Informatica®

Informatica® Cloud Data Integration
October 2023

関数リファレンス

Informatica Cloud Data Integration 関数リファレンス
October 2023
2022 年 11 月

発行日: 2024-01-10

目次

序文

データ統合を使用してデータを変換するための関数の記述に使用するトランスフォーメーション言語の詳細については、「*関数リファレンス*」を参照してください。簡単な、または複雑なトランスフォーメーション式を記述するための関数、演算子、および定数について学びます。

Informatica のリソース

Informatica は、Informatica Network やその他のオンラインポータルを通じてさまざまな製品リソースを提供しています。リソースを使用して Informatica 製品とソリューションを最大限に活用し、その他の Informatica ユーザーや各分野の専門家から知見を得ることができます。

Informatica マニュアル

Informatica マニュアルポータルでは、最新および最近の製品リリースに関するドキュメントの膨大なライブラリを参照できます。マニュアルポータルを利用するには、https://docs.informatica.com にアクセスしてください。

製品マニュアルに関する質問、コメント、ご意見については、Informatica マニュアルチーム（infa_documentation@informatica.com）までご連絡ください。

Informatica Intelligent Cloud Services Web サイト

Informatica Intelligent Cloud Services Web サイト（http://www.informatica.com/cloud）にアクセスできます。このサイトには、Informatica Cloud 統合サービスに関する情報が含まれます。

Informatica Intelligent Cloud Services コミュニティ

Informatica Intelligent Cloud Services コミュニティを使用して、技術的な問題について議論し、解決します。また、技術的なヒント、マニュアルの更新情報、FAQ（よくある質問）への答えを得ることもできます。

次の Informatica Intelligent Cloud Services コミュニティにアクセスします。

https://network.informatica.com/community/informatica-network/products/cloud-integration

開発者は、次の Cloud 開発者コミュニティで詳細情報を確認したり、ヒントを共有したりできます。

https://network.informatica.com/community/informatica-network/products/cloud-integration/cloud-developers

Informatica Intelligent Cloud Services マーケットプレース

Informatica マーケットプレイスにアクセスすると、データ統合コネクタ、テンプレート、およびマップレットを試用したり購入したりできます。

https://marketplace.informatica.com/

データ統合のコネクタのドキュメント

データ統合のコネクタのドキュメントには、マニュアルポータルからアクセスできます。マニュアルポータルを利用するには、https://docs.informatica.com にアクセスしてください。

Informatica ナレッジベース

Informatica ナレッジベースを使用して、ハウツー記事、ベストプラクティス、よくある質問に対する回答など、製品リソースを見つけることができます。

ナレッジベースを検索するには、https://search.informatica.com にアクセスしてください。ナレッジベースに関する質問、コメント、ご意見の連絡先は、Informatica ナレッジベースチーム（KB_Feedback@informatica.com）です。

Informatica Intelligent Cloud Services Trust Center

Informatica Intelligent Cloud Services Trust Center は、Informatica のセキュリティポリシーおよびリアルタイムでのシステムの可用性について情報を提供します。

Trust Center（https://www.informatica.com/trust-center.html）にアクセスします。

Informatica Intelligent Cloud Services Trust Center にサブスクライブして、アップグレード、メンテナンス、およびインシデントの通知を受信します。Informatica Intelligent Cloud Services Status ページには、すべての Informatica Cloud 製品の実稼働ステータスが表示されます。メンテナンスの更新はすべてこのページに送信され、停止中は最新の情報が表示されます。更新と停止の通知がされるようにするには、Informatica Intelligent Cloud Services の 1 つのコンポーネントまたはすべてのコンポーネントについて更新の受信をサブスクライブします。すべてのコンポーネントにサブスクライブするのが、更新を逃さないようにするための最良の方法です。

サブスクライブするには、Informatica Intelligent Cloud Services Status ページで **[サブスクライブして更新]** をクリックします。電子メール、SMS テキストメッセージ、Webhook、RSS フィード、またはこれらの 4 つの任意の組み合わせとして送信された通知を受信するという選択ができます。

Informatica グローバルカスタマサポート

グローバルサポートセンターには、Informatica Network または電話でお問い合わせください。

Informatica Network でオンラインサポートリソースを検索するには、Informatica Intelligent Cloud Services のヘルプメニューで **[サポートにお問い合わせください]** をクリックして、**Cloud Support** ページに移動します。**Cloud Support** ページには、システムステータス情報とコミュニティディスカッションが記載されています。追加のリソースを検索する場合や電子メールで Informatica グローバルカスタマサポートに問い合わせる場合は、Informatica Network にログインし、**[サポートが必要な場合]** をクリックしてください。

