにより解釈されます。

時間構成要素は、コロン (:) で区切られ、午前と午後を区別する A.M.、P.M.、a.m.、p.m. のいずれかを末尾に追加することもできます。

DATEFORMAT パラメータは、日時の値がフォーマット済みまたは変換済み文字列フォーマットで入力された場合に、日付要素 (月/日/年) の順序を指定します。このパラメータを使用すると、値の入力フォーマットが、その値に適用される変数のフォーマットに依存しなくなります。

数値文字列フォーマット

数値文字列フォーマットは、2 桁、4 桁、6 桁、8 桁のいずれかです。4 桁の文字列は西暦年 (世紀の指定必須) であると見なされ、月日は 1 月 1 日に設定されます。6 桁と 8 桁の文字列には、それぞれ 2 桁または 4 桁の西暦年が含まれ、その後ろに 2 桁の月と 2 桁の日が指定されます。このフォーマットでは構成要素の順序が固定されているため、DATEFORMAT 設定は無視されます。

9 桁以上の数値文字列フォーマットは、日付時間を組み合わせた文字列である H*nn* フォーマットとして処理されます。

例　数値文字列フォーマットの使用

数列日付定数の例は次のとおりです。

文字列	日付
`99`	January 1, 1999
`1999`	January 1, 1999
`19990201`	February 1, 1999

フォーマット済み文字列フォーマット

フォーマット済み文字列フォーマットには、1 桁または 2 桁の日、1 桁または 2 桁の月、および 2 桁または 4 桁の西暦年が含まれます。各構成要素は、ブランク、スラッシュ (/)、ハイフン (-)、ピリオド (.) のいずれかの区切り文字で区切られます。3 つの構成要素がすべて存在し、DATEFORMAT 設定に従う必要があります。3 つのフィールドの中で 4 桁のものは年と認識され、他の 2 つのフィールドは DATEFORMAT 設定で指定された順序に従う必要があります。

例　フォーマット済み文字列フォーマットの使用

以下は、フォーマット済み文字列日付定数の例です。これらの値は、いずれも 1999 年 5 月 20 日を示します。

```
1999/05/20
5 20 1999
99.05.20
1999-05-20
```

変換済み文字列フォーマット

変換済み文字列フォーマットには、月の完全な名前または略名が含まれています。年は、4 桁または 2 桁のフォーマットで表示する必要があります。日の値が欠落している場合は、月の 1 日目と見なされます。日の値は、1 桁または 2 桁にすることができます。文字列に 2 桁の年と 2 桁の日の両方が含まれる場合は、DATEFORMAT で設定された順序に従う必要があります。

例　変換済み文字列フォーマットの使用

次の日付は、変換済み文字列フォーマットで示されています。

```
January 6 2000
```

時間フォーマット

時間構成要素は、コロン (:) で区切られ、午前と午後を区別する A.M.、P.M.、a.m.、p.m. のいずれかを末尾に追加することもできます。

秒は、小数点で表すことができます。また、コロン (:) を末尾に追加することもできます。コロン (:) が秒の末尾に追加される場合、その次の値はミリ秒を表します。この表記法では、マイクロ秒やナノ秒を表すことはできません。

秒値の小数点以下第 1 位の値は、1/10 秒を表します。マイクロ秒は、6 桁で表すことができます。ナノ秒は、9 桁で表すことができます。

例　時間フォーマットの使用

以下は、利用可能な時間フォーマットの例です。

```
14:30:20:99       (99 milliseconds)
14:30
14:30:20.99       (99/100 seconds)
14:30:20.999999   (999999 microseconds)
02:30:20:500pm
```

例　標準日付時間入力値の使用

STANDARD および STANDARDU の DTSTANDARD 設定を使用して、次の日付時間値を入力値として読み取ることができます。

入力値	説明
`14:30[:20,99]`	時間要素を区切る場合は、ピリオド (.) の代わりにカンマ (,) を使用します。
`14:30[:20.99]Z`	世界標準時です。

`15:30[:20,99]+01`
`15:30[:20,99]+0100`
`15:30[:20,99]+01:00`
	中央ヨーロッパ標準時において、これらの入力値はすべて上記の値と等しくなります。

`09:30[:20.99]-05`	東部標準時において、この入力値は上記の値と等しくなります。

なお、これらの値は内部的に STANDARDU 設定と同様に格納されます。STANDARD 設定を使用する場合、Z、+、- の後に来るものはすべて無視されます。

日付時間値の割り当て

日付時間値は、次のいずれかにより割り当てられた文字フォーマットの定数です。

- シーケンシャルデータソース
- WHERE 条件または IF 条件を定義する式、あるいは DEFINE または COMPUTE コマンドで一時項目を作成する式

日付時間定数では、先頭または末尾、あるいは am/pm 標識の直前にブランクを配置できます。

構文　日付時間値の割り当て

文字ファイル

```
date_string [time_string]
```

または

```
time_string [date_string]
```

COMPUTE、DEFINE、または WHERE 式

```
DT(date_string [time_string])
```

または

```
DT(time_string [date_string])
```

IF 条件式

```
'date_string [time_string]'
```

または

```
'time_string [date_string]'
```

説明

`time_string`

利用可能なフォーマットの時間文字列です。時間文字列では、am/pm 標識の直前にブランクを配置できます。

`date_string`

数値文字列、フォーマット済み文字列、または変換済み文字フォーマットの日付文字列です。

IF 条件で値にブランクや特殊文字が含まれていない場合、値を一重引用符 (') で囲む必要はありません。

注意：日付時間文字列は、最低 1 つのブランクで区切られている必要があります。ブランクは、日付時間文字列の先頭または末尾に配置することもできます。

例　日付時間リテラルの割り当て

日付時間リテラルを日付時間フィールドに割り当てるために、COMPUTE、DEFINE、または WHERE 式に接頭語 DT を使用できます。以下はその例です。

```
DT2/HYYMDS = DT(20051226 05:45);
DT3/HYYMDS = DT(2005 DEC 26 05:45);
DT4/HYYMDS = DT(December 26 2005 05:45);
```

例　COMPUTE コマンドへの日付時間値の割り当て

次のリクエストは、COMPUTE コマンドで DT 関数を使用して、割り当てられた日付時間値を含む新しいフィールドを作成します。

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME AND COMPUTE
NEWSAL/D12.2M = CURR_SAL + (0.1 * CURR_SAL);
RAISETIME/HYYMDIA = DT(20000101 09:00AM);
WHERE CURR_JOBCODE LIKE 'B%'
END
```

出力結果は次のとおりです。

```
LAST_NAME      FIRST_NAME           NEWSAL   RAISETIME
---------      ----------           ------   ---------
SMITH          MARY            $14,520.00   2000/01/01  9:00AM
JONES          DIANE           $20,328.00   2000/01/01  9:00AM
ROMANS         ANTHONY         $23,232.00   2000/01/01  9:00AM
MCCOY          JOHN            $20,328.00   2000/01/01  9:00AM
BLACKWOOD      ROSEMARIE       $23,958.00   2000/01/01  9:00AM
MCKNIGHT       ROGER           $17,710.00   2000/01/01  9:00AM
```

例　WHERE 条件への日付時間値の割り当て

次のリクエストは、DT 関数を使用して、割り当てられた日付時間値を含む新しいフィールドを作成します。この値は、その後 WHERE 条件として使用されます。

```
DEFINE FILE EMPLOYEE
NEWSAL/D12.2M = CURR_SAL + (0.1 * CURR_SAL);
RAISETIME/HYYMDIA = DT(20000101 09:00AM);
END


TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME NEWSAL RAISETIME
WHERE RAISETIME EQ DT(20000101 09:00AM)
END
```

出力結果は次のとおりです。

```
LAST_NAME      FIRST_NAME           NEWSAL   RAISETIME
---------      ----------           ------   ---------
STEVENS        ALFRED          $12,100.00   2000/01/01  9:00AM
SMITH          MARY            $14,520.00   2000/01/01  9:00AM
JONES          DIANE           $20,328.00   2000/01/01  9:00AM
SMITH          RICHARD         $10,450.00   2000/01/01  9:00AM
BANNING        JOHN            $32,670.00   2000/01/01  9:00AM
IRVING         JOAN            $29,548.20   2000/01/01  9:00AM
ROMANS         ANTHONY         $23,232.00   2000/01/01  9:00AM
MCCOY          JOHN            $20,328.00   2000/01/01  9:00AM
BLACKWOOD      ROSEMARIE       $23,958.00   2000/01/01  9:00AM
MCKNIGHT       ROGER           $17,710.00   2000/01/01  9:00AM
GREENSPAN      MARY             $9,900.00   2000/01/01  9:00AM
CROSS          BARBARA         $29,768.20   2000/01/01  9:00AM
```

例　IF 条件への日付時間値の割り当て

次のリクエストは、DT 関数を使用して、割り当てられた日付時間値を含む新しいフィールドを作成します。この値は、後から IF 句で使用されます。

```
DEFINE FILE EMPLOYEE
NEWSAL/D12.2M = CURR_SAL + (0.1 * CURR_SAL);
RAISETIME/HYYMDIA = DT(20000101 09:00AM);
END

TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME NEWSAL RAISETIME
IF RAISETIME EQ '20000101 09:00AM'
END
```

出力結果は次のとおりです。

```
LAST_NAME      FIRST_NAME           NEWSAL   RAISETIME
---------      ----------           ------   ---------
STEVENS        ALFRED          $12,100.00   2000/01/01  9:00AM
SMITH          MARY            $14,520.00   2000/01/01  9:00AM
JONES          DIANE           $20,328.00   2000/01/01  9:00AM
SMITH          RICHARD         $10,450.00   2000/01/01  9:00AM
BANNING        JOHN            $32,670.00   2000/01/01  9:00AM
IRVING         JOAN            $29,548.20   2000/01/01  9:00AM
ROMANS         ANTHONY         $23,232.00   2000/01/01  9:00AM
MCCOY          JOHN            $20,328.00   2000/01/01  9:00AM
BLACKWOOD      ROSEMARIE       $23,958.00   2000/01/01  9:00AM
MCKNIGHT       ROGER           $17,710.00   2000/01/01  9:00AM
GREENSPAN      MARY             $9,900.00   2000/01/01  9:00AM
CROSS          BARBARA         $29,768.20   2000/01/01  9:00AM
```

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(datetime, 'component', increment, length, output)
```

説明

`datetime`
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名、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

返される値の長さです。有効な値は次のとおりです。

- **8** - 1 桁から 3 桁 (ミリ秒) を含む日付時間値です。
- **10** - 4 桁から 6 桁 (マイクロ秒) を含む日付時間値です。
- **12** - 7 桁から 9 桁 (ナノ秒) を含む日付時間値です。

`output`
日付時間

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。このフィールドは、日付時間フォーマット (データタイプ H) である必要があります。

例　日付時間フィールドの月構成要素を増加

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。

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
ADD_MONTH/HYYMDS = HADD(TRANSDATE, 'MONTH', 2, 8, 'HYYMDS');
WHERE DATE EQ 2000;
END
```

出力結果は次のとおりです。

```
CUSTID  DATE-TIME          ADD_MONTH
------  ---------          ---------
1237    2000/02/05 03:30   2000/04/05 03:30:00
1118    2000/06/26 05:45   2000/08/26 05:45:00
```

例　UNIX (エポック) 時間の日付時間値への変換

UNIX 時間 (エポック時間とも呼ばれる) は、協定世界時 (UTC) での 1970 年 1 月 1 日 (木曜日) 午前 0 時 0 分 0 秒からの経過秒数を、うるう秒を含めない時間として定義したものです。

次の DEFINE FUNCTION は、エポック時間を表す数値を取得し、その数値を日付時間値に変換します。ここでは、HADD 関数を使用して、エポック時間で表された入力値の秒数をエポック基準日に追加するという方法で日付時間値が計算されます。

```
DEFINE FUNCTION UNIX2GMT(INPUT/I9)
   UNIX2GMT/HYYMDS = HADD(DT(1970 JAN 1),'SECONDS',INPUT,8,'HYYMDS');
END
```

次のリクエストは、この DEFINE FUNCTION を使用して、エポック時間の 1449068652 を日付時間値に変換します。

```
DEFINE FILE GGSALES
INPUT/I9=1449068652;
OUTDATE/HMTDYYSb = UNIX2GMT(INPUT);
END
TABLE FILE GGSALES
PRINT DATE NOPRINT INPUT OUTDATE
WHERE RECORDLIMIT EQ 1
ON TABLE SET PAGE NOLEAD
END
```

下図は、出力結果を示しています。

INPUT	OUTDATE
1449068652	December 02 2015 3:04:12 pm

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(datetime, '(format)', length, output)
```

説明

`datetime`
日付時間

変換する日付時間値です。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。

`format`
文字

日付時間フィールドのフォーマットです。フォーマットは括弧と一重引用符 (') で囲みます。フォーマットは、日付時間フォーマット (データタイプ H、H23 まで) である必要があります。

`length`
整数

文字フィールドに返される値のバイト数です。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。

`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。このフィールドは、返される文字をすべて格納できる長さの文字フォーマットにする必要があります。

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。

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
ALPHA_DATE_TIME1/A20 = HCNVRT(TRANSDATE, '(H17)', 17, 'A20');
ALPHA_DATE_TIME2/A20 = HCNVRT(TRANSDATE, '(HYYMDS)', 20, 'A20');
WHERE DATE EQ 2000
END
```

出力結果は次のとおりです。

```
CUSTID  DATE-TIME         ALPHA_DATE_TIME1      ALPHA_DATE_TIME2
------  ---------        ----------------      ----------------
1237    2000/02/05 03:30  20000205033000000     2000/02/05 03:30:00
1118    2000/06/26 05:45  20000626054500000     2000/06/26 05:45:00
```

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(datetime, output)
```

説明

`datetime`
日付時間

変換する日付時間値です。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。

`output`
日付

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。結果を格納するフィールドを指定することもできます。

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。

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
TRANSDATE_DATE/YYMD = HDATE(TRANSDATE, 'YYMD');
WHERE DATE EQ 2000;
END
```

出力結果は次のとおりです。

```
CUSTID  DATE-TIME          TRANSDATE_DATE
------  ---------          --------------
1237    2000/02/05 03:30   2000/02/05
1118    2000/06/26 05:45   2000/06/26
```

HDIFF - 2 つの日付時間値の差を計算

HDIFF 関数は、指定した構成要素単位 (日付または時間) で 2 つの日付時間値の差を計算します。

参照　HDIFF 使用上の注意

HDIFF 関数の減算は、日付フィールドに格納されている日付構成要素を減算する DATEDIF 関数と異なります。DATEDIF 関数では、年単位の差 (満年数) または月単位の差 (満月数) が計算されます。そのため、次の 2 つの日付を減算し、月数または年数を求めると、結果は 0 (ゼロ) になります。

```
DATE1 12/25/2014, DATE2 1/5/2015
```

HDIFF 関数を使用して同一の計算を日付時間フィールドに対して実行すると、結果は 1 か月または 1 年になります。この場合、最初に月または年が各日付時間値から抽出され、次に減算が実行されます。

構文　2 つの日付時間値の差を計算

```
HDIFF(end_dt, start_dt, 'component', output)
```

説明

`end_dt`
日付時間

減算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。

start_dt
日付時間

減算する日付時間値です。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。

component
文字

計算に使用する構成要素名です。構成要素名は一重引用符 (') で囲みます。構成要素が週の場合、計算に WEEKFIRST パラメータ設定を使用します。

output
倍精度浮動小数点数

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。フォーマットは、倍精度浮動小数点数である必要があります。

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。

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
ADD_MONTH/HYYMDS = HADD(TRANSDATE, 'MONTH', 2, 8, 'HYYMDS');
DIFF_DAYS/D12.2 = HDIFF(ADD_MONTH, TRANSDATE, 'DAY', 'D12.2');
WHERE DATE EQ 2000;
END
```

出力結果は次のとおりです。

```
CUSTID  DATE-TIME         ADD_MONTH               DIFF_DAYS
------  ---------         ---------               ---------
1237    2000/02/05 03:30  2000/04/05 03:30:00     60.00
1118    2000/06/26 05:45  2000/08/26 05:45:00     61.00
```

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, output)
```

説明

`date`
日付

変換する日付です。値を含む日付フィールド名、または値を返す式を指定することもできます。日付は、完全な構成要素フォーマットで指定する必要があります。たとえば、使用可能なフォーマットには MDYY や YYJUL があります。

`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。
- **12** - ナノ秒を含む時間値です。

`output`
日付時間

生成される日付時間値です。フィールド名、または出力値のフォーマットを指定することもできます。フォーマットは一重引用符 (') で囲みます。値は、日付時間フォーマットで指定する必要があります (H データタイプ)。

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。

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
TRANSDATE_DATE/YYMD = HDATE(TRANSDATE, 'YYMD');
DT2/HYYMDIA = HDTTM(TRANSDATE_DATE, 8, 'HYYMDIA');
WHERE DATE EQ 2000;
END
```

出力結果は次のとおりです。

```
CUSTID  DATE-TIME          TRANSDATE_DATE  DT2
------  ---------          --------------  ---
1237    2000/02/05 03:30   2000/02/05      2000/02/05 12:00AM
1118    2000/06/26 05:45   2000/06/26      2000/06/26 12:00AM
```

HEXTR - 日付時間値の要素を抽出し、残りの要素を 0 (ゼロ) に設定

HEXTR 関数は、日付時間値から 1 つ以上の日付時間構成要素を抽出し、これらをターゲット日付時間フィールドに移動します。ターゲット日付時間フィールドのその他の日付時間構成要素はすべて 0 (ゼロ) に設定されます。

構文　　日付時間値からの複数構成要素の抽出

```
HEXTR(datetime, 'componentstring', length, output)
```

説明

`datetime`
日付時間
指定の構成要素を抽出する日付時間値です。

`componentstring`
文字
任意の順序のコード列で、抽出して出力日付時間フィールドに移動する構成要素を指定します。下表は有効値を示しています。下表にない文字は、いずれもコード列の終了と見なされます。

コード	説明
C	世紀 (4 桁年の上 2 桁のみ)
Y	年 (4 桁年の下 2 桁のみ)
YY	4 桁年
M	月
D	日
H	時間
I	分
S	秒
s	ミリ秒 (6 桁マイクロ秒値の上 3 桁)
u	マイクロ秒 (6 桁マイクロ秒値の下 3 桁)

コード	説明
m	マイクロ秒値の 6 桁すべて
n	9 桁の下 3 桁

`length`
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。
- **12** - ナノ秒を含む時間値です。

`output`
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。このフィールドは、日付時間フォーマット (データタイプ H) である必要があります。

例 HEXTR による時間および分構成要素の抽出

VIDEOTR2 データソースには、HYYMDI タイプの「TRANSDATE」という名前の日付時間フィールド名が含まれています。次のリクエストは、その他の構成要素の値とは関係なく、時間値が 09:18AM であるすべてのレコードを選択します。

```
TABLE FILE VIDEOTR2
PRINT TRANSDATE
BY LASTNAME
BY FIRSTNAME
WHERE HEXTR(TRANSDATE, 'HI', 8, 'HYYMDI') EQ DT(09:18AM)
END
```

出力結果は次のとおりです。

```
LASTNAME        FIRSTNAME   TRANSDATE
--------        ---------   ---------
DIZON           JANET       1999/11/05 09:18
PETERSON        GLEN        1999/09/09 09:18
```

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任意の日付時間フォーマットで返します。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, output)
```

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。
- **12** - ナノ秒を含む時間値です。

`output`
日付時間

返される日付時間値です。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。フォーマットは、日付時間フォーマット (データタイプ H) である必要があります。

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。

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
DT2/HYYMDm = HGETC(10, 'HYYMDm');
WHERE DATE EQ 2000;
END
```

出力結果は次のとおりです。

```
CUSTID  DATE-TIME         DT2
------  ---------         ---
1237    2000/02/05 03:30  2000/10/03 15:34:24.000000
1118    2000/06/26 05:45  2000/10/03 15:34:24.000000
```

HGETZ - 現在の協定世界時を日付時間フィールドに格納

HGETZ 関数は、現在の協定世界時 (UTC/GMT 時、Z 時とも呼ばれる) を取得します。協定世界時 (UTC) は、世界の時計および時刻の標準となる常用時です。

値は、指定した日付時間フォーマットで返されます。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　現在のグリニッジ標準日時を日付時間フィールドに格納

```
HGETZ(length, output)
```

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。
- **12** - ナノ秒を含む時間値です。

`output`
日付時間

返される日付時間値です。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。フォーマットは、日付時間フォーマット (データタイプ H) である必要があります。

例　現在のグリニッジ標準日時を日付時間フィールドに格納

HGETZ 関数は、現在のグリニッジ標準日時を DT2 フィールドに格納します。

```
TABLE FILE VIDEOTRK
PRINT CUSTID AND COMPUTE
DT2/HYYMDm = HGETZ(10, 'HYYMDm');
WHERE CUSTID GE '2000' AND CUSTID LE '3000';
END
```

出力結果は次のとおりです。

```
CUSTID  DT2
------  ---
2165    2015/05/08 14:43:08.740000
2187    2015/05/08 14:43:08.740000
2280    2015/05/08 14:43:08.740000
2282    2015/05/08 14:43:08.740000
2884    2015/05/08 14:43:08.740000
```

例　タイムゾーンの計算

タイムゾーンは、グリニッジ標準時 (GMT) を基準とした、時間単位の正または負のオフセットとして計算することができます。グリニッジ子午線より西の地域では、負のオフセットになります。次のリクエストは、HGETC 関数を使用して現地時間を取得し、HGETZ 関数を使用してグリニッジ標準時間 (GMT) を取得します。HDIFF 関数は、2 つの日付時間値の差を分単位で計算します。ゾーンを特定するには、この分単位の時間を 60 で除算します。

```
DEFINE FILE EMPLOYEE
LOCALTIME/HYYMDS = HGETC(8, LOCALTIME);
UTCTIME/HYYMDS = HGETZ(8, UTCTIME);
MINUTES/D4= HDIFF(LOCALTIME, UTCTIME, 'MINUTES', 'D4');
ZONE/P3 = MINUTES/60;
END
TABLE FILE EMPLOYEE
PRINT EMP_ID NOPRINT OVER
LOCALTIME  OVER
UTCTIME OVER
MINUTES OVER
ZONE
IF RECORDLIMIT IS 1
END
```

出力結果は次のとおりです。

```
  LOCALTIME  2015/05/12 12:47:04
  UTCTIME    2015/05/12 16:47:04
  MINUTES                  -240
  ZONE                       -4
```

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(output)
```

説明

`output`
文字 (A8 以上)

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。

例　現在の時間を取得

HHMMSS 関数は、現在の時間を取得し、ページの脚注に表示します。

```
TABLE FILE EMPLOYEE
SUM CURR_SAL AS 'TOTAL SALARIES' AND COMPUTE
NOWTIME/A8 = HHMMSS(NOWTIME); NOPRINT
BY DEPARTMENT
FOOTING
"SALARY REPORT RUN AT TIME <NOWTIME"
END
```

出力結果は次のとおりです。

```
DEPARTMENT   TOTAL SALARIES
----------   --------------
MIS             $108,002.00
PRODUCTION      $114,282.00

SALARY REPORT RUN AT TIME 15.21.14
```

HHMS - 日付時間値を時間値に変換

HHMS 関数は、日付時間値を時間値に変換します。

構文　日付時間値を時間値に変換

```
HHMS(datetime, length, output)
```

説明

`datetime`
日付時間

変換する日付時間値です。

`length`
数値

返される時間値の長さです。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。
- **12** - ナノ秒を含む時間値です。

output
時間

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。

例 日付時間値を時間値に変換

次のリクエストは、TRANSDATE 日付時間フィールドを、HHIS 時間フォーマットの時間フィールドに変換します。

```
DEFINE FILE VIDEOTR2
TRANSYEAR/I4 = HPART(TRANSDATE, 'YEAR', 'I4');
END
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
TRANS_TIME/HHIS = HHMS(TRANSDATE, 8, 'HHIS');
WHERE TRANSYEAR EQ 2000;
END
```

出力結果は次のとおりです。

```
CUSTID  DATE-TIME         TRANS_TIME
------  ---------         ----------
1118    2000/06/26 05:45  05:45:00
1237    2000/02/05 03:30  03:30:00
```

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(source_length, 'source_string', output_length, output)
```

説明

source_length
整数

変換するソース文字列のバイト数です。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。

source_string
文字

変換する文字列です。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。

output_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** - 1 桁から 3 桁 (ミリ秒) を含む時間値です。
- **10** - 4 桁から 6 桁 (マイクロ秒) を含む時間値です。
- **12** - 7 桁から 9 桁 (ナノ秒) を含む時間値です。

output
日付時間

返される日付時間値です。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。フォーマットは、日付時間フォーマット (データタイプ H) である必要があります。

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。

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
ALPHA_DATE_TIME/A20 = HCNVRT(TRANSDATE, '(H17)', 17, 'A20');
DT_FROM_ALPHA/HYYMDS = HINPUT(14, ALPHA_DATE_TIME, 8, 'HYYMDS');
WHERE DATE EQ 2000;
END
```

出力結果は次のとおりです。

```
CUSTID  DATE-TIME         ALPHA_DATE_TIME       DT_FROM_ALPHA
------  ---------        ---------------       -------------
1237    2000/02/05 03:30  20000205033000000     2000/02/05 03:30:00
1118    2000/06/26 05:45  20000626054500000     2000/06/26 05:45:00
```

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(datetime, length, output)
```

説明

`datetime`
日付時間

時間部分を午前零時に設定する日付時間値です。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。
- **12** - ナノ秒を含む時間値です。

`output`
日付時間

時間を午前零時に設定し、タイムスタンプから日付をコピーする日付時間の戻り値です。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。 フォーマットは、日付時間フォーマット (データタイプ H) である必要があります。

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。

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
TRANSDATE_MID_24/HYYMDS  = HMIDNT(TRANSDATE, 8, 'HYYMDS');
TRANSDATE_MID_12/HYYMDSA = HMIDNT(TRANSDATE, 8, 'HYYMDSA');
WHERE DATE EQ 2000;
END
```

出力結果は次のとおりです。

```
CUSTID  DATE-TIME          TRANSDATE_MID_24      TRANSDATE_MID_12
------  ---------          ----------------      ----------------
1118    2000/06/26 05:45   2000/06/26 00:00:00   2000/06/26 12:00:00AM
1237    2000/02/05 03:30   2000/02/05 00:00:00   2000/02/05 12:00:00AM
```

HMASK - 日付時間構成要素を抽出し、それ以外を保持

HMASK 関数は、日付時間値から 1 つ以上の日付時間構成要素を抽出し、これらをターゲット日付時間フィールドに移動します。ターゲット日付時間フィールドのその他の日付時間構成要素はすべて保持されます。

構文　　複数日付時間構成要素のターゲット日付時間フィールドの変更

```
HMASK(source, 'componentstring', input, length, output)
```

説明

`source`
指定の構成要素を抽出する日付時間値です。

`componentstring`
任意の順序のコード列で、抽出して出力日付時間フィールドに移動する構成要素を指定します。下表は有効値を示しています。下表にない文字は、いずれもコード列の終了と見なされます。

コード	説明
C	世紀 (4 桁年の上 2 桁のみ)
Y	年 (4 桁年の下 2 桁のみ)
YY	4 桁年
M	月
D	日
H	時間
I	分
S	秒
s	ミリ秒 (6 桁マイクロ秒値の上 3 桁)
u	マイクロ秒 (6 桁マイクロ秒値の下 3 桁)
m	マイクロ秒値の 6 桁すべて

コード	説明
n	9 桁の下 3 桁

`input`
変換される値を格納するフィールド名です。

`length`
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** - 1 桁から 3 桁 (ミリ秒) を含む時間値です。
- **10** - 4 桁から 6 桁 (マイクロ秒) を含む時間値です。
- **12** - 7 桁から 9 桁 (ナノ秒) を含む時間値です。

`output`
結果を格納するフィールド、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。このフィールドは、日付時間フォーマット (データタイプ H) である必要があります。

参照 HMASK 関数使用上の注意

HMASK の処理は DTSTRICT の設定に依存します。月 (M) 構成要素を伴わない日 (D) 構成要素の移動は無効な結果が発生します。この移動は DTSTRICT が ON に設定されている場合、禁止されます。日付時間値が無効の場合、すべての日付時間関数は 0 (ゼロ) を返します。

例 HMASKによる日付時間フィールドの変更

VIDEOTRK データソースには、HYYMDI フォーマットの「TRANSDATE」という名前の日付時間フィールド名が含まれています。次のリクエストは、12:00 以降の TRANSDATE 値をすべて翌日の 8:30 に変更します。HEXTR 関数は、最初に値の時間および分の部分を抽出し、12:00 と比較します。12:00 よりも大きい場合、HADD 関数は HMASK を呼び出し、時間を 08:30 に変更して、日付を 1 日追加します。

```
DEFINE FILE VIDEOTR2
ORIG_TRANSDATE/HYYMDI = TRANSDATE;
TRANSDATE =
IF HEXTR(TRANSDATE, 'HI', 8, 'HHI') GT DT(12:00)
   THEN HADD (HMASK(DT(08:30), 'HISs', TRANSDATE, 8, 'HYYMDI'), 'DAY',
    1,8, 'HYYMDI')
   ELSE TRANSDATE;
END

TABLE FILE VIDEOTR2
PRINT ORIG_TRANSDATE TRANSDATE
BY LASTNAME
BY FIRSTNAME
WHERE ORIG_TRANSDATE NE TRANSDATE
END
```

出力結果は次のとおりです。

```
LASTNAME        FIRSTNAME   ORIG_TRANSDATE    TRANSDATE
--------        ---------   --------------    ---------
BERTAL          MARCIA      1999/07/29 12:19  1999/07/30 08:30
GARCIA          JOANN       1998/05/08 12:48  1998/05/09 08:30
                            1999/11/30 12:12  1999/12/01 08:30
PARKER          GLENDA      1999/01/06 12:22  1999/01/07 08:30
RATHER          MICHAEL     1998/02/28 12:33  1998/03/01 08:30
WILSON          KELLY       1999/06/26 12:34  1999/06/27 08:30
```

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、指定した構成要素を日付時間値から抽出し、その値を文字フォーマットで返します。

構文　日付時間構成要素を文字フォーマットで取得

```
HNAME(datetime, 'component', output)
```

説明

`datetime`
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。

`output`
文字 (A2 以上)

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。フォーマットは、文字フォーマットである必要があります。

HNAME 関数は、月の引数を月の省略名に変換し、その他すべての構成要素を数字の文字列のみに変換します。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。

```
SET WEEKFIRST = 7
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
WEEK_COMPONENT/A10 = HNAME(TRANSDATE, 'WEEK', 'A10');
WHERE DATE EQ 2000;
END
```

WEEKFIRST が 7 に設定されると、出力は次のようになります。

```
CUSTID  DATE-TIME         WEEK_COMPONENT
------  ---------         --------------
1237    2000/02/05 03:30  06
1118    2000/06/26 05:45  26
```

WEEKFIRST が 3 に設定されると、出力は次のようになります。

```
CUSTID  DATE-TIME         WEEK_COMPONENT
------  ---------         --------------
1237    2000/02/05 03:30  05
1118    2000/06/26 05:45  25
```

WEEKFIRST についての詳細は、『ibi™ WebFOCUS® アプリケーション作成ガイド』を参照してください。

例　日構成要素を文字フォーマットで取得

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

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
DAY_COMPONENT/A2 = HNAME(TRANSDATE, 'DAY', 'A2');
WHERE DATE EQ 2000;
END
```

出力結果は次のとおりです。

```
CUSTID  DATE-TIME          DAY_COMPONENT
------  ---------          -------------
1237    2000/02/05 03:30   05
1118    2000/06/26 05:45   26
```

HPART - 日付時間構成要素を数値として取得

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(datetime, 'component', output)
```

説明

`datetime`
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。

`output`
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。

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
DAY_COMPONENT/I2 = HPART(TRANSDATE, 'DAY', 'I2');
WHERE DATE EQ 2000;
END
```

出力結果は次のとおりです。

```
CUSTID    DATE-TIME              DAY_COMPONENT

1237      2000/02/05 03:30       5
1118      2000/06/26 05:45       26
```

HSETPT - 日付時間値に構成要素を挿入

HSETPT 関数は、指定した構成要素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(datetime, 'component', value, length, output)
```

説明

`datetime`
日付時間

構成要素の挿入先の日付時間値です。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

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

指定した構成要素に挿入する数値です。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** - 1 桁から 3 桁 (ミリ秒) を含む時間値です。
- **10** - 4 桁から 6 桁 (マイクロ秒) を含む時間値です。
- **12** - 7 桁から 9 桁 (ナノ秒) を含む時間値です。

output
日付時間

返される日付時間値です。指定した構成要素の値のみが更新されます。その他の構成要素はすべて、元の日付時間値からコピーされます。

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。フォーマットは、日付時間フォーマット (データタイプ H) である必要があります。

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。

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
ADD_MONTH/HYYMDS = HADD(TRANSDATE, 'MONTH', 2, 8, 'HYYMDS');
INSERT_DAY/HYYMDS = HSETPT(ADD_MONTH, 'DAY', 28, 8, 'HYYMDS');
WHERE DATE EQ 2000;
END
```

出力結果は次のとおりです。

```
CUSTID  DATE-TIME          ADD_MONTH             INSERT_DAY
------  ---------          ---------             ----------
1118    2000/06/26 05:45   2000/08/26 05:45:00   2000/08/28 05:45:00
1237    2000/02/05 03:30   2000/04/05 03:30:00   2000/04/28 03:30:00
```

HTIME - 日付時間値の時間部分を数値に変換

HTIME 関数は、length 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ます。また、length 引数が 10 の場合はマイクロ秒の数値に、length 引数が 12 の場合はナノ秒の数値に変換します。

構文　日付時間値の時間部分を数値に変換

```
HTIME(length, datetime, output)
```

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** - 1 桁から 3 桁 (ミリ秒) を含む時間値です。
- **10** - 4 桁から 6 桁 (マイクロ秒) を含む時間値です。
- **12** - 7 桁から 9 桁 (ナノ秒) を含む時間値です。

`datetime`
日付時間

時間の変換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。

`output`
倍精度浮動小数点数

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。フォーマットは、倍精度浮動小数点数である必要があります。

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。

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
MILLISEC/D12.2 = HTIME(8, TRANSDATE, 'D12.2');
WHERE DATE EQ 2000;
END
```

出力結果は次のとおりです。

```
CUSTID  DATE-TIME              MILLISEC
------  ---------             --------
1237    2000/02/05 03:30   12,600,000.00
1118    2000/06/26 05:45   20,700,000.00
```

HTMTOTS または TIMETOTS - 時間をタイムスタンプに変換

HTMTOTS 関数は、現在の日付からタイムスタンプを取得し、その値の日付構成要素を返した上で、入力日付時間値から時間構成要素をコピーします。

注意：TIMETOTS は、HTMTOTS の同義語です。

構文　時間をタイムスタンプに変換

```
HTMTOTS(time, length, output)
```

または

```
TIMETOTS(time, length, output)
```

説明

`time`

日付時間

日付時間値です。この日付時間値の時間部分が使用されます。日付部分は無視されます。

`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 - ミリ秒を含む入力時間値です。
- 10 - マイクロ秒を含む入力時間値です。
- 12 - ナノ秒を含む入力時間値です。

`output_format`

日付時間

現在の日付に設定し、時間をコピーするタイムスタンプです。

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。

例　時間をタイムスタンプに変換

HTMTOTS 関数は、TRANSDATE フィールドの時間部分をタイムスタンプに変換します。また、戻り値の日付部分には現在の日付を使用します。

```
DEFINE FILE VIDEOTR2
  TSTMPSEC/HYYMDS = HTMTOTS(TRANSDATE, 8, 'HYYMDS');
END
TABLE FILE VIDEOTR2
PRINT TRANSDATE TSTMPSEC
BY LASTNAME BY FIRSTNAME
WHERE DATE EQ '1991'
END
```

出力結果は次のとおりです。

```
LASTNAME        FIRSTNAME   TRANSDATE          TSTMPSEC
--------        ---------   ---------          --------
CRUZ            IVY         1991/06/27 02:45   2011/01/11 02:45:00
GOODMAN         JOHN        1991/06/25 01:19   2011/01/11 01:19:00
GREEVEN         GEORGIA     1991/06/24 10:27   2011/01/11 10:27:00
HANDLER         EVAN        1991/06/20 05:15   2011/01/11 05:15:00
                            1991/06/21 07:11   2011/01/11 07:11:00
KRAMER          CHERYL      1991/06/21 01:10   2011/01/11 01:10:00
                            1991/06/19 07:18   2011/01/11 07:18:00
                            1991/06/19 04:11   2011/01/11 04:11:00
MONROE          CATHERINE   1991/06/25 01:17   2011/01/11 01:17:00
                PATRICK     1991/06/27 01:17   2011/01/11 01:17:00
SPIVEY          TOM         1991/11/17 11:28   2011/01/11 11:28:00
WILLIAMS        KENNETH     1991/06/24 04:43   2011/01/11 04:43:00
                            1991/06/24 02:08   2011/01/11 02:08:00
```

HYYWD - 日付時間値から年と週番号を取得

HNAME および HPART で返された週番号を入力日付の前後いずれかに配置して、年の値に含めることができます。

HYYWD 関数は、指定した日付時間値から年と週番号の両方を返します。

出力は、日付と週番号の ISO 標準フォーマットである yyyy-Www-d に適合するように編集されます。

構文　　日付時間値から年と週番号を取得

```
HYYWD(dtvalue, output)
```

説明

`dtvalue`

日付時間

編集する日付時間値です。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。

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。

出力フォーマットの長さは 10 バイト以上にする必要があります。以下は、出力のフォーマットです。

```
yyyy-Www-d
```

説明

yyyy

4 桁の年です。

ww

2 桁の週番号です (01 から 53)。

d

1 桁の曜日です (1 から 7)。d の値は、WEEKFIRST の設定に対応します。たとえば、WEEKFIRST が 2 または ISO2 (月曜日) の場合、出力では月曜日は 1、火曜日は 2 と表示されます。

EDIT 関数を使用して、この出力から個々のサブフィールドを抽出できます。

例 日付時間値から年と週番号を取得

次の例では、VIDEOTR2 データソースに対するリクエストで HYYWD を呼び出して、TRANSDATE 日付時間フィールドを ISO 標準フォーマットの日付と週番号に変換します。WEEKFIRST が ISO2 (月曜日) に設定され、ISO 標準の週番号が生成されます。

```
SET WEEKFIRST = ISO2
TABLE FILE VIDEOTR2
SUM TRANSTOT QUANTITY
COMPUTE ISODATE/A10 = HYYWD(TRANSDATE, 'A10');
BY TRANSDATE
WHERE QUANTITY GT 1
END
```

出力結果は次のとおりです。

```
TRANSDATE          TRANSTOT  QUANTITY  ISODATE
---------          --------  --------  -------
1991/06/24 04:43     16.00         2  1991-W26-1
1991/06/25 01:17      2.50         2  1991-W26-2
1991/06/27 02:45     16.00         2  1991-W26-4
1996/08/17 05:11      5.18         2  1996-W33-6
1998/02/04 04:11     12.00         2  1998-W06-3
1999/01/30 04:16     13.00         2  1999-W04-6
1999/04/22 06:19      3.75         3  1999-W16-4
1999/05/06 05:14      1.00         2  1999-W18-4
1999/08/09 03:17     15.00         2  1999-W32-1
1999/09/09 09:18     14.00         2  1999-W36-4
1999/10/16 09:11      5.18         2  1999-W41-6
1999/11/05 11:12      2.50         2  1999-W44-5
1999/12/09 09:47      5.18         2  1999-W49-4
1999/12/15 04:04      2.50         2  1999-W50-3
```

例　HYYWD によって返される日付から構成要素を抽出

次の例では、VIDEOTR2 データソースに対するリクエストで HYYWD を呼び出して、TRANSDATE 日付時間フィールドを ISO 標準フォーマットの日付と週番号に変換します。次に、EDIT 関数を使用して、この日付から週構成要素を抽出します。WEEKFIRST が ISO2 (月曜日) に設定され、ISO 標準の週番号が生成されます。

```
SET WEEKFIRST = ISO2
TABLE FILE VIDEOTR2
SUM TRANSTOT QUANTITY
COMPUTE ISODATE/A10 = HYYWD(TRANSDATE, 'A10');
COMPUTE WEEK/A2 = EDIT(ISODATE, '$$$$$$99$$');
BY TRANSDATE
WHERE QUANTITY GT 1 AND DATE EQ 1991
END
```

出力結果は次のとおりです。

```
TRANSDATE          TRANSTOT  QUANTITY  ISODATE     WEEK
---------          --------  --------  -------     ----
1991/06/24 04:43     16.00         2  1991-W26-1  26
1991/06/25 01:17      2.50         2  1991-W26-2  26
1991/06/27 02:45     16.00         2  1991-W26-4  26
```

13

簡略変換関数

簡略変換関数では、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 - 数値コードに基づいて文字を取得

CHAR 関数は、10 進整数を入力値として、オペレーティングシステム環境に応じて、変換された値が ASCII で識別される文字を返します。出力は、可変長文字として返されます。数値が有効な文字範囲を超える場合、NULL 値が返されます。

表示可能文字および対応する文字コードについての詳細は、39 ページの 「 ASCII 文字コード表 」 を参照してください。

構文　数値コードに基づいて文字を取得

```
CHAR(number_code)
```

説明

`number_code`

整数

フィールド、数値、数値式のいずれかです。この整数の絶対値が数値コードとして使用され、出力文字が取得されます。

たとえば、TAB 文字は、ASCII 環境では CHAR(9) で返されます。

例　CHAR 関数による文字列への制御文字の挿入

次のリクエストは、ASCII 環境でキャリッジリターン文字 (CHAR(13)) とラインフィード文字 (CHAR(10)) が「HELLO」と「GOODBYE」という語句の間に挿入されたフィールドを定義します。これらの文字が挿入されたことを示すために PDF フォーマットで出力し、これらの文字にスタイルシート属性の LINEBREAK='CRLF' を使用して、フィールド値を 2 行で表示します。

```
DEFINE FILE WF_RETAIL_LITE
MYFIELD/A20 WITH COUNTRY_NAME='HELLO' | CHAR(13) | CHAR(10) | 'GOODBYE';
END
TABLE FILE WF_RETAIL_LITE
SUM MYFIELD
ON TABLE PCHOLD FORMAT PDF
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
TYPE=REPORT,LINEBREAK='CRLF',$
ENDSTYLE
END
```

下図は、出力結果を示しています。

```
MYFIELD
---------

HELLO
GOODBYE
```

COMPACTFORMAT - 短縮形式での数値表示

COMPACTFORMAT 関数は、数値を短縮形式で表示します。

- K は 1000 の短縮形です。

- M は 100 万の短縮形です。
- B は 10 億の短縮形です。
- T は 1 兆の短縮形です。

COMPACTFORMAT 関数は、フィールドの最大値の桁数に基づいて、使用する省略形を計算します。結果は、文字値として返されます。この値を数値フォーマットで出力しようとすると、フォーマットエラーが発生し、値として 0 (ゼロ) が表示されます。

構文　短縮形式での数値表示