Informatica グローバルカスタマサポートの電話番号は、Informatica の Web サイト https://www.informatica.com/services-and-training/support-services/contact-us.html に掲載されています。

第 1 章

関数リファレンス

Informatica では、ソースデータを変換するために、SQL に似た関数を含むトランスフォーメーション言語が提供されています。これらの関数を使用して式を記述し、ユーザー定義関数を作成します。

式はデータを変更します。または、データが条件に一致するかテストします。例えば、AVG 関数を使用して従業員の平均給与を計算したり、SUM 関数を使用して特定の支店の総売上高を計算したりできます。AVG などの単一関数のみを含む簡易式を作成できます。関数内に関数をネストして複雑な式を記述することもできます。

ユーザー定義関数は、式の論理を使用して複雑な式を構築します。ユーザ定義関数は、他のユーザ定義関数や式に格納できます。ユーザ定義関数の詳細については、「*コンポーネント*」を参照してください。

関数の構成要素

トランスフォーメーション言語には以下の構成要素が含まれ、ユーザはこれらを使って簡単なトランスフォーメーション式から複雑なトランスフォーメーション式まで作成することができます。

- **関数。**マッピングでデータを変更するために、SQL に似た関数が 100 個以上用意されています。
- **演算子。**トランスフォーメーション演算子を使用することにより、トランスフォーメーション式で算術演算を実行したり、データを結合または比較できます。
- **定数。**一定の値を保つ値を参照するために、TRUE などの定数が用意されています。
- **システム変数。**変数を使用して、現在のタスク名や実行 ID などの変更されたシステム値を返します。

第 2 章

定数

この章では、以下の項目について説明します。

FALSE

条件式を明確にするために使用します。 FALSE は整数 0 と等価です。

例

次の例では、DECODE 式で FALSE を使用して、値が 25 から 30 であることを確認します。これは 1 つの検索値に基づいて複数の検索を行いたい場合に便利です。

```
DECODE (FALSE,
Var1 >= 25, 'Var1 not in desired range.',
Var1 <= 30, 'Var1 not in desired range.',
'Var1 in desired range.')
```

NULL

値が未知または未定義であることを示します。NULL は空の文字列（文字が入る列の場合）、またはゼロ（数値が入る列の場合）と等価ではありません。

NULL 値を返す式を記述することは可能ですが、NOT NULL または PRIMARY KEY 制約が指定された列では NULL は許可されません。したがって、データ統合がいずれかの制約のあるカラムに NULL 値を書き込もうとした場合、データベースではその行が却下され、データ統合はその NULL 値を却下ファイルに書き込みます。トランスフォーメーションを作成する際には、必ず NULL について考慮するようにしてください。

各関数は別々の方法で NULL を扱うことができます。関数に NULL 値を渡した場合、ゼロや NULL を返す関数もあれば、NULL 値を無視する関数もあります。

Boolean 式における Null 値の扱い

NULL 値とブール式を結合する式は ANSI に準拠した結果を生成します。以下に例を示します。

- NULL AND TRUE = NULL
- NULL AND FALSE = FALSE

フィルタ条件における NULL 値

フィルタ条件の評価結果が NULL となった場合、関数はレコードを選択しません。

NULL と演算子

演算子（文字列演算子 || を除く）を使用した式に NULL 値が含まれると、式の評価結果が常に NULL になります。たとえば、次の式を評価した結果は NULL になります。

```
8 * 10 - NULL
```

NULL かどうかをテストするには、ISNULL 関数を使用します。

TRUE

比較の結果に基づいて値を返します。TRUE は整数 1 と等価です。

例

次の例では、DECODE 式で TRUE を使用し、比較結果に基づいて値を返しています。これは 1 つの検索値に基づいて複数の検索を行いたい場合に便利です。

```
DECODE( TRUE,
Var1 = 22,'Variable 1 was 22!',
Var2 = 49,'Variable 2 was 49!',
Var1 < 23, 'Variable 1 was less than 23.',
Var2 > 30, 'Variable 2 was more than 30.',
'Variables were out of desired ranges.')
```