```
COMPACTFORMAT(input)
```

説明

`source_string`

数値フィールドの名前です。

例　短縮形式での数値表示

次の例では COMPACTFORMAT 関数を使用して、DAYSDELAYED、QUANTITY_SOLD、COGS_US フィールドの集計値を短縮形で表示します。

```
TABLE FILE WF_RETAIL_LITE
SUM DAYSDELAYED QUANTITY_SOLD COGS_US
COMPUTE
CDAYS/A30= COMPACTFORMAT(DAYSDELAYED);
CQUANT/A30= COMPACTFORMAT(QUANTITY_SOLD);
CCOGS/A30= COMPACTFORMAT(COGS_US);
ON TABLE SET PAGE NOPAGE
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果を示しています。

Days Delayed	Quantity Sold	Cost of Goods	CDAYS	CQUANT	CCOGS
5,355	13,923	$2,950,358.00	5,355	14K	$3M

CTRLCHAR - 非表示制御文字の取得

CTRLCHAR 関数は、サポートされているキーワードリストに基づいて、実行中のオペレーティングシステムに固有の非表示制御文字を返します。出力は、可変長文字として返されます。

構文　　非表示制御文字の取得

```
CTRLCHAR(ctrl_char)
```

説明

`ctrl_char`

次のキーワードのいずれかです。

- **NUL**　NULL 文字を返します。
- **SOH**　ヘッダ開始文字を返します。
- **STX**　テキスト開始文字を返します。
- **ETX**　テキスト終了文字を返します。
- **EOT**　伝送終了文字を返します。
- **ENQ**　問い合わせ文字を返します。
- **ACK**　確認文字を返します。
- **BEL**　ベル文字またはビープ文字を返します。
- **BS**　バックスペース文字を返します。
- **TAB** または **HT**　水平タブ文字を返します。
- **LF**　ラインフィード文字を返します。
- **VT**　垂直タブ文字を返します。
- **FF**　フォームフィード (ページの最上部) 文字を返します。
- **CR**　キャリッジ制御文字を返します。
- **SO**　シフトアウト文字を返します。
- **SI**　シフトイン文字を返します。
- **DLE**　データリンクエスケープ文字を返します。

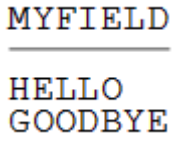
- **DC1** または **XON**　装置制御 1 文字を返します。
- **DC2**　装置制御 2 文字を返します。
- **DC3** または **XOFF**　装置制御 3 文字を返します。
- **DC4**　装置制御 4 文字を返します。
- **NAK**　否定の確認文字を返します。
- **SYN**　同期信号文字を返します。
- **ETB**　伝送ブロック終了文字を返します。
- **CAN**　取消し文字を返します。
- **EN**　媒体終了文字を返します。
- **SUB**　置換文字を返します。
- **ESC**　エスケープ文字、接頭文字、AltMode 文字を返します。
- **FS**　ファイル区切り文字を返します。
- **GS**　グループ区切り文字を返します。
- **RS**　レコード区切り文字を返します。
- **US**　ユニット区切り文字を返します。
- **DEL**　削除文字、抹消文字、割り込み文字を返します。

例　CTRLCHAR 関数による文字列への制御文字の挿入

次のリクエストは、キャリッジリターン文字 (CTRLCHAR(CR)) とラインフィード文字 (CTRLCHAR(LF)) が「HELLO」と「GOODBYE」という語句の間に挿入されたフィールドを定義します。これらの文字が挿入されたことを示すために PDF フォーマットで出力し、これらの文字にスタイルシート属性の LINEBREAK='CRLF' を使用して、フィールド値を 2 行で表示します。

```
DEFINE FILE WF_RETAIL_LITE
MYFIELD/A20 WITH COUNTRY_NAME='HELLO' | CTRLCHAR(CR) | CTRLCHAR(LF) |
'GOODBYE';
END
TABLE FILE WF_RETAIL_LITE
SUM MYFIELD
ON TABLE PCHOLD FORMAT PDF
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
TYPE=REPORT,LINEBREAK='CRLF',$
ENDSTYLE
END
```

下図は、出力結果を示しています。



DT_FORMAT - 日付または日付時間値を文字列に変換

DT_FORMAT 関数は、日付または日付時間値を、指定された日付または日付時間フォーマットの文字列に変換します。

構文　日付値を指定した日付フォーマットの文字列に変換

```
DT_FORMAT(date,'date_format')
```

説明

`date`
　数値、日付、日付時間

　変換する日付、日付時間フィールド、または値です。

`'date_format'`
　文字リテラル

入力フォーマットタイプに対応する日付または日付時間フォーマットを、一重引用符で囲んで指定します。

例　日付および日付時間値を文字フォーマットに変換

次のリクエストは、日付および日付時間値を、指定した日付、日付時間フォーマットの文字値に変換します。

```
DEFINE FILE VIDEOTRK
NEWDT1/A12 = DT_FORMAT(TRANSDAT,'YYMD');
NEWDT2/A30 = DT_FORMAT(DT_CURRENT_DATETIME(SECOND),'HYYMTDs');
NEWDT3/A30= DT_FORMAT('April 1, 2019','YYMDTr');
END
TABLE FILE VIDEOTRK
PRINT NEWDT1 NEWDT2 NEWDT3
BY TRANSDAT
WHERE OUTPUTLIMIT EQ 1
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
END
```

出力結果は次のとおりです。

TRANSDATE	NEWDT1	NEWDT2	NEWDT3
91/06/17	1991/06/17	2019 December 17 11:36:45.000	2019, APRIL 1

FPRINT - 指定したフォーマットでの値表示

FPRINT 簡略変換関数は、値を文字フォーマットに変換し、指定された出力フォーマットで表示します。

注意：FPRINT レガシー関数も使用可能で、従来どおりサポートされます。詳細は、430 ページの「 FPRINT - フィールドを文字フォーマットに変換 」を参照してください。レガシー関数には、戻り値の名前またはフォーマットを指定するための追加の引数があります。

構文　指定したフォーマットでの値表示

```
FPRINT(value, 'out_format')
```

説明

`value`
任意のデータタイプ

変換する値です。

'out_format'
固定長の文字

表示フォーマットです。

例　指定したフォーマットでの値表示

次のリクエストは、FPRINT 関数を使用して COGS_US および TIME_DATE フィールドの値を文字に変更することで、COGS_US を D9M フォーマット、TIME_DATE を YYMtrD フォーマットで表示します。

```
DEFINE FILE WF_RETAIL_LITE
COGS_A/A25 = FPRINT(COGS_US, 'D9M');
DATE1/A25 = FPRINT(TIME_DATE, 'YYMtrD');
END
TABLE FILE WF_RETAIL_LITE
PRINT LST.COGS_US COGS_A  DATE1
BY TIME_DATE
WHERE RECORDLIMIT EQ 10
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果を示しています。

Sale Date	LST Cost of Goods	COGS_A	DATE1
01/03/2009	$234.00	$234	2009, January 3
	$46.00	$46	2009, January 3
	$380.00	$380	2009, January 3
	$374.00	$374	2009, January 3
	$310.00	$310	2009, January 3
	$83.00	$83	2009, January 3
	$312.00	$312	2009, January 3
	$548.00	$548	2009, January 3
	$400.00	$400	2009, January 3
	$131.00	$131	2009, January 3

HEXTYPE - 入力値の 16 進数表記の取得

HEXTYPE 関数は、任意のデータタイプの入力値を 16 進数表記で返します。結果は、可変長文字として返されます。16 進数値が返される文字フィールドは、入力文字列の 1 文字につき 2 バイトを格納できる長さにする必要があります。返される値は、実行中のオペレーティングシステムに応じて異なります。

構文　入力値の 16 進数表記の取得

```
HEXTYPE(in_value)
```

説明

`in_value`

文字フィールドまたは整数フィールド、定数、式のいずれかです。

例　16 進数表記の取得

次のリクエストは、国名および遅延日数合計の 16 進数表記を返します。

```
DEFINE FILE WF_RETAIL_LITE
Days/I8 = DAYSDELAYED;
Country/A20 = COUNTRY_NAME;
HexCountry/A30 = HEXTYPE(Country);
END
TABLE FILE WF_RETAIL_LITE
SUM COUNTRY_NAME NOPRINT  Country HexCountry Days
COMPUTE HexDays/A40 = HEXTYPE(Days);
BY COUNTRY_NAME NOPRINT
WHERE COUNTRY_NAME LT 'P'
ON TABLE SET PAGE NOPAGE
END
```

下図は、出力結果を示しています。

Country	HexCountry	Days	HexDays
Argentina	417267656E74696E61202020202020	84	00000054
Australia	4175737472616C6961202020202020	27	0000001B
Austria	417573747269612020202020202020	798	0000031E
Belgium	42656C6769756D2020202020202020	14	0000000E
Brazil	4272617A696C202020202020202020	204	000000CC
Canada	43616E6164612020202020202020	584	00000248
Chile	4368696C65202020202020202020	45	0000002D
China	4368696E61202020202020202020	1	00000001
Colombia	436F6C6F6D6269612020202020202020	114	00000072
Denmark	44656E6D61726B2020202020202020	0	00000000
Egypt	45677970742020202020202020	3	00000003
Finland	46696E6C616E642020202020202020	3	00000003
France	4672616E6365202020202020202020	49	00000031
Germany	4765726D616E792020202020202020	498	000001F2
Greece	477265656365202020202020202020	9	00000009
Hungary	48756E67617279202020202020202020	7	00000007
India	496E6469612020202020202020202020	23	00000017
Ireland	4972656C616E642020202020202020	7	00000007
Israel	49737261656C202020202020202020	2	00000002
Italy	4974616C79202020202020202020202020	7	00000007
Japan	4A6170616E202020202020202020202020	12	0000000C
Luxembourg	4C7578656D626F7572672020202020	0	00000000
Malaysia	4D616C617973696120202020202020	20	00000014
Mexico	4D657869636F202020202020202020	170	000000AA
Netherlands	4E65746865726C616E647320202020	8	00000008
Norway	4E6F72776179202020202020202020	0	00000000

PHONETIC - 文字列の音声キーの取得

PHONETIC 関数は、文字列の音声キーを計算します。計算失敗時は NULL 値を返します。音声キーは、文字値に綴りのバリエーションがある場合に (例、氏名)、これらの文字値をグループ化する際に役立ちます。このグループ化は、発音に基づいて同一名のバリエーションに同一のインデックス番号を生成する方法で行われます。インデックスの生成には、2 つの音声アルゴリズム (Metaphone と Soundex) のいずれかを使用することができます。デフォルトのアルゴリズムは Metaphone です。

このアルゴリズムで、次のコマンドを使用するよう設定することができます。

```
SET PHONETIC_ALGORITHM = {METAPHONE|SOUNDEX}
```

ほとんどの音声アルゴリズムは、英語の発音に基づいて開発されました。そのため、別の言語の単語にルールを適用すると、意味のない結果が得られる場合があります。

Metaphone は、名前に使用する以外に、ほとんどの英単語に適しています。Metaphone アルゴリズムは、よく使用される多くのスペルチェッカーの基礎になっています。

注意：Metaphone は、生成された SQL では最適化されません。そのため、SQL DBMS のリクエストを最適化する必要がある場合は、SOUNDEX 設定を使用する必要があります。

Soundex は、英語発音の音声に基づいて名前をインデックス化する、従来の音声アルゴリズムです。

構文　音声キーの取得

```
PHONETIC(string)
```

説明

`string`

文字

キーを作成する文字列です。失敗時には NULL 値が返されます。

例　音声キーの生成

次のリクエストは、「MARY SMITH」の姓の綴りを「SMYTHE」に変更し、それぞれの姓の音声キーを生成します。

```
DEFINE FILE EMPLOYEE
LAST_NAME2/A16 = IF LAST_NAME EQ 'SMITH' AND FIRST_NAME EQ 'MARY' THEN
'SMYTHE' ELSE LAST_NAME;
PKEY/A10 = PHONETIC(LAST_NAME2);
END
TABLE FILE EMPLOYEE
PRINT FIRST_NAME LAST_NAME2
BY PKEY
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果を示しています。「SMITH」の 2 つの綴り (元の綴りと変更後の綴り) には、同一のインデックス番号が割り当てられます。

PKEY	FIRST_NAME	LAST_NAME2
B423	ROSEMARIE	BLACKWOOD
B552	JOHN	BANNING
C620	BARBARA	CROSS
G652	MARY	GREENSPAN
I615	JOAN	IRVING
J520	DIANE	JONES
M200	JOHN	MCCOY
M252	ROGER	MCKNIGHT
R552	ANTHONY	ROMANS
S315	ALFRED	STEVENS
S530	MARY	SMYTHE
	RICHARD	SMITH

TO_INTEGER - 文字列を整数値に変換

TO_INTEGER 関数は、文字列内の有効な数値が数字と小数点 (オプション) で構成されている場合に、この文字列を整数値に変換します。値に小数点が含まれている場合、小数点以下の値は切り捨てられます。値が有効な数値でない場合、0 (ゼロ) が返されます。

構文　文字列を整数値に変換

```
TO_INTEGER(string)
```

説明

`string`

一重引用符 (') で囲まれた文字列、または数字と小数点 (オプション) で構成された数値を表す文字フィールドです。

例　文字列を整数値に変換

次のリクエストは、文字列を整数値に変換します。小数点以下の数字は切り捨てられます。

```
DEFINE FILE WF_RETAIL_LITE
INT1/I8 = TO_INTEGER('56.78');
INT2/I8 = TO_INTEGER('.5678');
INT3/I8 = TO_INTEGER('5678');
END
TABLE FILE WF_RETAIL_LITE
PRINT INT1 INT2 INT3
BY BUSINESS_REGION AS Region
WHERE READLIMIT EQ 1
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果を示しています。

Region	INT1	INT2	INT3
EMEA	56	0	5678

TO_NUMBER - 文字列を数値に変換

TO_NUMBER 関数は、文字列内の有効な数値が数字と小数点 (オプション) で構成されている場合に、この文字列を数値に変換します。値が有効な数値でない場合、0 (ゼロ) が返されます。

構文　文字列を数値に変換

```
TO_NUMBER(string)
```

説明

`string`

一重引用符 (') で囲まれた文字列、または数字と小数点 (オプション) で構成された数値を表す文字フィールドです。この文字列は、倍精度小数点数に変換されます。

例　文字列を数値に変換

次のリクエストは、文字列を倍精度小数点数に変換します。

```
DEFINE FILE WF_RETAIL_LITE
NUM1/D12.1 = TO_NUMBER('56.78');
NUM2/D12.2 = TO_NUMBER('0.5678');
END
TABLE FILE WF_RETAIL_LITE
PRINT NUM1 NUM2
BY BUSINESS_REGION AS Region
WHERE READLIMIT EQ 1
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果を示しています。

Region	NUM1	NUM2
EMEA	56.8	.57

14

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。

多くの関数では、*output* 引数にフィールド名またはフォーマットを指定することができます。フォーマットを指定する場合、一重引用符 (') で囲みます。ただし、関数がダイアログマネージャコマンドから呼び出される場合、この引数には常にフォーマットを指定する必要があります。

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(source_string, length, output)
```

説明

`source_string`
文字

変換する文字列です。この文字列は、1 桁以上の数字と、オプションとして 1 つの符号と 1 つの小数点で構成されます。文字列を含むフィールドまたは変数を指定することもできます。

`length`
文字

ソース文字列の長さを 2 文字表記のバイト数で指定します。数値定数を指定することも、値を含むフィールドまたは変数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 (例、'12')。

`output`
倍精度浮動小数点数

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。

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。

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND FIRST_NAME AND
EMP_ID AND
COMPUTE D_EMP_ID/D12.2 = ATODBL(EMP_ID, '09', D_EMP_ID);
WHERE DEPARTMENT EQ 'MIS';
END
```

出力結果は次のとおりです。

```
LAST_NAME     FIRST_NAME    EMP_ID       D_EMP_ID
SMITH         MARY          112847612    112,847,612.00
JONES         DIANE         117593129    117,593,129.00
MCCOY         JOHN          219984371    219,984,371.00
BLACKWOOD     ROSEMARIE     326179357    326,179,357.00
GREENSPAN     MARY          543729165    543,729,165.00
CROSS         BARBARA       818692173    818,692,173.00
```

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。

この関数は、特定のフォーマットで被演算子を必要とする式を実行する際に、その式の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ォーマットに変換すると、フィールド内の符号および小数点は数値の一部として保存されます。

  数値以外のその他の文字は無効で、EDIT が 0 (ゼロ) を返します。

- 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。

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。

```
TABLE FILE EMPLOYEE
PRINT HIRE_DATE AND COMPUTE
ALPHA_HIRE/A17 = EDIT(HIRE_DATE); NOPRINT AND COMPUTE
HIRE_MDY/A17 = CHGDAT('YMD', 'MDYYX', ALPHA_HIRE, 'A17');
BY LAST_NAME BY FIRST_NAME
WHERE DEPARTMENT EQ 'MIS';
END
```

出力結果は次のとおりです。

```
LAST_NAME  FIRST_NAME  HIRE_DATE  HIRE_MDY
---------  ----------  ---------  --------
BLACKWOOD  ROSEMARIE   82/04/01   APRIL 01 1982
CROSS      BARBARA     81/11/02   NOVEMBER 02 1981
GREENSPAN  MARY        82/04/01   APRIL 01 1982
JONES      DIANE       82/05/01   MAY 01 1982
MCCOY      JOHN        81/07/01   JULY 01 1981
SMITH      MARY        81/07/01   JULY 01 1981
```

FPRINT - フィールドを文字フォーマットに変換

FPRINT 関数は、テキストフィールド以外のすべてのフィールドを、対応する文字フォーマットに変換して表示します。この文字フォーマットには、元のフィールドフォーマットで指定される任意の表示オプションを含めることができます。

構文　FPRINT によるフィールド変換

```
FPRINT(in_value, 'usageformat', output)
```

説明

`in_value`
TX 以外の任意のフォーマット

変換する値です。

`usageformat`
文字

表示オプションも含めた、変換する値の USAGE フォーマットです。この値は一重引用符 (') で囲む必要があります。

`output`
文字

一重引用符 (') で囲んだフォーマットまたは出力フィールド名です。

出力フォーマットには、十分な長さを指定する必要があります。これには、符号や小数点を含めた変換後の数値自体のほかに、カンマ (,)、通貨記号、パーセント記号 (%) など、表示オプションで生成される追加の文字も含まれます。

たとえば、D12.2 フォーマットは、小数点以下 2 桁、小数点、マイナス記号 (-)、8 桁までの整数、2 つのカンマ (,) を出力するため、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、その長さを超える右端の文字が切り捨てられる場合があります。

参照　FPRINT 関数使用上の注意

- USAGE フォーマットは、フィールドの実際のデータに一致させる必要があります。
- FPRINT の数値出力は、指定されたフォーマットに対応する最大の文字数が収まる領域内に、右揃えで表示されます。これにより、すべての値が小数点または単位桁位置で、縦方向に位置揃えされます。
- デフォルト設定では、文字フィールドの場合、列タイトルは左揃えになります。列タイトルを右揃えにするには、[/R] (フォーマット変更) オプションを使用します。

例　数値フィールドの文字フォーマットへの変換

次のリクエストは、EMPLOYEE データソースを使用し、FPRINT で CURR_SAL、ED_HRS、BANK_ACCT フィールドを文字に変換し、レポート出力に表示します。さらに、STRREP 関数によって、文字フォーマットの CURR_SAL のブランクが、アスタリスク (*) で置き換えられます。CURR_SAL は D12.2M フォーマットのため、文字フォーマットは A15 です。ED_HRS フィールドは F6.2 フォーマットのため、文字フォーマットは A6 です。BANK_ACCT フィールドは I9S フォーマットのため、文字フォーマットは A9 です。数値フィールドの文字フォーマットは、右揃えになります。PRINT コマンドで「/R」オプションを使用すると、列タイトルが値の上部に右揃えで表示されます。

```
DEFINE FILE EMPLOYEE
ASAL/A15 = FPRINT(CURR_SAL, 'D12.2M', ASAL);
ASAL/A15 = STRREP(15, ASAL, 1, ' ', 1, '*', 15, ASAL);
AED/A6 = FPRINT(ED_HRS, 'F6.2', AED);
ABANK/A9 = FPRINT(BANK_ACCT, 'I9S', ABANK);
END
TABLE FILE EMPLOYEE
PRINT CURR_SAL ASAL
ED_HRS AED/R
BANK_ACCT ABANK/R
WHERE BANK_NAME NE ' '
ON TABLE SET PAGE NOPAGE
END
```

出力結果は次のとおりです。

```
  CURR_SAL  ASAL              ED_HRS     AED  BANK_ACCT      ABANK
  --------  ----              ------  ------  ---------  ---------
$18,480.00  *****$18,480.00    50.00   50.00   40950036   40950036
$29,700.00  *****$29,700.00      .00     .00     160633     160633
$26,862.00  *****$26,862.00    30.00   30.00  819000702  819000702
$21,780.00  *****$21,780.00    75.00   75.00  122850108  122850108
$16,100.00  *****$16,100.00    50.00   50.00  136500120  136500120
$27,062.00  *****$27,062.00    45.00   45.00  163800144  163800144
```

例　文字数値日付フィールドを文字フォーマットに変換

次のリクエストは、EMPLOYEE データソースが使用され、HIRE_DATE フィールドが文字フォーマットに変換されます。また、「ADATE」という名前の文字日付フィールドも作成され、文字フォーマットに変換されます。HIRE_DATE フィールドのフォーマットは I6YMD で、ADATE フィールドのフォーマットは A6YMD であることから、日付構成要素間のスラッシュ (/) が加味され、A8 フォーマットになります。「/R」オプションにより、フィールド値上部の列タイトルが右揃えになります。

```
DEFINE FILE EMPLOYEE
AHDATE/A8 = FPRINT(HIRE_DATE,'I6YMD', AHDATE);
ADATE/A6YMD = EDIT(HIRE_DATE);
AADATE/A8 = FPRINT(ADATE,'A6YMD', AADATE);
END
TABLE FILE EMPLOYEE
PRINT HIRE_DATE AHDATE/R
ADATE AADATE/R
ON TABLE SET PAGE NOPAGE
END
```

出力結果は次のとおりです。

```
HIRE_DATE    AHDATE  ADATE       AADATE
---------  --------  -----     --------
 80/06/02  80/06/02  80/06/02  80/06/02
 81/07/01  81/07/01  81/07/01  81/07/01
 82/05/01  82/05/01  82/05/01  82/05/01
 82/01/04  82/01/04  82/01/04  82/01/04
 82/08/01  82/08/01  82/08/01  82/08/01
 82/01/04  82/01/04  82/01/04  82/01/04
 82/07/01  82/07/01  82/07/01  82/07/01
 81/07/01  81/07/01  81/07/01  81/07/01
 82/04/01  82/04/01  82/04/01  82/04/01
 82/02/02  82/02/02  82/02/02  82/02/02
 82/04/01  82/04/01  82/04/01  82/04/01
 81/11/02  81/11/02  81/11/02  81/11/02
```

例　日付フィールドの文字フォーマット変換

次のリクエストは、VIDEOTRK データソースが使用され、TRANSDATE (YMD) フィールドが文字フォーマットに変換されます。この文字フォーマットは、日付構成要素間のスラッシュ (/) が加味され A8 になります。

```
DEFINE FILE VIDEOTRK
ALPHA_DATE/A8  = FPRINT(TRANSDATE,'YMD', ALPHA_DATE);
END
TABLE FILE VIDEOTRK
PRINT TRANSDATE ALPHA_DATE
WHERE TRANSDATE LE '91/06/20'
ON TABLE SET PAGE NOPAGE
END
```

出力結果は次のとおりです。

```
TRANSDATE  ALPHA_DATE
---------  ----------
91/06/19   91/06/19
91/06/17   91/06/17
91/06/20   91/06/20
91/06/19   91/06/19
91/06/18   91/06/18
91/06/17   91/06/17
91/06/17   91/06/17
91/06/17   91/06/17
91/06/20   91/06/20
91/06/19   91/06/19
91/06/18   91/06/18
91/06/19   91/06/19
91/06/18   91/06/18
91/06/20   91/06/20
91/06/18   91/06/18
91/06/20   91/06/20
91/06/19   91/06/19
91/06/17   91/06/17
```

例　日付時間フィールドを文字フォーマットに変換し、HOLD ファイルを作成

次のリクエストは、VIDEOTR2 データソースが使用され、TRANSDATE (HYYMDI) フィールドが文字フォーマットに変換されます。4 桁の年、2 桁の月、2 桁の日、日付構成要素間の 2 つのスラッシュ (/)、日付と時間の間のブランク、2 桁の時間、時間要素と分要素の間のコロン (:)、2 桁の分が加味され、文字フォーマットは A16 になります。

```
DEFINE FILE VIDEOTR2
DATE/I4 = HPART(TRANSDATE, 'YEAR', 'I4');
ALPHA_DATE/A16 = FPRINT(TRANSDATE,'HYYMDI', ALPHA_DATE);
END
TABLE FILE VIDEOTR2
PRINT TRANSDATE ALPHA_DATE/R
WHERE DATE EQ '1991'
ON TABLE SET PAGE NOPAGE
END
```

出力結果は次のとおりです。

```
TRANSDATE              ALPHA_DATE
---------          ----------------
1991/06/27 02:45  1991/06/27 02:45
1991/06/20 05:15  1991/06/20 05:15
1991/06/21 07:11  1991/06/21 07:11
1991/06/21 01:10  1991/06/21 01:10
1991/06/19 07:18  1991/06/19 07:18
1991/06/19 04:11  1991/06/19 04:11
1991/06/25 01:19  1991/06/25 01:19
1991/06/24 04:43  1991/06/24 04:43
1991/06/24 02:08  1991/06/24 02:08
1991/06/25 01:17  1991/06/25 01:17
1991/06/27 01:17  1991/06/27 01:17
1991/11/17 11:28  1991/11/17 11:28
1991/06/24 10:27  1991/06/24 10:27
```

カンマ区切りやその他の文字ファイルに出力を保存する場合、元のフィールドでは、値の数字フォーマットのみが継承されるのに対し、変換後のフィールドでは、表示オプションも継承されます。

```
DEFINE FILE VIDEOTR2
DATE/I4 = HPART(TRANSDATE, 'YEAR', 'I4');
ALPHA_DATE/A16 = FPRINT(TRANSDATE,'HYYMDI', ALPHA_DATE);
END
TABLE FILE VIDEOTR2
PRINT TRANSDATE ALPHA_DATE/R
WHERE DATE EQ '1991'
ON TABLE HOLD FORMAT COMMA
END
```

HOLD ファイルは次のようになります。最初のフィールドは元のデータ、2 つ目のフィールドには表示オプションを含めた変換後の値が表示されます。

```
"19910627024500000","1991/06/27 02:45"
"19910620051500000","1991/06/20 05:15"
"19910621071100000","1991/06/21 07:11"
"19910621011000000","1991/06/21 01:10"
"19910619071800000","1991/06/19 07:18"
"19910619041100000","1991/06/19 04:11"
"19910625011900000","1991/06/25 01:19"
"19910624044300000","1991/06/24 04:43"
"19910624020800000","1991/06/24 02:08"
"19910625011700000","1991/06/25 01:17"
"19910627011700000","1991/06/27 01:17"
"19911117112800000","1991/11/17 11:28"
"19910624102700000","1991/06/24 10:27"
```

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、先頭にブランクを挿入することにより、数値を右揃えします。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)', output)
```

説明

`number`
数値 F または D (単精度および倍精度浮動小数点数)

変換する数値です。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

変換する数値のフォーマットです。フォーマットは括弧で囲みます。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、そのフォーマットを括弧で囲み、さらに一重引用符 (') で囲む必要があります。

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。

```
TABLE FILE EMPLOYEE
PRINT GROSS AND COMPUTE
ALPHA_GROSS/A15 = FTOA(GROSS, '(D12.2)', ALPHA_GROSS);
BY HIGHEST 1 PAY_DATE NOPRINT
BY LAST_NAME
WHERE (GROSS GT 800) AND (GROSS LT 2300);
END
```

出力結果は次のとおりです。

```
LAST_NAME  GROSS       ALPHA_GROSS
---------  -----       -----------
BLACKWOOD  $1,815.00      1,815.00
CROSS      $2,255.00      2,255.00
IRVING     $2,238.50      2,238.50
JONES      $1,540.00      1,540.00
MCKNIGHT   $1,342.00      1,342.00
ROMANS     $1,760.00      1,760.00
SMITH      $1,100.00      1,100.00
STEVENS      $916.67        916.67
```

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 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。指定する 10 進数は、構成されたコードページの文字に関連する値にする必要があります。HEXBYT 関数は、ASCII、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

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(decimal_value, output)
```

説明

`decimal_value`
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、decimal_value の値を 256 で除算した剰余として扱われます。指定する 10 進数は、構成されたコードページの文字に関連する値にする必要があります。

`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。

例　10 進数を ASCII および Unicode の文字に変換

次のリクエストは、HEXBYT 関数を使用し、10 進数の整数値「130」を ASCII コードページ 1252 のカンマ (,) に変換します。次にカンマ (,) は、LAST_NAME と FIRST_NAME の連結に使用され、NAME フィールドが生成されます。

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND
COMPUTE COMMA1/A1 = HEXBYT(130, COMMA1); NOPRINT
COMPUTE NAME/A40 = LAST_NAME || COMMA1| ' '| FIRST_NAME;
BY LAST_NAME NOPRINT
BY FIRST_NAME
WHERE DEPARTMENT EQ 'MIS';
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果を示しています。

FIRST_NAME	LAST_NAME	NAME
ROSEMARIE	BLACKWOOD	BLACKWOOD, ROSEMARIE
BARBARA	CROSS	CROSS, BARBARA
MARY	GREENSPAN	GREENSPAN, MARY
DIANE	JONES	JONES, DIANE
JOHN	MCCOY	MCCOY, JOHN
MARY	SMITH	SMITH, MARY

コードページ 65001 を使用するよう構成された Unicode 環境でこれと同じ出力結果を生成するには、COMMA1 フィールドの COMPUTE コマンドを次の構文で置換します。この場合、HEXBYT 関数の呼び出しによって、整数値「14844058」がカンマ (,) に変換されます。

```
COMPUTE COMMA1/A1 = HEXBYT(14844058, COMMA1); NOPRINT
```

例　10 進数を文字に変換

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。

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND
COMPUTE LAST_INIT_CODE/I3 = BYTVAL(LAST_NAME, 'I3');
COMPUTE LAST_INIT/A1 = HEXBYT(LAST_INIT_CODE, LAST_INIT);
WHERE DEPARTMENT EQ 'MIS';
END
```

ASCII プラットフォームの出力結果は次のとおりです。

```
LAST_NAME LAST_INIT_CODE LAST_INIT
--------- -------------- ---------
SMITH                 83 S
JONES                 74 J
MCCOY                 77 M
BLACKWOOD             66 B
GREENSPAN             71 G
CROSS                 67 C
```

ITONUM - 整数を倍精度小数点数フォーマットに変換

ITONUM 関数は、FOCUS 以外の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 FOCUS 以外のデータベースには、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, output)
```

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 (A8)

2 進数値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。

`output`
倍精度浮動小数点数 (D*n*)

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。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

外部ファイルの 2 進数のフォーマットが次の COBOL フォーマットであることを想定します。

```
PIC 9(8)V9(4) COMP
```

これは、EUROCAR マスターファイルの「BINARYFLD」という名前のフィールドに定義されています。このフィールドのフォーマットは、USAGE=A8、ACTUAL=A8 です。これは、このフィールドの長さが 4 バイトを超えるためです。

次のリクエストは、フィールドを倍精度フォーマットに変換します。

```
DEFINE FILE EUROCAR
MYFLD/D14 = ITONUM(6, BINARYFLD, MYFLD);
END
TABLE FILE EUROCAR
PRINT MYFLD BY CAR
END
```

ITOPACK - 整数をパック 10 進数フォーマットに変換

ITOPACK 関数は、FOCUS 以外の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 FOCUS 以外のデータベースには、倍長整数フォーマットを使用するものがあります。倍長整数フォーマットは FOCUS が使用する整数フォーマットに類似していますが、より大きい数値を使用することができます。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, output)
```

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 2 進入力フィールドの最大バイト数です。

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 (A8)

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。

`output`
数値

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。フォーマットは P*n* または P*n.d*.

例 整数をパック 10 進数フォーマットに変換

外部ファイルの 2 進数のフォーマットが次の COBOL フォーマットであることを想定します。

```
PIC 9(8)V9(4) COMP
```

これは、EUROCAR マスターファイルの「BINARYFLD」という名前のフィールドに定義されています。このフィールドのフォーマットは、USAGE=A8、ACTUAL=A8 です。これは、このフィールドの長さが 4 バイトを超えるためです。

次のリクエストは、フィールドをパック 10 進数フォーマットに変換します。

```
DEFINE FILE EUROCAR
PACKFLD/P14.4 = ITOPACK(6, BINARYFLD, PACKFLD);
END
TABLE FILE EUROCAR
PRINT PACKFLD BY CAR
END
```

ITOZ - 数値をゾーン 10 進数フォーマットに変換

ITOZ 関数は、数値フォーマット内の数をゾーン 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 数値をゾーン 10 進数フォーマットに変換

```
ITOZ(length, in_value, output)
```

説明

`length`
整数

in_value の長さをバイト数で指定します。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。

in_value
数値

変換する数値です。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。

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。

例 数値をゾーン 10 進数フォーマットに変換

次のリクエストは、従業員 ID と給与情報を含む抽出ファイルを作成します。このファイルは、COBOL プログラム用にゾーン 10 進数フォーマットで作成されます。

```
DEFINE FILE EMPLOYEE
ZONE_SAL/A8 = ITOZ(8, CURR_SAL, ZONE_SAL);
END

TABLE FILE EMPLOYEE
PRINT CURR_SAL ZONE_SAL BY EMP_ID
ON TABLE SAVE AS SALARIES
END
```

結果の抽出ファイルは次のとおりです。

```
NUMBER OF RECORDS IN TABLE= 12 LINES= 12

ALPHANUMERIC  RECORD  NAMED  SALARIES
FIELDNAME                          ALIAS          FORMAT         LENGTH
EMP_ID                             EID            A9              9
CURR_SAL                           CSAL           D12.2M         12
ZONE_SAL                                          A8              8
TOTAL                                                            29
```

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(in_value, length, output)
```

説明

`in_value`
　数値

　値を含む入力フィールドを指定することもできます。パック 10 進数、整数、単精度浮動小数点数、倍精度浮動小数点数フォーマットが使用できます。整数フォーマット以外の場合、端数処理により、値は最も近い整数になります。

`length`
　数値

　出力値の長さを 1 から 16 までのバイト数で指定します。

`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。フィールドにパック 10 進数データが格納されている場合でも、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。

```
DEFINE FILE EMPLOYEE
SHORT_SAL/A5 = PCKOUT(CURR_SAL, 5, SHORT_SAL);
END
TABLE FILE EMPLOYEE
PRINT LAST_NAME SHORT_SAL HIRE_DATE
ON TABLE SAVE
END
```

結果の抽出ファイルは次のとおりです。

```
 NUMBER OF RECORDS IN TABLE=       12 LINES=     12

ALPHANUMERIC  RECORD  NAMED  SAVE
FIELDNAME                         ALIAS          FORMAT           LENGTH
LAST_NAME                         LN             A15                15
SHORT_SAL                                        A5                  5
HIRE_DATE                         HDT            I6YMD               6
TOTAL                                                               26
```

PTOA - 数値を文字フォーマットに変換

PTOA 関数は、パック 10 進数の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。

構文　　数値を文字フォーマットに変換

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

説明

`number`

変換する数値です。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

数値のフォーマットです。フォーマットは括弧で囲み、さらに一重引用符 (') で囲みます。

パック 10 進数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。

フォーマット値には、元のフィールドと同じ長さや小数点以下の桁数を指定する必要はありません。小数点以下の桁数を変更すると、結果は端数処理されます。フィールドの長さが数値の整数部分より短い場合は、数値ではなくアスタリスク (*) が表示されます。

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。ただし、このフィールドに格納されたフォーマットの前後に括弧が含まれている必要があります。以下はその例です。

```
FMT/A10 = '(P12.2C)';
```

これにより、リクエストに関数を使用する際にこのフィールドをフォーマット引数として使用することができます。

```
COMPUTE ALPHA_GROSS/A20 = PTOA(PGROSS, FMT, ALPHA_GROSS);
```

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　パック 10 進数を文字フォーマットに変換

PTOA 関数は、PGROSS フィールドのフォーマットをパック 10 進数から文字に変換するために 2 回呼び出されます。関数の最初の呼び出しで指定されたフォーマットは、「FMT」という名前の一時項目に格納されます。関数の 2 回目の呼び出しで指定されたフォーマットは、小数点以下を含みません。このため、表示される値は端数処理されます。

```
DEFINE FILE EMPLOYEE
PGROSS/P18.2=GROSS;
FMT/A10='(P14.2C)';
END
TABLE FILE EMPLOYEE PRINT PGROSS NOPRINT
COMPUTE AGROSS/A17 = PTOA(PGROSS, FMT, AGROSS); AS ''
COMPUTE BGROSS/A37 = '<- THIS AMOUNT IS' |
                     PTOA(PGROSS, '(P5C)', 'A6') |
                     ' WHEN ROUNDED'; AS '' IN +1
BY HIGHEST 1 PAY_DATE NOPRINT
BY LAST_NAME NOPRINT
END
```

出力結果は次のとおりです。

```
2,475.00 <- THIS AMOUNT IS 2,475 WHEN ROUNDED
1,815.00 <- THIS AMOUNT IS 1,815 WHEN ROUNDED
2,255.00 <- THIS AMOUNT IS 2,255 WHEN ROUNDED
  750.00 <- THIS AMOUNT IS   750 WHEN ROUNDED
2,238.50 <- THIS AMOUNT IS 2,239 WHEN ROUNDED
1,540.00 <- THIS AMOUNT IS 1,540 WHEN ROUNDED
1,540.00 <- THIS AMOUNT IS 1,540 WHEN ROUNDED
1,342.00 <- THIS AMOUNT IS 1,342 WHEN ROUNDED
1,760.00 <- THIS AMOUNT IS 1,760 WHEN ROUNDED
1,100.00 <- THIS AMOUNT IS 1,100 WHEN ROUNDED
  791.67 <- THIS AMOUNT IS   792 WHEN ROUNDED
  916.67 <- THIS AMOUNT IS   917 WHEN ROUNDED
```

TSTOPACK - MSSQL または Sybase タイムスタンプフィールドをパック 10 進数に変換

この関数は、Microsoft SQL Server および Sybase アダプタにのみ適用されます。

Microsoft SQL Server および Sybase には、TIMESTAMP と呼ばれるデータタイプがあります。このデータタイプのフィールドには、実際のタイムスタンプが格納されるのではなく、データソース内で挿入または更新されたレコードごとに増分される数値が格納されます。このタイムスタンプは共通領域から取得されるため、データベース内の 2 つのテーブルに同一のタイムスタンプフィールド値が存在することはありません。この値は Binary(8) または Varbinary(8) フォーマットでテーブルに格納されますが、長さが 2 倍の文字フィールド (A16) として返されます。TSTOPACK 関数を使用して、タイムスタンプ値をパック 10 進数に変換することができます。

構文　MSSQL または Sybase タイムスタンプフィールドをパック 10 進数に変換

```
TSTOPACK(tscol, output);
```

説明

`tscol`

文字 (A16)

変換するタイムスタンプフィールドです。

`output`

パック 10 進数 (P21)

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。

例　Microsoft SQL Server タイムスタンプフィールドのパック 10 進数への変換

次の CREATE TABLE コマンドは、「TSTEST」と呼ばれる SQL Server テーブルを作成し、このテーブルに「I」という整数カウンタフィールドと「TS」というタイムスタンプフィールドを格納します。

```
SQL SQLMSS
  CREATE TABLE TSTEST (I INT, TS timestamp) ;
END
```

TSTEST データソースのマスターファイルは次のとおりです。フィールド名の TS が TIMESTAMP フィールドを表しています。

```
FILENAME=TSTEST, SUFFIX=SQLMSS  , $
  SEGMENT=TSTEST, SEGTYPE=S0, $
    FIELDNAME=I, ALIAS=I, USAGE=I11, ACTUAL=I4,
      MISSING=ON, $
    FIELDNAME=TS, ALIAS=TS, USAGE=A16, ACTUAL=A16, FIELDTYPE=R, $
```

注意：TIMESTAMP フィールドが含まれたテーブルのシノニムを作成すると、その TIMESTAMP フィールドは読み取り専用 (FIELDTYPE=R) として作成されます。

TSTOPACK 関数を使用して、TS タイムスタンプフィールドをパック 10 進数に変換します。

```
DEFINE FILE TSTEST
TSNUM/P21=TSTOPACK(TS,'P21');
END
TABLE FILE TEST64
PRINT I TS TSNUM
END
```

出力結果は次のとおりです。

I	TS	TSNUM
1	0000000000007815	30741
2	0000000000007816	30742
3	0000000000007817	30743
4	0000000000007818	30744
5	0000000000007819	30745
6	000000000000781A	30746
7	000000000000781B	30747
8	000000000000781C	30748
9	000000000000781D	30749
10	000000000000781E	30750

UFMT - 文字列を 16 進数に変換

UFMT 関数は、文字フォーマットのソース文字列の文字を 16 進数に変換します。この関数は、不明なフォーマットのデータの検査に役立ちます。データの長さが明らかであれば、その内容を検査することができます。

構文　文字列を 16 進数に変換

```
UFMT(source_string, length, output)
```

説明

`source_string`
文字

変換する文字列です。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。

`length`
整数

source_string の長さをバイト数で指定します。

`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。output は文字フォーマットにし、length で指定した長さの 2 倍にする必要があります。

例　文字列を 16 進数に変換

UFMT 関数は、JOBCODE の各値を 16 進数に変換し、結果を HEXCODE に格納します。

```
DEFINE FILE JOBFILE
HEXCODE/A6 = UFMT(JOBCODE, 3, HEXCODE);
END
TABLE FILE JOBFILE
PRINT JOBCODE HEXCODE
END
```

出力結果は次のとおりです。

```
JOBCODE  HEXCODE
-------  -------
A01      C1F0F1
A02      C1F0F2
A07      C1F0F7
A12      C1F1F2
A14      C1F1F4
A15      C1F1F5
A16      C1F1F6
A17      C1F1F7
B01      C2F0F1
B02      C2F0F2
B03      C2F0F3
B04      C2F0F4
B14      C2F1F4
```

XTPACK - 有効数字最大 31 桁のパック 10 進数値の出力ファイルへの書き込み

XTPACK 関数を使用して、最大で有効数字 31 桁のパック 10 進数値を、10 進数のデータを保持したまま文字フィールドに格納することができます。これにより、パック 10 進数の短いフィールドまたは長いフィールドを 1 から 16 バイトの任意の長さで出力ファイルに書き込むことが可能になります。

構文　文字フィールドでのパック 10 進数値の格納

```
XTPACK(in_value, outlength, outdec, output)
```

説明

`in_value`

数値

パック 10 進数値です。

`outlength`

数値

変換されたパック 10 進数フィールドを格納する文字フィールドの長さです。1 から 16 までの値を指定することができます。

`outdec`

数値

output の小数点の位置を示します。

`output`

文字

結果を格納するフィールド名、または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 (') で囲みます。

例　長いパック 10 進数値の出力ファイルへの書き込み

次のリクエストは、LONGPCK という名前の長いパック 10 進数フィールドを作成します。ALPHAPCK (フォーマット A13) は、XTPACK 関数を長いパック 10 進数フィールドに適用した結果です。PCT_INC、LONGPCK、および ALPHAPCK は、XTOUT という名前の SAVE ファイルに書き込まれます。

```
DEFINE FILE EMPLOYEE
LONGPCK/P25.2 = PCT_INC + 11111111111111111111;
ALPHAPCK/A13 = XTPACK(LONGPCK,13,2,'A13');
END
TABLE FILE EMPLOYEE
PRINT PCT_INC LONGPCK ALPHAPCK
WHERE PCT_INC GT 0
  ON TABLE SAVE AS XTOUT
END
```

SAVE ファイルのフィールドとフォーマットは、次のとおりです。

```
ALPHANUMERIC RECORD NAMED  XTOUT
FIELDNAME                       ALIAS         FORMAT        LENGTH
PCT_INC                         PI            F6.2           6
LONGPCK                                       P25.2          25
ALPHAPCK                                      A13            13
TOTAL                                                        44
SAVED...
```

15

簡略数値関数

新しい数値関数が導入され、関数が分かりやすくなり、必須の引数の入力が容易になりました。これらの関数では、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。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。

注意

- 簡略数値関数は、ダイアログマネージャでサポートされます。

トピックス

ASCII - 文字列の左端文字の ASCII コードを取得

ASCII は、文字例を取得し、左端文字の ASCII コードを整数フォーマットで返します。

構文　文字列の左端文字の ASCII コードを取得

```
ASCII(charexp)
```

説明

`charexp`

文字列です。

例　文字列の左端文字の ASCII コードを取得

次のリクエストでは、ASCII 関数によって、CATEGORY フィールドの左端の文字の ASCII コードを取得します。

```
TABLE FILE GGSALES
SUM DOLLARS NOPRINT
AND COMPUTE
ASCII_CODE/I9 = ASCII(CATEGORY);
BY CATEGORY
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果は次のとおりです。

Category	ASCII_CODE
Coffee	67
Food	70
Gifts	71

CEILING - 特定の値以上の最小整数値の取得

CEILING 関数は、特定の値以上の最小整数値を返します。

構文　特定の値以上の最小整数値の取得

```
CEILING(number)
```

説明

`number`

数値

数値です。この数値以上の最小整数値が返されます。出力データタイプは入力データタイプと同一です。

例　特定の値以上の最小整数値の取得

次のリクエストの CEILING 関数は、GROSS_PROFIT_US 値以上の最小整数値を返します。

```
DEFINE FILE WF_RETAIL_LITE
CEIL1/D7.2= CEILING(GROSS_PROFIT_US);
END
TABLE FILE WF_RETAIL_LITE
PRINT GROSS_PROFIT_US/D9.2  CEIL1
ON TABLE SET PAGE NOPAGE
END
```

以下は、出力の一部を示しています。返される値は整数ですが、出力フォーマットは CEIL1 フィールドのフォーマット (D7.2) と同一になります。

```
  Gross Profit     CEIL1
  ------------     -----
        165.00    165.00
         13.99     14.00
         60.99     61.00
        225.98    226.00
         79.99     80.00
         44.59     45.00
         94.30     95.00
        238.50    239.00
        199.99    200.00
         68.99     69.00
         63.58     64.00
        129.99    130.00
         37.49     38.00
         75.99     76.00
         13.99     14.00
        119.00    119.00
        -30.01    -30.00
         54.99     55.00
        189.98    190.00
         44.59     45.00
         91.98     92.00
         89.00     89.00
         59.50     60.00
        129.99    130.00
         54.00     54.00
        109.98    110.00
         98.99     99.00
         98.99     99.00
         99.99    100.00
         44.59     45.00
```

EXPONENT - 定数 e を指数でべき乗

EXPONENT 関数は、定数 e を指数でべき乗します。

構文　　定数 e を指数でべき乗

```
EXPONENT(power)
```

説明

`power`

数値

e をべき乗する指数です。出力データタイプは数値です。

例　定数 e を指数でべき乗

次のリクエストは、e の値、および e を 5 でべき乗した値を出力します。

```
DEFINE FILE WF_RETAIL_LITE
EXP1/D12.5 = EXPONENT(1);
EXP2/D12.5 = EXPONENT(5);
END
TABLE FILE WF_RETAIL_LITE
PRINT EXP1 EXP2
BY BUSINESS_REGION AS Region
WHERE BUSINESS_REGION EQ 'EMEA'
WHERE RECORDLIMIT EQ 1
ON TABLE SET PAGE NOPAGE
END
```

下図は、出力結果を示しています。

Region	EXP1	EXP2
EMEA	2.71828	148.41316

FLOOR - 特定の値以下の最大整数値を取得

FLOOR 関数は、特定の値以下の最大整数値を返します。

構文　特定の値以下の最大整数値の取得

```
FLOOR(number)
```

説明

`number`

数値

数値です。この値数値以下の最大整数値が返されます。出力データタイプは入力データタイプと同一です。

例　特定の値以下の最大整数値の取得

次のリクエストの FLOOR 関数は、GROSS_PROFIT_US 値以下の最大整数値を返します。

```
DEFINE FILE WF_RETAIL_LITE
FLOOR1/D7.2= FLOOR(GROSS_PROFIT_US);
END
TABLE FILE WF_RETAIL_LITE
PRINT GROSS_PROFIT_US/D9.2  FLOOR1
ON TABLE SET PAGE NOPAGE
END
```

以下は、出力結果の一部を示しています。返される値は整数ですが、出力フォーマットは FLOOR1 フィールドのフォーマット (D7.2) と同一になります。

```
  Gross Profit    FLOOR1
  ------------    ------
        165.00    165.00
         13.99     13.00
         60.99     60.00
        225.98    225.00
         79.99     79.00
         44.59     44.00
         94.30     94.00
        238.50    238.00
        199.99    199.00
         68.99     68.00
         63.58     63.00
        129.99    129.00
         37.49     37.00
         75.99     75.00
         13.99     13.00
        119.00    119.00
        -30.01    -31.00
         54.99     54.00
        189.98    189.00
         44.59     44.00
         91.98     91.00
         89.00     89.00
         59.50     59.00
        129.99    129.00
         54.00     54.00
        109.98    109.00
         98.99     98.00
         98.99     98.00
         99.99     99.00
         44.59     44.00
```

LOG10 - 10 を底とする対数の計算

LOG10 関数は、数式から 10 を底とする対数を返します。

構文　10 を底とする対数の計算

```
LOG10(num_exp)
```

説明

`num_exp`

数値

10 を底とする対数を計算する数値です。

例　10 を底とする対数の計算

次のリクエストは、現在の給与の 10 を底とする対数を計算します。

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL AND COMPUTE
LOG_CURR_SAL/D12.6 = LOG10(CURR_SAL);
BY LAST_NAME BY FIRST_NAME
WHERE DEPARTMENT EQ 'PRODUCTION';
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果は次のとおりです。

LAST_NAME	FIRST_NAME	CURR_SAL	LOG_CURR_SAL
BANNING	JOHN	$29,700.00	4.472756
IRVING	JOAN	$26,862.00	4.429138
MCKNIGHT	ROGER	$16,100.00	4.206826
ROMANS	ANTHONY	$21,120.00	4.324694
SMITH	RICHARD	$9,500.00	3.977724
STEVENS	ALFRED	$11,000.00	4.041393

MOD - 除算の剰余を計算

MOD 関数は、除算の剰余を計算します。出力データタイプは入力データタイプと同一です。

構文　除算の剰余を計算

```
MOD(dividend, divisor)
```

説明

dividend

数値

除算される値です (被除数)。

注意：返される値の符号は、除算される値の符号と同一です。

divisor

数値

除算する値です (除数)。

除数が 0 (ゼロ) の場合、被除数が返されます。

例　除算の剰余を計算

次のリクエストの MOD 関数は、PRICE_DOLLARS 値を DAYSDELAYED 値で除算した剰余を返します。

```
DEFINE FILE WF_RETAIL_LITE
MOD1/D7.2= MOD(PRICE_DOLLARS, DAYSDELAYED);
END
TABLE FILE WF_RETAIL_LITE
PRINT PRICE_DOLLARS/D7.2  DAYSDELAYED/I5 MOD1
WHERE DAYSDELAYED GT 1
ON TABLE SET PAGE NOPAGE
ON TABLE PCHOLD FORMAT WP
END
```

以下は、出力結果の一部を示しています。

```
  Price     Days
Dollars  Delayed      MOD1
-------  -------      ----
 399.00        3       .00
 489.99        3       .99
 786.50        2       .50
 599.99        4      3.99
  29.99        4      1.99
 169.00        2      1.00
 219.99        2      1.99
 280.00        3      1.00
  79.99        4      3.99
 145.99        2      1.99
 399.99        3       .99
 349.99        3      1.99
 169.00        3      1.00
```

POWER - 値を指数でべき乗

POWER 関数は、底の値を指数でべき乗します。

構文　値を指数でべき乗

```
POWER(base, power)
```

説明

`base`

数値

指数でべき乗する値です。出力値のデータタイプは、底の値のデータタイプと同一になります。底の値が整数の場合、負の指数値を指定すると、結果の末尾が切り捨てられます。

`power`

数値

底の値をべき乗する指数です。

例　底の値を指数でべき乗

次のリクエストの POWER 関数は、COGS_US/20.00 値を底として、DAYSDELAYED に格納されている指数でべき乗した結果を返します。

```
DEFINE FILE WF_RETAIL_LITE
BASE=COGS_US/20.00;
POWER1= POWER(COGS_US/20.00,DAYSDELAYED);
END
TABLE FILE WF_RETAIL_LITE
PRINT BASE IN 15 DAYSDELAYED POWER1
BY PRODUCT_CATEGORY
WHERE PRODUCT_CATEGORY EQ 'Computers'
WHERE DAYSDELAYED NE 0
ON TABLE SET PAGE NOPAGE
END
```

以下は、出力結果の一部を示しています。

```
  Product                                Days
  Category                 BASE          Delayed          POWER1
  --------                 ----          -------          ------
  Computers               12.15                3       1,793.61
                          16.70                2         278.89
                           8.35                1           8.35
                           8.10                2          65.61
                           4.05                1           4.05
                           4.05                2          16.40
                           4.05                4         269.04
                           8.35                1           8.35
                          16.70                1          16.70
                           8.35                3         582.18
                           8.35                1           8.35
                           4.05                1           4.05
                           4.05                1           4.05
                           8.35                4       4,861.23
                           8.35               -1            .12
                           8.35                1           8.35
                           8.35                3         582.18
```

ROUND - 桁数を指定した数値の端数処理

ROUND 関数は、指定された数値式および整数から、この整数の桁数で端数処理された数値式を返します。小数点以下桁数が負の値の場合、小数点の左側で四捨五入されます。

構文　　指定された桁数での数値の端数処理

```
ROUND(num_exp, count)
```

説明

`num_exp`

数値

端数処理を行う数値式です。

`count`

数値

数値式を端数処理する小数点以下の桁数です。小数点以下の桁数が負の値の場合、小数点の左側で四捨五入されます。

例　指定された桁数での数値の端数処理

次のリクエストは LISTPR フィールドを小数点以下 0 (ゼロ) 桁、NEWLISTPR フィールドを小数点以下 1 桁および -2 桁で四捨五入します。

```
TABLE FILE MOVIES
PRINT LISTPR
AND  COMPUTE
NEWLISTPR/D12.3 = LISTPR * 99;
ROUND_ZERO/D12.3 = ROUND(LISTPR, 0);
ROUND_PLUS1/D12.3 = ROUND(NEWLISTPR, 1);
ROUND_MINUS1/D12.3 = ROUND(NEWLISTPR, -2);
BY MOVIECODE
WHERE RECORDLIMIT EQ 3
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果は次のとおりです。

MOVIECODE	LISTPR	NEWLISTPR	ROUND_ZERO	ROUND_PLUS1	ROUND_MINUS2
001MCA	19.95	1,975.050	20.000	1,975.100	2,000.000
005WAR	24.98	2,473.020	25.000	2,473.000	2,500.000
020TUR	39.99	3,959.010	40.000	3,959.000	4,000.000

SIGN - 数値の符号を取得

SIGN 関数は、数値の引数に対して、負の数の場合は -1、0 (ゼロ) の場合は 0、正の数の場合は 1 の値を返します。

構文　数値の符号を取得

```
SIGN(number)
```

説明

`number`

数値を含むフィールドまたは数字です。

例　数値の符号を取得

次のリクエストは、正の数、負の数、および 0 (ゼロ) の符号を返します。

```
TABLE FILE GGSALES
SUM DOLLARS NOPRINT AND COMPUTE
PLUSDOLL/I9 = IF DOLLARS GT 12000000 THEN DOLLARS ELSE 0;
SIGN1/I5 = SIGN(PLUSDOLL);
NEGDOLL/I9 = IF DOLLARS LT 12000000 THEN 0 ELSE -DOLLARS;
SIGN2/I5 = SIGN(NEGDOLL);
BY CATEGORY
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果は次のとおりです。

Category	PLUSDOLL	SIGN1	NEGDOLL	SIGN2
Coffee	17231455	1	-17231455	-1
Food	17229333	1	-17229333	-1
Gifts	0	0	0	0

TRUNCATE - 指定された小数点以下桁数での数値の切り捨て

TRUNCATE 関数は、指定された数値式および整数から、この整数の桁数で切り捨てられた数値式を返します。小数点以下桁数がマイナスの場合、小数点の左側で切り捨てられます。

構文　指定された小数点以下桁数での数値の切り捨て

```
TRUNCATE(num_exp, count)
```

説明

`num_exp`

数値

切り捨てを行う数値式です。

`count`

数値

数値式を切り捨てる小数点以下の桁数です。TRUNCATE 関数は、小数点以下桁数がマイナスの場合、小数点の左側で切り捨てます。

例　指定された小数点以下桁数による数値の切り捨て

次のリクエストは、LISTPR フィールドの小数点以下 1 桁および -1 桁を切り捨てます。

```
TABLE FILE MOVIES
PRINT LISTPR
AND  COMPUTE
TRUNCATE_PLUS1/D12.3 = TRUNCATE(LISTPR, 1);
TRUNCATE_MINUS1/D12.3 = TRUNCATE(LISTPR, -1);
BY MOVIECODE
WHERE RECORDLIMIT EQ 3
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果は次のとおりです。

MOVIECODE	LISTPR	TRUNCATE_PLUS1	TRUNCATE_MINUS1
001MCA	19.95	19.900	10.000
005WAR	24.98	24.900	20.000
020TUR	39.99	39.900	30.000

16

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。

多くの関数では、output 引数にフィールド名またはフォーマットを指定することができます。フォーマットを指定する場合、一重引用符 (') で囲みます。 ただし、関数がダイアログマネージャコマンドから呼び出される場合、この引数には常にフォーマットを指定する必要があります。関数の呼び出しおよび引数の指定についての詳細は、45 ページの 「 関数へのアクセスと呼び出し 」 を参照してください。

注意：コンチネンタル 10 進表記 (CDN=ON) では、複数の数値引数は、カンマ (,) とそれに続くブランクで区切る必要があります。

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(in_value)
```

説明

`in_value`
　数値

　絶対値を返す数値です。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　絶対値を計算

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

```
TABLE FILE SALES
PRINT UNIT_SOLD AND DELIVER_AMT AND
COMPUTE DIFF/I5 = DELIVER_AMT - UNIT_SOLD; AND
COMPUTE ABS_DIFF/I5 = ABS(DIFF);BY PROD_CODE
WHERE DATE LE '1017';
END
```

出力結果は次のとおりです。

```
PROD_CODE  UNIT_SOLD  DELIVER_AMT  DIFF  ABS_DIFF
---------  ---------  -----------  ----  --------
B10                30           30     0         0
B17                20           40    20        20
B20                15           30    15        15
C17                12           10    -2         2
D12                20           30    10        10
E1                 30           25    -5         5
E3                 35           25   -10        10
```

ASIS - ブランクと 0 (ゼロ) を区別

ASIS 関数は、ダイアログマネージャ内のブランクと 0 (ゼロ) を区別します。ASIS 関数は、数値文字列定数、数値文字列として定義された変数、および数値として定義されたフィールドを区別します。

ASIS についての詳細は、181 ページの 「 ASIS - ブランクと 0 (ゼロ) を区別 」 を参照してください。

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', output)
```

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。

`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。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。

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL AND COMPUTE
SAL_BAR/A30 = BAR(30, CURR_SAL, 30000, '=', SAL_BAR);BY LAST_NAME BY
FIRST_NAME
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

出力結果は次のとおりです。

```
LAST_NAME       FIRST_NAME          CURR_SAL  SAL_BAR
---------       ----------          --------  -------
BANNING         JOHN              $29,700.00  ============================
IRVING          JOAN              $26,862.00  ==========================
MCKNIGHT        ROGER             $16,100.00  ================
ROMANS          ANTHONY           $21,120.00  =====================
SMITH           RICHARD            $9,500.00  ==========
STEVENS         ALFRED            $11,000.00  ===========
```

例　目盛り付き棒グラフを作成

BAR 関数は、CURR_SAL フィールドの棒グラフを作成します。リクエストは、AS 句を使用して「SAL_BAR」というフィールド名を目盛りで置き換えます。

デフォルトの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このリクエストを実行する場合、プロポーショナル以外の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。

```
SET STYLE=OFF


TABLE FILE EMPLOYEE
HEADING
"CURRENT SALARIES OF EMPLOYEES IN PRODUCTION DEPARTMENT"
"GRAPHED IN THOUSANDS OF DOLLARS"
" "
PRINT CURR_SAL AS 'CURRENT SALARY'
AND COMPUTE
    SAL_BAR/A30 = BAR(30, CURR_SAL, 30000, '=', SAL_BAR);
    AS '    5   10   15   20   25   30,----+----+----+----+----+----+'
BY LAST_NAME AS 'LAST NAME'
BY FIRST_NAME AS 'FIRST NAME'
WHERE DEPARTMENT EQ 'PRODUCTION';
ON TABLE SET PAGE-NUM OFF
ON TABLE SET STYLE * GRID=OFF, $
END
```

出力結果は次のとおりです。

```
CURRENT SALARIES OF EMPLOYEES IN PRODUCTION DEPARTMENT
GRAPHED IN THOUSANDS OF DOLLARS
                                                    5 10 15 20 25 30
LAST NAME     FIRST NAME     CURRENT SALARY ----+----+----+----+----+----+
---------     ----------     -------------- ------------------------------
BANNING       JOHN           $29,700.00     ==============================
IRVING        JOAN           $26,862.00     ==========================
MCKNIGHT      ROGER          $16,100.00     ================
ROMANS        ANTHONY        $21,120.00     =====================
SMITH         RICHARD         $9,500.00     ==========
STEVENS       ALFRED         $11,000.00     ===========
```

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(length, in_value, error, output)
```

説明

`length`
　数値

　パック 10 進数フィールドの長さです。1 から 16 バイトの値を指定します。

`in_value`
　文字

　パック 10 進数フィールドの名前、またはパック 10 進数として確認する値です。この値は、パック 10 進数ではなく、文字として記述されている必要があります。

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。

output
パック 10 進数

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。

例　パック 10 進数を検査

1. 無効なパック 10 進数を含むデータソースを用意します。次の例は、TESTPACK を作成します。TESTPACK には、PACK_SAL フィールドが含まれます。PACK_SAL は文字として定義されていますが、実際にはパック 10 進数を含みます。無効なパック 10 進数は AAA として格納されます。

```
DEFINE FILE EMPLOYEE
PACK_SAL/A8 = IF EMP_ID CONTAINS '123'
      THEN 'AAA' ELSE PCKOUT(CURR_SAL, 8, 'A8');
END


TABLE FILE EMPLOYEE
PRINT DEPARTMENT PACK_SAL BY EMP_ID
ON TABLE SAVE AS TESTPACK
END
```

出力結果は次のとおりです。

```
 NUMBER OF RECORDS IN TABLE=       12  LINES=     12
ALPHANUMERIC  RECORD NAMED  TESTPACK
FIELDNAME                          ALIAS          FORMAT         LENGTH
EMP_ID                             EID            A9                9
DEPARTMENT                         DPT            A10              10
PACK_SAL                                          A8                8
TOTAL                                                               27
```

2. TESTPACK データソース用マスターファイルを作成します。PACK_SAL フィールドを USAGE および ACTUAL 属性で文字として定義します。

```
FILE  = TESTPACK,  SUFFIX = FIX
FIELD = EMP_ID    ,ALIAS = EID,USAGE = A9 ,ACTUAL = A9 ,$
FIELD = DEPARTMENT,ALIAS = DPT,USAGE = A10,ACTUAL = A10,$
FIELD = PACK_SAL  ,ALIAS = PS ,USAGE = A8 ,ACTUAL = A8 ,$
```

3. CHKPCK を使用するリクエストを作成し、PACK_SAL フィールドの値を検査します。その結果を 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。

```
DEFINE FILE TESTPACK
GOOD_PACK/P8CM = CHKPCK(8, PACK_SAL, -999, GOOD_PACK);
END

TABLE FILE TESTPACK
PRINT DEPARTMENT GOOD_PACK BY EMP_ID
END
```

出力結果は次のとおりです。

```
EMP_ID     DEPARTMENT    GOOD_PACK
------     ----------    ---------
071382660  PRODUCTION      $11,000
112847612  MIS             $13,200
117593129  MIS             $18,480
119265415  PRODUCTION       $9,500
119329144  PRODUCTION      $29,700
123764317  PRODUCTION        -$999
126724188  PRODUCTION      $21,120
219984371  MIS             $18,480
326179357  MIS             $21,780
451123478  PRODUCTION        -$999
543729165  MIS              $9,000
818692173  MIS             $27,062
```

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, output)
```

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。

`output`
数値

結果です。この結果のフォーマットは、使用する関数によって決定されます。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。

除数が 0 (ゼロ) の場合、被除数が返されます。

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。

```
TABLE FILE EMPLOYEE
PRINT ACCTNUMBER AND COMPUTE
LAST3_ACCT/I3L = IMOD(ACCTNUMBER, 1000, LAST3_ACCT);
BY LAST_NAME BY FIRST_NAME
WHERE (ACCTNUMBER NE 000000000) AND (DEPARTMENT EQ 'MIS');
END
```

出力結果は次のとおりです。

```
LAST_NAME       FIRST_NAME  ACCTNUMBER  LAST3_ACCT
---------       ----------  ----------  ----------
BLACKWOOD       ROSEMARIE   122850108          108
CROSS           BARBARA     163800144          144
GREENSPAN       MARY        150150302          302
JONES           DIANE       040950036          036
MCCOY           JOHN        109200096          096
SMITH           MARY        027300024          024
```

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, output)
```

説明

`power`
数値

「e」 をべき乗する指数です。

`output`
倍精度浮動小数点数

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。

例　「e」 を N でべき乗

EXP 関数は、「e」を &POW 変数により指定された値 (ここでは 3) でべき乗します。結果の値は 0.5 を定数として端数処理され、最も近い整数が変数 &RESULT に返されます。出力値のフォーマットは D15.3 です。

```
-SET &POW = '3';
-SET &RESULT = EXP(&POW, 'D15.3') + 0.5;-HTMLFORM BEGIN
<HTML>
<BODY>
E TO THE &POW POWER IS APPROXIMATELY &RESULT
</BODY>
</HTML>
-HTMLFORM END
```

出力結果は次のとおりです。

```
E TO THE 3 POWER IS APPROXIMATELY 20
```

EXPN - 指数表記の数値を評価

EXPN 関数は、指数 (科学) 表記の数値リテラルまたはダイアログマネージャ変数を評価します。

構文　　指数表記の数値を評価

```
EXPN(n.nn {E|D} {+|-} p)
```

説明

`n.nn`

数値

数値リテラルです。この定数は、整数、小数点、小数部分の順序で構成されます。

`E, D`

指数表記を表します。E と D は交換することができます。

`+, -`

p の正負を表します。

`p`

整数

n.nn の 10 乗指数です。

注意：EXPN では、output 引数は使用されません。結果のフォーマットは、倍精度浮動小数点数です。

例　　指数表記での数値の評価

次の EXPN 関数は、「1.03E+2」を評価します。

```
EXPN(1.03E+2)
```

結果は、103 です。

FMLCAP - FML 階層キャプションを抽出

FMLCAP 関数は、FML 階層リクエスト内の各行にキャプション値を返します。キャプション値を抽出するためには、マスターファイルで FML 階層を定義し、リクエストで GET CHILDREN、ADD、または WITH CHILDREN オプションを使用して階層データを抽出する必要があります。リクエストの FOR フィールドにキャプションフィールドが定義されていない場合、FMLCAP はブランクの文字列を返します。

FMLCAP 関数は、COMPUTE では使用できますが、DEFINE での使用はお勧めしません。

構文　FMLCAP 関数により FML リクエスト内のキャプションを抽出

```
FMLCAP(fieldname|'format')
```

説明

`fieldname`
　入力フィールド名です。

`'format'`
　キャプション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 (') で囲みます。

例　FMLCAP 関数により FML 階層キャプションを抽出

次のリクエストは、親値 2000 で開始する FML 階層を抽出、統合します。FMLCAP 関数はキャプションを抽出しますが、実際のアカウント数は FOR 値として表示されます。

```
SET FORMULTIPLE = ON
TABLE FILE CENTSTMT
SUM ACTUAL_AMT
COMPUTE CAP1/A30= FMLCAP(GL_ACCOUNT_CAPTION);
FOR GL_ACCOUNT
2000 WITH CHILDREN 2 ADD
END
```

出力結果は次のとおりです。

```
                Actual  CAP1
                ------  ----
2000       313,611,852.  Gross Margin
  2100     187,087,470.  Sales Revenue
    2200    98,710,368.  Retail Sales
    2300    13,798,832.  Mail Order Sales
    2400    12,215,780.  Internet Sales
  2500     100,885,159.  Cost Of Goods Sold
    2600    54,877,250.  Variable Material Costs
    2700     6,176,900.  Direct Labor
    2800     3,107,742.  Fixed Costs
```

FMLFOR - FML タグ値を抽出

FMLFOR 関数は、FML リクエストの各行に関連付けられたタグ値を抽出します。FML 行が OR 句によるデータレコードの合計として生成された場合、FMLFOR はリストに指定された 1 つ目の値を返します。OR 句が FML 階層 ADD コマンドにより生成された場合、FMLFOR は ADD コマンドに指定された親に関連付けられたタグ値を返します。

FMLFOR 関数は、COMPUTE で使用できますが、DEFINE では使用できません。DEFINE で使用すると、ブランク値が生成されます。

構文　FML タグ値を抽出

```
FMLFOR(output)
```

説明

`output`
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。

例　FMLFOR により FML タグ値を抽出

```
SET FORMULTIPLE = ON
TABLE FILE LEDGER
SUM AMOUNT
COMPUTE RETURNEDFOR/A8 = FMLFOR('A8');
FOR ACCOUNT
1010                    OVER
1020                    OVER
1030                    OVER
BAR                     OVER
1030 OR 1020 OR 1010
END
```

出力結果は次のとおりです。

```
      AMOUNT  RETURNEDFOR
      ------  -----------
1010   8,784  1010
1020   4,494  1020
1030   7,961  1030
      ------  --------
1010  21,239  1030
```

FMLINFO - FOR 値を取得

FMLINFO 関数は、FML レポートの各行に関連付けられた FOR 値を返します。FMLINFO 関数を使用することにより、COMPUTE コマンドで適切な FOR 値を使用してレポートの各行にドリルダウンや符号の変更を実行することができます。これは、行が OR リストや FML (Financial Modeling Language) 階層 ADD コマンドにより作成された集計行でも可能です。

注意：SET パラメータ FORMULTIPLE=ON により、受信されるレコードを FML レポートの複数行で使用することができます。

構文　FML リクエストにより FOR 値を取得

```
FMLINFO('FORVALUE', output)
```

説明

`'FORVALUE'`
文字

FML レポートで各行に関連付けられた FOR 値を返します。FML 行が OR 句によるデータレコードの合計として生成された場合、FMLINFO は値リストに指定された 1 つ目の FOR 値を返します。OR 句が FML 階層 ADD コマンドにより生成された場合、FMLINFO は ADD コマンドに指定された親に関連付けられた FOR 値を返します。

`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。

例　取得した FOR 値を FML 階層行に出力

次のリクエストは、CENTSYSF データソースでのアカウント数が 2500 未満の NAT_AMOUNT フィールドの負のフィールドである PRINT_AMT を作成します。CENTGL データソースには、CENTSYSF の階層情報が含まれています。このため、このリクエストでは、CENTGL は CENTSYSF に結合されます。

```
SET FORMULTIPLE = ON
JOIN SYS_ACCOUNT IN CENTGL TO ALL SYS_ACCOUNT IN CENTSYSF
TABLE FILE CENTGL
SUM NAT_AMOUNT/D10 IN 30
COMPUTE PRINT_AMT/D10 = IF FMLINFO('FORVALUE','A7') LT '2500'
     THEN 0-NAT_AMOUNT ELSE NAT_AMOUNT;
COMPUTE FORV/A4 = FMLINFO('FORVALUE', 'A4');
COMPUTE ACTION/A9  = IF FORV LT '2500'
      THEN 'CHANGED' ELSE 'UNCHANGED';
FOR GL_ACCOUNT
2000 WITH CHILDREN 2 ADD AS CAPTION
END
```

注意：WITH CHILDREN ADD コマンドに指定された親値 (2000) がレポートの 1 行目に返されます。2 行目以降も FMLINFO により返された親値により階層にサブセクションとして統合されます。

```
                                  Month
                                 Actual      PRINT_AMT  FORV  ACTION
                                 ------      ---------  ----  ------
Gross Margin                  -25,639,223    25,639,223  2000  CHANGED
  Sales Revenue               -62,362,490    62,362,490  2100  CHANGED
    Retail Sales              -49,355,184    49,355,184  2200  CHANGED
    Mail Order Sales           -6,899,416     6,899,416  2300  CHANGED
    Internet Sales             -6,107,890     6,107,890  2400  CHANGED
  Cost Of Goods Sold           36,723,267    36,723,267  2500  UNCHANGED
    Variable Material Costs    27,438,625    27,438,625  2600  UNCHANGED
    Direct Labor                6,176,900     6,176,900  2700  UNCHANGED
    Fixed Costs                 3,107,742     3,107,742  2800  UNCHANGED
```

例　OR 句での FMLINFO の使用

集計行に出力された FOR 値は 1010 ですが、FMLINFO は OR リストに指定された 1 つ目の値である 1030 を返します。

```
SET FORMULTIPLE = ON
TABLE FILE LEDGER
SUM AMOUNT
COMPUTE RETURNEDFOR/A8 = FMLINFO('FORVALUE','A8');
FOR ACCOUNT
1010                    OVER
1020                    OVER
1030                    OVER
BAR                     OVER
1030 OR 1020 OR 1010
END
```

出力結果は次のとおりです。

```
           AMOUNT      RETURNEDFOR

 1010       8,784      1010
 1020       4,494      1020
 1030       7,961      1030

           ------      ---------

 1010      21,239      1030
```

FMLLIST - FML タグリストを抽出

FMLLIST 関数は、FML リクエスト内の各行のタグの完全なリストを含む文字列を返します。行にタグ値が 1 つだけ存在する場合は、その値が返されます。

FMLLIST 関数は、COMPUTE で使用できますが、DEFINE では使用できません。DEFINE で使用すると、ブランク値が生成されます。

構文　FML タグリストを抽出

```
FMLLIST('A4096V')
```

説明

`'A4096V'`
　必須の引数です。

例　FMLLIST により FML タグリストを抽出

```
SET FORMULTIPLE=ON
TABLE FILE LEDGER
HEADING
"TEST OF FMLLIST"
" "
SUM AMOUNT
COMPUTE LIST1/A36 = FMLLIST('A4096V');
FOR ACCOUNT
'1010'                 OVER
'1020'                 OVER
'1030'                 OVER
BAR                    OVER
'1030' OR '1020' OR '1010'
END
```

出力結果は次のとおりです。

```
TEST OF FMLLIST
      AMOUNT  LIST1
      ------  -----
1010   8,784  1010
1020   4,494  1020
1030   7,961  1030
      ------  -----------------------------------
1010  21,239  1010 OR 1020 OR 1030
```

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(in_value)
```

説明

`in_value`
数値

整数構成要素を取得する値です。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。

```
TABLE FILE EMPLOYEE
SUM DED_AMT AND COMPUTE
INT_DED_AMT/I9 = INT(DED_AMT);BY LAST_NAME BY FIRST_NAME
WHERE (DEPARTMENT EQ 'MIS') AND (PAY_DATE EQ 820730);
END
```

出力結果は次のとおりです。

```
LAST_NAME          FIRST_NAME           DED_AMT   INT_DED_AMT
---------          ----------           -------   -----------
BLACKWOOD          ROSEMARIE          $1,261.40          1261
CROSS              BARBARA            $1,668.69          1668
GREENSPAN          MARY                 $127.50           127
JONES              DIANE                $725.34           725
SMITH              MARY                 $334.10           334
```

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(in_value)
```

説明

`in_value`
数値

自然対数を計算する値です。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。in_value 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。

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL AND COMPUTE
LOG_CURR_SAL/D12.2 = LOG(CURR_SAL);BY LAST_NAME BY FIRST_NAME
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

出力結果は次のとおりです。

```
LAST_NAME       FIRST_NAME         CURR_SAL     LOG_CURR_SAL
---------       ----------         --------     ------------
BANNING         JOHN            $29,700.00            10.30
IRVING          JOAN            $26,862.00            10.20
MCKNIGHT        ROGER           $16,100.00             9.69
ROMANS          ANTHONY         $21,120.00             9.96
SMITH           RICHARD          $9,500.00             9.16
STEVENS         ALFRED          $11,000.00             9.31
```

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}(value1, value2, ...)
```

説明

`MAX`
最大値を返します。

`MIN`
最小値を返します。

`value1, value2`
数値

最大値または最小値を取得する値です。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　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。

```
TABLE FILE EMPLOYEE
PRINT ED_HRS AND COMPUTE
MIN_EDHRS_30/D12.2 = MIN(ED_HRS, 30);BY LAST_NAME BY FIRST_NAME
WHERE DEPARTMENT EQ 'MIS';
END
```

出力結果は次のとおりです。

```
LAST_NAME       FIRST_NAME  ED_HRS     MIN_EDHRS_30
---------       ----------  ------     ------------
BLACKWOOD       ROSEMARIE    75.00            30.00
CROSS           BARBARA      45.00            30.00
GREENSPAN       MARY         25.00            25.00
JONES           DIANE        50.00            30.00
MCCOY           JOHN           .00              .00
SMITH           MARY         36.00            30.00
```

MIRR - 修正内部利益率を計算

MIRR 関数は定期的キャッシュフローの修正内部収益率の演算を実行します。

構文　修正内部利益率を計算

```
TABLE FILE ...
{PRINT|SUM} field ...COMPUTE rrate/fmt = MIRR(cashflow, finrate,
reinvrate, output);
WITHIN {sort_field|TABLE}
```

説明

`field ...`
レポートの出力結果に表示するフィールドです。

`rrate`
演算済み利益率を含むフィールドです。

`fmt`
利益率のフォーマットです。データタイプは必ず D (倍精度小数点数) に設定します。

`cashflow`
数値フィールドです。各値は、1 つの期間においての支出 (負の値) 、または収入 (正の値) を表します。一連のキャッシュフローの計算を正しく実行するためには、値を正しい順序で配列します。各キャッシュフローに対応する日付は、等間隔で、時系列順に配列します。計算には *cashflow* フィールドに少なくとも 1 つの負の値および 1 つの正の値が必要です。値がすべて正の値、またはすべて負の値である場合、結果として 0 (ゼロ) が返されます。

`finrate`
負のキャッシュフローの利率です。この値は 0 から 1 の小数で表し、負の数値は無効です。この値は、収益率の演算を実行する各ソートグループで一定である必要がありますが、ソートグループ間に異なる値を使用することも可能です。

`reinvrate`
正のキャッシュフローの再投資値です。この値は 0 から 1 の小数で表し、負の数値は無効です。この値は各ソートグループで一定である必要がありますが、ソートグループ間での変更は可能です。この値は、収益率の演算を実行する各ソートグループで一定である必要がありますが、ソートグループ間に異なる値を使用することも可能です。

`output`
変換後の日付を含むフィールド名、またはフォーマットです。フォーマットは一重引用符 (') で囲みます。

`sort_field`
レポート出力結果のソート、および関数の計算を別に実行する行のサブセットにグループ化するためのフィールドです。関数の計算にレポート出力結果のすべての行を含める場合は、WITHIN TABLE を使用します。これには WITHIN 句が必要です。

参照　MIRR 関数使用上の注意

- この関数は、WITHIN 句を伴う COMPUTE コマンドのみで使用可能です。
- キャッシュフローフィールドには、少なくとも 1 つの負の値および 1 つの正の値を含めます。
- 日付は等間隔に配置します。
- キャッシュフローおよび日付の値を省略することはできません。

例　修正内部利益率を計算

次のリクエストは、製品カテゴリの修正内部利益率の計算を実行します。ここでは金融諸費用として 10 パーセント、および再投資率 10 パーセントを想定します。このリクエストは、日付でソートすることで、正しいキャッシュフローの計算が実行されるようにします。この関数で返された率を 100 倍し、小数値をパーセントで表します。フォーマットにパーセント記号 (%) が含まれています。ここではパーセント記号の表示は行われますが、パーセントの計算は行われません。

日付単位のキャッシュフローを作成するために、値を加算します。各カテゴリに関数で必要な負の値 NEWDOLL を定義します。

```
DEFINE FILE GGSALES
 SDATE/YYM = DATE;
 SYEAR/Y = SDATE;
 NEWDOLL/D12.2 = IF DATE LT '19970401' THEN -1 * DOLLARS ELSE DOLLARS;
END
TABLE FILE GGSALES
  SUM NEWDOLL
  COMPUTE RRATE/D7.2% = MIRR(NEWDOLL, .1, .1, RRATE) * 100;
  WITHIN CATEGORY
  BY CATEGORY
  BY SDATE
  WHERE SYEAR EQ 97
END
```

WITHIN CATEGORY のため、カテゴリごとに異なる率が計算されています。下図は出力結果の一部です。

```
Category      SDATE           NEWDOLL      RRATE
--------      -----           -------      -----
Coffee        1997/01     -801,123.00     15.11%
              1997/02     -682,340.00     15.11%
              1997/03     -765,078.00     15.11%
              1997/04      691,274.00     15.11%
              1997/05      720,444.00     15.11%
              1997/06      742,457.00     15.11%
              1997/07      747,253.00     15.11%
              1997/08      655,896.00     15.11%
              1997/09      730,317.00     15.11%
              1997/10      724,412.00     15.11%
              1997/11      620,264.00     15.11%
              1997/12      762,328.00     15.11%
Food          1997/01     -672,727.00     16.24%
              1997/02     -699,073.00     16.24%
              1997/03     -642,802.00     16.24%
              1997/04      718,514.00     16.24%
              1997/05      660,740.00     16.24%
              1997/06      734,705.00     16.24%
              1997/07      760,586.00     16.24%
```

レポートデータのすべての対する修正内部利益率の計算を実行するには、WITHIN TABLE を使用します。この場合、データを CATEGORY でソートする必要はありません。

```
DEFINE FILE GGSALES
 SDATE/YYM = DATE;
 SYEAR/Y = SDATE;
 NEWDOLL/D12.2 = IF DATE LT '19970401' THEN -1 * DOLLARS ELSE DOLLARS;
END

TABLE FILE GGSALES
  SUM NEWDOLL
  COMPUTE RRATE/D7.2% = MIRR(NEWDOLL, .1, .1, RRATE) * 100;
  WITHIN TABLE
  BY SDATE
  WHERE SYEAR EQ 97
END
```

出力結果は次のとおりです。

```
SDATE            NEWDOLL       RRATE
-----            -------       -----
1997/01   -1,864,129.00      15.92%
1997/02   -1,861,639.00      15.92%
1997/03   -1,874,439.00      15.92%
1997/04    1,829,838.00      15.92%
1997/05    1,899,494.00      15.92%
1997/06    1,932,630.00      15.92%
1997/07    2,005,402.00      15.92%
1997/08    1,838,863.00      15.92%
1997/09    1,893,944.00      15.92%
1997/10    1,933,705.00      15.92%
1997/11    1,865,982.00      15.92%
1997/12    2,053,923.00      15.92%
```

NORMSDST および NORMSINV - 標準正規分布の計算

NORMSDST 関数と NORMSINV 関数は、標準正規分布曲線上で計算を実行します。NORMSDST 関数は、正規化値以下のデータ値の百分率を計算します。NORMSINV 関数は、NORMSDST 関数の逆で、標準正規分布曲線の百分位数の上限境界となる正規化値を計算します。

NORMSDST - 累積標準正規分布関数を計算

NORMSDST 関数は、標準正規分布上で計算を実行し、正規化値以下のデータ値の百分率を求めます。正規化数値は、標準正規分布曲線の平均値の標準偏差内の X 軸上の点です。正規分布データ内の百分位数を決定する際に役立ちます。

NORMSINV 関数は、NORMSDST の逆です。NORMSINV についての詳細は、489 ページの「 NORMSINV - 逆累積標準正規分布を計算 」 を参照してください。

NORMSDST の結果は 6 桁の倍精度小数点数として返されます。

標準正規分布曲線は、平均値が 0 (ゼロ) で、標準偏差が 1 の正規分布です。この曲線下の領域の合計値は 1 です。標準正規分布の X 軸上の点は、「正規化数」と呼ばれます。データが正規分布されている場合、データ点を正規化数に変換することにより、ロースコア以下のスコアの百分率を得ることができます。