第 3 章

演算子

この章では、以下の項目について説明します。

演算子の優先順位

式を作成するときに、複数の演算子を使用したり、ネストされた式内で演算子を使用したりできます。

複数の演算子を含む式を記述した場合、データ統合では、次の順序で式が評価されます。

1. 複合演算子
2. 算術演算子
3. 文字列演算子
4. 比較演算子
5. 論理演算子

データ統合では、次の表に示す順序で演算子が評価されます。1 つの式の中の演算子については、左から右にすべての演算子を等しい優先順位で評価します。

以下の表に、トランスフォーメーション言語のすべての演算子の優先順位を示します。

演算子	意味
[ ], .	添字、ドット。
( )	かっこ。
+, -, NOT	単項のプラスとマイナス、および論理否定演算子。
*, /, %	乗算、除算、剰余。
+, -	加算、減算。

演算子	意味
||	連結。
<, <=, >, >=	より小さい、以下、より大きい、以上。
=, <>, ! =, ^=	等しい、等しくない、等しくない、等しくない。
AND	論理演算子 AND（条件の指定時に使用）。
OR	論理演算子 OR（条件の指定時に使用）。

ネストされた式内で演算子を使用できます。式にかっこが含まれている場合、データ統合はかっこの外の演算の前にかっこ内の演算を評価します。最も内側のかっこ内の演算が最初に評価されます。

例えば、演算をどのようにネストするかによって、式 8 + 5 - 2 * 8 の返す値が異なります。

式	戻り値
8 + 5 - 2 * 8	-3
8 + (5 - 2) * 8	32

複合演算子

複合演算子を使用して、配列、マップまたは構造体内の要素にアクセスします。複雑な演算子は詳細モードでのみ使用できます。

次の表に、複合演算子の一覧を示します。

演算子	意味
[ ]	添字演算子。 添字演算子は、配列またはマップで使用します。 - 配列で添字演算子を使用し、1 つ以上の要素にアクセスします。 - マップで添字演算子を使用し、キーと値のペアの特定のキーに対応する値にアクセスします。
.	ドット演算子。 ドット演算子は、構造体または構造体の配列で使用します。 - 構造体でドット演算子を使用し、要素にアクセスします。 - 構造体の配列でドット演算子を使用し、各構造体の要素にアクセスします。演算子は、各構造体内の同じ名前の要素を配列として返します。

ネストされた階層にアクセスするには、複合演算子を組み合わせて使用できます。

添字演算子（配列）

添字演算子を使用して、配列内の要素にアクセスします。特定の要素または要素の範囲にアクセスできます。

構文

配列の特定の要素にアクセスするには、次の構文を使用します。

```
array[ index ]
```

配列の要素の範囲にアクセスするには、次の構文を使用します。

```
array[ start_index , end_index ]
```

以下の表に、構文の引数を示します。

引数	説明
array	配列データ型。アクセスする 1 つ以上の要素が含まれる配列。 配列を求める有効な式を必要に応じて入力できます。
index	整数データ型。アクセスする要素の位置。例えば、インデックス 0 は配列の最初の要素を示します。
start_index	整数データ型。アクセスする要素の範囲の開始インデックス。添字演算子には、開始インデックスが表す要素が含まれます。
end_index	整数データ型。アクセスする要素の範囲の終了インデックス。添字演算子は、終了インデックスが表す要素を除外します。

整数値を返すインデックス式を使用できます。式が負の値を返す場合、インデックスは 0 であると見なされます。

指定したインデックスが配列のサイズから 1 引いた値よりも大きい場合、インデックスは配列の最後の要素にアクセスします。

戻り値

配列の要素。戻り値の型は、要素のデータ型と同じです。

[i, j]のようにカンマで区切られた 2 つのインデックスを指定した場合、この式は i から j-1 までの要素の配列を返します。i が j または配列のサイズより大きい場合、式は空の配列を返します。

次の状況では NULL です。

- インデックスが配列のサイズより大きい。
- インデックスは NULL である。
- [i,j]のように複数の添字を指定し、i と j のいずれかが NULL である。
- 配列が NULL である。

例

次の文字列配列があります。

```
drinks = [‘milk’, ‘coffee’, ‘tea’, ‘chai’]
```

次の式では、添字演算子を使用して配列の文字列要素にアクセスします。

Input Value	RETURN VALUE
drinks[0]	'milk'
drinks[2]	'tea'
drinks[NULL]	NULL
drinks[1,3]	['coffee','tea']
drinks[2,NULL]	NULL
drinks[3,1]	[ ]

添字演算子（マップ）

添字演算子を使用し、キーと値のペアの特定のキーに対応する値にアクセスします。

構文

マップ内の特定のキーに対応する値にアクセスするには、次の構文を使用します。

```
map[ key ]
```

以下の表に、構文の引数を示します。

引数	説明
map	Map データ型。キーに対応する値を取得するマップ。
key	キーのデータ型。値を取得するキー要素。 マップデータのキー値を求める有効な式を入力できます。

戻り値

マップ内のキーに関連付けられている値。戻り値の型は、値のデータ型と同じです。

キーがマップに存在しない場合は NULL。

例

次のマップがあります。

```
country_currency = [‘England’ -> ‘Pound’, ‘France’ -> ‘Euro’, ‘Japan’ -> ‘Yen’>, ‘USA’ -> ‘Dollar’]
```

次の式では、添字演算子を使用してマップの値にアクセスします。

Input Value	RETURN VALUE
country_currency [‘Japan’]	‘Yen’
country_currency [‘India’]	NULL
country_currency [‘England’]	‘Pound’

ドット演算子（構造体）

ドット演算子を使用して、構造体の要素にアクセスします。

構文

構造の要素にアクセスするには、次の構文を使用します。

```
struct.element
```

以下の表に、構文の引数を示します。

引数	説明
struct	Struct データ型。アクセスする要素が含まれる構造体。 構造を求める有効なトランスフォーメーション式を必要に応じて入力できます。
element	アクセスする構造体要素の名前。

戻り値

構造体の要素。戻り値の型は、要素のデータ型と同じです。

次の状況では NULL です。

- 構造体の要素に NULL 値がある。
- 構造体が NULL である。

例

次のような構造があるとします。

```
location{
    street: NULL
    city : 'NEWYORK'
    state: 'NY'
    zip : 12345
}
```

次の式では、ドット演算子を使用して構造の要素にアクセスします。

Input Value	RETURN VALUE
location.street	NULL
location.city	'NEWYORK'
location.state	'NY'
location.zip	12345

ドット演算子（構造体の配列）

構造体の配列でドット演算子を使用して、配列内の各構造体の要素にアクセスします。

構文

構造の配列の要素にアクセスするには、次の構文を使用します。

```
array_of_structs.element
```

以下の表に、構文の引数を示します。

引数	説明
array_of_structs	配列データ型。アクセスする各構造体の要素が含まれる構造体の配列。 配列を求める有効なトランスフォーメーション式を必要に応じて入力できます。
element	アクセスする構造体要素の名前。

戻り値

各構造体から指定された要素を含む配列。

次の状況では NULL です。

- 構造体の要素に NULL 値がある。
- 構造体が NULL である。

例

3 つの構造体要素を持つ次の配列があり、各構造体に 3 つの要素があります。

```
employee_info_array = [

 derrick_struct{
     name: 'Derrick'
     city: NULL
     state: 'NY'
 },

 kevin_struct{
     name: 'Kevin'
     city: 'Redwood City'
     state: 'CA'
 },

 lauren_struct{
     name: 'Lauren'
     city: 'Woodcliff Lake'
     state: NULL
 }

]
```

次の式では、ドット演算子を使用して配列の各構造体の要素にアクセスします。

Input Value	RETURN VALUE
employee_info_array.name	['Derrick','Kevin','Lauren']
employee_info_array.city	[NULL,'Redwood City','Woodcliff Lake']
employee_info_array.state	['NY','CA',NULL]

ネストされた階層の復号演算子

ネストされた階層には、構造体の配列などの階層データも含む要素が含まれます。ネストされた階層の要素にアクセスするには、複合演算子を組み合わせて使用します。

次のタイプのネストされた階層の要素にアクセスできます。

- 多次元配列

- 構造体の配列
- 配列要素を持つ構造体
- ネストされた構造体

多次元配列

多次元配列は配列の配列です。添字演算子を使用して、最も内側のレベルにある配列のプリミティブ要素にアクセスできます。添字演算子を使用して、任意のレベルの配列にアクセスできます。

添字演算子を使用して、次の値を返すことができます。

- 最も内側のレベルにある配列内のプリミティブ要素。
- 任意のレベルにある 1 つ以上の配列。
- 任意のレベルにある 1 つ以上の配列のサブセット。

最も内側のレベルにある配列のプリミティブ要素にアクセスするには、複数の添字演算子を使用します。多次元配列の次元数によって、使用する添字演算子の数が決まります。各添字演算子には 1 つのインデックス値を含める必要があります。戻り値のデータ型は、配列のプリミティブ要素のデータ型と同じです。