正規分布データの値 (ロースコア) を対応する正規化数 (z スコア) に変換するには、次の式を実行します。

```
z = (raw_score - mean)/standard_deviation
```

z スコアをロースコアに変換するには、次の式を使用します。

```
raw_score = z * standard_deviation + mean
```

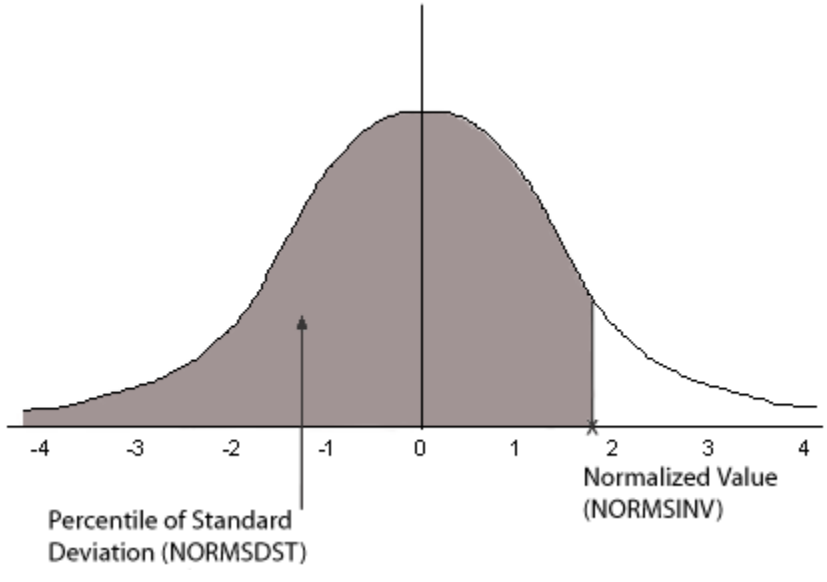
データ点 xi (i は 1 から n) の平均値は、次のようになります。

$$(\Sigma x_i)/n$$

データ点 xi (i は 1 から n) の標準偏差は、次のようになります。

$$SQRT(((\Sigma x_i^2 - (\Sigma x_i)^2/n)/(n - 1)))$$

下図は、NORMSDST 関数と NORMSINV 関数の結果を示しています。



参照　正規分布の特性

多くの一般的な測定は、正規分布です。データ値の正規分布は、釣鐘曲線になります。正規分布の作成には、平均値と標準偏差の 2 つの指標が必要です。

- 平均値は曲線の中心点です。
- 標準偏差は、平均値から変曲点 (曲線が方向を変更する点) までの距離で、曲線の広がりを示します。

構文　累積標準正規分布関数を計算

```
NORMSDST(value, 'D8');
```

説明

`value`

正規化された値です。

`D8`

結果のフォーマットです。関数により返される値のフォーマットは倍精度小数点数です。有効な任意の数値フォーマットを割り当てることができます。

例　NORMSDST 関数の使用

NORMSDST 関数は、Z 値を計算し、百分位数を取得します。

```
DEFINE FILE GGPRODS
-* CONVERT SIZE FIELD TO DOUBLE PRECISION
X/D12.5 = SIZE;
END
TABLE FILE GGPRODS
SUM X NOPRINT CNT.X NOPRINT
-* CALCULATE MEAN AND STANDARD DEVIATION
COMPUTE NUM/D12.5 = CNT.X; NOPRINT
COMPUTE MEAN/D12.5 = AVE.X; NOPRINT
COMPUTE VARIANCE/D12.5 = ((NUM*ASQ.X) - (X*X/NUM))/(NUM-1); NOPRINT
COMPUTE STDEV/D12.5 = SQRT(VARIANCE); NOPRINT
PRINT SIZE X NOPRINT
-* COMPUTE NORMALIZED VALUES AND USE AS INPUT TO NORMSDST FUNCTION
COMPUTE Z/D12.5 = (X - MEAN)/STDEV;
COMPUTE NORMSD/D12.5 = NORMSDST(Z, 'D8');
BY PRODUCT_ID NOPRINT
END
```

出力結果は次のとおりです。

```
Size              Z         NORMSD
----              -         ------
  16        -.07298         .47091
  12        -.80273         .21106
  12        -.80273         .21106
  20         .65678         .74434
  24        1.38654         .91721
  20         .65678         .74434
  24        1.38654         .91721
  16        -.07298         .47091
  12        -.80273         .21106
   8       -1.53249         .06270
```

NORMSINV - 逆累積標準正規分布を計算

NORMSINV 関数は、標準正規分布で計算を実行し、標準正規分布上での百分位数の上限を形成する正規化値を求めます。これは、NORMSDST の逆です。NORMSDST についての詳細は、486 ページの 「 NORMSDST - 累積標準正規分布関数を計算 」 を参照してください。

NORMSINV の結果は、6 桁の倍精度小数点数として返されます。

構文　逆累積標準正規分布関数を計算

```
NORMSINV(value, 'D8');
```

説明

`value`

標準正規分布上の百分位数を表す 0 (ゼロ) と 1 の間の数値です。

`D8`

結果のフォーマットです。関数により返される値のフォーマットは倍精度小数点数です。有効な任意の数値フォーマットを割り当てることができます。

例　NORMSINV 関数の使用

NORMSDST 関数は、Z フィールドの百分位数を求めます。次に NORMSINV 関数が、この百分位数を正規化値として返します。

```
DEFINE FILE GGPRODS
-* CONVERT SIZE FIELD TO DOUBLE PRECISION
X/D12.5 = SIZE;
END
TABLE FILE GGPRODS
SUM X NOPRINT CNT.X NOPRINT
-* CALCULATE MEAN AND STANDARD DEVIATION
COMPUTE NUM/D12.5 = CNT.X; NOPRINT
COMPUTE MEAN/D12.5 = AVE.X; NOPRINT
COMPUTE VARIANCE/D12.5 = ((NUM*ASQ.X) - (X*X/NUM))/(NUM-1); NOPRINT
COMPUTE STDEV/D12.5 = SQRT(VARIANCE); NOPRINT
PRINT SIZE X NOPRINT
-* COMPUTE NORMALIZED VALUES AND USE AS INPUT TO NORMSDST FUNCTION
-* THEN USE RETURNED VALUES AS INPUT TO NORMSINV FUNCTION
-* AND CONVERT BACK TO DATA VALUES
COMPUTE Z/D12.5 = (X - MEAN)/STDEV;
COMPUTE NORMSD/D12.5 = NORMSDST(Z, 'D8');
COMPUTE NORMSI/D12.5 = NORMSINV(NORMSD, 'D8');
COMPUTE DSIZE/D12 = NORMSI * STDEV + MEAN;
BY PRODUCT_ID NOPRINT
END
```

出力では、NORMSINV 関数が NORMSDST 関数と逆方向の計算を実行した結果、元の値が返されています。

```
Size              Z          NORMSD          NORMSI           DSIZE
----              -          ------          ------           -----
  16        -.07298          .47091        -.07298              16
  12        -.80273          .21106        -.80273              12
  12        -.80273          .21106        -.80273              12
  20         .65678          .74434         .65678              20
  24        1.38654          .91721        1.38654              24
  20         .65678          .74434         .65678              20
  24        1.38654          .91721        1.38654              24
  16        -.07298          .47091        -.07298              16
  12        -.80273          .21106        -.80273              12
   8       -1.53249          .06270       -1.53249               8
```

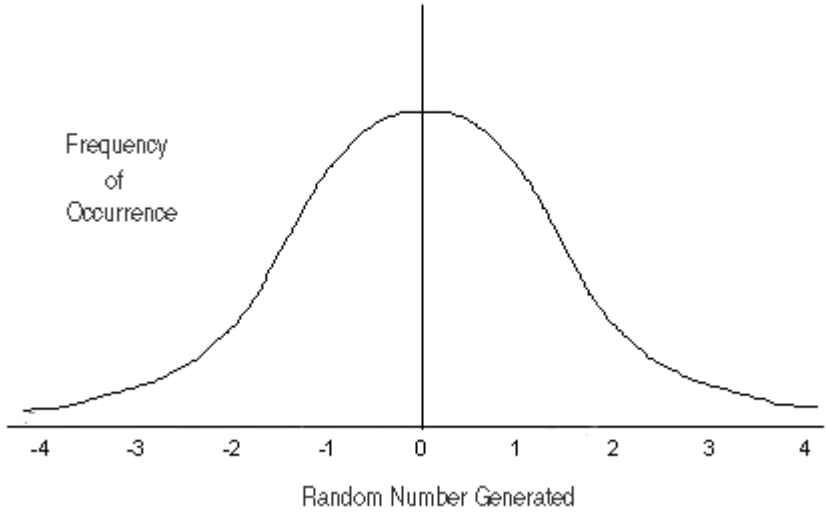
PRDNOR および PRDUNI - 再生可能な乱数を生成

PRDNOR および PRDUNI 関数は、再生可能な乱数を倍精度浮動小数点数で生成します。

- PRDNOR は、相加平均が 0 (ゼロ) で標準偏差が 1 の、正規分布を持つ再生可能乱数を倍精度浮動小数点数で生成します。

PRDNOR 関数が数値を多数生成した場合、これらの数値は次のようになります。

- これらの数値は、下図のように釣鐘曲線で表されます。釣鐘曲線の最高点は 0 (ゼロ) です。これは、0 (ゼロ) に近い数値が 0 (ゼロ) から遠い数値よりも多いことを示します。



  - 数値の平均値は 0 (ゼロ) に近似します。
  - 数値の大きさに制限はありませんが、ほとんどの数値は 3 から -3 までの値となります。
- PRDUNI は、0 (ゼロ) と 1 の間に均等に分布する再生可能な乱数を生成します。0 (ゼロ) と 1 の間の数値は、すべて等しい確率で生成されます。

構文　再生可能な乱数を生成

```
{PRDNOR|PRDUNI}(seed, output)
```

説明

`PRDNOR`
相加平均が 0 (ゼロ) で標準偏差が 1 の、正規分布を持つ再生可能乱数を倍精度浮動小数点数で生成します。

`PRDUNI`
0 (ゼロ) と 1 の間に均等に分布する再生可能な乱数を倍精度浮動小数点数で生成します。

seed
数値

9 バイト以内のシード、またはシードを含むフィールドです。シードは整数に切り捨てられます。

output
倍精度浮動小数点数

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。

例 再生可能な乱数を生成

PRDNOR 関数は、乱数を割り当て、RAND に格納します。これらの値は、LAST_NAME および FIRST_NAME フィールド内の値により識別された従業員のレコードを任意に 5 つ抽出するために使用されます。シードは 40 です。異なる数値セットを生成するには、シードを変更します。

```
DEFINE FILE EMPLOYEE
RAND/D12.2 WITH LAST_NAME = PRDNOR(40, RAND);END

TABLE FILE EMPLOYEE
PRINT LAST_NAME AND FIRST_NAME
BY HIGHEST 5 RAND
END
```

出力結果は次のとおりです。

```
RAND   LAST_NAME         FIRST_NAME
----   ---------         ----------
1.38   STEVENS           ALFRED
1.12   MCCOY             JOHN
 .55   SMITH             RICHARD
 .21   JONES             DIANE
 .01   IRVING            JOAN
```

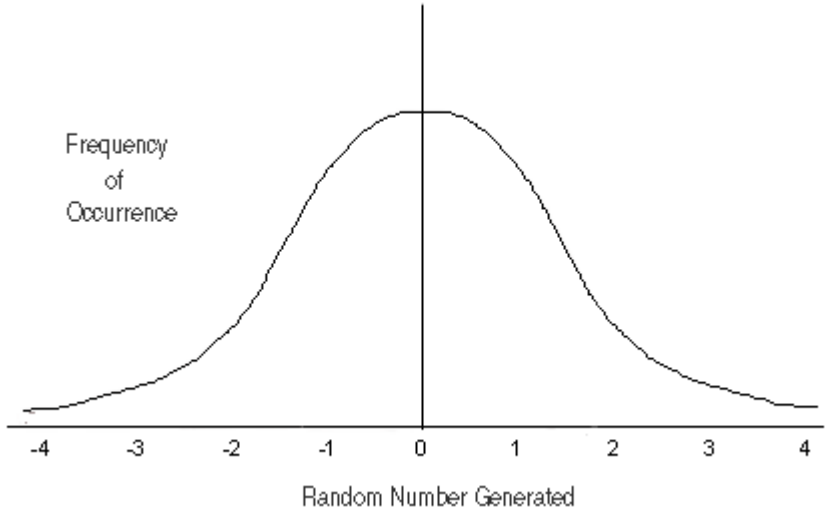
RDNORM および RDUNIF - 乱数を生成

RDNORM および RDUNIF 関数は、乱数を生成します。

- RDNORM は、相加平均が 0 (ゼロ) で標準偏差が 1 の、正規分布を持つ乱数を倍精度浮動小数点数で生成します。

  RDNORM 関数は、数値 (1 から 32768) を多数生成します。

  - これらの数値は、下図のように釣鐘曲線で表されます。釣鐘曲線の最高点は 0 (ゼロ) です。これは、0 (ゼロ) に近い数値が 0 (ゼロ) から遠い数値よりも多いことを示します。



  - 数値の平均値は 0 (ゼロ) に近似します。
  - 数値の大きさに制限はありませんが、ほとんどの数値は 3 から -3 までの値となります。

- RDUNIF は、0 と 1 の間に均等に分布する乱数を生成します。0 と 1 の間の数値は、すべて等しい確率で生成されます。

構文　乱数を生成

```
{RDNORM|RDUNIF}(output)
```

説明

`RDNORM`
相加平均が 0 (ゼロ) で標準偏差が 1 の、正規分布を持つ乱数を倍精度浮動小数点数で生成します。

RDUNIF
0 と 1 の間に均等に分布する乱数を倍精度浮動小数点数で生成します。

output
倍精度浮動小数点数

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。

例 乱数を生成

RDNORM 関数は、乱数を割り当て、RAND に格納します。これらの値は、LAST NAME および FIRST NAME フィールド内の値により識別された従業員のレコードを任意に 5 つ抽出するために使用されます。

```
DEFINE FILE EMPLOYEE
RAND/D12.2 WITH LAST_NAME = RDNORM(RAND);END
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND FIRST_NAME
BY HIGHEST 5 RAND
END
```

リクエストは次のような出力を生成します。

```
RAND   LAST_NAME         FIRST_NAME
----   ---------         ----------
 .65   CROSS             BARBARA
 .20   BANNING           JOHN
 .19   IRVING            JOAN
 .00   BLACKWOOD        ROSEMARIE
-.14   GREENSPAN         MARY
```

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(in_value)
```

説明

in_value
数値

平方根を計算する値です。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。

```
TABLE FILE MOVIES
PRINT LISTPR AND COMPUTE
SQRT_LISTPR/D12.2 = SQRT(LISTPR);BY TITLE
WHERE CATEGORY EQ 'MUSICALS';
END
```

出力結果は次のとおりです。

```
TITLE                              LISTPR      SQRT_LISTPR
-----                              ------      -----------
ALL THAT JAZZ                       19.98             4.47
CABARET                             19.98             4.47
CHORUS LINE, A                      14.98             3.87
FIDDLER ON THE ROOF                 29.95             5.47
```

17

簡略統計関数

簡略統計関数は、COMPUTE コマンドで呼び出され、TABLE リクエストの処理で生成される内部マトリックスに対して統計的計算を実行します。STDDEV および CORRELATION 関数は、表示コマンドの動詞オブジェクトとして呼び出すこともできます。リクエストにソートフィールドが含まれる場合は、統計関数を呼び出す前に、これらの関数が処理するパーティションのサイズを指定する必要があります。

注意：これらの関数に対してパラメータとして使用される数値およびフィールドはすべて倍精度浮動小数点数にすることをお勧めします。

トピックス

簡略統計関数のパーティションサイズの指定

```
SET PARTITION_ON = {FIRST|PENULTIMATE|TABLE}
```

説明

`FIRST`

リクエストの 1 つ目のソートフィールド (主ソートフィールド) を使用して、値を分割します。

`PENULTIMATE`

一時項目 (COMPUTE) が評価される最後のソートフィールドの 1 つ前のソートフィールドを使用して、値を分割します。これがデフォルト値です。

TABLE

内部マトリックス全体を使用して、統計関数を計算します。

CORRELATION - 2 つのデータセット間の相関度を計算

CORRELATION 関数は、2 つの数値フィールド間の相関係数を計算します。この関数は、ゼロ (-1.0) と 1.0 の間の数値を返します。

構文　2 つのフィールド間の相関係数の計算

```
CORRELATION(field1, field2)
```

説明

field1

数値

相関度を計算する 1 つ目のデータセットです。

field2

数値

相関度を計算する 2 つ目のデータセットです。

注意：CORRELATION 関数の引数は、接頭語付きフィールドにすることはできません。演算接頭語を適用したフィールドを使用する必要がある場合は、演算接頭語を COMPUTE コマンドのフィールドに適用して結果を HOLD ファイルに保存します。次に、この HOLD ファイルに対して相関度の計算を実行します。

例　相関度の計算

次のリクエストは、DOLLARS フィールドと BUDDOLLARS フィールドとの相関度を計算し、倍精度小数点数に変換します。

```
DEFINE FILE ibisamp/ggsales
DOLLARS/D12.2 = DOLLARS;
BUDDOLLARS/D12.2 = BUDDOLLARS;
END
TABLE FILE ibisamp/ggsales
SUM DOLLARS BUDDOLLARS
CORRELATION(DOLLARS, BUDDOLLARS)
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果を示しています。

DOLLARS	BUDDOLLARS	CORRELATION DOLLARS BUDDOLLARS
46,156,290.00	46,220,778.00	.895691073

KMEANS_CLUSTER - 最近傍平均値に基づき観測値をクラスタに分割

KMEANS_CLUSTER 関数は、最近傍平均値に基づき観測値を特定数のクラスタに分割します。この関数は、渡されたフィールド値に割り当てられたクラスタ数をパラメータとして返します。

注意：要求されたクラスタ数を生成するために十分なデータポイントが存在しない場合は、作成不可能なクラスタ数に対して値 -10 が返されます。

構文　最近傍の平均値に基づく観測値のクラスタへの分割

```
KMEANS_CLUSTER(number, percent, iterations, tolerance,
        [prefix1.]field1[, [prefix1.]field2 ...])
```

説明

`number`

整数

取得するクラスタの数です。

`percent`

数値

トレーニングセットのサイズのパーセントです (計算に使用するデータ全体に対するパーセント)。デフォルト値は AUTO です (内部デフォルトパーセントを使用)。

`iterations`

整数

生成済みの平均値を使用して再計算する最大回数です。デフォルト値は AUTO です (内部デフォルト反復回数を使用)。

`tolerance`

数値

0 (ゼロ) から 1.0 までの加重値です。この値が AUTO の場合は、tolerance の内部デフォルト値を使用します。

`prefix1, prefix2`

フィールドに適用するオプションの集計演算子を定義します。この演算子が演算に使用されます。有効な演算子には次のものがあります。

- **SUM** フィールド値の合計を計算します。SUM がデフォルト値です。
- **CNT** フィールド値の個数を計算します。
- **AVE** フィールド値の平均を計算します。
- **MIN** フィールド値の最小値を計算します。
- **MAX** フィールド値の最大値を計算します。
- **FST** フィールドの最初の値を取得します。
- **LST** フィールドの最後の値を取得します。

注意：PCT.、RPCT.、TOT.、MDN.、MDE.、RNK.、DST. 演算子はサポートされません。

`field1`

数値

分析対象のデータセットです。

`field2`

数値

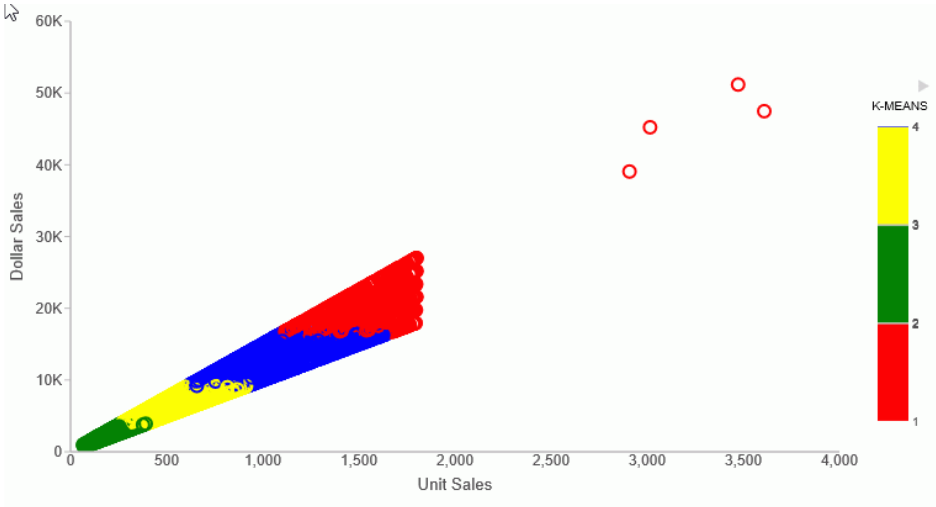
分析対象のデータセットです (オプション)。

例　データ値のクラスタへの分割

次のリクエストは、DOLLARS フィールドの値を 4 つのクラスタに分割し、クラスタごとに色分けされた散布図として結果を表示します。このリクエストでは、percent、iterations、tolerance の各パラメータでデフォルト値が使用され、これらの値が 0 (ゼロ) として渡されています。

```
SET PARTITION_ON = PENULTIMATE
GRAPH FILE GGSALES
PRINT UNITS DOLLARS
COMPUTE KMEAN1/D20.2 TITLE 'K-MEANS'=  KMEANS_CLUSTER(4, AUTO, AUTO, AUTO,
DOLLARS);
ON GRAPH SET LOOKGRAPH SCATTER
ON GRAPH PCHOLD FORMAT JSCHART
ON GRAPH SET STYLE *
INCLUDE=IBFS:/FILE/IBI_HTML_DIR/ibi_themes/Warm.sty,$
type = data, column = N2, bucket=y-axis,$
type=data, column= N1, bucket=x-axis,$
type=data, column=N3, bucket=color,$
GRID=OFF,$
*GRAPH_JS_FINAL
colorScale: {
        colorMode: 'discrete',
        colorBands: [{start: 1, stop: 1.99, color: 'red'}, {start: 2, stop:
2.99, color: 'green'},
                {start: 3, stop: 3.99, color: 'yellow'}, {start: 3.99, stop:
4, color: 'blue'} ]
    }
*END
ENDSTYLE
END
```

下図は、出力結果を示しています。



MULTIREGRESS - 線形重回帰フィールドの作成

MULTIREGRESS 関数は、一連の数値データ点に最適な一次方程式を導き出し、その方程式を使用してレポートの出力結果に新しいフィールドを作成します。方程式は、1 つまたは複数の独立変数を使用して作成することができます。

方程式は次の形式で作成され、y が従属変数、x1、x2、x3 が独立変数です。

```
y = a1*x1 [+ a2*x2 [+ a3*x3] ...] + b
```

独立変数が 1 つの場合は、この方程式は直線を表します。この方程式は、独立変数が 2 つの場合は面を表し、独立変数が 3 つの場合は超平面を表します。独立変数を線形に組み合わせることにより、従属変数の近似値を求めることができると考えられる場合は、その方法を使用するようにします。

構文　　線形重回帰フィールドの作成

```
MULTIREGRESS(input_field1, [input_field2, ...])
```

説明

`input_field1, input_field2 ...`

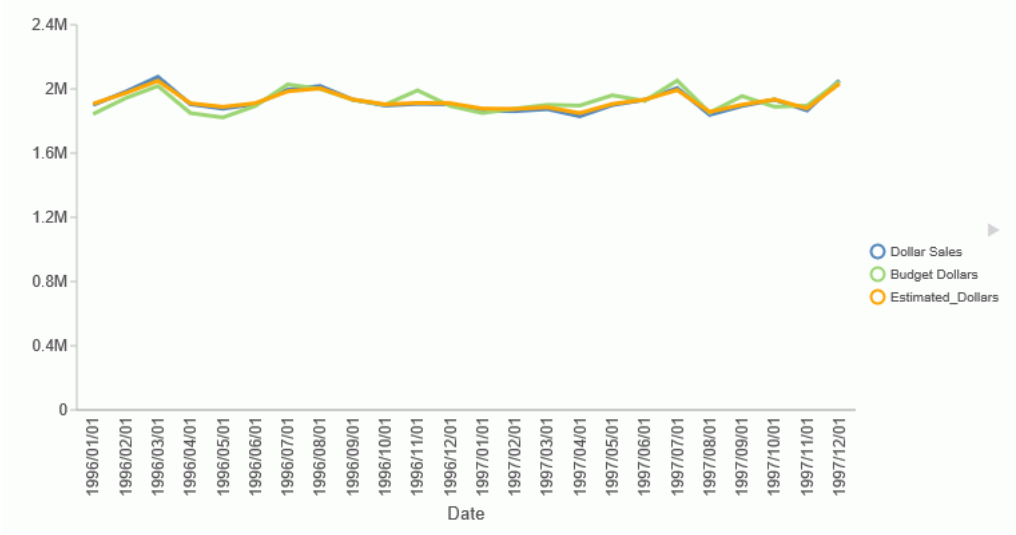
独立変数として使用する任意の数のフィールド名です。これらは互いに独立している必要があります。入力フィールドが数値以外の場合は分類され、線形回帰演算に使用できるよう数値に変換されます。

例　線形重回帰フィールドの作成

次のリクエストは、DOLLARS フィールドと BUDDOLLARS フィールドを使用して、Estimated_Dollars という名前の回帰フィールドを作成します。

```
GRAPH FILE GGSALES
SUM BUDUNITS UNITS BUDDOLLARS DOLLARS
COMPUTE Estimated_Dollars/F8 = MULTIREGRESS(DOLLARS, BUDDOLLARS);
BY DATE
ON GRAPH SET LOOKGRAPH LINE
ON GRAPH PCHOLD FORMAT JSCHART
ON GRAPH SET STYLE *
INCLUDE=IBFS:/FILE/IBI_HTML_DIR/ibi_themes/Warm.sty,$
type=data, column = n1, bucket = x-axis,$
type=data, column= dollars, bucket=y-axis,$
type=data, column= buddollars, bucket=y-axis,$
type=data, column= Estimated_Dollars, bucket=y-axis,$
*GRAPH_JS
"series":[
{"series":2, "color":"orange"}]
*END
ENDSTYLE
END
```

下図は、出力結果を示しています。オレンジ色の線が、回帰方程式を表します。



OUTLIER - 数値データの異常値の検出

データの異常値の検出には、一般に 1.5 x IQR の原則を使用します (この場合の IQR は四分位範囲)。この原則は、データの第 3 四分位から四分位範囲の 1.5 倍を上回る、または第 1 四分位から四分位範囲の 1.5 倍を下回る値として異常値を定義します。四分位範囲とは、データ値のソートに基づき値を 4 等分後、第 3 四分位 (ソートデータを 4 等分した 3 つ目の値) から第 1 四分位 (ソートデータを 4 等分した 1 つ目の値) を差し引いた値の範囲です。第 1 四分位から四分位範囲の 1.5 倍小さい値は「下側境界」、第 3 四分位から四分位範囲の 1.5 倍大きい値は「上側境界」と呼ばれます。

OUTLIER 関数は、DEFINE 式では使用できません。COMPUTE 式、または WHERE 句、WHERE TOTAL 句、WHERE_GROUP 句では使用できます。

数値フィールドに値を入力すると、OUTLIER は、1.5 x IQR の原則を使用して、各フィールドの値に対して次のいずれかの値を返します。

- **0 (ゼロ)** 値は異常値ではありません。
- **-1** 値は下側境界を下回ります。
- **1** 値は上側境界を上回ります。

構文　数値データの異常値の検出

```
OUTLIER(input_field)
```

説明

`input_field`

数値

分析する数値フィールドです。

例　異常値の検出

次のリクエストは、店舗コードによって値の異なる SALES フィールドを定義し、OUTLIER を使用して各フィールドの値が異常値かどうかを決定します。

```
DEFINE FILE GGSALES
SALES/D12 = IF ((CATEGORY EQ 'Coffee') AND (STCD EQ 'R1019')) THEN 19000
  ELSE IF ((CATEGORY EQ 'Coffee') AND (STCD EQ 'R1020')) THEN 20000
  ELSE IF ((CATEGORY EQ 'Coffee') AND (STCD EQ 'R1040')) THEN 7000
  ELSE DOLLARS;
END
TABLE FILE GGSALES
SUM SALES
COMPUTE OUT1/I3 = OUTLIER(SALES);
BY CATEGORY
BY STCD
WHERE CATEGORY EQ 'Coffee'
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果を示しています。2,000,000 より大きい値は上側境界を上回る値、1,000,000 より小さい値は下側境界を下回る値です。その他の値は異常値ではありません。

Category	Store ID	SALES	OUT1
Coffee	R1019	2,280,000	1
	R1020	2,400,000	1
	R1040	840,000	-1
	R1041	1,576,915	0
	R1044	1,340,437	0
	R1088	1,375,040	0
	R1100	1,364,420	0
	R1109	1,459,160	0
	R1200	1,463,453	0
	R1244	1,553,962	0
	R1248	1,535,631	0
	R1250	1,386,124	0

STDDEV - 一連のデータ値の標準偏差を計算

STDDEV 関数は、データの散らばりの度合いを表す数値を返します。データセットは、母集団全体として指定することも標本として指定することもできます。標準偏差は、分散の平方根です。分散は、観測値の予測値 (平均値) からの乖離を表します。母集団を指定した場合、標準偏差の計算の除数 (自由度とも呼ばれる) は、データポイントの総数 N になります。標本を指定した場合、除数は N-1 になります。

xi が観測値、N が観測値の数、μ がすべての観測値の平均である場合、母集団の標準偏差を計算する式は次のとおりです。

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_1-\mu)^2}$$

標本の標準偏差を計算する場合、平均は標本観測値を使用して計算し、除数は N ではなく N-1 となります。

参照　一連のデータ値の標準偏差の計算

```
STDDEV(field, sampling)
```

説明

`field`

数値

標準偏差の計算に使用する一連の観測値です。

`sampling`

キーワード

データセットの元データを示します。次のいずれかの値です。

- **P**　母集団全体
- **S**　母集団の標本

注意：STDDEV 関数の引数は、接頭語付きフィールドにすることはできません。演算接頭語を適用したフィールドを使用する必要がある場合は、演算接頭語を COMPUTE コマンドのフィールドに適用して結果を HOLD ファイルに保存します。次に、この HOLD ファイルに対して標準偏差の計算を実行します。

例　標準偏差の計算

次のリクエストは、DOLLARS フィールドの標準偏差を計算し、倍精度小数点数に変換します。

```
DEFINE FILE ibisamp/ggsales
DOLLARS/D12.2 = DOLLARS;
END
TABLE FILE ibisamp/ggsales
SUM DOLLARS STDDEV(DOLLARS,S)
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果を示しています。

DOLLARS	STDS DOLLARS
46,156,290.00	6,157.711080272

18

機械学習 (Python ベース) 関数

機械学習 (Python ベース) 関数は、Python スクリプトとして実装された FOCUS 関数です。これらの Python スクリプトは、Python の機能を機械学習にまで拡張する scipy、numpy、scikit-learn、pandas などの Python パッケージを利用します。

機械学習 (Python ベース) 関数は、機械学習のさまざまな手法を用いて、回帰、分類、勾配ブースティング、異常値の検出を実行します。Python スクリプトは、必要なデータのスケーリングなど一連の標準機械学習タスクを実行します。これらは、相互検証付きグリッド検索を中心に構築されています。つまり、最適値を決定するために、いくつかのハイパーパラメータ (学習プロセスに影響を及ぼすが、モデルパラメータではない) が特定され、各ハイパーパラメータの複数の値を使用してモデルが作成されます。最適値を決定するためには相互検証が適用されます。これは、トレーニングデータのサブセットとは異なる評価データのサブセットに基づいてパフォーマンスが測定されるようにするためです。ターゲットフィールドにミッシング値が含まれる行は、トレーニングおよび評価に使用されませんが、予測値はトレーニングデータを使用したモデルで計算されます。

Reporting Server ブラウザインターフェースの [WebFOCUS - データサイエンスデモ] を実行することで、例で使用する .cvs ファイルおよび付属するマスターファイルを生成することができます。

トピックス

❑ RUN_MODEL、RUN_MODEL2 - 保存済み Python モデルの実行

ANOMALY_IF - 異常値の検出

ANOMALY_IF は、アイソレーションフォレストを使用して、異常値を検出します。アイソレーションフォレストでは、決定木群を使用して、観測値 X0、X1、. . . が分布する空間をランダムかつ再帰的にハイパー矩形に分割し、これらに 1 つまたは微小な数のサンプルを格納します。異常値サンプルは、その他多数のサンプル近傍のサンプルよりも少数の分割による、ハイパー矩形で分離できます。1 つのサンプルに到達するために必要な分割回数が、異常スコアに換算されます。

構文　　異常スコアの計算

```
ANOMALY_IF('options' predictor_field1[, predictor_field2, ...])
```

説明

`'options'`

モデル属性を制御する詳細パラメータのディクショナリを、一重引用符 (') で囲んで指定します。これらのパラメータのほとんどにはデフォルト値が設定されているため、デフォルト値を使用する場合は、リクエストでは省略することができます。詳細パラメータを指定しない場合でも、一重引用符 (') が必要です。詳細パラメータディクショナリのフォーマットは次のとおりです。

```
'{"parm_name1": "parm_value1", ... ,"parm_namei": "parm_valuei"}'
```

次の詳細パラメータがサポートされています。

`"trees"`

決定木群の決定木の数です。使用可能な値は、11 以上の整数です。デフォルト値は 100 です。

`"score"`

この関数から返される値のタイプを定義します。スコアが "binary" の場合、この関数は、異常サンプルに -1、正常サンプルに +1 を返します。スコアが "grade" の場合、-1.0 から 1.0 までの連続異常スコアが返されます。ここで、数値がより大きい負の値になるほど、そのポイントの異常度が高まります。有効値は、"binary" および "grade" です。デフォルト値は "binary" です。

`"max_samples"`

各木が使用するトレーニングセット内の行の割合です。使用可能な値は 0 (ゼロ) から 1 までの小数値です。デフォルト値は "0.5" です。

"contamination"
スコアが "binary" の場合にのみ適用されます。トレーニングセット内で、異常値として特定されるサンプルの割合です。使用可能な値は 0 (ゼロ) から 0.5 までの小数値です。デフォルト値は "0.1" です。

"train_ratio"
モデルのトレーニングに使用するデータの割合を指定する 0 から 1 までの値です。デフォルト値は "1.0" です。

predictor_field1[, predictor_field2, ...]
数値

1 つまたは複数の予測子のフィールド名です。

例 ANOMALY_IF による異常値の検出

次のプロシジャは、ANOMALY_IF を使用して 異常値を検出します。バイナリモードで、予測子として馬力、最高出力 (RPM)、街中燃費 (MPG)、高速燃費 (MPG)、価格を使用します。戻り値 -1.00 により、異常値が特定されます。

```
TABLE FILE imports85
PRINT horsepower peakRpm cityMpg highwayMpg price
COMPUTE AnomalyBinaryScore/D5.2 =
ANOMALY_IF('{"trees":"123","score":"binary","contamination":"0.2"}',
              horsepower, peakRpm, cityMpg, highwayMpg, price);
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果の一部を示しています。

horsepower	peakRpm	cityMpg	highwayMpg	price	AnomalyBinaryScore
111	5000	21	27	13495	1.00
111	5000	21	27	16500	1.00
154	5000	19	26	16500	1.00
102	5500	24	30	13950	1.00
115	5500	18	22	17450	1.00
110	5500	19	25	15250	1.00
110	5500	19	25	17710	1.00
110	5500	19	25	18920	1.00
140	5500	17	20	23875	-1.00
160	5500	16	22	.	-1.00
101	5800	23	29	16430	1.00
101	5800	23	29	16925	1.00
121	4250	21	28	20970	1.00
121	4250	21	28	21105	1.00
121	4250	20	25	24565	-1.00
182	5400	16	22	30760	-1.00
182	5400	16	22	41315	-1.00
182	5400	15	20	36880	-1.00
48	5100	47	53	5151	-1.00
70	5400	38	43	6295	1.00
70	5400	38	43	6575	1.00
68	5500	37	41	5572	1.00
68	5500	31	38	6377	1.00
102	5500	24	30	7957	1.00

次のリクエストでは、grade モードで、上記と同一の詳細パラメータおよび予測子が使用されています。

```
TABLE FILE imports85
PRINT horsepower peakRpm cityMpg highwayMpg price
COMPUTE AnomalyGradeScore/D5.2 =
ANOMALY_IF('{"trees":"123","score":"grade","contamination":"0.2"}',
                                                horsepower, peakRpm, cityMpg,
highwayMpg, price);
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果の一部を示しています。

horsepower	peakRpm	cityMpg	highwayMpg	price	AnomalyGradeScore
111	5000	21	27	13495	.09
111	5000	21	27	16500	.08
154	5000	19	26	16500	.07
102	5500	24	30	13950	.08
115	5500	18	22	17450	.05
110	5500	19	25	15250	.09
110	5500	19	25	17710	.09
110	5500	19	25	18920	.08
140	5500	17	20	23875	-.01
160	5500	16	22	.	.00
101	5800	23	29	16430	.03
101	5800	23	29	16925	.03
121	4250	21	28	20970	.02
121	4250	21	28	21105	.02
121	4250	20	25	24565	-.01
182	5400	16	22	30760	-.03
182	5400	16	22	41315	-.09
182	5400	15	20	36880	-.06
48	5100	47	53	5151	-.16
70	5400	38	43	6295	.00
70	5400	38	43	6575	.00
68	5500	37	41	5572	.03
68	5500	31	38	6377	.10
102	5500	24	30	7957	.09
68	5500	31	38	6229	.10
68	5500	31	38	6692	.10
68	5500	31	38	7609	.09
102	5500	24	30	8558	.09
88	5000	24	30	8921	.10

CLASSIFY_BLR - バイナリロジスティック回帰

バイナリロジスティック回帰は、予測子によって範囲が定められた空間の 2 クラス間の最適な線形分離を導きます。ターゲット変数は、2 つのクラスに対して 2 つの値 (0 および 1) が与えられます。予測値は、クラス割り当て (0 または 1) かクラス 1 の確率のいずれかです。

構文　　バイナリロジスティック回帰の演算

```
CLASSIFY_BLR('options',
        predictor_field1[, predictor_field2, ...] target_field)
```

説明

`'options'`
モデル属性を制御する詳細パラメータのディクショナリを、一重引用符 (') で囲んで指定します。これらのパラメータのほとんどにはデフォルト値が設定されているため、デフォルト値を使用する場合は、リクエストでは省略することができます。詳細パラメータを指定しない場合でも、一重引用符 (') が必要です。詳細パラメータディクショナリのフォーマットは次のとおりです。

```
'{"parm_name1": "parm_value1", ... ,"parm_namei": "parm_valuei"}'
```

次の詳細パラメータがサポートされています。

`"proba"`
ターゲットクラスまたはクラス予測に所属する確率を表示するかどうかを指定します。有効値は "no" (デフォルト) または "yes" です。値を "yes" に設定すると、空間上の各点が "pos_label" (デフォルトは "1") で指定されるクラスに所属する確率 Y (0<=Y<=1) が生成されます。値を "no" に設定すると、空間上の各点のクラス予測 Y (0 または 1) が生成されます。

`"pos_label"`
"proba" を "yes" に設定した場合に使用します。所属の確率を計算するクラスを指定します。有効値は "0" および "1" です。デフォルト値は "1" です。

`"train_ratio"`
モデルのトレーニングに使用するデータの割合を指定する 0 から 1 までの値です。デフォルト値は "0.8" です。

`"test_ratio"`
モデルのテストに使用するデータの割合を指定する 0 から 1 までの値です。デフォルト値は "0.2" です。

`"l2_grid"`
L2 正規化強度に使用する、カンマ区切りの正の数値で構成されるグリッドです。デフォルト値は "0,1,1,10" です。最適値は、交差検定によって選択されます。

`"kfold"`
交差検定で使用する分割サブセット数です。推奨値は "2" から "10" の整数です。デフォルト値は "4" です。

`predictor_field1[, predictor_field2, ...]`
数値

1 つまたは複数の予測子のフィールド名です。

`target_field`
数値

ターゲットフィールドです。

例 CLASSIFY_BLR によるドア数の予測

次の TABLE リクエストは、CLASSIFY_BLR を使用して自動車のドア数を予測します。詳細パラメータおよび予測子 (価格、車高、馬力、最高出力 (RPM)、街中燃費 (MPG)、高速燃費 (MPG)) のデフォルト値を使用します。

```
TABLE FILE imports85
PRINT numOfDoors
COMPUTE predictedNumOfDoors/A6 =
CLASSIFY_BLR('{"proba":"no","kfold":"4","test_ratio":"0.2"}',
                  price, height, horsepower, peakRpm,
                  cityMpg, highwayMpg, numOfDoors);
WHERE price LT 30000
ON TABLE SET PAGE NOPAGE
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果の一部を示しています。

numOfDoors	predictedNumOfDoors
two	two
two	two
two	two
four	four
four	four
two	four
four	four
four	four
four	four
two	two
two	four
four	four
two	four
four	four
four	four
two	two
two	two
four	two
two	two
two	two
two	two
four	two
four	two
four	two
.	two
four	four
two	two

CLASSIFY_KNN - k 近傍法分類

k 近傍法分類は、予測子によって範囲が定められた空間の任意のデータポイントにクラスメンバーシップを割り当てる方法です。分類は、k 近傍で最も一般的なクラスの割り当てで実行されます。この分類法には、距離空間の定義が必要です。

参照　k 近傍法分類の演算

```
CLASSIFY_KNN('options',
        predictor_field1[, predictor_field2, ...] target_field)
```

説明

`'options'`
モデル属性を制御する詳細パラメータのディクショナリを、一重引用符 (') で囲んで指定します。これらのパラメータのほとんどにはデフォルト値が設定されているため、デフォルト値を使用する場合は、リクエストでは省略することができます。詳細パラメータを指定しない場合でも、一重引用符 (') が必要です。詳細パラメータディクショナリのフォーマットは次のとおりです。

```
'{"parm_name1": "parm_value1", ... ,"parm_namei": "parm_valuei"}'
```

次の詳細パラメータがサポートされています。

`"k"`
近傍数です。使用可能な値は、2 以上の整数です。デフォルト値は "5" です。

`"p"`
Lp ノルムの乗数 (p) です。有効値は正の整数です。推奨値は "1" および "2" です。デフォルト値は "2" です。

- power=1 の場合、座標間の差の絶対値の合計として距離を計算します (マンハッタン距離)。
- power=2 の場合、座標間の差の 2 平方和の平方根として距離を計算します (ユークリッド距離)。これがデフォルト値です。

`"prediction_ratio"`
予測に使用するデータの割合です。有効値は 0 から 1 までの数値です。デフォルト値は "0.8" です。

`"test_ratio"`
モデルのテストに使用するデータの割合を指定する 0 から 1 までの値です。デフォルト値は "0.2" です。

"kfold"
交差検定のグリッドサーチで使用する分割の回数です。最近傍グリッド K/2、K、2K に対するグリッドサーチが実行されます。推奨値は "2" から "10" の整数です。デフォルト値は "4" です。

predictor_field1[, predictor_field2, ...]
数値

1 つまたは複数の予測子のフィールド名です。

target_field
数値

ターゲットフィールドです。

例 CLASSIFY_KNN によるクラス割り当ての予測

次のリクエストは、自動車のドア数に基づいてクラス 0 (ゼロ) および 1 を作成し、次に CLASSIFY_KNN で 10 個の最近傍およびユークリッド距離を使用してクラス割り当てを予測します。予測子は、価格、車高、馬力、最高出力 (RPM)、街中燃費 (MPG)、高速燃費 (MPG) です。データセットには、ターゲット値がミッシング値の行が含まれます。

```
DEFINE FILE imports85
OUR_CLASSES/I2 MISSING ON = IF numOfDoors EQ MISSING THEN MISSING
                            ELSE IF numOfDoors EQ 'two' THEN 0
                            ELSE 1;
END
TABLE FILE imports85
PRINT numOfDoors BODYSTYLE OUR_CLASSES
COMPUTE predictedCLASSES/I2 =
CLASSIFY_KNN('{"K":"10","p":"2","kfold":"4","test_ratio":"0.2"}',
              price,  height, horsepower, peakRpm,
              cityMpg, highwayMpg, OUR_CLASSES);

ON TABLE SET PAGE NOPAGE
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果の一部を示しています。

numOfDoors	bodyStyle	OUR_CLASSES	predictedCLASSES
two	convertible	0	0
two	convertible	0	0
two	hatchback	0	0
four	sedan	1	1
four	sedan	1	1
two	sedan	0	1
four	sedan	1	1
four	wagon	1	1
four	sedan	1	1
two	hatchback	0	0
two	sedan	0	1
four	sedan	1	1
two	sedan	0	1
four	sedan	1	1
four	sedan	1	1
four	sedan	1	1
two	sedan	0	1
four	sedan	1	1
two	hatchback	0	0
two	hatchback	0	0
four	sedan	1	0
two	hatchback	0	0
two	hatchback	0	0
two	hatchback	0	0
four	hatchback	1	0
four	sedan	1	0
four	sedan	1	0
.	sedan	.	0
four	wagon	1	1
two	hatchback	0	0

CLASSIFY_RF - ランダムフォレスト分類

CLASSIFY_RF 関数は、決定木の集合であるランダムフォレスト (決定木群) を作成します。決定木がそれぞれ独立した分類予測を生成し、決定木群の予測は個々の予測の多数決予測になります。

構文　　ランダムフォレスト分類の演算

```
CLASSIFY_RF('options', number_of_trees,
        predictor_field1[, predictor_field2, ...] target_field)
```

説明

`'options'`
モデル属性を制御する詳細パラメータのディクショナリを、一重引用符 (') で囲んで指定します。これらのパラメータのほとんどにはデフォルト値が設定されているため、デフォルト値を使用する場合は、リクエストでは省略することができます。詳細パラメータを指定しない場合でも、一重引用符 (') が必要です。詳細パラメータディクショナリのフォーマットは次のとおりです。

```
'{"parm_name1": "parm_value1", ... ,"parm_namei": "parm_valuei"}'
```

次の詳細パラメータがサポートされています。

`"trees"`
決定木群の決定木の数です。使用可能な値は、11 以上の整数です。デフォルト値は 100 です。

`"feature_importances"`
機能の重要度を計算するかどうかを指定します。有効値は "yes" および "no" です。デフォルト値は "yes" です。

`"train_ratio"`
モデルのトレーニングに使用するデータの割合を指定する 0 から 1 までの値です。デフォルト値は "0.8" です。

`"test_ratio"`
モデルのテストに使用するデータの割合を指定する 0 から 1 までの値です。デフォルト値は "0.2" です。

`"scoring"`
トレーニングは F1 スコア (適合率と再現率の加重平均)、正解率 (合計観測値に対する予測正解率) のいずれかを最適化します。有効値は、"f1_score" および "accuracy" です。デフォルト値は "f1_score" です。

`"min_values_leaf_grid"`
ノードの分割に必要な単一ノードの最小サンプル数のグリッドです。最適値は、交差検定によって選択されます。デフォルト値は "1,3,5" です。

`predictor_field1[, predictor_field2, ...]`
数値 または文字

1 つまたは複数の予測子のフィールド名です。

`target_field`
数値 または文字

ターゲットフィールドです。

例 CLASSIFY_RF によるクラス割り当ての予測

次のプロシジャは、CLASSIFY_RF を使用して自動車のドア数のクラス割り当てを予測します。100 個の決定木群によるランダムフォレストで、予測子として価格、ボディスタイル、車高、馬力、最高出力 (RPM)、街中燃費 (MPG)、高速燃費 (MPG) を使用します。

```
TABLE FILE imports85
PRINT numOfDoors
COMPUTE predictedNumOfDoors/A6 =
CLASSIFY_RF('{"trees":"100","kfold":"4","test_ratio":"0.2"}',
                        price, bodyStyle, height, horsepower,
                        peakRpm, cityMpg, highwayMpg, numOfDoors);
WHERE price LT 30000
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果の一部を示しています。

numOfDoors	predictedNumOfDoors
two	two
two	two
two	two
four	four
four	four
two	four
four	four
four	four
four	four
two	two
two	four
four	four
two	four
four	four
four	four
two	two
two	two
four	four
two	two
two	two
two	two
four	two
four	four
four	four
.	four
four	four
two	two
two	two

CLASSIFY_XGB - 勾配ブースティング分類

CLASSIFY_XGB 関数は、一連の決定木群を作成し、新しいツリーのそれぞれが前の木群の予測精度の向上を試みます。各決定木は、バイナリ分割の再帰的な連続を使用して、予測子がまたがる空間の分割に基づいて予測に到達します。各分割で、単一予測値の値は 2 つのセットに分割されます。これらの分割は、各分割内のデータポイントでの Y 値近似度の測定 (純度と呼ばれる) を伴う目的関数貪欲最大化手法 (分割予測子および分割点の選択) によって求められます。予測機能の精度が劣化したところで、早期停止が適用されます。

構文　　勾配ブースティング分類の計算

```
CLASSIFY_XGB('options',
        predictor_field1[, predictor_field2, ...] target_field)
```

説明

`'options'`
モデル属性を制御する詳細パラメータのディクショナリを、一重引用符 (') で囲んで指定します。これらのパラメータのほとんどにはデフォルト値が設定されているため、デフォルト値を使用する場合は、リクエストでは省略することができます。詳細パラメータを指定しない場合でも、一重引用符 (') が必要です。詳細パラメータディクショナリのフォーマットは次のとおりです。

```
'{"parm_name1": "parm_value1", ... ,"parm_namei": "parm_valuei"}'
```

次の詳細パラメータがサポートされています。

`"trees"`
決定木群の決定木の数です。使用可能な値は、11 以上の整数です。デフォルト値は "300" です。

`"train_ratio"`
モデルのトレーニングに使用するデータの割合を指定する 0 から 1 までの値です。デフォルト値は "0.8" です。

`"test_ratio"`
モデルのテストに使用するデータの割合を指定する 0 から 1 までの値です。デフォルト値は "0.2" です。

`"early_stopping_rounds"`
アルゴリズムのパフォーマンスがこれ以上向上しない場合に、木群の追加を停止するまでの回数を指定します。有効値は、1 からこの木群の数までの整数です。デフォルト値は 10 です。

"l2_grid"
L2 正規化強度に使用する、カンマ区切りの正の数値で構成されるグリッドです。デフォルト値は "0,1,1,10" です。最適値は、交差検定によって選択されます。

"kfold"
交差検定のグリッドサーチで使用する分割の回数です。推奨値は "2" から "10" の整数です。デフォルト値は "4" です。

"max_depth_grid"
各決定木の最大の深さです。"4,5,6" 形式のグリッドが使用できます。デフォルト値は "5" です。最適値は、交差検定によって選択されます。

"scoring"
トレーニングは F1 スコア (適合率と再現率の加重平均)、正解率 (合計観測値に対する予測正解率) のいずれかを最適化します。有効値は、"f1_score" および "accuracy" です。デフォルト値は "f1_score" です。

predictor_field1[, predictor_field2, ...]
数値 または文字

1 つまたは複数の予測子のフィールド名です。

target_field
数値 または文字

ターゲットフィールドです。

例 CLASSIFY_XGB による価格の予測

次のプロシジャは、CLASSIFY_XGB を使用して数値を予測します。400 個の決定木群を使用し、早期停止回数として 10、その他の詳細パラメータとしてデフォルト値、予測子として価格、ボディスタイル、車高、馬力、最高出力 (RPM)、街中燃費 (MPG)、高速燃費 (MPG) を使用します。

```
TABLE FILE imports85
PRINT numOfDoors
COMPUTE predictedNumOfDoors/A6 =
CLASSIFY_XGB('{"trees":"400","early_stopping_rounds":"10",
                      "kfold":"4","test_ratio":"0.2"}',
                       price, bodyStyle, height, horsepower,
                       peakRpm, cityMpg, highwayMpg, numOfDoors);
WHERE price LT 30000
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果の一部を示しています。

numOfDoors	predictedNumOfDoors
two	four
two	four
two	two
four	four
four	four
two	four
four	four
four	four
four	four
two	two
two	four
four	four
two	four
four	four
four	four
two	two
two	two
four	four
two	two
two	two
two	two
four	two
four	four
four	four
.	four
four	four
two	two

REGRESS_KNN - k 近傍法回帰

k 近傍法回帰は、予測子によって範囲が定められた空間の任意のデータポイントに対するターゲット値を予測する方法です。予測値は、k 近傍のターゲット値の平均です。この分類法には、距離空間の定義が必要です。

参照　　k 近傍法回帰の演算

```
REGRESS_KNN('options',
        predictor_field1[, predictor_field2, ...] target_field)
```

説明

`'options'`
モデル属性を制御する詳細パラメータのディクショナリを、一重引用符 (') で囲んで指定します。これらのパラメータのほとんどにはデフォルト値が設定されているため、デフォルト値を使用する場合は、リクエストでは省略することができます。詳細パラメータを指定しない場合でも、一重引用符 (') が必要です。詳細パラメータディクショナリのフォーマットは次のとおりです。

```
'{"parm_name1": "parm_value1", ... ,"parm_namei": "parm_valuei"}'
```

次の詳細パラメータがサポートされています。

`"K"`
予測に使用する最近傍の数です。使用可能な値は、2 以上の整数です。デフォルト値は "5" です。

`"p"`
Lp ノルムの乗数 (p) です。有効値は正の整数です。推奨値は "1" および "2" です。デフォルト値は "2" です。

- power=1 の場合、座標間の差の絶対値の合計として距離を計算します (マンハッタン距離)。
- power=2 の場合、座標間の差の 2 平方和の平方根として距離を計算します (ユークリッド距離)。

`"prediction_ratio"`
予測に使用するデータの割合です。有効値は 0 から 1 までの数値です。デフォルト値は "0.8" です。

`"test_ratio"`
モデルのテストに使用するデータの割合を指定する 0 から 1 までの値です。デフォルト値は "0.2" です。

"kfold"
交差検定のグリッドサーチで使用する分割の回数です。最近傍グリッド K/2、K、2K に対するグリッドサーチが実行されます。推奨値は "2" から "10" の整数です。デフォルト値は "4" です。

predictor_field1[, predictor_field2, ...]
数値

1 つまたは複数の予測子のフィールド名です。

target_field
数値

ターゲットフィールドです。

例 REGRESS_KNN を使用した価格の予測

次のリクエストは、REGRESS_KNN でデフォルト詳細パラメータ (10 個の最近傍およびユークリッド距離) を使用して価格を予測します。予測子として車高、馬力、最高出力 (RPM)、街中燃費 (MPG)、高速燃費 (MPG) を使用します。

```
TABLE FILE imports85
PRINT price
COMPUTE predictedPrice/I5 =
REGRESS_KNN('{"K":"10","p":"2","kfold":"4","prediction_ratio":"0.8","test_ra
tio":"0.2"}',
                height, horsepower, peakRpm,
                cityMpg, highwayMpg, price);
WHERE price LT 30000
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果の一部を示しています。

price	predictedPrice
13495	12461
16500	12461
16500	16099
13950	12951
17450	17266
15250	17266
17710	17612
18920	17612
23875	18281
.	16099
16430	13541
16925	13541
20970	20690
21105	20690
24565	24088
5151	6851
6295	5934
6575	5934
5572	5724
6377	6677
7957	8765
6229	6685
6692	6685

REGRESS_POLY - 多項式回帰

多項式回帰は、予測子フィールドの多項式にターゲットフィールドを適合させます。多項式の次数は、関数の入力引数として指定されます。

参照　多項式回帰フィールドの演算

```
REGRESS_POLY('options',
       predictor_field1, predictor_field2, [...,] target_field)
```

説明

`'options'`
モデル属性を制御する詳細パラメータのディクショナリを、一重引用符 (') で囲んで指定します。これらのパラメータのほとんどにはデフォルト値が設定されているため、デフォルト値を使用する場合は、リクエストでは省略することができます。詳細パラメータを指定しない場合でも、一重引用符 (') が必要です。詳細パラメータディクショナリのフォーマットは次のとおりです。

```
'{"parm_name1": "parm_value1", ... ,"parm_namei": "parm_valuei"}'
```

次の詳細パラメータがサポートされています。

`"degree"`
オプション。多項式の次数です。低次多項式の使用が推奨されます。デフォルト値は "1" です。

`"interaction_only"`
オプション。多項式で生成される項を制御します。デフォルト値は "no" です。有効値は次のとおりです。

- "no" - 予測子フィールドに基づいて、最も一般的な次数の多項式を生成します。
- "yes" - 各予測子 (X0, X1, . . .) の一次多項式および予測子の最大次数の交差項 (X0*X1*X2 形式) を含む多項式のみを生成します。

`"train_ratio"`
オプション。モデルのトレーニングに使用するデータの割合を指定する 0 から 1 までの値です。デフォルト値は "0.8" です。

`"test_ratio"`
オプション。モデルのテストに使用するデータの割合を指定する 0 から 1 までの値です。デフォルト値は "0.2" です。

`"l2_grid"`
オプション。L2 正規化強度に使用する、カンマ区切りの正の数値で構成されるグリッドです。デフォルト値は "0,1,1,10" です。最適値は、交差検定によって選択されます。

"kfold"
オプション。交差検定で使用する分割サブセット数です。推奨値は "2" から "10" の整数です。デフォルト値は "4" です。

predictor_field1, predictor_field2, [...,]
数値

2 つ以上の予測子フィールドの名前です。

target_field
数値

ターゲットフィールドです。

例 REGRESS_POLY を使用した価格の予測

次のリクエストでは、Regress_POLY を使用して予測価格を計算します。次数 4 の多項式回帰、予測子として車高、馬力、最高出力 (RPM)、街中燃費 (MPG)、ピーク回転数、都市部の MPG、高速燃費 (MPG) を使用します。

```
TABLE FILE imports85
PRINT price
COMPUTE predictedPrice/I5 = REGRESS_POLY('{"degree":"4"}',
                height, horsepower, peakRpm,
                cityMpg, highwayMpg, price);
WHERE price LT 30000
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果の一部を示しています。

price	predictedPrice
13495	13981
16500	13981
16500	14360
13950	11755
17450	18201
15250	14163
17710	17545
18920	17545
23875	22230
.	15711
16430	13639
16925	13639
20970	19351
21105	19351
24565	23792
5151	9471
6295	7992
6575	7992
5572	5927
6377	6849
7957	9107
6229	6624
6692	6624
7609	6624
8558	8940
8921	8948

REGRESS_RF - ランダムフォレスト回帰

REGRESS_RF 関数は、決定木の集合であるランダムフォレスト (決定木群) を作成します。決定木がそれぞれ独立した回帰予測を生成し、決定木群の予測は個々の予測の平均予測になります。

構文　　ランダムフォレスト回帰の演算

```
REGRESS_RF('options',
        predictor_field1[, predictor_field2, ...] target_field)
```

説明

`'options'`
モデル属性を制御する詳細パラメータのディクショナリを、一重引用符 (') で囲んで指定します。これらのパラメータのほとんどにはデフォルト値が設定されているため、デフォルト値を使用する場合は、リクエストでは省略することができます。詳細パラメータを指定しない場合でも、一重引用符 (') が必要です。詳細パラメータディクショナリのフォーマットは次のとおりです。

```
'{"parm_name1": "parm_value1", ... ,"parm_namei": "parm_valuei"}'
```

次の詳細パラメータがサポートされています。

`"trees"`
決定木群の決定木の数です。使用可能な値は、11 以上の整数です。デフォルト値は 100 です。

`"feature_importances"`
機能の重要度を計算するかどうかを指定します。有効値は "yes" および "no" です。デフォルト値は "yes" です。

`"train_ratio"`
モデルのトレーニングに使用するデータの割合を指定する 0 から 1 までの値です。デフォルト値は "0.8" です。

`"test_ratio"`
モデルのテストに使用するデータの割合を指定する 0 から 1 までの値です。デフォルト値は "0.2" です。

`"min_values_leaf_grid"`
ノードの分割に必要な単一ノードの最小サンプル数のグリッドです。最適値は、交差検定によって選択されます。デフォルト値は "1,3,5" です。

`predictor_field1[, predictor_field2, ...]`
数値 または文字

1 つまたは複数の予測子のフィールド名です。

`target_field`
数値

ターゲットフィールドです。

例 REGRESS_RF による価格の予測

次のプロシジャは、REGRESS_RF を使用して価格を予測します。ランダムフォレストですべての詳細パラメータとしてデフォルト値、予測子としてドア数、ボディスタイル、車高、馬力、最高出力 (RPM)、街中燃費 (MPG)、高速燃費 (MPG) を使用します。

```
TABLE FILE imports85
PRINT price
COMPUTE predictedPrice/I5 = REGRESS_RF('',
                numOfDoors, bodyStyle, height, horsepower,
                peakRpm, cityMpg, highwayMpg, price);
WHERE price LT 30000
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果の一部を示しています。

price	predictedPrice
13495	15210
16500	15210
16500	14983
13950	13184
17450	16074
15250	15459
17710	18590
18920	18573
23875	21104
.	16532
16430	12923
16925	14432
20970	19721
21105	20481
24565	23470
5151	5910
6295	6810
6575	8448
5572	6043
6377	6745
7957	8249
6229	6627
6692	7152
7609	7152
8558	9234
8921	9668

REGRESS_XGB - 勾配ブースティング回帰

REGRESS_XGB 関数は、一連の決定木群を作成し、新しいツリーのそれぞれが前の木群の予測精度の向上を試みます。各決定木は、バイナリ分割の再帰的な連続を使用して、予測子がまたがる空間の分割に基づいて予測に到達します。各分割で、単一予測値の値は 2 つのセットに分割されます。これらの分割は、各分割内のデータポイントでの Y 値近似度の測定 (純度と呼ばれる) を伴う目的関数貪欲最大化手法 (分割予測子および分割点の選択) によって求められます。予測機能の精度が劣化したところで、早期停止が適用されます。

構文　　勾配ブースティング回帰の計算

```
REGRESS_XGB('options',
        predictor_field1[, predictor_field2, ...] target_field)
```

説明

`'options'`
モデル属性を制御する詳細パラメータのディクショナリを、一重引用符 (') で囲んで指定します。これらのパラメータのほとんどにはデフォルト値が設定されているため、デフォルト値を使用する場合は、リクエストでは省略することができます。詳細パラメータを指定しない場合でも、一重引用符 (') が必要です。詳細パラメータディクショナリのフォーマットは次のとおりです。

```
'{"parm_name1": "parm_value1", ... ,"parm_namei": "parm_valuei"}'
```

次の詳細パラメータがサポートされています。

`"trees"`
決定木群の決定木の数です。使用可能な値は、11 以上の整数です。デフォルト値は "300" です。

`"train_ratio"`
モデルのトレーニングに使用するデータの割合を指定する 0 から 1 までの値です。デフォルト値は "0.8" です。

`"test_ratio"`
モデルのテストに使用するデータの割合を指定する 0 から 1 までの値です。デフォルト値は "0.2" です。

`"early_stopping_rounds"`
アルゴリズムのパフォーマンスがこれ以上向上しない場合に、木群の追加を停止するまでの回数を指定します。有効値は、1 からこの木群の数までの整数です。デフォルト値は 10 です。

"l2_grid"
L2 正規化強度に使用する、カンマ区切りの正の数値で構成されるグリッドです。デフォルト値は "0,1,1,10" です。最適値は、交差検定によって選択されます。

"kfold"
交差検定のグリッドサーチで使用する分割の回数です。推奨値は "2" から "10" の整数です。デフォルト値は "4" です。

"max_depth_grid"
各決定木の最大の深さです。"4,5,6" 形式のグリッドが使用できます。デフォルト値は "5" です。最適値は、交差検定によって選択されます。

predictor_field1[, predictor_field2, ...]
数値 または文字

1 つまたは複数の予測子のフィールド名です。

target_field
数値

ターゲットフィールドです。

例 REGRESS_XGB による価格の予測

次のプロシジャは、REGRESS_XGB を使用して価格を予測します。400 個の決定木群を使用し、その他の詳細パラメータとしてデフォルト値、予測子としてドア数、ボディスタイル、車高、馬力、最高出力 (RPM)、街中燃費 (MPG)、高速燃費 (MPG) を使用します。

```
TABLE FILE imports85
PRINT price
COMPUTE predictedPrice/I5 =
REGRESS_XGB('{"trees":"400","early_stopping_rounds":"10",
                      "kfold":"4","test_ratio":"0.2"}',
                      numOfDoors, bodyStyle, height, horsepower,
                      peakRpm, cityMpg, highwayMpg, price);
WHERE price LT 30000
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果の一部を示しています。

price	predictedPrice
13495	15634
16500	15634
16500	14517
13950	13522
17450	14939
15250	15343
17710	19065
18920	19097
23875	21800
.	17161
16430	13499
16925	13738
20970	21822
21105	21352
24565	24267
5151	5704
6295	6414
6575	7402
5572	6201
6377	6821
7957	8410
6229	6778
6692	7566
7609	7566
8558	9300
8921	9204
12964	14641

RUN_MODEL、RUN_MODEL2 - 保存済み Python モデルの実行

機械学習関数を使用して、トレーニングデータおよびテストデータによるモデルを作成後、そのモデルを保存しておくことで、新しいデータで実行することができます。モデルの実行では、WebFOCUS で読み取り可能な任意の種類のデータソースを使用することができます。

新しいデータソースの予測子フィールド名が、モデルの生成に使用されたものと同一の場合は、RUN_MODEL 関数を使用します。フィールド名が異なる場合は、RUN_MODEL2 を使用します。いずれの場合も、データタイプと長さは同一にする必要があります。

構文　Python ベースモデルの保存

```
ON TABLE HOLD FORMAT PYTHON MODEL fieldname AS app/modelname
```

説明

`fieldname`
　機械学習関数から値を取得した COMPUTE コマンドのフィールド名です。

`app`
　モデルを保存するアプリケーションフォルダです。

`modelname`
　保存するモデルの名前です。

例　Python モデルの保存

次のリクエストでは、REGRESS_RF 関数を使用して「predictedPrice」というフィールドを返し、アプリケーションフォルダ app1/mymodels に「model5」という名前のモデルとして保存します。

```
TABLE FILE imports85
PRINT price
COMPUTE predictedPrice/I5 =
REGRESS_RF('{"trees":"123","test_ratio":"0.20","train_ratio":"0.80"}',
             make, numOfDoors, bodyStyle, height, horsepower,
             peakRpm, cityMpg, highwayMpg, price);
WHERE OUTPUTLIMIT EQ 1
ON TABLE HOLD FORMAT PYTHON MODEL predictedPrice
    AS app1/mymodels/model5
END
```

構文　保存済み Python モデルの実行

予測子フィールドの名前がモデルおよびデータと同一の場合は、RUN_MODEL 関数を使用します。

```
COMPUTE fieldname/fmt = RUN_MODEL('app/modelname');
```

予測子フィールドの名前がモデルとデータで異なる場合は、RUN_MODEL2 関数を使用します。

```
COMPUTE fieldname/fmt = RUN_MODEL2('app/modelname',
                pfield1, ..., pfieldn);
```

説明

`fieldname`
モデルから取得された予測値を格納するフィールドの名前です。

`app/modelname'`
アプリケーションフォルダおよび保存済みモデル名です。

`pfield1, ..., pfieldn`
モデルの実行に使用されるデータのフィールド名です。このモデルの生成に使用した予測子フィールド名と一致します。

例 RUN_MODEL によるモデルの実行

次のリクエストでは、リクエストのデータソースには、モデルの作成時と同一のフィールド名が使用されています。

```
TABLE FILE imports85
PRINT price COMPUTE predictedPriceFromSaved/I5 =
      RUN_MODEL('app1/mymodels/model5');
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果の一部を示しています。

price	predictedPriceFromSaved
13495	15594
16500	15594
16500	15786
13950	13479
17450	17685
15250	15888
17710	17916
18920	18267
23875	22390
.	18363
16430	14593
16925	15589
20970	20472
21105	20939
24565	23194

例　RUN_MODEL2 によるモデルの実行

次のリクエストでは、リクエストのデータソースに、モデルの作成時とは異なるフィールド名の使用が想定されています。

```
TABLE FILE imports2
PRINT price COMPUTE predictedPriceFromSaved/I5 =
   RUN_MODEL2('app1/mymodels/model5',
              type, doors, Style, ht, hpower, Rpm, CMpg, HMpg);
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果の一部を示しています。

price	predictedPriceFromSaved
13495	15594
16500	15594
16500	15786
13950	13479
17450	17685
15250	15888
17710	17916
18920	18267
23875	22390
.	18363
16430	14593
16925	15589
20970	20472
21105	20939
24565	23194

19

簡略システム関数

簡略システム関数では、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

EDAPRINT - EDAPRINT ログファイルへのカスタムメッセージの挿入

EDAPRINT 関数を使用して、EDAPRINT ログファイルにテキストメッセージを追加し、そのメッセージにメッセージタイプを割り当てることができます。この関数の戻り値は 0 (ゼロ) です。

構文　EDAPRINT ログファイルへのメッセージの挿入

```
EDAPRINT(message_type, 'message')
```

説明

`message_type`

キーワード

次の 3 つのメッセージタイプのいずれかを選択します。

- I　情報メッセージ

- **W** 警告メッセージ
- **E** エラーメッセージ

`'message'`

挿入するメッセージです。一重引用符 (') で囲みます。

例 EDAPRINT ログファイルへのカスタムメッセージの挿入

次のプロシジャは、EDAPRINT ログファイルに 3 つのメッセージを挿入します。

```
-SET &I = EDAPRINT(I, 'This is a test informational message');
-SET &W = EDAPRINT(W, 'This is a test warning message');
-SET &E = EDAPRINT(E, 'This is a test error message');
```

下図は、出力結果を示しています。

```
01/18/2018 15:28:42.892 I disconnect cmrpht000008 tscomid=11,sesid=15,fctkt=5a6102d5:1-5,fcdir=ht000002010
01/18/2018 15:28:42.892 I request by t3rp11708 to notify disconnect of sesid=15
01/18/2018 15:28:42.892 I statistics cmrpht000008 sesid=15,cpu=0.000s,dbms=0.000s,srv=0.008s
01/18/2018 15:28:42.922 I request by cmrpht000008 to exec <ibiweb> session=5a6102d5:1-5
01/18/2018 15:28:42.922 I request by cmrpht000008 to connect to agent (WC_DEFAULT)
01/18/2018 15:28:42.922 I connecting cmrpht000008 tscomid=11,sesid=16,fctkt=5a6102d5:1-5,fcdir=ht000002010
01/18/2018 15:28:42.927 I This is a test informational message
01/18/2018 15:28:42.927 W This is a test warning message
01/18/2018 15:28:42.927 E This is a test error message
01/18/2018 15:28:42.927 I disconnect cmrpht000008 tscomid=11,sesid=16,fctkt=5a6102d5:1-5,fcdir=ht000002010
01/18/2018 15:28:42.927 I request by t3rp11708 to notify disconnect of sesid=16
01/18/2018 15:28:42.927 I statistics cmrpht000008 sesid=16,cpu=0.000s,dbms=0.000s,srv=0.005s
01/18/2018 15:29:58.170 I accepting  cmrpht000009 tcp=fe80::641f:60b7:e7cc:3e56%2:52689
01/18/2018 15:29:58.170 I accepting  cmrpht000010 tcp=fe80::641f:60b7:e7cc:3e56%2:52690
01/18/2018 15:29:58.170 I request by cmrpht000009 to exec <webconsole> session=5a6102d5:1-5, page=UPDATELAYOUT
01/18/2018 15:30:21.546 I request by cmrpht000009 to exec <webconsole> session=5a6102d5:1-5, page=WSCONFIGURATION
01/18/2018 15:30:21.718 I request by cmrpht000009 to exec <webconsole> session=5a6102d5:1-5, page=WKSRIBBON
01/18/2018 15:30:21.859 I request by cmrpht000009 to exec <webconsole> session=5a6102d5:1-5, page=WKSFRAMES
01/18/2018 15:30:22.468 I request by cmrpht000009 to exec <webconsole> session=5a6102d5:1-5, page=CONFTREE
```

ENCRYPT - パスワードの暗号化

ENCRYPT 関数は、ibi™ WebFOCUS® Reporting Server で構成された暗号化アルゴリズムを使用して文字入力値を暗号化します。結果は、可変長文字として返されます。

構文 パスワードの暗号化

```
ENCRYPT(password)
```

説明

`password`

固定長の文字

暗号化する値です。

例　パスワードの暗号化

次のリクエストは、WebFOCUS Reporting Server で構成された暗号化アルゴリズムを使用して「guestpassword」という値を暗号化します。

```
-SET &P1 = ENCRYPT('guestpassword');
-TYPE &P1
```

暗号化された値は、「{AES}963AFA754E1763ABE697E8C5E764115E」として返されます。

GETENV - 環境変数の値を取得

GETENV 関数は、環境変数名を入力値として、その値を可変長文字値として返します。

構文　環境変数の値を取得

```
GETENV(var_name)
```

説明

`var_name`

固定長の文字

値を取得する環境変数名です。

例　環境変数の値を取得

次のリクエストは、EDAEXTSEC WebFOCUS Reporting Server 変数の値を取得します。

```
-SET &E1 = GETENV('EDAEXTSEC');
-TYPE &E1
```

WebFOCUS Reporting Server がセキュリティ ON で開始されている場合、返される値は ON、WebFOCUS Reporting Server がセキュリティ OFF で開始されている場合、返される値は OFF になります。

PUTENV - 環境変数に値を割り当て

PUTENV 関数は、環境変数に値を割り当てます。この関数は、割り当てに失敗した場合は 1、割り当てに成功した場合は 0 (ゼロ) の整数リターンコードを返します。

構文　環境変数に値を割り当て

```
PUTENV(var_name, var_value)
```

説明

`var_name`

固定長の文字

設定する環境変数の名前です。

`var_value`

文字

変数に割り当てる値です。

例 UNIX PS1 変数に値を割り当て

次のリクエストは、UNIX PS1 変数に「FOCUS/Shell:」という値を割り当てます。

```
-SET &P1 = PUTENV('PS1','FOCUS/Shell:');
```

これにより、ユーザが UNIX シェルコマンドの SH を発行した際に、UNIX に次のプロンプトが表示されます。

```
FOCUS/Shell:
```

次のリクエストは、「xxxx」という変数を作成し、その変数に割り当てる値を「this is a test」に設定します。次に、GETENV を使用して値を取得します。

```
-SET &XXXX=PUTENV(xxxx,'this is a test');
-SET &YYYY=GETENV(xxxx);
-TYPE  Return Code: &XXXX,  Variable value: &YYYY
```

出力結果は次のとおりです。

```
Return Code: 0,  Variable value: this is a test
```

SLACK - Slack チャンネルへのメッセージの投稿

SLACK 関数は、WebFOCUS プロシジャから Slack チャンネルにメッセージを投稿します。

- メッセージが正常に送信された場合、関数は true の値を返します。
- メッセージが正常に送信されなかった場合、関数はブランクを返します。

構文 Slack チャンネルへのメッセージの投稿

```
SLACK(workspace, channel, message)
```

説明

`workspace`

ワークスペース名です。

`channel`

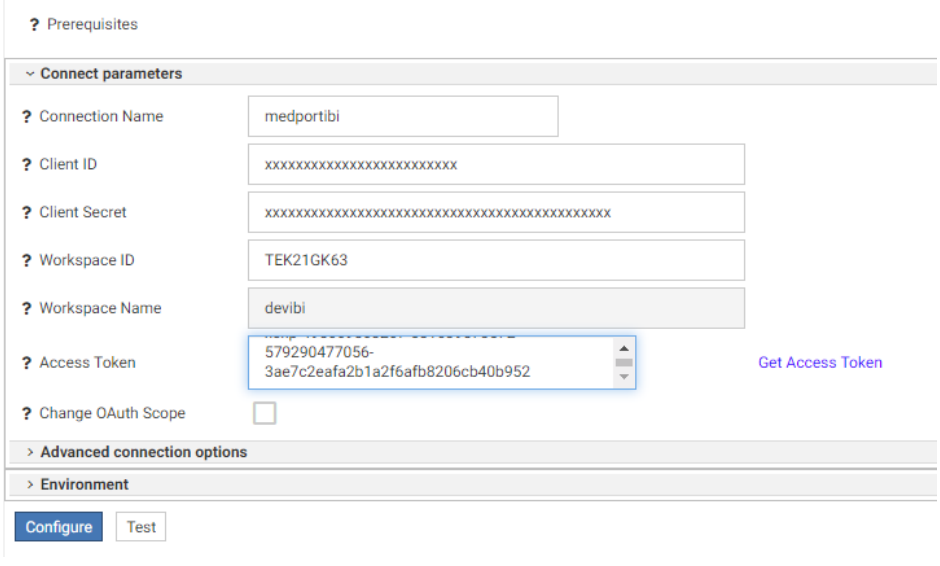
チャンネル名です。

`message`

メッセージを含む文字フィールドです。

例 WebFOCUS リクエストからの Slack メッセージの送信

Slack アダプタが構成され、下図のように、devibi ワークスペースへの接続が可能になりました。



次のリクエストは、DEPARTMENT が MIS の場合に、devibi ワークスペースの一般チャンネルに Slack メッセージを送信します。

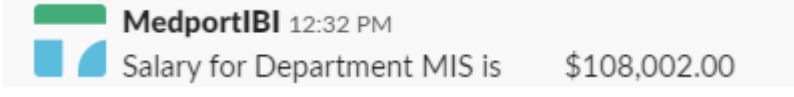
```
TABLE FILE ibisamp/EMPLOYEE
SUM
  CURR_SAL
  AND COMPUTE SLACK_MESSAGE/A200 = 'Salary for Department ' | DEPARTMENT ||
' is ' | LJUST(20, FPRINT(CURR_SAL,'D12.2M'), 'A20');
  AND COMPUTE CURR_SAL_SLACK/A20=IF DEPARTMENT EQ 'MIS'
      THEN SLACK('devibi', 'general', SLACK_MESSAGE) ELSE 'false';
      AS 'Message Sent,to Slack highlighting,Salary'
BY DEPARTMENT
HEADING
"Slack"
"Slack Function Example"
ON TABLE SET PAGE-NUM NOLEAD
ON TABLE NOTOTAL
ON TABLE SET STYLE *
INCLUDE=IBFS:/FILE/IBI_HTML_DIR/javaassist/intl/EN/ENIADefault_combine.sty,
$
ENDSTYLE
END
```

下図は、出力結果を示しています。

Slack
Slack Function Example

DEPARTMENT	CURR_SAL	SLACK_MESSAGE	Message Sent to Slack highlighting Salary
MIS	$108,002.00	Salary for Department MIS is $108,002.00	true
PRODUCTION	$114,282.00	Salary for Department PRODUCTION is $114,282.00	false

下図は、Slack チャンネルのメッセージを示しています。



20

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ついての情報を取得したり、システムサービスを使用することを可能にします。

多くの関数では、output 引数にフィールド名またはフォーマットを指定することができます。フォーマットを指定する場合、一重引用符 (') で囲みます。 ただし、関数がダイアログマネージャコマンドから呼び出される場合、この引数には常にフォーマットを指定する必要があります。関数の呼び出しおよび引数の指定についての詳細は、45 ページの 「 関数へのアクセスと呼び出し 」 を参照してください。

トピックス

CHECKPRIVS - 接続ユーザの権限ステータスの取得

CHECKPRIVS 関数は、指定された権限コードから、接続ユーザがその権限を所有する場合は Y、その権限を所有しない場合または権限が存在しない場合は N を返します。

注意：ユーザの全般権限のリストを表示するには、Web コンソール左上の [C] ボタンをクリックし、[マイコンソール]、[このユーザの全般権限を表示] を順に選択します。サーバ管理者権限を所有するユーザは、[アクセスコントロール] ページで特定のユーザの全般権限リストを表示することもできます。その場合は、ユーザ ID を右クリックし、コンテキストメニューから [プロパティ] を選択した後、[プロパティ] ページで [全般権限] タブをクリックします。

構文　接続ユーザの権限ステータスの取得

```
CHECKPRIVS(privcode, output)
```

説明

`privcode`

ステータスを取得する権限コードです。

`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。

例　接続ユーザの権限ステータスの取得

次のリクエストは、ADPTP (データアダプタの構成) 権限の権限ステータスを取得します。

```
-SET &PRIVSTATE = CHECKPRIVS(ADPTP,'A1');
-TYPE Privilege State is: &PRIVSTATE
```

出力結果は次のとおりです。

```
Privilege State is: Y
```

CLSDDREC - PUTDDREC 関数が開いたすべてのファイルを閉じる

CLSDDREC 関数は、PUTDDREC 関数が開いたすべてのファイルを閉じます。PUTDDREC がダイアログマネージャ -SET コマンドで呼び出された場合、リクエストまたは接続が終了するまでは、PUTDDREC が開いたファイルが自動的に閉じられることはありません。この場合、CLSDDREC 関数を呼び出すことにより、ファイルを閉じて、開いたファイルの情報の保存に使用したメモリを解放することができます。

構文　PUTDDREC 関数が開いたすべてのファイルを閉じる

```
CLSDDREC(output)
```

説明

`output`

整数

リターンコードです。次のいずれかの値が含まれます。

- **0** ファイルが閉じていることを示します。
- **1** ファイルを閉じる際にエラーが発生したことを示します。

例 PUTDDREC 関数が開いたファイルを閉じる

次の例では、PUTDDREC 関数で開いたファイルを閉じます。

```
CLSDDREC('I1')
```

FEXERR - エラーメッセージを取得

FEXERR 関数は、エラーメッセージを取得します。この関数は、出力メッセージを表示しないコマンドを使用するプロシジャで特に役立ちます。

エラーメッセージは、4 行以内のテキストで構成されます。先頭行にはメッセージが含まれ、それ以外の 3 行が存在する場合、詳細な説明が記述されています。FEXERR は、エラーメッセージの先頭行を取得します。

構文 エラーメッセージを取得

```
FEXERR(error, 'A72')
```

説明

`error`
数値

5 桁以内のエラー番号です。

`'A72'`
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。このフォーマットは、エラーメッセージの最大長である A72 です。

例　エラーメッセージを取得

FEXERR は、&ERR 変数に含まれている番号が割り当てられているエラーメッセージを取得します。ここでは、エラーメッセージ番号は 650 です。結果は変数 &&MSGVAR に A72 のフォーマットで返されます。

```
-SET &ERR = 650;
-SET &&MSGVAR = FEXERR(&ERR, 'A72');
-TYPE &&MSGVAR
```

出力結果は次のとおりです。

```
(FOC650) ディスクにアクセスできません。
```

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(length, 'varname', outlen, output)
```

説明

`length`
整数

環境変数名の長さをバイト数で指定します。

`varname`
文字

値を取得する環境変数名です。

`outlen`
整数

返される環境変数値の長さです。環境変数値を格納するフィールドを指定することもできます。

`output`
文字

環境変数値を格納するフィールドのフォーマットです。

FPUTENV - 環境変数に値を割り当て

利用可能なオペレーティングシステム - UNIX、Windows

FPUTENV 関数は、オペレーティングシステムの環境変数に値を設定します。FPUTENV 関数は、システム以外の部分で使用されている値を設定します。

制限：FOCPRINT 関数、FOCPATH 関数、および USERPATH 関数の設定または変更には、FPUTENV 関数は使用できません。これらの変数は、開始後にメモリに保持されるため、環境から再度読み込むことはできません。

構文　環境変数に値を割り当て

```
FPUTENV (varname_length,'varname',value_length, 'value', output)
```

説明

`varname_length`
整数

環境変数名の最大長をバイト数で指定します。

`varname`
文字

環境変数名です。一重引用符 (') で囲みます。名前は右揃えにし、varname_length で指定した最大長になるまでブランクを挿入する必要があります。

`value_length`
環境変数値の最大の長さ

注意：varname_length と value_length の合計長が 64 バイトを超えることはできません。

`value`
文字

環境変数 varname に割り当てる値です。文字列は右揃えにします。ブランクを含めることはできません。ブランクが含まれている場合、文字列の 1 つ目のブランク以降の部分は切り捨てられます。

`output`
整数

リターンコードです。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。変数が正しく設定されると、リターンコードは 0 (ゼロ) になります。これ以外の値は失敗を示します。

例　環境変数に値を割り当て

FPUTENV 関数は、値 FOCUS/Shell を PS1 変数に割り当て、結果をフォーマットが I4 のフィールドに格納します。

```
-SET &RC = FPUTENV(3,'PS1', 12, 'FOCUS/Shell:', 'I4');
```

ユーザが、変数値を表示する UNIX シェルコマンドを発行すると、リクエストは以下を表示します。

```
FOCUS/Shell:
```

GETCOOKI - ブラウザの Cookie 値を取得

セキュリティ認証情報は、さまざまなソースから、いくつかのフォーマットで提供することができます。他社製シングルサインオン製品のセキュリティ認証情報には、ブラウザの Cookie として渡されるものがあります。WebFOCUS Reporting Server は、GETCOOKI 関数を使用して、WebFOCUS Client からサーバに渡されたブラウザの Cookie の値を取得することができます。

構文　Cookie の値を取得

```
GETCOOKI('cookie_name', length)
```

説明

`cookie_name`

文字

値を取得するブラウザ Cookie 名です。Cookie 名の最大長は 80 バイトです。Cookie が設定されていない場合、または Cookie 名が 80 バイトを超える場合、この関数は「Invalid Cookie Name」というメッセージを返します。

`length`

文字 (An)

Cookie の長さです。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。指定した長さ n が、取得された Cookie の実際の長さより大きい場合、結果の末尾にブランクが追加されます。GETCOOKI から返された結果から末尾のブランクを削除するには、常に TRUNCATE(arg1) 関数を使用することをお勧めします。

例　ブラウザの Cookie 値を取得

次の関数を呼び出して、Oracle Access Manager (旧 Oblix) が作成した ObSSOCookie の値を取得します。