例えば、2 次元配列では 2 つの添字演算子を使用します。最初の添字演算子は親配列にアクセスします。2 番目の添字演算子は、親配列内の子配列にアクセスします。

例

3 つの子配列を含み、各子配列に文字列要素が含まれる次の 2 次元の親配列を考えてみます。

```
menu_array = [
 ['milk','coffee','tea','chai'],
 ['ham','turkey',NULL],
 ['caesar','cobb','greek','chipotle']
]
```

添字演算子を使用して、次のタイプの要素にアクセスできます。

プリミティブ要素

次の式では、2 つの添字演算子を使用して、親配列 menu_array 内の各子配列の特定文字列要素にアクセスします。

Input Value	RETURN VALUE
menu_array[0][1]	'coffee'
menu_array[2][3]	'chipotle'
menu_array[1][2]	NULL

配列要素

次の式では、単一の添字演算子を使用して、親配列 menu_array 内の子配列にアクセスします。

Input Value	RETURN VALUE
menu_array[0]	['milk','coffee','tea','chai']

Input Value	RETURN VALUE
menu_array[0,2]	[['milk','coffee','tea','chai'], ['ham','turkey',NULL]]
menu_array[1,0]	[ ]
menu_array[NULL,2]	NULL

配列要素のサブセット

次の式では、2 つの添字演算子を使用して、親配列 menu_array 内の子配列のサブセットにアクセスします。

Input Value	RETURN VALUE
menu_array[0][0,2]	['milk','coffee']
menu_array[2][0,3]	['caesar','cobb','greek']
menu_array[0,2][0,3]	[['milk','coffee','tea'], ['ham','turkey',NULL]]

構造体の配列

構造体の配列は構造体要素を持つ配列です。添字演算子とドット演算子の組み合わせを使用して、子構造体と子構造体の要素にアクセスします。

親配列内の子構造体の要素にアクセスするには、添字演算子の後にドット演算子を使用します。戻り値を変更せずに、演算子の順序を反転することもできます。

例

次のような構造体 employee_info_array の配列があります。

```
employee_info_array = [
derrick_struct{
    name: 'Derrick'
    city: NULL
    state: 'NY'
},

kevin_struct{
    name: 'Kevin'
    city: 'Redwood City'
    state: 'CA'
},

lauren_struct{
    name: 'Lauren'
    city: 'Woodcliff Lake'
    state: NULL
}
]
```

次のいずれかの順序で複合演算子を使用して、いずれかの子構造体の要素にアクセスできます。

添字演算子を使用してから、ドット演算子を使用する。

演算子は、次の順序で構造体の配列にアクセスします。

1. 添字演算子が配列内のインデックス付き要素にアクセスし、構造体を返します。
2. ドット演算子が構造体内の要素にアクセスします。

例えば、次の式では、添字演算子とその後にドット演算子を使用して、配列 employee_info_array の要素にアクセスします。

Input Value	RETURN VALUE
employee_info_array[0].name	'Derrick'
employee_info_array[1].city	'Redwood City'
employee_info_array[2].state	NULL

ドット演算子を使用してから、添字演算子を使用する。

演算子は、次の順序で構造体の配列にアクセスします。

1. ドット演算子は、各構造体から同じ名前の要素を検索し、配列を返します。
2. 添字演算子が配列内のインデックス付き要素にアクセスします。

例えば、次の式は、ドット演算子を使用して配列 employee_info_array の要素にアクセスする場合の戻り値を示しています。

Input Value	RETURN VALUE
employee_info_array.name	['Derrick','Kevin','Lauren']
employee_info_array.city	[NULL,'Redwood City','Woodcliff Lake']
employee_info_array.state	['NY','CA',NULL]

次の式は、ドット演算子とその後に添字演算子を使用して配列 employee_info_array の要素にアクセスする場合の戻り値を示しています。

Input Value	RETURN VALUE
employee_info_array.name[0]	'Derrick'
employee_info_array.city[1]	'Redwood City'
employee_info_array.state[2]	NULL

添字演算子とドット演算子のどちらを先に使用しても戻り値が同じであることに注意してください。例えば、式 employee_info_array[0].name と employee_info_array.name[0]の戻り値は同じ'Derrick'です。

配列要素を持つ構造体

構造体内にある配列の要素にアクセスするには、ドット演算子の後に添字演算子を使用します。ドット演算子が先に構造体の指定した配列にアクセスします。次に、添字演算子がインデックス値に基づいて配列の要素にアクセスします。