```
GETCOOKI('ObSSOCookie', 'A400')
```

GETHEADR - HTTP ヘッダ変数を取得

HTTP ヘッダ変数には、Web サーバ環境を記述した値や、Web サーバまたは他社製サインオン製品から取得される認証情報を指定した値が格納されます。WebFOCUS Reporting Server は、GETHEADR 関数を使用して、WebFOCUS Client からサーバに渡された HTTP ヘッダ変数の値を取得することができます。

構文　HTTP ヘッダ変数を取得

```
GETHEADR('varname', output)
```

説明

`varname`

文字

値を取得する HTTP ヘッダ変数名です。

`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。

例　HTTP ヘッダ変数値を取得

下図は、HTTP ヘッダのサンプルです。

Show incoming http header

Header	Value
cookie	JSESSIONID=3576041321067E425A4E2AC87D6E425E
connection	Keep-Alive
accept-encoding	gzip, deflate
referer	http://edamvt4:8080/ibi_apps_77/webconsole/webconsole/admin?IBIS_page=NODETREE
accept	image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*
content-length	62
cache-control	no-cache
accept-language	en-us
user-agent	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322; .NET CLR 2.0.50727; MS-RTC LM
content-type	application/x-www-form-urlencoded
host	edamvt4:8080

次の関数を呼び出して、HTTP「content-type」ヘッダから「application/x-www-form-urlencoded」という値を取得します。

```
GETHEADR('content-type', 'A150')
```

次の関数を呼び出して、HTTP ヘッダ「accept-language」から「en-us」という値を取得します。

```
GETHEADR('accept-language', 'A10')
```

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(output)
```

説明

`output`
文字 (A8 以上)

出力結果フィールドです。長さは、関数を発行するプラットホームに依存します。プラットホームで必要な長さを指定します。十分な長さを準備しないと、出力結果は切り捨てられます。

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。

```
DEFINE FILE EMPLOYEE
USERID/A8 WITH EMP_ID = GETUSER(USERID);
END

TABLE FILE EMPLOYEE
SUM CURR_SAL AS 'TOTAL SALARIES'
BY DEPARTMENT
HEADING
"SALARY REPORT RUN FROM USERID: <USERID"
" "
END
```

出力結果は次のとおりです。

```
SALARY REPORT RUN FROM USERID: doccar

DEPARTMENT            TOTAL SALARIES
----------            --------------
MIS                      $108,002.00
PRODUCTION               $114,282.00
```

GRPLIST - 接続ユーザのグループリストを取得

GRPLIST 関数は、接続しているユーザのグループ名、またはグループ名のリスト (コロン (:) で区切られたリスト) を返します。この関数は、LDAP セキュリティのすべてのタイプの接続でサポートされます。

グループリストに何も存在しない場合、または関数パラメータにエラーがある場合、この関数はブランクを返します。

構文　接続ユーザのグループメンバーシップリストの取得

```
GRPLIST(outputLength, outformat)
```

説明

`outputLength`
　出力文字列の長さです。

`outformat`
　出力文字列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　接続ユーザのグループリストを取得

次のリクエストは、接続しているユーザのグループリストを、&LIST というダイアログマネージャ変数に取得します。

```
-SET &LIST = GRPLIST(300, 'A300');
-TYPE &LIST
```

出力結果は次のとおりです。

```
#All_Technical_Staff;#CTSS_ADV;#CTSS_ADV;#CTSS_ADVT;#DSEDA
```

pgmuser1 ユーザに対して同一のリクエストを発行すると、このユーザが単一グループに属していることが分かります。

```
pgmgrp1
```

JOBNAME - 現在のプロセス ID 文字列の取得

JOBNAME 関数は、オペレーティングシステムから現在のプロセス ID 文字列を取得します。オペレーティングシステムレベルでは、この文字列はプロセス PID とも呼ばれます。この関数はすべての環境で有効ですが、通常はダイアログマネージャで使用され、オペレーティングシステムで使用される PID が数値の場合でも、値を文字列として返します。

注意：JOBNAME 文字列の表記および長さは、オペレーティングシステムによって異なります。たとえば、Windows、UNIX でのジョブ名は数値 (通常は最大 8 バイト長) です。アプリケーションが将来的に別の (予定外の) 環境で実行される可能性を考慮して、文字列の末尾が誤って削除されないように、最大 26 バイト長を使用することをお勧めします。この関数を ID 取得以外の目的で使用するアプリケーションでは、アプリケーションコードでこの相違点についても考慮する必要があります。

構文　現在のプロセス ID 文字列の取得

```
JOBNAME(length, output)
```

説明

`length`

整数

PID システムコールから返される文字列の最大バイト数です。

`output`

文字

返されるプロセス ID 文字列です。この文字列の長さは、この関数が発行されるプラットフォームによって異なります。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　プロセス ID 文字列の取得

次のリクエストでは、JOBNAME 関数を使用して現在のプロセス ID 文字列を A26 フォーマットで取得し、末尾のブランクを削除した上で、その文字列を -TYPE ステートメントで使用します。

```
-SET &JOBNAME = JOBNAME(26, 'A26');
-SET &JOBNAME = TRUNCATE(&JOBNAME);
-TYPE The Current system PID &JOBNAME is processing.
```

たとえば、Windows では次のように出力されます。

```
The Current system PID 2536 is processing.
```

PUTDDREC - 文字列をシーケンシャルファイルのレコードとして書き込み

PUTDDREC 関数は、文字列をシーケンシャルファイルのレコードとして書き込みます。このファイルは、FILEDEF コマンドで識別する必要があります。ファイルを (APPEND オプションで) 既存のファイルとして定義すると、新しいレコードが追加されます。新しいファイル (NEW) として定義し、このファイルが既に存在する場合、新しいレコードは既存のファイルを上書きします。

ファイルが開かれていない場合、PUTDDREC はこのファイルを開きます。PUTDDREC への各呼び出しには、同一ファイルまたは新しいファイルを使用することができます。PUTDDREC で開いたファイルはすべて、リクエストまたは接続が終了するまで開いたままになります。PUTDDREC で開いたファイルはすべて、リクエストまたは接続の終了時に自動的に閉じられます。

- 開く、閉じる、書き込む操作は、オペレーティングシステムにより処理されます。このため、ファイルへの書き込みにおいての要件、および PUTDDREC の呼び出し時の説明とは異なる結果は、使用する環境により異なります。入出力操作を実行するには、オペレーティングシステムのガイドラインを理解し、その手順に従う必要があります。
- PUTDDREC は、DEFINE FILE コマンド、またはマスターファイルの DEFINE で呼び出すことができます。ただし、PUTDDREC は、ファイルの 1 つ目のフィールド名がリクエストにより参照されるまでファイルを開きません。

PUTDDREC がダイアログマネージャ -SET コマンドで呼び出された場合、リクエストまたは接続が終了するまでは、PUTDDREC が開いたファイルが自動的に閉じられることはありません。この場合、CLSDDREC 関数を呼び出すことにより、ファイルを閉じて、開いたファイルの情報の保存に使用したメモリを解放することができます。

構文　シーケンシャルファイルのレコードとしての文字列の書き込み

```
PUTDDREC(ddname, dd_len, record_string, record_len, output)
```

説明

`ddname`
　文字

　FILEDEF コマンドでシーケンシャルファイルに割り当てられる論理名です。

`dd_len`
　数値

　論理名のバイト数です。

`record_string`
　文字

　シーケンシャルファイルの新しいレコードとして追加される文字列です。

`record_len`
　数値

　新しいレコードとして追加されるバイト数です。

　この値は、*record_string* の値以下である必要があります。すべての *record_string* をファイルに書き込むには、*record_len* の値は *record_string* の値と等しく、FILEDEF コマンドで宣言されたレコード長以下である必要があります。*record_len* の値が、宣言された長さ未満である場合、出力ファイルの各レコードの末尾に余分な文字が含まれる場合があります。*record_string* の値が、宣言された長さを超える場合、出力ファイルの *record_string* の末尾が切り捨てられる場合があります。

`output`
　整数

リターンコードです。次のいずれかの値が含まれます。

0 - レコードが追加されました。
-1 - FILEDEF ステートメントが見つかりません。
-2 - ファイルを開こうとしてエラーが発生しました。
-3 - ファイルにレコードを追加しようとしてエラーが発生しました。

例 TABLE リクエストでの PUTDDREC の呼び出し

次の例では、論理名が PUTDD1 である新しいファイルの定義を示します。続いて、TABLE リクエストは EMPLOYEE (従業員) データソースの各従業員に対して PUTDDREC を呼び出し、従業員の姓名、ID、ジョブコード、給料 (EDIT 関数により文字に変換済み) で構成されるファイルにレコードを書き込みます。0 (ゼロ、OUT1 内) のリターンコードは、PUTDDREC の呼び出しが成功したことを示します。

```
FILEDEF PUTDD1 DISK putdd1.datTABLE FILE EMPLOYEE
PRINT EMP_ID CURR_JOBCODE AS 'JOB' CURR_SAL
COMPUTE SALA/A12 = EDIT(CURR_SAL); NOPRINT
COMPUTE EMP1/A50= LAST_NAME|FIRST_NAME|EMP_ID|CURR_JOBCODE|SALA;
NOPRINT
COMPUTE OUT1/I1 = PUTDDREC('PUTDD1',6, EMP1, 50, OUT1);
BY LAST_NAME BY FIRST_NAME
END
```

出力結果は次のとおりです。

```
LAST_NAME       FIRST_NAME  EMP_ID     JOB         CURR_SAL   OUT1
---------       ----------  ---------  ---         --------   ----
BANNING         JOHN        119329144  A17       $29,700.00      0
BLACKWOOD       ROSEMARIE   326179357  B04       $21,780.00      0
CROSS           BARBARA     818692173  A17       $27,062.00      0
GREENSPAN       MARY        543729165  A07        $9,000.00      0
IRVING          JOAN        123764317  A15       $26,862.00      0
JONES           DIANE       117593129  B03       $18,480.00      0
MCCOY           JOHN        219984371  B02       $18,480.00      0
MCKNIGHT        ROGER       451123478  B02       $16,100.00      0
ROMANS          ANTHONY     126724188  B04       $21,120.00      0
SMITH           MARY        112847612  B14       $13,200.00      0
                RICHARD     119265415  A01        $9,500.00      0
STEVENS         ALFRED      071382660  A07       $11,000.00      0
```

このリクエストの実行後、シーケンシャルファイルには次のレコードが含まれます。

```
BANNING         JOHN      119329144A17000000029700
BLACKWOOD       ROSEMARIE 326179357B04000000021780
CROSS           BARBARA   818692173A17000000027062
GREENSPAN       MARY      543729165A07000000009000
IRVING          JOAN      123764317A15000000026862
JONES           DIANE     117593129B03000000018480
MCCOY           JOHN      219984371B02000000018480
MCKNIGHT        ROGER     451123478B02000000016100
ROMANS          ANTHONY   126724188B04000000021120
SMITH           MARY      112847612B14000000013200
SMITH           RICHARD   119265415A01000000009500
STEVENS         ALFRED    071382660A07000000011000
```

例　ダイアログマネージャ -SET コマンドでの PUTDDREC と CLSDDREC の呼び出し

次の例では、論理名が PUTDD1 である新しいファイルの定義を示します。1 つ目の -SET コマンドは、このファイルに追加するレコードを 1 つ作成します。2 つ目の -SET コマンドは、レコードを追加する PUTDDREC を呼び出します。3 つ目の -SET コマンドは、ファイルを閉じる CLSDDREC を呼び出します。操作が成功したことを示すリターンコードが表示されます。

```
FILEDEF PUTDD1 DISK putdd1.dat -SET &EMP1 = 'SMITH'|'MARY'|'A07'|'27000';
-TYPE DATA = &EMP1
-SET &OUT1 = PUTDDREC('PUTDD1',6, &EMP1, 17, 'I1');
-TYPE PUT RESULT = &OUT1
-SET &OUT1 = CLSDDREC('I1');
-TYPE CLOSE RESULT = &OUT1
```

出力結果は次のとおりです。

```
DATA = SMITHMARYA0727000
PUT RESULT = 0
CLOSE RESULT = 0
```

このプロシジャの実行後、シーケンシャルファイルには次のレコードが格納されます。

```
SMITHMARYA0727000
```

SLEEP - 指定した時間 (秒数) の実行保留

SLEEP 関数は、入力引数として指定した時間 (秒数) だけ実行を保留します。

この関数は、特定のプロシジャの開始を待機させる必要のある場合にダイアログマネージャで使用すると便利です。たとえば、FOCUS Database Server を開始した後、このサーバが稼動状態になるまでクライアントアプリケーションの開始を待機させることができます。

構文　指定した時間 (秒数) の実行保留

```
SLEEP(delay, output);
```

説明

`delay`
数値

実行を保留する時間 (秒数) です。ミリ秒の単位まで指定することができます。

`output`
数値

一重引用符 (') で囲んだフィールド名またはフォーマットです。戻り値は、保留時間として指定した値と同一です。

例　4 秒間の実行の保留

次の例は、現在の日時を計算し、実行を 4 秒間保留した後、保留後の現在の日時を計算します。

```
TABLE  FILE VIDEOTRK
PRINT TRANSDATE NOPRINT
COMPUTE
START_TIME/HYYMDSa = HGETC(8, START_TIME);
DELAY/I2 = SLEEP(4.0, 'I2');
END_TIME/HYYMDSa = HGETC(8, END_TIME);
IF RECORDLIMIT EQ 1
END
```

出力結果は次のとおりです。

```
START_TIME             DELAY  END_TIME
----------             -----  --------
2007/10/26  5:04:36pm      4  2007/10/26  5:04:40pm
```

SYSTEM - システムプログラムを呼び出し

利用可能なオペレーティングシステム - Windows

SYSTEM 関数は、プロシジャから DOS プログラム、バッチプログラム、または Windows アプリケーションを呼び出します。SYSTEM 関数は、DOS または Windows へコマンド文字列を渡します。プログラムは、コマンド文字列が DOS コマンドラインまたは Windows の [ファイル名を指定して実行] ダイアログボックスの [名前] テキストボックスに入力された場合と同様に実行されます。プログラムが終了すると、制御は WebFOCUS に戻ります。

FOCUS によるその後のコマンドの実行は、SYSTEM 関数により、ユーザがアプリケーションを終了するまで停止されます。プロシジャからの DOS プログラムおよび Windows アプリケーションの呼び出しは FOCUS DOS コマンドを使用しても可能ですが、SYSTEM 関数の機能はこの点において優れています。

SYSTEM 関数からコマンドを実行すると、コマンドは次のように実行します。

- SYSTEM に渡された文字列のコマンド名に拡張子 (.COM または .EXE) が含まれる場合、コマンドは直接呼び出されます。DOS コマンドインタプリタは使用しません。
- 文字列のコマンド名に拡張子が含まれていない、または .BAT 拡張子が含まれている場合、SYSTEM は DOS コマンドインタプリタである COMMAND.COM を呼び出し、指定したコマンドを実行し、終了します。
- SYSTEM 関数は、コマンド （CD、CLS、COPY、DEL、DIR、*drive:*、REN、TYPE) を FOCUS に転送します。DOS コマンドインタプリタには転送しません。その結果、これらのコマンドの変換は、DOS ではなく、FOCUS により直接実行されます。その動作は多少異なる場合があります。SYSTEM 関数でコマンドを DOS コマンドインタプリタに転送させるには、次の構文を使用します。

  ```
  SYSTEM(length, 'COMMAND /C string', returncode)
  ```

構文　DOS または Windows プログラムの呼び出し

```
SYSTEM(length, 'string ', returncode)
```

説明

`length`
整数

string で指定する文字列の長さです。

`string`
文字

コマンドラインパラメータを含む有効な Windows または DOS コマンドです。コマンドは一重引用符 (') で囲みます。

`returncode`
倍精度浮動小数点数

DOS エラーレベル値を含む変数の名前または長さです。

例　DIR コマンドの実行

SYSTEM 関数は、DIR コマンドを DOS コマンドインタプリタへ転送し、見出し情報や概要を含まないディレクトリリストを作成後、出力結果を「DIR.LIS」という名前のファイルに格納します。

```
-SET &RETCODE = SYSTEM(31,'COMMAND /C DIR /O-N /B >DIR.LIS','D4');
```

例　デフォルトディレクトリの変更

SYSTEM 関数は、デフォルトのディレクトリを変更し、作業が完了するまで処理を保留します。

```
-SET &ERRORLEVEL = SYSTEM(15,'CHDIR ¥CARDATA','D4');
```

例　Check Disk プログラムの実行

SYSTEM 関数は、Check Disk プログラムを実行し、出力結果を「CHKDSK.TXT」という名前のファイルに格納します。出力結果をファイルに格納することにより、読み取る必要のあるプログラムからのアクセスが可能になります。

```
-SET &RETCODE=SYSTEM(19,'CHKDSK > CHKDSK.TXT','D4');
```

21

簡略地理関数

簡略地理関数は、さまざまなタイプの位置データに対して位置ベースの演算を実行し、ジオコードされた地点を返します。これらの関数は、マップやグラフを生成するWebFOCUS ロケーションインテリジェンス製品で使用されます。一部の地理関数にはGIS サービスが使用され、ESRI ArcGIS 専有データにアクセスするための有効な認証情報が必要です。

トピックス

サンプル地理ファイル

地理関数の一部の例では、サンプルの地理ファイルが使用されます。1 つ目の esri-citibke.csv ファイルには、自転車シェアリングサービスのステーション名、緯度、経度、移動の開始時間、終了時間が格納されています。2 つ目の esri-geo10036.ftm ファイルには、ジオメトリデータが格納されています。これらのファイルを使用する例を実行するには、「esri」というアプリケーションを作成し、次のファイルをそのアプリケーションフォルダに配置する必要があります。

esri-citibike.mas

```
FILENAME=ESRI-CITIBIKE, SUFFIX=DFIX    ,
 DATASET=esri/esri-citibike.csv, $
  SEGMENT=CITIBIKE_TRIPDATA, SEGTYPE=S0, $
    FIELDNAME=TRIPDURATION, ALIAS=tripduration, USAGE=I7, ACTUAL=A5V,
      TITLE='tripduration', $
    FIELDNAME=STARTTIME, ALIAS=starttime, USAGE=HMDYYS, ACTUAL=A18,
      TITLE='starttime', $
    FIELDNAME=STOPTIME, ALIAS=stoptime, USAGE=HMDYYS, ACTUAL=A18,
      TITLE='stoptime', $
    FIELDNAME=START_STATION_ID, ALIAS='start station id', USAGE=I6, ACTUAL=A4V,
      TITLE='start station id', $
    FIELDNAME=START_STATION_NAME, ALIAS='start station name', USAGE=A79V,
      ACTUAL=A79BV, TITLE='start station name', $
    FIELDNAME=START_STATION_LATITUDE, ALIAS='start station latitude', USAGE=P20.15,
      ACTUAL=A18V, TITLE='start station latitude',
      GEOGRAPHIC_ROLE=LATITUDE,  $
    FIELDNAME=START_STATION_LONGITUDE, ALIAS='start station longitude', USAGE=P20.14,
      ACTUAL=A18V, TITLE='start station longitude',
      GEOGRAPHIC_ROLE=LONGITUDE,  $
    FIELDNAME=END_STATION_ID, ALIAS='end station id', USAGE=I6,
      ACTUAL=A4V, TITLE='end station id', $


    FIELDNAME=END_STATION_NAME, ALIAS='end station name', USAGE=A79V,
      ACTUAL=A79BV, TITLE='end station name', $
    FIELDNAME=END_STATION_LATITUDE, ALIAS='end station latitude', USAGE=P20.15,
      ACTUAL=A18V, TITLE='end station latitude',
      GEOGRAPHIC_ROLE=LATITUDE,  $
    FIELDNAME=END_STATION_LONGITUDE, ALIAS='end station longitude', USAGE=P20.14,
      ACTUAL=A18V, TITLE='end station longitude',
      GEOGRAPHIC_ROLE=LONGITUDE,  $
    FIELDNAME=BIKEID, ALIAS=bikeid, USAGE=I7, ACTUAL=A5,
      TITLE='bikeid', $
    FIELDNAME=USERTYPE, ALIAS=usertype, USAGE=A10V, ACTUAL=A10BV,
      TITLE='usertype', $
    FIELDNAME=BIRTH_YEAR, ALIAS='birth year', USAGE=I6, ACTUAL=A4,
      TITLE='birth year', $
    FIELDNAME=GENDER, ALIAS=gender, USAGE=I3, ACTUAL=A1,
      TITLE='gender', $
  SEGMENT=ESRIGEO, SEGTYPE=KU, SEGSUF=FIX, PARENT=CITIBIKE_TRIPDATA,
     DATASET=esri/esri-geo10036.ftm (LRECL 80 RECFM V, CRFILE=ESRI-GEO10036, $
```

esri-citibike.acx

```
SEGNAME=CITIBIKE_TRIPDATA,
  DELIMITER=',',
  ENCLOSURE=",
  HEADER=NO,
  CDN=OFF, $
```

esri-citibike.csv

注意：各レコード全体を 1 行に収める必要があります。そのため、このマニュアルのページ幅の制限により行末に改行記号が挿入されている場合は、その改行記号を削除する必要があります。

```
1094,11/1/2015 0:00,11/1/2015 0:18,537,Lexington Ave & E 24 St,
40.74025878,-73.98409214,531,Forsyth St & Broome St,
40.71893904,-73.99266288,23959,Subscriber,1980,1

520,11/1/2015 0:00,11/1/2015 0:08,536,1 Ave & E 30 St,
40.74144387,-73.97536082,498,Broadway & W 32 St,
40.74854862,-73.98808416,22251,Subscriber,1988,1

753,11/1/2015 0:00,11/1/2015 0:12,229,Great Jones St,
40.72743423,-73.99379025,328,Watts St & Greenwich St,
40.72405549,-74.00965965,15869,Subscriber,1981,1

353,11/1/2015 0:00,11/1/2015 0:06,285,Broadway & E 14 St,
40.73454567,-73.99074142,151,Cleveland Pl & Spring St,
40.72210379,-73.99724901,21645,Subscriber,1987,1

1285,11/1/2015 0:00,11/1/2015 0:21,268,Howard St & Centre St,
40.71910537,-73.99973337,476,E 31 St & 3 Ave,40.74394314,-73.97966069,14788,Customer,,0

477,11/1/2015 0:00,11/1/2015 0:08,379,W 31 St & 7 Ave,40.749156,-73.9916,546,E 30 St &
Park Ave S,40.74444921,-73.98303529,21128,Subscriber,1962,2

362,11/1/2015 0:00,11/1/2015 0:06,407,Henry St & Poplar St,
40.700469,-73.991454,310,State St & Smith St,40.68926942,-73.98912867,21016,Subscriber,
1978,1

2316,11/1/2015 0:00,11/1/2015 0:39,147,Greenwich St & Warren St,
40.71542197,-74.01121978,441,E 52 St & 2 Ave,40.756014,-73.967416,24117,Subscriber,
1988,2

627,11/1/2015 0:00,11/1/2015 0:11,521,8 Ave & W 31 St,
40.75096735,-73.99444208,285,Broadway & E 14 St,
40.73454567,-73.99074142,17048,Subscriber,1986,2

1484,11/1/2015 0:01,11/1/2015 0:26,281,Grand Army Plaza & Central Park S,
40.7643971,-73.97371465,367,E 53 St & Lexington Ave,
40.75828065,-73.97069431,16779,Customer,,0
```

```
284,11/1/2015 0:01,11/1/2015 0:06,247,Perry St & Bleecker St,
40.73535398,-74.00483091,453,W 22 St & 8 Ave,40.74475148,-73.99915362,17272,Subscriber,
1976,1


886,11/1/2015 0:01,11/1/2015 0:16,492,W 33 St & 7 Ave,40.75019995,-73.99093085,377,6
Ave & Canal St,40.72243797,-74.00566443,23019,Subscriber,1982,1


1379,11/1/2015 0:01,11/1/2015 0:24,512,W 29 St & 9 Ave,40.7500727,-73.99839279,445,E
10 St & Avenue A,40.72740794,-73.98142006,23843,Subscriber,1962,2


179,11/1/2015 0:01,11/1/2015 0:04,319,Fulton St & Broadway,
40.711066,-74.009447,264,Maiden Ln & Pearl St,
40.70706456,-74.00731853,22538,Subscriber,1981,1


309,11/1/2015 0:01,11/1/2015 0:07,160,E 37 St & Lexington Ave,
40.748238,-73.978311,362,Broadway & W 37 St,40.75172632,-73.98753523,22042,Subscriber,
1988,1


616,11/1/2015 0:02,11/1/2015 0:12,479,9 Ave & W 45 St,40.76019252,-73.9912551,440,E 45
St & 3 Ave,40.75255434,-73.97282625,22699,Subscriber,1982,1


852,11/1/2015 0:02,11/1/2015 0:16,346,Bank St & Hudson St,
40.73652889,-74.00618026,375,Mercer St & Bleecker St,
40.72679454,-73.99695094,21011,Subscriber,1991,1


1854,11/1/2015 0:02,11/1/2015 0:33,409,DeKalb Ave & Skillman St,
40.6906495,-73.95643107,3103,N 11 St & Wythe Ave,
40.72153267,-73.95782357,22011,Subscriber,1992,1


1161,11/1/2015 0:02,11/1/2015 0:21,521,8 Ave & W 31 St,40.75096735,-73.99444208,461,E
20 St & 2 Ave,40.73587678,-73.98205027,19856,Subscriber,1957,1


917,11/1/2015 0:02,11/1/2015 0:17,532,S 5 Pl & S 4 St,40.710451,-73.960876,393,E 5 St
& Avenue C,40.72299208,-73.97995466,18598,Subscriber,1991,1
```

esri-geo10036.mas

```
FILENAME=ESRI-GEO10036, SUFFIX=FIX     ,
 DATASET=esri/esri-geo10036.ftm (LRECL 80 RECFM V, IOTYPE=STREAM, $
  SEGMENT=ESRIGEO, SEGTYPE=S0, $
    FIELDNAME=GEOMETRY, ALIAS=GEOMETRY, USAGE=TX80L, ACTUAL=TX80,
      MISSING=ON, $
```

esri-geo10036.ftm


```
{"rings":[[[-73.9803889998524,40.7541490002762],[-73.9808779999197,40.7534830001
404],[-73.9814419998484,40.7537140000011],[-73.9824040001445,40.7541199998382],[
-73.982461000075,40.7541434001978],[-73.9825620002361,40.7541850001377],[-73.983
2877000673,40.7544888999428],[-73.9833499997027,40.7545150000673],[-73.983644399
969,40.7546397998869],[-73.9836849998628,40.7546570003204],[-73.9841276003085,40
.7548161002829],[-73.984399700086,40.7544544999752],[-73.9846140004357,40.754165
0001147],[-73.984871999743,40.7542749997914],[-73.9866590003126,40.7550369998577
],[-73.9874449996869,40.7553720000178],[-73.9902640001834,40.756570999552],[-73.
9914340001789,40.7570449998269],[-73.9918260002697,40.7572149995726],[-73.992429
0001982,40.7574769999636],[-73.9927679996434,40.7576240004473],[-73.993069000034
3,40.7578009996165],[-73.9931059999419,40.7577600004237],[-73.9932120003335,40.7
576230004012],[-73.9933250001486,40.7576770001934],[-73.9935390001247,40.7577669

998472],[-73.993725999755,40.7578459998931],[-73.9939599997542,40.757937999639],
[-73.9940989998689,40.7579839999617],[-73.9941529996611,40.7579959996157],[-73.9
942220001452,40.7580159996387],[-73.9943040003293,40.7580300002843],[-73.9943650
004444,40.7580330004227],[-73.99446499966,40.7580369997078],[-73.9945560002591,4
0.7580300002843],[-73.9946130001898,40.7580209998693],[-73.9945689999594,40.7580
809999383],[-73.9945449997519,40.7581149997075],[-73.9944196999092,40.7582882001
404],[-73.9943810002829,40.7583400001909],[-73.9953849998179,40.7587409997973],[
-73.9959560000693,40.7589690004191],[-73.9960649996999,40.7590149998424],[-73.99
68730000888,40.7593419996336],[-73.996975000296,40.7593809996335],[-73.997314999
7874,40.7595379996789],[-73.9977009996014,40.7597030000935],[-73.998039999946,40
.7598479995856],[-73.998334000014,40.7599709998618],[-73.9987769997587,40.760157
0003453],[-73.9990089996656,40.7602540003219],[-74.0015059997021,40.761292999672

2],[-74.0016340002089,40.7613299995799],[-74.0015350001401,40.7614539999022],[-7
4.0014580001865,40.7615479997405],[-74.0013640003483,40.7616560002242],[-74.0013
050003255,40.7617199995784],[-74.0011890003721,40.7618369995779],[-74.0010579997
269,40.7619609999003],[-74.0009659999808,40.7620389999],[-74.0008649998198,40.76
21230001764],[-74.0008390004195,40.7621430001993],[-74.0006839995669,40.76226100
0245],[-74.000531999752,40.7623750001062],[-74.0003759997525,40.7624849997829],[
-74.0002840000066,40.7625510001286],[-73.9998659996161,40.762850999574],[-73.999
8279996624,40.7628779999198],[-73.9995749996864,40.7630590001727],[-73.999312000
1487,40.7632720001028],[-73.9991639996189,40.7633989996642],[-73.998941000127,40
.7636250001936],[-73.9987589998279,40.7638580001466],[-73.9986331999622,40.76402
77004181],[-73.9986084002574,40.7640632002565],[-73.9984819996445,40.76423400039
89],[-73.9983469997142,40.7644199999831],[-73.998171999738,40.7646669996823],[-7
3.9980319995771,40.7648580003964],[-73.9979881998955,40.7649204996813],[-73.9979
368000432,40.7649942000224],[-73.9978947999051,40.7650573998791],[-73.9977017001
```

```
733,40.7653310995507],[-73.9975810003629,40.765481000348],[-73.9975069996483,40.
7654519999099],[-73.9956019999323,40.7646519998899],[-73.9955379996789,40.764625
0004434],[-73.9954779996099,40.7646030003282],[-73.9949389999348,40.764369000329
1],[-73.9936289997785,40.7638200001929],[-73.9934620001711,40.7637539998473],[-7
3.9931520002646,40.7636270002859],[-73.992701000151,40.7634409998023],[-73.99244
19000736,40.7633312995998],[-73.9898629996777,40.7622390001298],[-73.98861200044
34,40.761714000201],[-73.988021000169,40.761460000179],[-73.987028000242,40.7610
439998808],[-73.9867690998141,40.7609346998765],[-73.9848240002274,40.7601130001
149],[-73.9841635003452,40.7598425002312],[-73.9813259998949,40.7586439998208],[
-73.9805479999902,40.7583159999834],[-73.9793569999256,40.757814000216],[-73.978
1150002071,40.7572939996184],[-73.9785670003668,40.7566709996669],[-73.979014000
2958,40.7560309998308],[-73.9794719998329,40.7554120000638],[-73.9799399998311,4
0.7547649999048],[-73.9802380000836,40.7543610001601],[-73.9803889998524,40.7541
490002762]]]}
%$
```

GIS_DISTANCE - 2 地点間の距離の計算

GIS_DISTANCE 関数は、GIS サービスを使用して、2 地点間の距離を計算します。

構文　　2 地点間の距離の計算

```
GIS_DISTANCE(geo_point1,geo_point2)
```

説明

`geo_point1,geo_point2`

固定長の文字、JSON 形式で記述された地点の格納に必要な長さ (例、A200)

距離を計算する地点です。

注意：地点を生成するには、GIS_POINT 関数を使用することができます。

例　2 地点間の距離の計算

次のリクエストは、citibike.csv ファイルを使用します。このファイルには、自転車シェアリングサービスのステーション名、緯度、経度、移動の開始時間、終了時間が格納されています。このリクエストでは、GIS_POINT 関数を使用して、出発ステーションと到着ステーションの地点を定義します。次に、GIS_DISTANCE 関数を使用して、2 地点間の距離を計算します。

```
DEFINE FILE esri/esri-citibike
STARTPOINT/A200 = GIS_POINT('4326', START_STATION_LONGITUDE,
START_STATION_LATITUDE);
ENDPOINT/A200 = GIS_POINT('4326', END_STATION_LONGITUDE,
END_STATION_LATITUDE);
Distance/P10.2 = GIS_DISTANCE(ENDPOINT, STARTPOINT);
END
TABLE FILE esri/esri-citibike
PRINT END_STATION_NAME AS End Distance
BY START_STATION_NAME AS Start
ON TABLE SET PAGE NOLEAD
END
```

下図は、出力結果を示しています。

Start	End	Distance
1 Ave & E 30 St	Broadway & W 32 St	.83
8 Ave & W 31 St	Broadway & E 14 St	1.15
	E 20 St & 2 Ave	1.23
9 Ave & W 45 St	E 45 St & 3 Ave	1.10
Bank St & Hudson St	Mercer St & Bleecker St	.83
Broadway & E 14 St	Cleveland Pl & Spring St	.92
DeKalb Ave & Skillman St	N 11 St & Wythe Ave	2.13
E 37 St & Lexington Ave	Broadway & W 37 St	.54
Fulton St & Broadway	Maiden Ln & Pearl St	.30
Grand Army Plaza & Central Park S	E 53 St & Lexington Ave	.45
Great Jones St	Watts St & Greenwich St	.87
Greenwich St & Warren St	E 52 St & 2 Ave	3.62
Henry St & Poplar St	State St & Smith St	.78
Howard St & Centre St	E 31 St & 3 Ave	2.01
Lexington Ave & E 24 St	Forsyth St & Broome St	1.54
Perry St & Bleecker St	W 22 St & 8 Ave	.71
S 5 Pl & S 4 St	E 5 St & Avenue C	1.32
W 29 St & 9 Ave	E 10 St & Avenue A	1.80
W 31 St & 7 Ave	E 30 St & Park Ave S	.55
W 33 St & 7 Ave	6 Ave & Canal St	2.07

GIS_DRIVE_ROUTE - 2 地点間の走行経路の計算

GIS_DRIVE_ROUTE 関数は、GIS サービスを使用して、2 地点間の走行経路を計算します。

注意：この関数では、GIS サービスが使用され、指定ログイン情報を使用する ESRI ArcGIS アダプタ接続が必要です。

構文　　2 地点間の走行経路の計算

```
GIS_DRIVE_ROUTE(geo_start_point,geo_end_point)
```

説明

`geo_start_point,geo_point2`

固定長の文字、JSON 形式で記述された地点の格納に必要な長さ (例、A200)

走行経路の計算に使用する出発地点です。

注意：地点を生成するには、GIS_POINT 関数を使用することができます。

`geo_end_point,geo_point2`

固定長の文字、JSON 形式で記述された地点の格納に必要な長さ (例、A200)

走行経路の計算に使用する到着地点です。

注意：地点を生成するには、GIS_POINT 関数を使用することができます。

走行経路が返されるフィールドのフォーマットは TX です。

例　2 地点間の走行経路の計算

次のリクエストは、citibike.csv ファイルを使用します。このファイルには、自転車シェアリングサービスのステーション名、緯度、経度、移動の開始時間、終了時間が格納されています。このリクエストでは、GIS_POINT 関数を使用して、出発ステーションと到着ステーションの地点を定義します。次に、GIS_DRIVE_ROUTE 関数を使用して、出発地点から到着地点までの経路を計算します。

```
DEFINE FILE esri/esri-citibike
STARTPOINT/A200 = GIS_POINT('4326', START_STATION_LONGITUDE,
START_STATION_LATITUDE);
ENDPOINT/A200 = GIS_POINT('4326', END_STATION_LONGITUDE,
END_STATION_LATITUDE);
Route/TX140 (GEOGRAPHIC_ROLE=GEOMETRY_LINE) =
          GIS_DRIVE_ROUTE(ENDPOINT, STARTPOINT);
END
TABLE FILE esri/esri-citibike
PRINT START_STATION_NAME AS Start END_STATION_NAME AS End Route
WHERE START_STATION_ID EQ 147
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF,SIZE-11,$
ENDSTYLE
END
```

下図は、出力結果を示しています。

Start	End	Route
Greenwich St & Warren St	E 52 St & 2 Ave	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPolyline","geometry": {"paths":[[[-73.967401732999974,40.756047761000048],[-73.965229999999963,40.755130000000065],[-73.964739999999949,40.755790000000047],[-73.962869999999953,40.758340000000032],[-73.961959999999976,40.759610000000066],[-73.961609999999951,40.760090000000048],[-73.961499999999944,40.760240000000067],[-73.959829999999954,40.759510000000034],[-73.959569999999985,40.759400000000028],[-73.959259999999972,40.759280000000047],[-73.959179999999947,40.759390000000053],[-73.958949999999959,40.759720000000073],[-73.958869999999933,40.759830000000079],[-73.958789999999965,40.759940000000029],[-73.958589999999958,40.760210000000029],[-73.958339999999964,40.760530000000074],[-73.958189999999945,40.760730000000081],[-73.957859999999982,40.761150000000043],[-73.957319999999982,40.760950000000037],[-73.957139999999981,40.760880000000043],[-73.956929999999943,40.760660000000003],[-73.957879999999989,40.759760000000028],[-73.957919999999945,40.759710000000041],[-73.958119999999951,40.759400000000028],[-73.958489999999983,40.758960000000059],[-73.958609999999965,40.758770000000027],[-73.958639999999946,40.758720000000039],[-73.958689999999933,40.758650000000046],[-73.958739999999977,40.758580000000052],[-73.958969999999965,40.758280000000007],[-73.959369999999979,40.757710000000031],[-73.959849999999996,40.757350000000031],[-73.960149999999942,40.756990000000003],[-73.960489999999936,40.756500000000074],[-73.961799999999982,40.755220000000065],[-73.961939999999997,40.755090000000052],[-73.963129999999978,40.753890000000069],[-73.963889999999935,40.753060000000062],[-73.964119999999998,40.752800000000036],[-73.964639999999974,40.752230000000054],[-73.965099999999995,40.751780000000053],[-73.966719999999953,40.749800000000005],[-73.968079999999986,40.748140000000035],[-73.968179999999961,40.748020000000054],[-73.968259999999987,40.747930000000053],[-73.968379999999968,40.747780000000034],[-73.968639999999937,40.747440000000004],[-73.970579999999984,40.745410000000049],[-73.971699999999942,40.743880000000047],[-73.972109999999986,40.743210000000033],[-73.972149999999942,40.743130000000065],[-73.972659999999962,40.741790000000037],[-73.972819999999956,40.741010000000074],[-73.973059999999975,40.739860000000078],[-73.973479999999938,40.738830000000064],[-73.974089999999933,40.737920000000031],[-73.974449999999933,40.737460000000056],[-73.974729999999965,40.737050000000067],[-73.975049999999953,40.736270000000047],[-73.974979999999996,40.735410000000059],[-73.974409999999978,40.733260000000003],[-73.974259999999958,40.732670000000041],[-73.974179999999933,40.732340000000079],[-73.973819999999989,40.731240000000071],[-73.972349999999949,40.729860000000031],[-73.972029999999961,40.729450000000043],[-73.971819999999998,40.728950000000054],[-73.971769999999935,40.728270000000066],[-73.971829999999954,40.727830000000004],[-73.971959999999967,40.726850000000007],[-73.972009999999955,40.726610000000051],[-73.973299999999938,40.724280000000078],[-73.974449999999933,40.722320000000025],[-73.974659999999972,40.721390000000042],[-73.974729999999965,40.720750000000066],[-73.974959999999953,40.718960000000038],[-73.974979999999996,40.718760000000032],[-73.975229999999954,40.717580000000055],[-73.976309999999955,40.715240000000051],[-73.976549999999975,40.714770000000044],[-73.976699999999937,40.714480000000037],[-73.976859999999988,40.714160000000049],[-73.977689999999939,40.712550000000078],[-73.978309999999965,40.711790000000065],[-73.978529999999978,40.711630000000071],[-73.978689999999972,40.711520000000064],[-73.979819999999961,40.711020000000076],[-73.981799999999964,40.710820000000069],[-73.985259999999982,40.710560000000044],[-73.991719999999987,40.709740000000068],[-73.992799999999988,40.709590000000048],[-73.994009999999946,40.709420000000008],[-73.994579999999985,40.709340000000054],[-73.995429999999942,40.709230000000048],[-73.995919999999956,40.709160000000054],[-73.996199999999988,40.709120000000041],[-73.996899999999982,40.709020000000066],[-73.997399999999971,40.708940000000041],[-73.998499999999979,40.708630000000028],[-73.999509999999987,40.708120000000065],[-74.002469999999996,40.706520000000069],[-74.003479999999968,40.705930000000008],[-74.004169999999988,40.705470000000048],[-74.007859999999937,40.703110000000038],[-74.009209999999939,40.702280000000003],[-74.009979999999985,40.701950000000068],[-74.010919999999942,40.701730000000055],[-74.011149999999986,40.701680000000067],[-74.011989999999969,40.701480000000006],[-74.012239999999963,40.701420000000041],[-74.012679999999989,40.701320000000067],[-74.013289999999984,40.701210000000006],[-74.014119999999934,40.701220000000035],[-74.014759999999967,40.701400000000035],[-74.015359999999987,40.701750000000061],[-74.015429999999981,40.701800000000048],[-74.015479999999968,40.701850000000036],[-74.015729999999962,40.702140000000043],[-74.015999999999963,40.702620000000024],[-74.016049999999995,40.702770000000044],[-74.016299999999944,40.703400000000045],[-74.016639999999938,40.704650000000072],[-74.016609999999957,40.704950000000053],[-74.016569999999945,40.705120000000079],[-74.016009999999937,40.706470000000024],[-74.015719999999988,40.707210000000032],[-74.015689999999995,40.707280000000026],[-74.015619999999956,40.707460000000026],[-74.015589999999975,40.707530000000077],[-74.015399999999943,40.707970000000046],[-74.014719999999954,40.709780000000008],[-74.014599999999973,40.710360000000037],[-74.014489999999967,40.710890000000063],[-74.014469999999996,40.711010000000044],[-74.014229999999941,40.712080000000071],[-74.013749999999959,40.713680000000068],[-74.013429999999971,40.714420000000075],[-74.013369999999952,40.714590000000044],[-74.013229999999965,40.715210000000007],[-74.013079999999945,40.715750000000071],[-74.012929999999983,40.716390000000047],[-74.011109999999974,40.715590000000077],[-74.011159794999969,40.715405758000031]]]}}

例　2 地点間の走行経路の描画

次のリクエストは、GIS_DRIVE_ROUTE 関数を使用して出発ステーションから到着ステーションまでの走行経路を生成し、その経路を ESRI マップ上に描画します。

```
DEFINE FILE esri-citibike
STARTPOINT/A200 = GIS_POINT('4326', START_STATION_LONGITUDE,
START_STATION_LATITUDE);
ENDPOINT/A200 = GIS_POINT('4326', END_STATION_LONGITUDE,
END_STATION_LATITUDE);
Route/TX80 (GEOGRAPHIC_ROLE=GEOMETRY_LINE) =
          GIS_DRIVE_ROUTE(ENDPOINT, STARTPOINT);
END
```

```
GRAPH FILE ESRI-CITIBIKE
 PRINT
   START_STATION_NAME
   END_STATION_NAME
WHERE START_STATION_ID EQ 147
 ON TABLE PCHOLD FORMAT JSCHART
 ON TABLE SET LOOKGRAPH CHOROPLETH
 ON TABLE SET EMBEDHEADING ON
 ON TABLE SET AUTOFIT ON
 ON TABLE SET STYLE *
  TYPE=REPORT, TITLETEXT='Map', PAGESIZE=E, CHART-LOOK=com.esri.map, $
  TYPE=DATA, COLUMN=N1, /*START_STATION_NAME*/
   BUCKET=tooltip, $
  TYPE=DATA, COLUMN=N2, /*END_STATION_NAME*/

  *GRAPH_JS_FINAL
"legend": {"visible": true},
"extensions" : { "com.esri.map" :
  { "scalebar" :
{
    "scalebarUnit": "dual",
    "attachTo" : "bottom-left"
  },
  "baseMapInfo": {
       "drawBasemapControl" : false,
       "showArcGISBasemaps" : false,
            "customBaseMaps" : [
            {"ibiBaseLayer" : "dark-gray"}
        ]
   },
  "overlayLayers":
  [{
  "ibiDataLayer": {"map-geometry" :  {"map_by_field" : "Route"}}, "title" :
"Chart"}]
  },
"introAnimation": "{¥"enabled¥":false}"
}

  *END
 ENDSTYLE
 HEADING
  "Chart Drive Route"
END
```

下図は、出力結果を示しています。

GIS_GEOCODE_ADDR - 完全な住所のジオコード

GIS_GEOCODE_ADDR 関数は、GIS ジオコードサービスを使用して、完全な住所の地点を取得します。

注意：この関数では、GIS サービスが使用され、指定ログイン情報を使用する ESRI ArcGIS アダプタ接続が必要です。

構文　　完全な住所のジオコード

```
GIS_GEOCODE_ADDR(address[, country])
```

説明

`address`

固定長の文字

ジオコードする完全な住所です。

`country`

固定長の文字

国名です。国名がアメリカ合衆国の場合は、オプションとして指定します。

例　完全な住所のジオコード

次のリクエストは、番地、市、州、郵便番号を連結して完全な住所を作成します。次に、GIS_GEOCODE_ADDR 関数を使用して、その住所の GIS 地点を作成します。

```
DEFINE FILE WF_RETAIL_LITE
GADDRESS/A200 =ADDRESS_LINE_1 || '  ' | CITY_NAME || '  ' | STATE_PROV_NAME
|| ' ' | POSTAL_CODE;
GEOCODE1/A200 = GIS_GEOCODE_ADDR(GADDRESS);
END
TABLE FILE WF_RETAIL_LITE
PRINT ADDRESS_LINE_1 AS Address GEOCODE1
BY POSTAL_CODE AS Zip
WHERE CITY_NAME EQ 'New York'
WHERE POSTAL_CODE FROM '10013' TO '10020'
ON TABLE SET PAGE NOPAGE
END
```

下図は、出力結果を示しています。

Zip	Address	GEOCODE1
10013	125 Worth St	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-74.00269, "y":40.71543}}
10016	139 E 35Th St	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.97911, "y":40.74705}}
10017	2 United Nations Plz	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.97115, "y":40.75111}}
	405 E 42Nd St	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.96956, "y":40.74867}}
	405 E 42Nd St	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.96956, "y":40.74867}}
	219 E 42Nd St	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.97333, "y":40.75030}}
	330 Madison Ave	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.97906, "y":40.75316}}
10018	119 W 40Th St Fl 10	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.98599, "y":40.75398}}
	11 West 40Th Street	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.98235, "y":40.75245}}
10019	31 West 52Nd Street	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.97776, "y":40.76044}}
	1301 Ave Of The Americas	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.97945, "y":40.76125}}
	1345 Avenue Of The Americas	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.97843, "y":40.76264}}
	745 7Th Ave	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.98340, "y":40.76077}}
10020	1221 Avenue Of The Americas	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.98129, "y":40.75874}}
	1271 Avenue Of The Americas	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.98018, "y":40.76025}}

GIS_GEOCODE_ADDR_CITY - 番地、市、州のジオコード

GIS_GEOCODE_ADDR_CITY 関数は、GIS ジオコードサービスを使用して、番地、市、州、国 (オプション) から地点を取得します。返される値は、JSON 形式で記述された地点の格納に必要な長さの固定長文字フォーマットです (例、A200)。

注意：この関数では、GIS サービスが使用され、指定ログイン情報を使用する ESRI ArcGIS アダプタ接続が必要です。

構文　番地、市、州のジオコード

```
GIS_GEOCODE_ADDR_CITY( street_addr, city , state [, country])
```

説明

`street_addr`

固定長の文字

ジオコードする番地です。

`city`

固定長の文字

番地に関連する都市名です。

`state`

固定長の文字

番地に関連する州名です。

`country`

固定長の文字

国名です。国名がアメリカ合衆国の場合は、オプションとして指定します。

例　番地、市、州のジオコード

次のリクエストは、GIS_GEOCODE_ADDR_CITY 関数を使用して、住所をジオコードします。

```
DEFINE FILE WF_RETAIL_LITE
GEOCODE1/A200 = GIS_GEOCODE_ADDR_CITY(ADDRESS_LINE_1, CITY_NAME ,
STATE_PROV_NAME);
END
TABLE FILE WF_RETAIL_LITE
PRINT ADDRESS_LINE_1 AS Address GEOCODE1
BY POSTAL_CODE AS Zip
WHERE CITY_NAME EQ 'New York'
WHERE POSTAL_CODE FROM '10013' TO '10020'
ON TABLE SET PAGE NOPAGE
END
```

下図は、出力結果を示しています。

Zip	Address	GEOCODE1
10013	125 Worth St	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-74.00269, "y":40.71543}}
10016	139 E 35Th St	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.94483, "y":40.65194}}
10017	2 United Nations Plz	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.97115, "y":40.75111}}
	405 E 42Nd St	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.96956, "y":40.74867}}
	405 E 42Nd St	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.96956, "y":40.74867}}
	219 E 42Nd St	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.97333, "y":40.75030}}
	330 Madison Ave	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.97906, "y":40.75316}}
10018	119 W 40Th St Fl 10	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.98599, "y":40.75398}}
	11 West 40Th Street	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.98235, "y":40.75245}}
10019	31 West 52Nd Street	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.97776, "y":40.76044}}
	1301 Ave Of The Americas	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.97945, "y":40.76125}}
	1345 Avenue Of The Americas	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.97843, "y":40.76264}}
	745 7Th Ave	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.98340, "y":40.76077}}
10020	1221 Avenue Of The Americas	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.98129, "y":40.75874}}
	1271 Avenue Of The Americas	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.98018, "y":40.76025}}

GIS_GEOCODE_ADDR_POSTAL - 番地、郵便番号のジオコード

GIS_GEOCODE_ADDR_POSTAL 関数は、GIS ジオコードサービスを使用して、番地、郵便番号、国 (オプション) の地点を取得します。返される値は、JSON 形式で記述された地点の格納に必要な長さの固定長文字フォーマットです (例、A200)。

注意：この関数では、GIS サービスが使用され、指定ログイン情報を使用する ESRI ArcGIS アダプタ接続が必要です。

構文　番地、郵便番号のジオコード

```
GIS_GEOCODE_ADDR_POSTAL( street_addr, postal_code [, country])
```

説明

`street_addr`

固定長の文字

ジオコードする番地です。

`postal_code`

固定長の文字

番地に関連する郵便番号です。

```
country
```

固定長の文字

国名です。国名がアメリカ合衆国の場合は、オプションとして指定します。

例　番地、郵便番号のジオコード

次のリクエストは、GIS_GEOCODE_ADDR_POSTAL 関数を使用して、住所をジオコードします。

```
DEFINE FILE WF_RETAIL_LITE
GEOCODE1/A200 = GIS_GEOCODE_ADDR_POSTAL(ADDRESS_LINE_1, POSTAL_CODE);
END
TABLE FILE WF_RETAIL_LITE
PRINT ADDRESS_LINE_1 AS Address GEOCODE1
BY POSTAL_CODE AS Zip
WHERE CITY_NAME EQ 'New York'
WHERE POSTAL_CODE FROM '10013' TO '10020'
ON TABLE SET PAGE NOPAGE
END
```

下図は、出力結果を示しています。

Zip	Address	GEOCODE1
10013	125 Worth St	{"spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-74.00269, "y":40.71543}}
10016	139 E 35Th St	{"spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.97911, "y":40.74705}}
10017	2 United Nations Plz	{"spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.97115, "y":40.75111}}
	405 E 42Nd St	{"spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.96956, "y":40.74867}}
	405 E 42Nd St	{"spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.96956, "y":40.74867}}
	219 E 42Nd St	{"spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.97333, "y":40.75030}}
	330 Madison Ave	{"spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.97906, "y":40.75316}}
10018	119 W 40Th St Fl 10	{"spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.98599, "y":40.75398}}
	11 West 40Th Street	{"spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.98235, "y":40.75245}}
10019	31 West 52Nd Street	{"spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.97776, "y":40.76044}}
	1301 Ave Of The Americas	{"spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.97945, "y":40.76125}}
	1345 Avenue Of The Americas	{"spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.97806, "y":40.76309}}
	745 7Th Ave	{"spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.98340, "y":40.76077}}
10020	1221 Avenue Of The Americas	{"spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.98129, "y":40.75874}}
	1271 Avenue Of The Americas	{"spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.98018, "y":40.76025}}

GIS_GEOMETRY - JSON ジオメトリオブジェクトの作成

GIS_GEOMETRY 関数は、指定されたジオメトリタイプ、WKID、ジオメトリ (GEOMETRY) から、JSON ジオメトリオブジェクトを作成します。

構文　JSON ジオメトリオブジェクトの作成

```
GIS_GEOMETRY(geotype, wkid, geometry)
```

説明

`geotype`

文字

ジオメトリタイプです。たとえば、'esriGeometryPolygon', 'esriGeometryPolyline'、'esriGeometryMultipoint'、'EsriGeometryPoint'、'EsriGeometryExtent' があります。

`wkid`

文字

有効な空間参照 ID です。WKID (Well Known ID の略語) は、投影座標系または地理座標系を識別する ID です。

`geometry`

TX

JSON のジオメトリ (GEOMETRY) です。

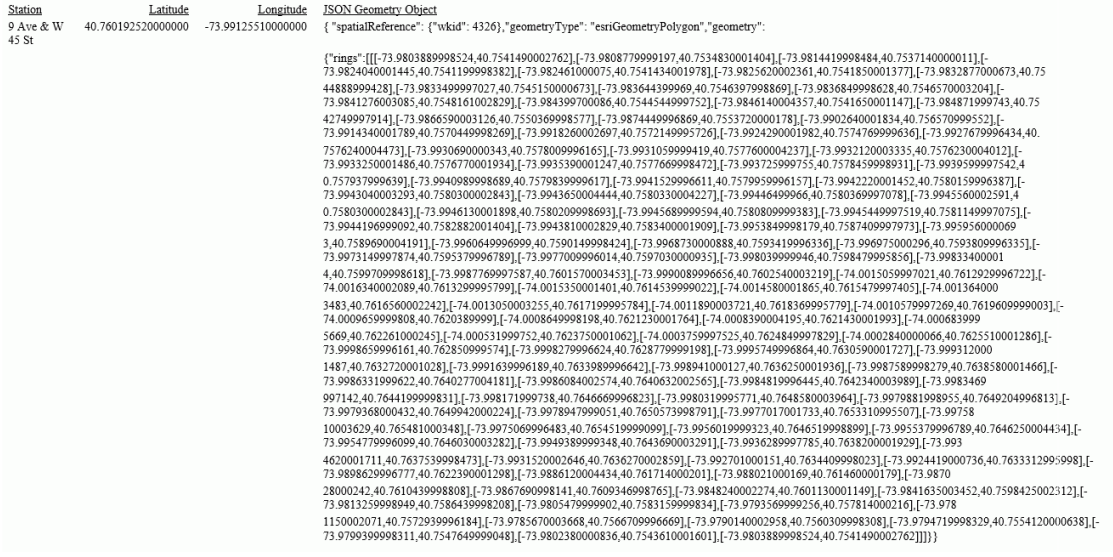
出力は、TX として返されます。

例 JSON ジオメトリオブジェクトの作成

次のリクエストは、マンハッタンで郵便番号が 10036 の地域を包囲する多角形を作成します。入力ジオメトリオブジェクトは、esri-citibike マスターファイルでクロスリファレンスされているテキストファイル (.ftm) に格納されています。ジオメトリオブジェクトが格納されるフィールドは GEOMETRY です。

```
DEFINE FILE esri/esri-citibike
WKID/A10  = '4326';
 MASTER_GEOMETRY/TX256 (GEOGRAPHIC_ROLE=GEOMETRY_AREA) =
    GIS_GEOMETRY( 'esriGeometryPolygon', WKID , GEOMETRY );
END
TABLE FILE esri/esri-citibike
 PRINT
    START_STATION_NAME AS Station
    START_STATION_LATITUDE AS Latitude
    START_STATION_LONGITUDE AS Longitude
    MASTER_GEOMETRY AS 'JSON Geometry Object'
 WHERE START_STATION_ID EQ 479
ON TABLE SET PAGE NOLEAD
 ON TABLE SET STYLE *
type=report, grid=off, size=10,$
 ENDSTYLE
END
```

下図は、出力結果を示しています。



Station	Latitude	Longitude	JSON Geometry Object
9 Ave & W 45 St	40.760192520000000	-73.99125510000000	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPolygon","geometry":

{"rings":[[[-73.9803889998524,40.75414900027 62],[-73.9808779999197,40.7534830001404],[-73.9814419998484,40.75371400000111],[-73.9824040001445,40.7541199998382],[-73.98246100075,40.75414340019 78],[-73.9825620002361,40.7541850001377],[-73.9832877000673,40.7544888999428],[-73.9833499997027,40.7545150000673],[-73.9836443999669,40.7546397998869],[-73.9836849998628,40.7546570003204],[-73.9841276003085,40.7548161002829],[-73.98439970 0086,40.7544544999752],[-73.9846140004357,40.7541650001147],[-73.9848711999743,40.7542749979914],[-73.9866590003126,40.7550369998577],[-73.9874449996869,40.7553720000178],[-73.9902640001834,40.7565709995552],[-73.9914340001789,40.7570449998269],[-73.9918260002697,40.7572149995726],[-73.9924290001982,40.7574769999636],[-73.9927679996434,40.7576240004473],[-73.9930690000343,40.7578009996165],[-73.9931059999419,40.7577600004237],[-73.9932120003335,40.7576230004012],[-73.9933250001486,40.7576770001934],[-73.9935390001247,40.7577669998472],[-73.9937259997755,40.7578459998931],[-73.9939599997542,40.7579379996 39],[-73.9940989998689,40.7579839999617],[-73.9941529996611,40.7579959996157],[-73.9942220001452,40.7580159996387],[-73.9943040003293,40.7580300002843],[-73.9943650004444,40.7580330004227],[-73.99446499966,40.7580369997078],[-73.9945560002591,40.7580300002843],[-73.9946130001898,40.7580209998693],[-73.9945689999594,40.7580809999383],[-73.9945449997519,40.7581149997075],[-73.9944196999092,40.7582882001404],[-73.9943810002829,40.7583400001909],[-73.9953849998179,40.7587409997973],[-73.995956000069 3,40.7589690004191],[-73.9960649996999,40.7590149998424],[-73.9968730000888,40.7593419996336],[-73.9969750002 96,40.7593809996335],[-73.9973149997874,40.7595379996789],[-73.9977009996014,40.7597030000935],[-73.9980399999946,40.7598479995856],[-73.99833400001 4,40.7599709998618],[-73.9987769997587,40.7601570003453],[-73.9990089996656,40.7602540003219],[-74.0015059997021,40.7612929996722],[-74.0016340002089,40.7613299995799],[-74.0015350001401,40.7614539999022],[-74.0014580001865,40.7615479997405],[-74.001364000 3483,40.7616560002242],[-74.0013050003255,40.7617199995784],[-74.0011890003721,40.7618369995779],[-74.0010579997269,40.7619609999003],[-74.0009659999808,40.7620389999],[-74.0008649998198,40.7621230001764],[-74.0008390004195,40.7621430001993],[-74.000683999 5669,40.762261000245],[-74.0005319997 52,40.7623750001062],[-74.0003759997525,40.7624849997829],[-74.0002840000066,40.7625510001286],[-73.9998659996161,40.7628509995 74],[-73.9998279996624,40.7628779999198],[-73.9995749996864,40.7630590001727],[-73.999312000 1487,40.7632720001028],[-73.9991639996189,40.7633989996642],[-73.998941000127,40.7636250001936],[-73.9987589998279,40.7638580001466],[-73.9986331999622,40.7640277004181],[-73.9986084002574,40.7640632002565],[-73.9984819996445,40.7642340003989],[-73.9983469 997142,40.7644199999831],[-73.9981719997 38,40.7646669996823],[-73.9980319995771,40.7648580003964],[-73.9979881998955,40.7649204996813],[-73.9979368000432,40.7649942000224],[-73.9978947999051,40.7650573998791],[-73.9977017001733,40.7653310995507],[-73.997 5810003629,40.7654810003481],[-73.9975069996483,40.7654519999099],[-73.9956019999323,40.7646519998899],[-73.9955379996789,40.7646250004434],[-73.9954779996099,40.7646030003282],[-73.9949389999348,40.7643690003291],[-73.9936289997785,40.7638200001929],[-73.993 4620001711,40.7637539998473],[-73.9931520002646,40.7636270002859],[-73.9927010001 51,40.7634409998023],[-73.9924419000736,40.7633312995998],[-73.9898629996777,40.7622390001298],[-73.9886120004434,40.7617140002 01],[-73.9880210001 69,40.7614600001 79],[-73.9870 28000242,40.7610439998808],[-73.9867690998141,40.7609346998765],[-73.9848240002274,40.7601130001149],[-73.9841635003452,40.7598425002312],[-73.9813259998949,40.7586439998208],[-73.9805479999902,40.7583159999834],[-73.9793569999256,40.7578140002 16],[-73.978 1150002071,40.7572939996184],[-73.9785670003668,40.7566709996669],[-73.9790140002958,40.7560309998308],[-73.9794719998329,40.7554120000638],[-73.9799399998311,40.7547649999048],[-73.9802380000836,40.7543610001601],[-73.9803889998524,40.7541490002762]]]}}

例　ジオメトリオブジェクトの描画

次のリクエストは、GIS_GEOMETRY 関数を使用してジオメトリオブジェクトを作成し、そのオブジェクトを ESRI マップ上に描画します。

```
DEFINE FILE esri-citibike
WKID/A10  = '4326';
 MASTER_GEOMETRY/TX256 (GEOGRAPHIC_ROLE=GEOMETRY_AREA) =
    GIS_GEOMETRY( 'esriGeometryPolygon', WKID , GEOMETRY );
END

 GRAPH FILE ESRI-CITIBIKE
 PRINT
   START_STATION_NAME
   END_STATION_NAME
   ON TABLE PCHOLD FORMAT JSCHART
 ON TABLE SET LOOKGRAPH CHOROPLETH
 ON TABLE SET EMBEDHEADING ON
 ON TABLE SET AUTOFIT ON
 ON TABLE SET STYLE *
  TYPE=REPORT, TITLETEXT='Map', PAGESIZE=E, CHART-LOOK=com.esri.map, $
  TYPE=DATA, COLUMN=N1, /*START_STATION_NAME*/
   BUCKET=tooltip, $
  TYPE=DATA, COLUMN=N2, /*END_STATION_NAME*/
```

```
  *GRAPH_JS_FINAL

"legend": {"visible": true},
"extensions" : { "com.esri.map" :
  { "scalebar" :
{
    "scalebarUnit": "dual",
    "attachTo" : "bottom-left"
  },
  "baseMapInfo": {
       "drawBasemapControl" : false,
       "showArcGISBasemaps" : false,
            "customBaseMaps" : [
            {"ibiBaseLayer" : "dark-gray"}
        ]
   },
  "overlayLayers":
  [{
  "ibiDataLayer": {"map-geometry" :  {"map_by_field" : "MASTER_GEOMETRY"}},
"title" : "Chart"}]
  },
"introAnimation": "{¥"enabled¥":false}"
}

  *END
 ENDSTYLE
 HEADING
  "Chart Geometry Object"
END
```

下図は、出力結果を示しています。

GIS_IN_POLYGON - 複雑な多角形内の地点の有無を特定

GIS_IN_POLYGON 関数は、指定された地点と多角形定義から、地点が多角形内に存在する場合は 1 (TRUE)、存在しない場合は 0 (ゼロ) を返します。値は、整数フォーマットで返されます。

構文　　複雑な多角形内の地点の有無を特定

```
GIS_IN_POLYGON(point, polygon_definition)
```

説明

`point`

文字またはテキスト

地点です。

`polygon_definition`

テキスト

面 (多角形) 定義です。

例　多角形内の地点の有無を特定

次の例では、郵便番号が 10036 に該当する地域内にステーションが存在するかどうかを特定します。GIS_IN_POLYGON 関数は、多角形定義の内側に地点が存在する場合は 1、外側に存在する場合は 0 (ゼロ) を返します。渡される多角形定義は、上記の GIS_GEOMETRY 関数の例で使用されたものと同一で、ニューヨーク市マンハッタンの郵便番号 10036 に該当する地域を多角形で定義します。出力結果では、1 は Yes、0 (ゼロ) は No に変換されて表示されます。

```
DEFINE FILE esri/esri-citibike
WKID/A10  = '4326';
MASTER_GEOMETRY/TX256 (GEOGRAPHIC_ROLE=GEOMETRY_AREA) =
  GIS_GEOMETRY( 'esriGeometryPolygon', WKID , GEOMETRY );
START_STATION_POINT/A200=GIS_POINT(WKID, START_STATION_LONGITUDE,
START_STATION_LATITUDE);
STATION_IN_POLYGON/I4=GIS_IN_POLYGON(START_STATION_POINT, MASTER_GEOMETRY);
IN_POLYGON/A5 = IF STATION_IN_POLYGON EQ 1 THEN 'Yes' ELSE 'No';
END
TABLE FILE esri/esri-citibike
 PRINT
     START_STATION_NAME AS Station
    IN_POLYGON AS 'Station in zip, code 10036?'
BY START_STATION_ID AS 'Station ID'
ON TABLE SET PAGE NOLEAD
 ON TABLE SET STYLE *
type=report, grid=off, size=10,$
type=data, column=in_polygon, style=bold, color=red, when = in_polygon eq
'Yes',$
 ENDSTYLE
END
```

下図は、出力結果を示しています。

Station ID	Station	Station in zip code 10036?
147	Greenwich St & Warren St	No
160	E 37 St & Lexington Ave	No
229	Great Jones St	No
247	Perry St & Bleecker St	No
268	Howard St & Centre St	No
281	Grand Army Plaza & Central Park S	No
285	Broadway & E 14 St	No
319	Fulton St & Broadway	No
346	Bank St & Hudson St	No
379	W 31 St & 7 Ave	No
407	Henry St & Poplar St	No
409	DeKalb Ave & Skillman St	No
479	9 Ave & W 45 St	Yes
492	W 33 St & 7 Ave	No
512	W 29 St & 9 Ave	No
521	8 Ave & W 31 St	No
	8 Ave & W 31 St	No
532	S 5 Pl & S 4 St	No
536	1 Ave & E 30 St	No
537	Lexington Ave & E 24 St	No

GIS_LINE - JSON 線の作成

GIS_LINE 関数は、指定された 2 地点または 2 本線から、JSON 形式の線を作成します。出力はテキストフォーマットで返されます。

構文　　JSON 線の作成

```
GIS_LINE(geometry1, geometry2)
```

説明

geometry1

文字またはテキスト

新しい線の開始位置を定義する 1 つ目の地点または線です。

geometry2

文字またはテキスト

新しい線の連結位置を定義する 2 つ目の地点または線です。

例 JSON 線の作成

次のリクエストは、出発ステーションと到着ステーションを表示し、これらの地点を接続する JSON 形式の線を作成します。

```
DEFINE FILE ESRI/ESRI-CITIBIKE
STARTPOINT/A200 = GIS_POINT('4326', START_STATION_LONGITUDE,
START_STATION_LATITUDE);
ENDPOINT/A200 = GIS_POINT('4326', END_STATION_LONGITUDE,
END_STATION_LATITUDE);
CONNECTION_LINE/TX80 (GEOGRAPHIC_ROLE=GEOMETRY_LINE) =
   GIS_LINE(STARTPOINT, ENDPOINT);
END
TABLE FILE ESRI/ESRI-CITIBIKE
PRINT END_STATION_NAME AS End CONNECTION_LINE AS 'Connecting Line'
BY START_STATION_NAME AS Start
WHERE START_STATION_NAME LE 'D'
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF,$
ENDSTYLE
END
```

下図は、出力結果を示しています。

Start	End	Connecting Line
1 Ave & E 30 St	Broadway & W 32 St	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPolyline","geometry": {"paths": [[[-73.97536082000000,40.741443870000000],[-73.98808416000000,40.748548620000000]]]}}
8 Ave & W 31 St	Broadway & E 14 St	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPolyline","geometry": {"paths": [[[-73.99444208000000,40.750967350000000],[-73.99074142000000,40.734545670000000]]]}}
	E 20 St & 2 Ave	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPolyline","geometry": {"paths": [[[-73.99444208000000,40.750967350000000],[-73.98205027000000,40.735876780000000]]]}}
9 Ave & W 45 St	E 45 St & 3 Ave	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPolyline","geometry": {"paths": [[[-73.99125510000000,40.760192520000000],[-73.97282625000000,40.752554340000000]]]}}
Bank St & Hudson St	Mercer St & Bleecker St	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPolyline","geometry": {"paths": [[[-74.00618026000000,40.736528890000000],[-73.99695094000000,40.726794540000000]]]}}
Broadway & E 14 St	Cleveland Pl & Spring St	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPolyline","geometry": {"paths": [[[-73.99074142000000,40.734545670000000],[-73.99724901000000,40.722103790000000]]]}}

例　線の描画

次のリクエストは、複数の線を生成し、これらの線を ESRI マップ上に描画します。

```
DEFINE FILE ESRI-CITIBIKE
CONNECTION_LINE/TX80 (GEOGRAPHIC_ROLE=GEOMETRY_LINE)
=GIS_LINE(START_STATION_POINT, END_STATION_POINT);
DISTANCE/P33.11 TITLE 'Distance'=GIS_DISTANCE(START_STATION_POINT,
END_STATION_POINT);
END
```

```
GRAPH FILE ESRI-CITIBIKE
 PRINT
   START_STATION_NAME
   END_STATION_NAME
   DISTANCE
 ON TABLE PCHOLD FORMAT JSCHART
 ON TABLE SET LOOKGRAPH BUBBLEMAP
 ON TABLE SET EMBEDHEADING ON
 ON TABLE SET AUTOFIT ON
 ON TABLE SET STYLE *
  TYPE=REPORT, TITLETEXT='Map', PAGESIZE=E, CHART-LOOK=com.esri.map, $
  TYPE=DATA, COLUMN=N1, /*START_STATION_NAME*/
   BUCKET=tooltip, $
  TYPE=DATA, COLUMN=N2, /*END_STATION_NAME*/
   BUCKET=tooltip, $
  TYPE=DATA, COLUMN=N3, /*DISTANCE*/
   BUCKET=tooltip, $

  *GRAPH_JS_FINAL
"legend": {"visible": true},
"extensions" : { "com.esri.map" :
  { "scalebar" :
{
    "scalebarUnit": "dual",
    "attachTo" : "bottom-left"
  },
  "baseMapInfo": {
       "drawBasemapControl" : false,
       "showArcGISBasemaps" : false,
            "customBaseMaps" : [
            {"ibiBaseLayer" : "dark-gray"}
        ]
   },
  "overlayLayers":
  [{
  "ibiDataLayer": {"map-geometry" :  {"map_by_field" : "CONNECTION_LINE"}},
"title" : "Chart"}]
  },
"introAnimation": "{¥"enabled¥":false}"
}

  *END
 ENDSTYLE
 HEADING
  "Chart Geometry Lines"
END
```

下図は、出力結果を示しています。

GIS_POINT - 地点の作成

GIS_POINT 関数は、指定された WKID 空間参照コード、経度、緯度から、ジオメトリオブジェクトを定義する JSON 形式の点を作成します。この関数は、JSON ジオメトリオブジェクトを作成可能な SQL エンジン用に最適化されます。

地点が返されるフィールドは、JSON 形式で記述された地点の格納に必要な長さにする必要があります (例、A200)。

構文　　地点の作成

```
GIS_POINT(wkid, longitude, latitude)
```

説明

`wkid`

固定長の文字

空間参照コード (WKID) です。WKID (Well Known ID の略語) は、投影座標系または地理座標系を識別する ID です。

`longitude`

D20.8

地点の経度です。

`latitude`

D20.8

地点の緯度です。

例　地点の作成

次のリクエストは、空間参照コード 4326 (10 進数 (度)) および州都の経度と緯度を使用して、地点を作成します。

```
DEFINE FILE WF_RETAIL_LITE
GPOINT/A200 = GIS_POINT('4326', STATE_PROV_CAPITAL_LONGITUDE,
STATE_PROV_CAPITAL_LATITUDE);
END
TABLE FILE WF_RETAIL_LITE
SUM FST.STATE_PROV_CAPITAL_LONGITUDE AS Longitude
FST.STATE_PROV_CAPITAL_LATITUDE AS Latitude
FST.GPOINT AS Point
BY STATE_PROV_CAPITAL_NAME AS Capital
WHERE COUNTRY_NAME EQ 'United States'
WHERE STATE_PROV_CAPITAL_NAME LT 'C'
ON TABLE SET PAGE NOPAGE
END
```

下図は、出力結果を示しています。

Capital	Longitude	Latitude	Point
Albany	-73.76000000	42.66000000	{"spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-73.76000000, "y":42.66000000}}
Annapolis	-76.49000000	38.95000000	{"spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-76.49000000, "y":38.95000000}}
Atlanta	-84.27000000	33.94000000	{"spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-84.27000000, "y":33.94000000}}
Augusta	-69.77000000	44.32000000	{"spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-69.77000000, "y":44.32000000}}
Austin	-97.75000000	30.40000000	{"spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-97.75000000, "y":30.40000000}}
Baton Rouge	-91.17000000	30.38000000	{"spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-91.17000000, "y":30.38000000}}
Bismarck	-100.77000000	46.82000000	{"spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-100.77000000, "y":46.82000000}}
Boise	-116.16000000	43.60000000	{"spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-116.16000000, "y":43.60000000}}
Boston	-71.10000000	42.35000000	{"spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPoint","geometry": {"x":-71.10000000, "y":42.35000000}}

例　地点の描画

次のリクエストは、GIS_POINT 関数を使用して地点を生成し、それらの地点を ESRI マップ上に描画します。

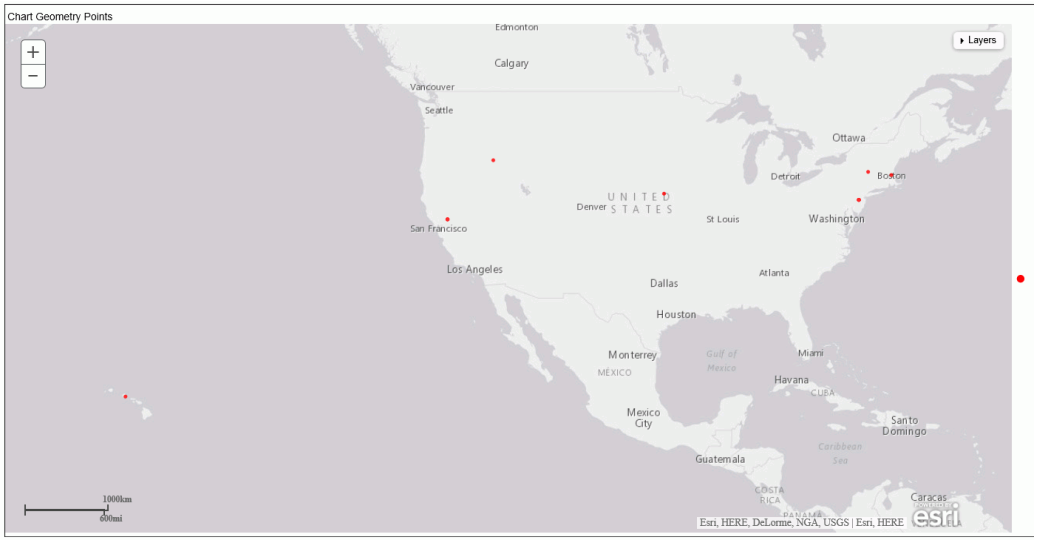
```
DEFINE FILE WF_RETAIL
GPOINT/A200 = GIS_POINT('4326', STATE_PROV_CAPITAL_LONGITUDE,
STATE_PROV_CAPITAL_LATITUDE);
END


 GRAPH FILE WF_RETAIL
 PRINT
 STATE_PROV_NAME
 WHERE STATE_PROV_CAPITAL_LONGITUDE NE MISSING
 ON TABLE PCHOLD FORMAT JSCHART
 ON TABLE SET LOOKGRAPH BUBBLEMAP
 ON TABLE SET EMBEDHEADING ON
 ON TABLE SET AUTOFIT ON
 ON TABLE SET STYLE *
  TYPE=REPORT, TITLETEXT='Map', PAGESIZE=E, CHART-LOOK=com.esri.map, $
  TYPE=DATA, COLUMN=N1,
   BUCKET=tooltip, $


  *GRAPH_JS_FINAL
"bubbleMarker": {"maxSize": "10%"},
"legend": {"visible": true},
"extensions" : { "com.esri.map" :
  { "scalebar" :
{
    "scalebarUnit": "dual",
    "attachTo" : "bottom-left"
  },
  "baseMapInfo": {
       "drawBasemapControl" : false,
       "showArcGISBasemaps" : false,
            "customBaseMaps" : [
            {"ibiBaseLayer" : "gray"}
        ]
   },
  "overlayLayers":
  [{
  "ibiDataLayer": {"map-geometry" :  {"map_by_field" : "GPOINT"}},
"title" : "Report"}]
  },
"introAnimation": "{¥"enabled¥":false}"
}

  *END
 ENDSTYLE
 HEADING
  "Chart Geometry Points"
END
```

下図は、出力結果を示しています。



GIS_REVERSE_COORDINATE - 地理コンポーネントを取得

緯度と経度の値および地理コンポーネント名から、GIS_REVERSE_COORDINATE は、それらの座標に関連付けられた特定の地理コンポーネントを取得します。

注意：この関数では、GIS サービスが使用され、指定ログイン情報を使用する ESRI ArcGIS アダプタ接続が必要です。

構文　地理コンポーネントを取得

```
GIS_REVERSE_COORDINATE(longitude, latitude, component)
```

説明

`longitude`

数値

取得するコンポーネントの経度です。

`latitude`

数値

取得するコンポーネントの緯度です。

component

キーワード

次のいずれかのコンポーネントです。

- MATCH_ADDRESS - 一致する住所を取得します。
- METROAREA - 都市圏の名前を取得します。
- REGION - 地域名を取得します。
- SUBREGION - 詳細地域名を取得します。
- CITY - 都市名を取得します。
- POSTAL - 郵便番号を取得します。

値は文字列で取得され、テキストまたは文字 (固定長または可変長) フォーマットのフィールドに割り当てられます。

例　座標に関連付けられた地理コンポーネントの取得

次のリクエストは、都市の緯度と経度を使用して一致する住所、郵便番号、地域名および詳細地域名を取得します。

```
TABLE FILE WF_RETAIL_GEOGRAPHY
SUM FST.CITY_LONGITUDE AS Longitude FST.CITY_LATITUDE AS Latitude
COMPUTE
MatchingAddress/A250 = GIS_REVERSE_COORDINATE(CITY_LONGITUDE,
                            CITY_LATITUDE, MATCH_ADDRESS);
PostalCode/A250 = GIS_REVERSE_COORDINATE(CITY_LONGITUDE,
                  CITY_LATITUDE, POSTAL);
Region/A250 = GIS_REVERSE_COORDINATE(CITY_LONGITUDE, CITY_LATITUDE,
              REGION);
Subregion/A250 = GIS_REVERSE_COORDINATE(CITY_LONGITUDE, CITY_LATITUDE,
                 SUBREGION);
BY CITY_NAME AS City
WHERE COUNTRY_NAME EQ 'United States'
WHERE TOTAL PostalCode NE ' '
WHERE RECORDLIMIT EQ 20
ON TABLE SET PAGE NOLEAD
END
```

下図は、出力結果を示しています。

City	Longitude	Latitude	MatchingAddress	PostalCode	Region	Subregion
Annapolis	-76.54540000	38.98790000	Annapolis Mall, Annapolis, Maryland, 21401	21401	Maryland	Anne Arundel County
Baton Rouge	-91.09780000	30.44990000	233 E Parkland Dr, Baton Rouge, Louisiana, 70806	70806	Louisiana	East Baton Rouge Parish
Cincinnati	-84.45690000	39.16200000	2070-2098 Elm Ave, Cincinnati, Ohio, 45212	45212	Ohio	Hamilton County
Daytona Beach	-81.04740000	29.19310000	511 S Clyde Morris Blvd, Daytona Beach, Florida, 32114	32114	Florida	Volusia County
Detroit	-83.05980000	42.34630000	133 Davenport St, Detroit, Michigan, 48201	48201	Michigan	Wayne County
Harrington Park	-73.98330000	40.99070000	247 Lynn St, Harrington Park, New Jersey, 07640	07640	New Jersey	Bergen County
Johnston	-93.72040000	41.70310000	Camp Dodge	50131	Iowa	Polk County
Lake Mary	-81.33970000	28.75780000	127-129 E Plantation Blvd, Lake Mary, Florida, 32746	32746	Florida	Seminole County
Laredo	-99.50350000	27.51330000	1501-1599 Santa Ursula Ave, Laredo, Texas, 78040	78040	Texas	Webb County
Latham	-73.78040000	42.75260000	1 Lear Jet Ln, Latham, New York, 12110	12110	New York	Albany County
Louisville	-85.69180000	38.20850000	2714 Lamont Rd, Louisville, Kentucky, 40205	40205	Kentucky	Jefferson County
Medley	-80.38390000	25.85880000	33178, Miami, Florida	33178	Florida	Miami-Dade County
North Kansas City	-94.56220000	39.13000000	1201-1499 Quebec St, Kansas City, Missouri, 64116	64116	Missouri	Clay County
Rochester	-77.61560000	43.15480000	1-35 Plymouth Ave S, Rochester, New York, 14614	14614	New York	Monroe County
Waco	-97.13820000	31.55100000	1200 Austin Ave, Waco, Texas, 76701	76701	Texas	McLennan County

GIS_SERVICE_AREA - 特定の地点を囲む領域の計算

GIS_SERVICE_AREA 関数は、指定された地点から特定の時間で移動可能な領域または特定の距離を境界とする領域を計算します。出力はテキストフォーマットで返されます。

注意：この関数では、GIS サービスが使用され、指定ログイン情報を使用する ESRI ArcGIS アダプタ接続が必要です。

構文　特定の地点を囲む領域の計算

```
GIS_SERVICE_AREA(geo_point, distance, travel_mode)
```

説明

`geo_point`

文字

開始地点です。

`distance`

文字

移動の最大範囲を時間単位または距離単位で指定します。

`travel_mode`

文字

Catalog ディレクトリ (*drive*:¥ibi¥WebFOCUS¥srv¥home¥catalog) 内の gis_serv_area.mas ファイルで定義されている有効な移動モードです。有効な移動モードには次のものがあります。

- **'Miles'** これがデフォルト値です。
- **'TravelTime'**
- **'TruckTravelTime'**
- **'WalkTime'**
- **'Kilometers'**

例 特定の地点を囲む領域の計算

次のリクエストは、ステーションから徒歩 5 分で移動可能な領域を計算します。

```
DEFINE FILE esri/esri-citibike
WKID/A10='4326';
START_STATION_POINT/A200=GIS_POINT(WKID, START_STATION_LONGITUDE,
START_STATION_LATITUDE);
DISTANCE/A10='5';
TRAVEL_MODE/A10='WalkTime';
STATION_SERVICE_AREA/TX80 (GEOGRAPHIC_ROLE=GEOMETRY_AREA)=
 GIS_SERVICE_AREA(START_STATION_POINT, DISTANCE, TRAVEL_MODE);
END
TABLE FILE esri/esri-citibike
 PRINT
    START_STATION_ID AS 'Station ID'
    START_STATION_NAME AS 'Station Name'
    STATION_SERVICE_AREA AS '5-Minute Walk Service Area Around Station'
 WHERE START_STATION_ID EQ 479 OR 512;
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF, SIZE=12,$
ENDSTYLE
END
```

下図は、出力結果を示しています。

Station ID	Station Name	5-Minute Walk Service Area Around Station
512	W 29 St & 9 Ave	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPolygon","geometry": {"rings":[[[-73.995542525999952,40.749246597000081],[-73.995094298999959,40.748346329000071],[-73.995542525999952,40.74767494200006],[-73.996665954999969,40.747449875000029],[-73.99778938299994,40.748571396000045],[-73.998462676999964,40.748571396000045],[-73.998462676999964,40.747449875000029],[-73.999135970999987,40.746999741000025],[-73.999586104999935,40.747224808000055],[-74.000932692999982,40.746103287000039],[-74.00160789499995,40.746549606000031],[-74.002056121999942,40.748121262000041],[-74.000484466999978,40.749471664000055],[-74.000259389999958,40.749471664000055],[-74.000034331999984,40.749917984000035],[-74.002729415999966,40.750818253000034],[-74.00317954999997,40.751489639000056],[-74.002729415999966,40.752614975000029],[-74.001831054999968,40.752614975000029],[-74.000932692999982,40.75328636200004],[-74.000034331999984,40.752840042000059],[-73.999811171999966,40.75171470600003],[-73.99778938299994,40.751043320000065],[-73.997564315999966,40.75036811800004],[-73.995542525999952,40.749246597000081]]]}}
479	9 Ave & W 45 St	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPolygon","geometry": {"rings":[[[-73.990602492999983,40.760248184000034],[-73.988132476999965,40.759351730000049],[-73.98768234299996,40.758451462000039],[-73.988580703999958,40.757555008000054],[-73.98992919899996,40.757780075000028],[-73.990827559999957,40.756658554000069],[-73.992399215999967,40.75732994100008],[-73.992849349999972,40.756433487000038],[-73.993745803999957,40.756208420000064],[-73.994644164999954,40.757104874000049],[-73.994421004999936,40.758230209000033],[-73.995094298999959,40.760026932000073],[-73.994195937999962,40.760923386000059],[-73.992624282999941,40.760248184000034],[-73.991950988999974,40.760923386000059],[-73.991725921999944,40.760923386000059],[-73.99150085399998,40.760923386000059],[-73.991500853999998,40.761148453000033],[-73.990602492999983,40.760698318000038],[-73.990602492999983,40.760248184000034]]]}}

例　特定の地点を囲む領域の描画

次のリクエストは、出発ステーション地点から徒歩 5 分で移動可能な領域を生成し、その領域を ESRI マップ上に描画します。

```
DEFINE FILE esri-citibike
WKID/A10='4326';
START_STATION_POINT/A200=GIS_POINT(WKID, START_STATION_LONGITUDE,
START_STATION_LATITUDE);
DISTANCE/A10='5';
TRAVEL_MODE/A10='WalkTime';
STATION_SERVICE_AREA/TX80 (GEOGRAPHIC_ROLE=GEOMETRY_AREA)=
 GIS_SERVICE_AREA(START_STATION_POINT, DISTANCE, TRAVEL_MODE);
END
```

```
GRAPH FILE ESRI-CITIBIKE
 PRINT
   START_STATION_NAME
   END_STATION_NAME
   DISTANCE
 ON TABLE PCHOLD FORMAT JSCHART
 ON TABLE SET LOOKGRAPH CHOROPLETH
 ON TABLE SET EMBEDHEADING ON
 ON TABLE SET AUTOFIT ON
 ON TABLE SET STYLE *
  TYPE=REPORT, TITLETEXT='Map', PAGESIZE=E, CHART-LOOK=com.esri.map, $
  TYPE=DATA, COLUMN=N1, /*START_STATION_NAME*/
   BUCKET=tooltip, $
  TYPE=DATA, COLUMN=N2, /*END_STATION_NAME*/
   BUCKET=tooltip, $
  TYPE=DATA, COLUMN=N3, /*DISTANCE*/
   BUCKET=tooltip, $

  *GRAPH_JS_FINAL
"legend": {"visible": true},
"extensions" : { "com.esri.map" :
  { "scalebar" :
{
    "scalebarUnit": "dual",
    "attachTo" : "bottom-left"
  },
  "baseMapInfo": {
       "drawBasemapControl" : false,
       "showArcGISBasemaps" : false,
            "customBaseMaps" : [
            {"ibiBaseLayer" : "dark-gray"}
        ]
   },
  "overlayLayers":
  [{
  "ibiDataLayer": {"map-geometry" :  {"map_by_field" :
"STATION_SERVICE_AREA"}}, "title" : "Chart"}]
  },
"introAnimation": "{¥"enabled¥":false}"
}

  *END
 ENDSTYLE
 HEADING
  "Chart Geometry Service Area"
END
```

下図は、出力結果を示しています。

GIS_SERV_AREA_XY - 特定の座標点を囲む領域の計算

GIS_SERV_AREA_XY 関数は、GIS サービスを使用して、指定された座標点から特定の時間で移動可能な領域または特定の距離を境界とする領域を計算します。出力はテキストフォーマットで返されます。

注意：この関数では、GIS サービスが使用され、指定ログイン情報を使用する ESRI ArcGIS アダプタ接続が必要です。

構文　特定の座標点を囲む領域の計算

```
GIS_SERV_AREA_XY(longitude, latitude, distance, travel_mode[, wkid])
```

説明

`longitude`

文字

開始地点の経度です。

`latitude`

文字

開始地点の緯度です。

`distance`

整数

移動の最大範囲を時間単位または距離単位で指定します。

`travel_mode`

文字

Catalog ディレクトリ (*drive*:¥ibi¥WebFOCUS¥srv¥home¥catalog) 内の gis_serv_area.mas ファイルで定義されている有効な移動モードです。有効な移動モードには次のものがあります。

- **'Miles'** これがデフォルト値です。
- **'TravelTime'**
- **'TruckTravelTime'**
- **'WalkTime'**
- **'Kilometers'**

`wkid`

文字

座標の空間参照 ID です。WKID (Well Known ID の略語) は、投影座標系または地理座標系を識別する ID です。デフォルト値は '4326' です。この値は、10 進数 (度) を表します。

例　特定の座標点を囲む領域の計算

次のリクエストは、ステーションの地点を示す経度と緯度を使用して、そのステーションから徒歩 5 分で移動可能な領域を計算します。

```
DEFINE FILE esri/esri-citibike
DISTANCE/I4=5;
WKID/A10='4326';
TRAVEL_MODE/A10='WalkTime';
STATION_SERVICE_AREA/TX80 (GEOGRAPHIC_ROLE=GEOMETRY_AREA)=
   GIS_SERV_AREA_XY(START_STATION_LONGITUDE, START_STATION_LATITUDE,
DISTANCE, TRAVEL_MODE, WKID);
END
TABLE FILE esri/esri-citibike
 PRINT
    START_STATION_ID AS 'Station ID'
    START_STATION_NAME AS 'Station Name'
    STATION_SERVICE_AREA
       AS '5-Minute Walk Service Area Around Station Coordinate'
 WHERE START_STATION_ID EQ 479 OR 512;
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF, SIZE=12,$
ENDSTYLE
END
```

下図は、出力結果を示しています。

Station ID	Station Name	5-Minute Walk Area Around Station Coordinate
512	W 29 St & 9 Ave	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPolygon","geometry": {"rings":[[[-73.996217727999976,40.748571396000045],[-73.996891021999943,40.748121262000041],[-73.998462676999964,40.748571396000045],[-73.998237609999933,40.747900009000034],[-73.998687743999938,40.747224808000055],[-74.000932692999982,40.746999741000025],[-74.001382827999976,40.748121262000041],[-74.000034331999984,40.749917984000035],[-74.002281188999973,40.750818253000034],[-74.002504348999935,40.75171470600003],[-74.002056121999942,40.752389908000055],[-74.001831054999968,40.752389908000055],[-74.001382827999976,40.752614975000029],[-74.001382827999976,40.752840042000059],[-73.996665954999969,40.750143051000066],[-73.995992660999946,40.749246597000081],[-73.996217727999976,40.748571396000045]]]}}
479	9 Ave & W 45 St	{ "spatialReference": {"wkid": 4326},"geometryType": "esriGeometryPolygon","geometry": {"rings":[[[-73.988357543999939,40.75867652900007],[-73.989255904999936,40.757780075000028],[-73.99127578699995,40.758451462000039],[-73.991725921999944,40.757555008000054],[-73.993297576999964,40.756658554000069],[-73.994195937999962,40.757555008000054],[-73.993745803999957,40.758451462000039],[-73.994195937999962,40.759576797000079],[-73.993745803999957,40.760248184000034],[-73.992399215999967,40.760248184000034],[-73.99150085399998,40.760923386000059],[-73.99150085399998,40.761148453000033],[-73.990827559999957,40.760923386000059],[-73.990602492999983,40.760248184000034],[-73.988805770999988,40.759801865000043],[-73.988357543999939,40.75867652900007]]]}}

例　特定の座標点を囲む領域の描画

次のリクエストは、出発ステーションの座標点から徒歩 5 分で移動可能な領域を生成し、その領域を ESRI マップ上に描画します。

```
DEFINE FILE esri-citibike
WKID/A10='4326';
DISTANCE/A10='5';
TRAVEL_MODE/A10='WalkTime';
STATION_SERVICE_AREA/TX80 (GEOGRAPHIC_ROLE=GEOMETRY_AREA)=
   GIS_SERV_AREA_XY(START_STATION_LONGITUDE, START_STATION_LATITUDE,
DISTANCE, TRAVEL_MODE, WKID);
END
```

```
GRAPH FILE ESRI-CITIBIKE
 PRINT
   START_STATION_NAME
   END_STATION_NAME
   DISTANCE
 ON TABLE PCHOLD FORMAT JSCHART
 ON TABLE SET LOOKGRAPH CHOROPLETH
 ON TABLE SET EMBEDHEADING ON
 ON TABLE SET AUTOFIT ON
 ON TABLE SET STYLE *
  TYPE=REPORT, TITLETEXT='Map', PAGESIZE=E, CHART-LOOK=com.esri.map, $
  TYPE=DATA, COLUMN=N1, /*START_STATION_NAME*/
   BUCKET=tooltip, $
  TYPE=DATA, COLUMN=N2, /*END_STATION_NAME*/
   BUCKET=tooltip, $
  TYPE=DATA, COLUMN=N3, /*DISTANCE*/
   BUCKET=tooltip, $

  *GRAPH_JS_FINAL
"legend": {"visible": true},
"extensions" : { "com.esri.map" :
  { "scalebar" :
{
    "scalebarUnit": "dual",
    "attachTo" : "bottom-left"
  },
  "baseMapInfo": {
       "drawBasemapControl" : false,
       "showArcGISBasemaps" : false,
            "customBaseMaps" : [
            {"ibiBaseLayer" : "dark-gray"}
        ]
   },
  "overlayLayers":
  [{
  "ibiDataLayer": {"map-geometry" :  {"map_by_field" :
"STATION_SERVICE_AREA"}}, "title" : "Chart"}]
  },
"introAnimation": "{¥"enabled¥":false}"
}

  *END
 ENDSTYLE
 HEADING
  "Chart Geometry Service Area"
END
```

下図は、出力結果を示しています。

22

SQL 文字列関数

文字列関数は、文字と数字を含むフィールドと文字列を操作します。

これらは、SQL 関数のリクエストや、DBMS がサポートしている場合は、ダイレクト SQL パススルーリクエストで使用できます。

トピックス

LOCATE - 文字列内のサブ文字列の位置を取得

LOCATE 関数は、指定されたサブ文字列、ソース文字列、開始位置 (デフォルト値は 1) から、開始位置から検索を開始して 1 つ目のサブ文字列の位置を返します。サブ文字列が検出されない場合、LOCATE 関数は 0 (ゼロ) を返します。検索では大文字と小文字が区別されません。

構文　文字列内のサブ文字列の位置を取得

```
LOCATE(substr, source [,start])
```

説明

`substr`

文字

検索文字列です。

`source`

文字

ソース文字列です。

`start`

数値

検索の開始位置です (オプション)。これを省略すると、デフォルト値の 1 になります。

例　文字列内のサブ文字列の開始位置を取得

次の SQL SELECT ステートメントは、FULLNAME 内の「a」という文字を検索します。開始位置は 3 および 1 です。

```
SQL
SELECT FULLNAME,
LOCATE('a', FULLNAME, 3) AS 'START AT 3',
LOCATE('a', FULLNAME) AS 'START AT 1'
FROM
WF_RETAIL_CUSTOMER T1
FETCH FIRST 5 ROWS ONLY;
TABLE
HEADING CENTER
"Search for the Character 'a'"
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
TYPE=HEADING, STYLE=BOLD, SIZE=16, $
ENDSTYLE
END
```

出力結果は次のとおりです。

Search for the Character 'a'

Full Name	START AT 3	START AT 1
Tyler Nolan	10	10
Joshua Bull	6	6
Zara Wood	4	2
Callum McKenzie	0	2
Bradley Charlton	3	3

23

その他の SQL 関数

この章で説明する SQL 関数は、さまざまな変換、テスト、操作を実行します。

これらは、SQL 関数のリクエストや、DBMS がサポートしている場合は、ダイレクト SQL パススルーリクエストで使用できます。

トピックス

CHR - 数値コードに対応する ASCII 文字の取得

CHR 関数は、引数として指定された数値コードに対応する ASCII 文字を返します。

構文　数値コードに対応する ASCII 文字の取得

```
CHR(number)
```

説明

`number`

数値

ASCII 文字に変換する数値コードです。

例　数値コードに対する ASCII 文字の取得

次の SELECT ステートメントは、姓 (ラストネーム) と名前 (ファーストネーム) の間にコロン (:) を配置します。

```
SQL
SELECT
LAST_NAME AS ' ', CHR(58) AS ' ', FIRST_NAME AS ' '
FROM EMPLOYEE
;
TABLE
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果は次のとおりです。

```
STEVENS     :  ALFRED
SMITH       :  MARY
JONES       :  DIANE
SMITH       :  RICHARD
BANNING     :  JOHN
IRVING      :  JOAN
ROMANS      :  ANTHONY
MCCOY       :  JOHN
BLACKWOOD   :  ROSEMARIE
MCKNIGHT    :  ROGER
GREENSPAN   :  MARY
CROSS       :  BARBARA
```

24

三角関数

三角関数は、三角関数計算、逆三角関数、および角度変換関数を提供します。

トピックス

ACOS - コサインに対する角度の計算

ACOS (アークコサイン) 関数は、指定されたラジアン単位のコサインに基づいて、0 (ゼロ) から pi ラジアンの間の角度を返します。

構文　コサインに対する角度の計算

```
ACOS(expression)
```

説明

`expression`

数値

角度のコサインです。

例　値に対するアークコサインの計算

次のリクエストは、0 (ゼロ)、PI/2 ラジアン、PI/4 ラジアン、PI ラジアンのアークコサインを計算します。

```
DEFINE FILE ggsales
PI1 = PI();
PI2 = PI()/2;
PI4 = PI()/4;
COS1 = COS(0);
COS2 = COS(PI2);
COS3 = COS(PI4);
COS4 = COS(PI1);
END

TABLE FILE ggsales
PRINT
COS1 COS2 COS3 COS4
OVER
COMPUTE
ARCCOS1/D12.2 = ACOS(COS1);
ARCCCOS2/D12.2 = ACOS(COS2);
ARCCOS3/D12.2 = ACOS(COS3);
ARCCOS4/D12.2 = ACOS(COS4);
BY DATE
WHERE RECORDLIMIT EQ 1
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果は次のとおりです。

Date								
1997/04/01	COS1	1.00	COS2	.00	COS3	.71	COS4	-1.00
	ARCCOS1	.00	ARCCCOS2	1.57	ARCCOS3	.79	ARCCOS4	3.14

ASIN - サインに対する角度の計算

ASIN (アークサイン) 関数は、指定されたラジアン単位のサインに基づいて、-(pi/2) ラジアンと pi/2 ラジアンの間の角度を返します。

構文　サインに対する角度の計算

```
ASIN(expression)
```

説明

`expression`

数値

角度のサインです。

例 値に対するアークサインの計算

次のリクエストは、0 (ゼロ)、PI/2 ラジアン、PI/4 ラジアン、PI ラジアンのアークサインを計算します。

```
DEFINE FILE ggsales
PI1 = PI();
PI2 = PI()/2;
PI4 = PI()/4;
SIN1 = SIN(0);
SIN2 = SIN(PI2);
SIN3 = SIN(PI4);
SIN4 = SIN(PI1);
END

TABLE FILE ggsales
PRINT
SIN1 SIN2 SIN3 SIN4
OVER
COMPUTE
ARCSIN1/D12.2 = ASIN(SIN1);
ARCCSIN2/D12.2 = ASIN(SIN2);
ARCSIN3/D12.2 = ASIN(SIN3);
ARCSIN4/D12.2 = ASIN(SIN4);
BY DATE
WHERE RECORDLIMIT EQ 1
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果は次のとおりです。

Date								
1997/04/01	SIN1	.00	SIN2	1.00	SIN3	.71	SIN4	.00
	ARCSIN1	.00	ARCCSIN2	1.57	ARCSIN3	.79	ARCSIN4	.00

ATAN - タンジェントに対する角度の計算

ATAN (アークタンジェント) 関数は、指定されたラジアン単位のタンジェントに基づいて、 -(pi/2) ラジアンと pi/2 ラジアンの間の角度を返します。

構文　タンジェントに対する角度の計算

```
ATAN(expression)
```

説明

`expression`
　数値

　角度のタンジェントです。

例　値に対するアークタンジェントの計算

次のリクエストは、0 (ゼロ)、PI/4 ラジアン、PI ラジアンのアークタンジェントを計算します。

```
DEFINE FILE ggsales
PI1 = PI();
PI4 = PI()/4;
TAN1 = TAN(0);
TAN3 = TAN(PI4);
TAN4 = TAN(PI1);
END

TABLE FILE ggsales
PRINT
TAN1 TAN3 TAN4
OVER
COMPUTE
ARCTAN1/D12.2 = ATAN(TAN1);
ARCTAN3/D12.2 = ATAN(TAN3);
ARCTAN4/D12.2 = ATAN(TAN4);
BY DATE
WHERE RECORDLIMIT EQ 1
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果は次のとおりです。

Date						
1997/04/01	TAN1	.00	TAN3	1.00	TAN4	.00
	ARCTAN1	.00	ARCTAN3	.79	ARCTAN4	.00

ATAN2 - タンジェントの座標に対する角度の計算

ATAN2 (アークタンジェント 2) 関数は、指定されたラジアン単位のタンジェントの座標に基づいて、-pi ラジアンと pi ラジアンの間の角度を返します。

構文　タンジェントの座標に対する角度の計算

```
ATAN2(x,y)
```

説明

`x`
　数値

　角度のタンジェントのラジアン単位の X 座標です。

`y`
　数値

　角度のタンジェントのラジアン単位の Y 座標です。

例　座標セットに対するアークタンジェントの計算

次のリクエストは、(PI,0)、(PI/4,PI/4)、(PI,PI) のアークタンジェントを計算します。

```
DEFINE FILE ggsales
PI4 = PI()/4;
END

TABLE FILE ggsales
PRINT
COMPUTE
ATAN2A/D12.2 = ATAN2(PI(),0);
ATAN2B/D12.2 = ATAN2(PI4,PI4);
ATAN2C/D12.2 = ATAN2(PI(),PI());

BY DATE
WHERE RECORDLIMIT EQ 1
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果は次のとおりです。

Date	ATAN2A	ATAN2B	ATAN2C
1997/04/01	1.57	.79	.79

COS - 角度のコサインを計算

COS 関数は、ラジアンで指定された角度から、そのコサインを計算します。

構文　角度のコサインを計算

```
COS(expression)
```

説明

`expression`

数値

ラジアン単位の角度です。

例　角度のコサインを計算

次のリクエストは、0 (ゼロ)、PI/2 ラジアン、PI/4 ラジアン、PI ラジアンのコサインを計算し、それらのコサインのアークコサインを計算します。

```
DEFINE FILE ggsales
PI1 = PI();
PI2 = PI()/2;
PI4 = PI()/4;
END

TABLE FILE ggsales
PRINT COMPUTE
COSINE1 = COS(0);
COSINE2 = COS(PI2);
COSINE3 = COS(PI4);
COSINE4 = COS(PI1);
BY DATE
WHERE RECORDLIMIT EQ 1
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果は次のとおりです。

Date	COSINE1	COSINE2	COSINE3	COSINE4
1997/04/01	1.00	.00	.71	-1.00

COT - 角度のコタンジェントを計算

COT 関数は、指定されたラジアン単位の角度に基づいて、その角度のコタンジェントを計算します。

構文　角度のコタンジェントを計算

```
COT(expression)
```

説明

`expression`

数値

ラジアン単位の角度です。

例　角度のコタンジェントを計算

次のリクエストは、PI/2 および PI/4 ラジアンのコタンジェントを計算します。

```
DEFINE FILE ggsales
PI2 = PI()/2;
PI4 = PI()/4;
END

TABLE FILE ggsales
PRINT COMPUTE
COTGENT2 = COT(PI2);
COTGENT3 = COT(PI4);
BY DATE
WHERE RECORDLIMIT EQ 1
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果は次のとおりです。

Date	COT1	COT2
1997/04/01	.00	1.00

DEGREES - ラジアンから度数への変換

DEGREES は、角度 (ラジアン単位) を角度 (度単位) に変換します。

構文　　ラジアンを度数に変換

```
DEGREES(expression)
```

説明

`expression`

数値

ラジアン単位の角度です。

例　　ラジアンを度数に変換

次のリクエストは、0 (ゼロ)、PI/2、PI/4、および PI ラジアンを度に変換します。

```
DEFINE FILE ggsales
PI2 = PI()/2;
PI4 = PI()/4;
END

TABLE FILE ggsales
PRINT COMPUTE
DEG1/D12.2 = DEGREES(0);
DEG2/D12.2 = DEGREES(PI2);
DEG3/D12.2 = DEGREES(PI4);
DEG4/D12.2 = DEGREES(PI());
BY DATE
WHERE RECORDLIMIT EQ 1
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果は次のとおりです。

Date	DEG1	DEG2	DEG3	DEG4
1997/04/01	.00	90.00	45.00	180.00

PI - 定数 Pi を取得

PI 関数は、定数 pi を浮動小数点数として取得します。

構文　　値 Pi を取得

```
PI()
```

例　定数 Pi を取得

次のリクエストは、定数 pi を小数点以下 2 桁に端数処理し、さらに小数点以下 10 桁に端数処理した数値を取得します。

```
TABLE FILE ggsales
PRINT COMPUTE
PI2/D12.2 = PI();
PI10/D12.10 = PI();
BY DATE
WHERE RECORDLIMIT EQ 1
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果は次のとおりです。

Date	PI2	PI10
1997/04/01	3.14	3.1415926536

RADIANS - 度数をラジアンに変換

RADIANS 関数は、角度 (度単位) を角度 (ラジアン単位) に変換します。

構文　度数をラジアンに変換

```
RADIANS(expression)
```

説明

`expression`

数値

度単位の角度です。

例　ラジアンを度数に変換

次のリクエストは、0 (ゼロ) 度、45 度、90 度、180 度をラジアンに変換します。

```
TABLE FILE ggsales
PRINT COMPUTE
RAD1/D12.2 = RADIANS(0);
RAD2/D12.2 = RADIANS(45);
RAD3/D12.2 = RADIANS(90);
RAD4/D12.2 = RADIANS(180);
BY DATE
WHERE RECORDLIMIT EQ 1
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果は次のとおりです。

Date	RAD1	RAD2	RAD3	RAD4
1997/04/01	.00	.79	1.57	3.14

SIN - 角度のサインを計算

SIN 関数は、指定されたラジアン単位の角度に基づいて、そのサインを計算します。

構文　角度のサインを計算

```
SIN(expression)
```

説明

`expression`

数値

ラジアン単位の角度です。

例　角度のサインを計算

次のリクエストは、0 (ゼロ)、PI/2 ラジアン、PI/4 ラジアン、および PI ラジアンのサインを計算します。

```
DEFINE FILE ggsales
PI1 = PI();
PI2 = PI()/2;
PI4 = PI()/4;
END

TABLE FILE ggsales
PRINT COMPUTE
SINE1 = SIN(0);
SINE2 = SIN(PI2);
SINE3 = SIN(PI4);
SINE4 = SIN(PI1);
BY DATE
WHERE RECORDLIMIT EQ 1
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果は次のとおりです。

Date	SINE1	SINE2	SINE3	SINE4
1997/04/01	.00	1.00	.71	.00

TAN - 角度のタンジェントを計算

TAN 関数は、指定されたラジアン単位の角度に基づいて、そのタンジェントを計算します。

構文　角度のタンジェントを計算

```
TAN(expression)
```

説明

`expression`

数値

ラジアン単位の角度です。

例　角度のタンジェントを計算

次のリクエストは、0 (ゼロ)、PI/4 ラジアン、PI ラジアンのタンジェントを計算します。

```
DEFINE FILE ggsales
PI1 = PI();
PI4 = PI()/4;
END

TABLE FILE ggsales
PRINT COMPUTE
TANGENT1 = TAN(0);
TANGENT2 = TAN(PI4);
TANGENT3 = TAN(PI1);
BY DATE
WHERE RECORDLIMIT EQ 1
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

出力結果は次のとおりです。

Date	TANGENT1	TANGENT2	TANGENT3
1997/04/01	.00	1.00	.00

A

サブルーチンの作成

この章では、WebFOCUS に付属の関数とともに使用可能なカスタムサブルーチンの作成について説明します。サブルーチンの作成は、次の作業で構成されています。

- サブルーチンコールをサポートする言語を使用したサブルーチンの作成。一般的に使用される言語には、FORTRAN、COBOL、PL/I、アセンブラ、C 言語などがあります。詳細は、623 ページの 「 サブルーチンの作成 」 を参照してください。
- サブルーチンのコンパイル。
- サブルーチンの個別ファイルへの格納。サブルーチンは、メインプログラムには格納しないでください。
- サブルーチンのテスト。

サブルーチンの作成

サブルーチンの作成には、サブルーチンをサポートする任意の言語を使用することができます。サブルーチンを他のユーザに利用可能にするには、サブルーチンの動作、使用する引数、これらの引数のフォーマットやサブルーチンコール時に使用する順序を定義する必要があります。

サブルーチンを作成する際、次の要件や制限事項を考慮する必要があります。これらは次のとおりです。

- 名前の付け方。詳細は、624 ページの 「 サブルーチン名の指定 」 を参照してください。
- 引数の考慮事項。詳細は、625 ページの 「 引数の作成 」 を参照してください。
- 言語の考慮事項。
- プログラミングの考慮事項。詳細は、626 ページの 「 サブルーチンのプログラミング 」 を参照してください。

単利の利子を得る口座の金額を計算する「INTCOMP」という名前のプログラムを作成する場合、プログラムはレコードの読み取りとデータの受容性のテストを実行し、「SIMPLE」という名前のサブルーチンを呼び出して金額を計算します。プログラムとサブルーチンは、同一ファイル内に格納されます。

次の例に示すプログラムとサブルーチンは疑似コードで記述されています。擬似コードとは、コンピュータコードの一般的な記述方法です。

```
Begin program INTCOMP.
Execute this loop until end-of-file.
   Read next record, fields: PRINCPAL, DATE_PUT, YRRATE.
   If PRINCPAL is negative or greater than 100,000,
      reject record.
   If DATE_PUT is before January 1, 1975, reject record.
   If YRRATE is negative or greater than 20%, reject record.
   Call subroutine SIMPLE (PRINCPAL, DATE_PUT, YRRATE, TOTAL).
   Print PRINCPAL, YEARRATE, TOTAL.
End of loop.
End of program.

Subroutine SIMPLE (AMOUNT, DATE, RATE, RESULT).
Retrieve today's date from the system.
Let NO_DAYS = Days from DATE until today's date.
Let DAY_RATE = RATE / 365 days in a year.
Let RESULT = AMOUNT * (NO_DAYS * DAY_RATE + 1).
End of subroutine.
```

SIMPLE サブルーチンをメインプログラムとは異なるファイルに移動し、コンパイルすると、サブルーチンとして呼び出すことができます。次のレポートリクエストは、12 パーセントの利子を払う口座に従業員が給与を投資した場合に得られる金額を計算します。

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME DAT_INC SALARY AND COMPUTE
   INVESTED/D10.2 = SIMPLE(SALARY, DAT_INC, 0.12, INVESTED);
BY EMP_ID
END
```

注意：このサブルーチンは投資金額のみを返すよう設計されており、現在の日付は返されません。これは、サブルーチンが呼び出し時に返すことのできる値は 1 つだけであるためです。

サブルーチン名の指定

サブルーチン名は、8 バイト以内で指定します。ただし、サブルーチンの作成に使用した言語によっては、7 バイト以下にする必要がある場合があります。名前は文字で始める必要がありますが、文字と数字の任意の組み合わせを使用することができます。特殊記号は使用できません。

引数の作成

サブルーチンの引数を作成する場合、次の問題を考慮する必要があります。

- **引数の最大数**　サブルーチンには、200 個以内の引数を含めることができます。複数の呼び出しが可能なサブルーチンを作成することにより、200 個を超える引数を含めることができます。
- **引数タイプ**　サブルーチンには、関数と同種の引数を使用することができます。引数のタイプについての詳細は、47 ページの 「 引数の種類 」 を参照してください。
- **入力引数**　入力引数は、標準規則に従ってサブルーチンに渡されます。引数リストにポイントを 1 つ登録します。

  入力パラメータは連続するメモリに格納されるとは考えられません。
- **出力引数**　サブルーチンが返す出力引数は 1 つだけです。この引数は、サブルーチンの最後に指定する必要があります。出力引数には任意のフォーマットを選択することができますが、ダイアログマネージャでは、引数は出力フィールドのフォーマットである必要があります。
- **内部処理**　サブルーチンの引数は、次のように処理されます。
  - 文字引数は変更されません。
  - 数値引数は、オペレーティングシステムの RUN コマンド、または変数に出力を格納する場合を除き、倍精度浮動小数点数フォーマットに変換されます。
- **ダイアログマネージャの要件**　ダイアログマネージャに特化したサブルーチンを作成している場合、サブルーチンは変換を実行する必要があります。ダイアログマネージャでのサブルーチンの使用についての詳細は、54 ページの 「 ダイアログマネージャコマンドからの関数の呼び出し 」 を参照してください。

WebFOCUS に定義される呼び出し引数の長さは、サブルーチンに定義された対応する引数の長さに一致する必要があります。

これらの規則に従わない場合、サブルーチンの使用時にエラーが発生することがあります。

- AMODE 31 (アドレッシングモード - 31-bit アドレッシング)
- RMODE ANY (システムはこのルーチンを任意の場所にロード可能)

サブルーチンのプログラミング

プログラミング要件を計画する際、次の点を考慮してください。

- 出力フィールドを指定する引数をサブルーチンに記述する。
- サブルーチンが変数を初期化する場合、サブルーチンが実行されるたびに変数を初期化する必要があります (連続再利用性)。
- 1 つのリクエストがサブルーチンを何回も実行する場合があるため、サブルーチンはできるだけ効率的に記述します。
- サブルーチンをテキストファイルまたはテキストライブラリで作成する場合、サブルーチンは 31 ビットのアドレスに対応できる必要があります。
- 最後の引数は通常、サブルーチンの結果を返すために使用されます。サブルーチンからの入力指定にも使用することができます。

プログラミングの手法を活用することにより、サブルーチンに柔軟性を持たせることができます。次の手法が使用できます。

- エントリポイントでのサブルーチンの実行。エントリポイントにより、1 つのアルゴリズムで複数の結果を生成することができます。詳細は、626 ページの 「 エントリポイントでのサブルーチンの実行 」 を参照してください。
- 複数のサブルーチンコールによる単一サブルーチンの作成。複数回呼び出すことにより、サブルーチンは 200 個を超える引数を処理することが可能になります。詳細は、628 ページの 「 200 個を超える引数を含むサブルーチンコール 」 を参照してください。

エントリポイントでのサブルーチンの実行

サブルーチンは通常、1 つ目のステートメントから実行されます。ただし、サブルーチンは、エントリポイントとして指定されたコード内の任意の位置で実行することも可能です。 これにより、サブルーチンは 1 つの基本アルゴリズムを使用して、複数の結果を生成することができます。たとえば、DOWK サブルーチンは、ある日付の曜日を計算します。サブルーチン名 DOWK を指定すると、3 バイトの曜日の略名が出力されます。エントリ名 DOWKL を指定すると、曜日の完全な名前が出力されます。ただし、実行される計算は同じです。

各エントリポイントには名前が指定されています。ある特定のエントリポイントでサブルーチンを実行するには、サブルーチンコールにそのエントリポイント名を指定します。サブルーチン名は指定しません。エントリポイントの指定方法は、使用する言語により異なります。

構文　エントリポイントでのサブルーチンの実行

```
{subroutine|entrypoint}  (input1, input2,...outfield)
```

説明

`subroutine`
サブルーチン名です。

`entrypoint`
サブルーチンを実行するエントリポイント名です。

`input1, input2,...`
サブルーチンの引数です。

`outfield`
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。

ダイアログマネージャでは、フォーマットを指定する必要があります。

例　エントリポイントでのサブルーチンの実行

次の擬似コードに記述されている FTOC サブルーチンは、華氏で表された温度を摂氏に変換します。エントリポイント FTOK (エントリコマンドが指定) は、摂氏から 273 を引くフラグを設定し、華氏を計算します。サブルーチンは次のようになります。

```
Subroutine FTOC (FAREN, CENTI).
Let FLAG = 0.
Go to label X.
Entry FTOK (FAREN, CENTI).
Let FLAG = 1.
Label X.
Let CENTI = (5/9) * (FAREN - 32).
If FLAG = 1 then CENTI = CENTI - 273.
Return.
End of subroutine.
```

次の記述は、上記のサブルーチンを簡略化したものです。エントリポイントに記述された出力引数 *KELV* は、サブルーチンの開始位置に記述された出力引数 *CENTI* とは異なります。

```
Subroutine FTOC (FAREN, CENTI).
Entry FTOK (FAREN, KELV).
Let CENTI = (5/9) * (FAREN - 32).
KELV = CENTI - 273.
Return.
End of Subroutine.
```

摂氏温度を計算するには、サブルーチンコールにサブルーチン名 FTOC を指定します。サブルーチンは、次のように処理されます。

```
CENTIGRADE/D6.2 = FTOC (TEMPERATURE, CENTIGRADE);
```

華氏温度を計算するには、サブルーチンコールにエントリ名 FTOK を指定します。サブルーチンは、次のように処理されます。

```
KELVIN/D6.2 = FTOK (TEMPERATURE, KELVIN);
```

200 個を超える引数を含むサブルーチンコール

サブルーチンには、出力引数を含めて 200 個以内の引数を指定することができます。200 個を超える引数を処理するには、サブルーチンにコールステートメントを複数指定することで、サブルーチンに引数を渡す必要があります。

複数の呼び出しが可能なサブルーチンを作成するには、次の手順を実行します。

1. サブルーチンをセグメントに分割します。各セグメントには、対応するサブルーチンコールにより引数が渡されます。

   サブルーチンの開始位置に記述された引数リストには、サブルーチンコールに指定された引数リストと同数の引数が指定されている必要があります。この引数には、コール番号引数と出力引数も含まれます。

   各コールには、同数の引数が含まれます。これは、各コールの引数リストには、サブルーチンの開始位置の引数リストと同数の引数が指定されている必要があるためです。引数の個数が一致しない場合は、ダミー引数を使用することが可能です。たとえば、32 個の引数を 6 つのセグメントに分割した場合、各セグメントは引数を 6 つ処理します。6 つ目のセグメントは、引数を 2 つとダミー引数を 4 つ処理することになります。

   サブルーチンには、作成者の判断により、追加の引数が必要な場合があります。

2. サブルーチンの開始位置からコール番号 (1 つ目の引数) を読み取り、対応するセグメントに分岐するステートメントを記述します。各セグメントは、1 つのコールから引数を処理します。たとえば、1 つ目のコールは 1 つ目のセグメントへ、2 つ目のコールは 2 つ目のセグメントへ分岐されます。
3. 各セグメントが渡された引数を他の変数に格納するか、現在の合計に追加します。引数が格納された変数は、最後のセグメントにより処理されます。

   各セグメントの最後に RETURN コマンド (制御をリクエストに戻すコマンド) を記述します。
4. 最後のセグメントは、リクエストに最終出力値を返します。

また、エントリポイントを使用することにより、200 個を超える引数を処理するサブルーチンを作成することもできます。詳細は、626 ページの 「 エントリポイントでのサブルーチンの実行 」 を参照してください。

構文　複数のコールステートメントによるサブルーチンの作成

```
field = subroutine (1, group1, field)
;field = subroutine (2, group2, field);
   .
   .
   .outfield = subroutine (n, groupn, outfield);
```

説明

`field`
セグメントの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。このフィールドのフォーマットには、*outfield* に指定されたフォーマットを使用する必要があります。

最後のコールステートメントには *field* を指定せず、*outfield* を使用してください。

`subroutine`
サブルーチン名です。サブルーチン名は 8 バイト以内で指定します。

`n`
各サブルーチンコールを識別する番号です。各サブルーチンコールの 1 つ目の引数として記述されている必要があります。サブルーチンは、このコール番号を使用して、コードのセグメントへの分岐を実行します。

`group1, group2,...`
各サブルーチンコールにより渡される入力引数のリストです。各グループには、同数 (26 以下) の引数が含まれています。

最終グループには、ダミー引数を含めることができます。

`outfield`
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。

ダイアログマネージャでは、フォーマットを指定する必要があります。

例　セグメントに分割したサブルーチンの作成

擬似コードに記述された ADD32 サブルーチンは、32 個の数値を合計します。合計数は 6 つのセグメントに分割されます。各セグメントは、サブルーチンコールから 6 つの数値を追加します。入力引数の合計数は 36 ですが、最後の 4 つはダミー引数です。6 つ目のセグメントは、SUM 変数に 2 つの引数を追加し、結果を返します。6 つ目のセグメントは、ダミー引数に指定された値は処理しません。

サブルーチンは次のようになります。

```
Subroutine ADD32 (NUM, A, B, C, D, E, F, TOTAL).
If NUM is 1 then goto label ONE
else if NUM is 2 then goto label TWO
else if NUM is 3 then goto label THREE
else if NUM is 4 then goto label FOUR
else if NUM is 5 then goto label FIVE
else goto label SIX.

Label ONE.
Let SUM = A + B + C + D + E + F.
Return.

Label TWO
Let SUM = SUM + A + B + C + D + E + F
Return

Label THREE
Let SUM = SUM + A + B + C + D + E + F
Return

Label FOUR
Let SUM = SUM + A + B + C + D + E + F
Return

Label FIVE
Let SUM = SUM + A + B + C + D + E + F
Return

Label SIX
LET TOTAL = SUM + A + B
Return
End of subroutine
```

ADD32 サブルーチンを使用するには、6 つの数値を指定するコールステートメントの 6 つすべてを記述します。最後の 4 つの数値はダミー引数で、0 (ゼロ) で表されます。DEFINE コマンドは 32 個の数値を SUM32 フィールドに格納します。

```
DEFINE FILE EMPLOYEE
DUMMY/D10 = ADD32 (1, 5, 7, 13, 9, 4, 2, DUMMY);
DUMMY/D10 = ADD32 (2, 5, 16, 2, 9, 28, 3, DUMMY);
DUMMY/D10 = ADD32 (3, 17, 12, 8, 4, 29, 6, DUMMY);
DUMMY/D10 = ADD32 (4, 28, 3, 22, 7, 18, 1, DUMMY);
DUMMY/D10 = ADD32 (5, 8, 19, 7, 25, 15, 4, DUMMY);
SUM32/D10 = ADD32 (6, 3, 27, 0, 0, 0, 0, SUM32);
END
```

B

ASCII コード

ここに記載されている表は、表示可文字および表示不可文字の ASCII コードを示しています。

ASCII 文字コード表

下表は、標準 ASCII 文字を対応するコード番号順に、10 進数および 16 進数のコード値とともに記載しています。

10 進コード	16 進数	ASCII	
0	00	NUL	null
1	01	SOH	ヘッディング開始
2	02	STX	テキスト開始
3	03	ETX	テキスト終結
4	04	EOT	伝送終了
5	05	ENQ	問い合わせ
6	06	ACK	肯定応答
7	07	BEL	ベル
8	08	BS	バックスペース
9	09	HT	水平タブ
10	0A	LF	改行
11	0B	VT	垂直タブ
12	0C	FF	用紙送り

10 進コード	16 進数	ASCII	
13	0D	CR	復帰
14	0E	SO	シフトアウト
15	0F	SI	シフトイン
16	10	DLE	伝送制御拡張
17	11	DC1	装置制御 1
18	12	DC2	装置制御 2
19	13	DC3	装置制御 3
20	14	DC4	装置制御 4
21	15	NAK	否定応答
22	16	SYN	同期信号
23	17	ETB	伝送ブロック終結
24	18	CAN	取り消し
25	19	EM	メディア終端
26	1A	SUB	置換
27	1B	ESC	エスケープ
28	1C	FS	ファイル分離文字
29	1D	GS	グループ分離文字
30	1E	RS	レコード分離文字
31	1F	US	ユニット分離文字
32	20	SP	ブランク
33	21	!	感嘆符
34	22	"	直線二重引用符

10 進コード	16 進数	ASCII	
35	23	#	ナンバー
36	24	$	ドル
37	25	%	パーセント記号
38	26	&	アンパサンド
39	27	'	アポストロフィ
40	28	(	左括弧
41	29	)	右括弧
42	2A	*	アスタリスク
43	2B	+	加算記号
44	2C	,	カンマ
45	2D	-	減算記号
46	2E	.	ピリオド
47	2F	/	右スラッシュ
48	30	0	0
49	31	1	1
50	32	2	2
51	33	3	3
52	34	4	4
53	35	5	5
54	36	6	6
55	37	7	7
56	38	8	8

10 進コード	16 進数	ASCII	
57	39	9	9
58	3A	:	コロン
59	3B	;	セミコロン
60	3C	<	より小さい
61	3D	=	等号
62	3E	>	より大きい
63	3F	?	疑問符
64	40	@	アットマーク
65	41	A	A
66	42	B	B
67	43	C	C
68	44	D	D
69	45	E	E
70	46	F	F
71	47	G	G
72	48	H	H
73	49	I	I
74	4A	J	J
75	4B	K	K
76	4C	L	L
77	4D	M	M
78	4E	N	N

10 進コード	16 進数	ASCII	
79	4F	O	O
80	50	P	P
81	51	Q	Q
82	52	R	R
83	53	S	S
84	54	T	T
85	55	U	U
86	56	V	V
87	57	W	W
88	58	X	X
89	59	Y	Y
90	5A	Z	Z
91	5B	[	左大括弧
92	5C	¥	円記号
93	5D	]	右大括弧
94	5E	^	ハット、曲折アクセント記号
95	5F	_	アンダースコア
96	60	`	アクサングラーブ
97	61	a	a
98	62	b	b
99	63	c	c
100	64	d	d

10 進コード	16 進数	ASCII	
101	65	e	e
102	66	f	f
103	67	g	g
104	68	h	h
105	69	i	i
106	6A	j	j
107	6B	k	k
108	6C	l	l
109	6D	m	m
110	6E	n	n
111	6F	o	o
112	70	p	p
113	71	q	q
114	72	r	r
115	73	s	s
116	74	t	t
117	75	u	u
118	76	v	v
119	77	w	w
120	78	x	x
121	79	y	y
122	7A	z	z

10 進コード	16 進数	ASCII	
123	7B	{	左中括弧
124	7C	|	縦線
125	7D	}	右中括弧
126	7E	~	チルダ

SOME CLOUD SOFTWARE GROUP, INC. (“CLOUD SG”) SOFTWARE AND CLOUD SERVICES EMBED, BUNDLE, OR OTHERWISE INCLUDE OTHER SOFTWARE, INCLUDING OTHER CLOUD SG SOFTWARE (COLLECTIVELY, “INCLUDED SOFTWARE”). USE OF INCLUDED SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED CLOUD SG SOFTWARE AND/OR CLOUD SERVICES. THE INCLUDED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER CLOUD SG SOFTWARE AND/OR CLOUD SERVICES OR FOR ANY OTHER PURPOSE.

USE OF CLOUD SG SOFTWARE AND CLOUD SERVICES IS SUBJECT TO THE TERMS AND CONDITIONS OF AN AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER AGREEMENT WHICH IS DISPLAYED WHEN ACCESSING, DOWNLOADING, OR INSTALLING THE SOFTWARE OR CLOUD SERVICES (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH LICENSE AGREEMENT OR CLICKWRAP END USER AGREEMENT, THE LICENSE(S) LOCATED IN THE “LICENSE” FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE SAME TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

Cloud SG 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 a specific version of Cloud SG software on a specific operating system platform.