例

配列 drinks、sandwiches、salads を持つ次の構造体があります。

```
menu_struct{
drinks: ['milk','coffee','tea','chai']
sandwiches: ['ham','turkey',NULL]
salads: ['caesar','cobb','greek','chipotle']
}
```

式 menu_struct.drinks[0]を使用する場合、演算子は次の順序で親構造体と子配列にアクセスします。

1. ドット演算子は、配列 drinks にアクセスします。
2. 添字演算子が配列 drinks: ['milk','coffee','tea','chai']の位置 0 にある値にアクセスし、'milk'を返します。

次の式は、ドット演算子とその後に添字演算子を使用して、親構造体 menu_struct の子配列の値にアクセスするその他の例を示しています。

Input Value	RETURN VALUE
menu_struct.drinks[1]	'coffee'
menu_struct.sandwiches[2]	NULL
menu_struct.salads[3]	'chipotle'
menu_struct.drinks[0,3]	['milk','coffee','tea']

ネストされた構造体

ネストされた構造体は、1 つ以上のレベルの構造を含む構造体です。ドット演算子を使用して、最も内側のレベルにある構造体のプリミティブ要素にアクセスできます。ドット演算子を使用して、任意のレベルの構造体にアクセスすることもできます。

ドット演算子を使用して、次の値を返すことができます。

- 最も内側のレベルにある構造体のプリミティブ要素。
- 任意のレベルにある 1 つ以上の構造。

最も内側のレベルにある構造体のプリミティブ要素にアクセスするには、複数のドット演算子を使用します。ネストされた構造体のレベル数によって、使用するドット演算子の数が決まります。戻り値のデータ型は、構造体の要素のデータ型と同じです。

例えば、2 つのレベルの構造体を持つネストされた構造体では 2 つのドット演算子を使用します。最初のドット演算子は、親構造体にアクセスして子構造体を見つけます。次に、2 番目のドット演算子が子構造体にアクセスして、子構造体の特定のプリミティブ要素を返します。

例

2 つの子構造体 home_address_info および department_info を含む、次の構造 employee_info_struct があります。

```
employee_info_struct{

emp_name: 'Derrick'
```

```
home_address_info{
    city: 'New York'
    state: NULL

department_info{
    NULL
}

}
```

次の式では、ドット演算子を使用して、構造体 employee_info_struct から値にアクセスします。

Input Value	RETURN VALUE
employee_info_struct.emp_name	'Derrick'

employee_info_struct.home_address_info	{
	city: 'New York'
	state: NULL
	}

employee_info_struct.department_info	NULL
employee_info_struct.home_address_info.city	'New York'
employee_info_struct.home_address_info.state	NULL

算術演算子

算術演算子は、数値データに対して算術計算を実行するときに使用します。

以下の表に、優先度順のトランスフォーメーション言語の算術演算子を示します。

演算子	意味
+, -	単項のプラスおよびマイナス。単項のプラスは正の値を示します。単項のマイナスは負の値を示します。
*, /, %	乗算、除算、剰余。剰余とは、整数を整数で割ったときの余りです。たとえば、13 % 2 = 1 となります。13 を 2 で割ると商が 6 で余りが 1 だからです。
+, -	加算、減算。 加算演算子（+）で文字列を連結することはできません。文字列を連結するには、文字列演算子 || を使用します。日付値に算術演算を実行するには、日付関数を使用します。

NULL 値に対して算術演算を実行すると、関数は NULL を返します。

式で算術演算子を使用する場合、式中のすべてのオペランドは数値でなければなりません。例えば、式 1 + '1'は、文字列に整数を加算しているため無効です。式 1.23 + 4 / 2 は、すべてのオペランドが数値であるため、有効です。

文字列演算子

2 つの文字列を連結するには、文字列演算子 || を使用します。|| 演算子は、任意のデータ型（Binary を除く）のオペランドを String データ型に変換してから連結します。

入力値	戻り値
'alpha' || 'betical'	alphabetical
'alpha' || 2	alpha2
'alpha' || NULL	alpha

|| 演算子は、文字列の先頭や末尾にあるスペースもそのまま連結します。2 つの文字列を連結する前に先頭や末尾のスペースを削除するには、LTRIM 関数および RTRIM 関数を使用します。

NULL

|| 演算子は NULL 値を無視します。ただし、演算子の両側がともに NULL である場合には、NULL を返します。

文字列演算子の例

次の例は、2 つの列から従業員の名前と姓を連結する式を示しています。この式は、名前の末尾と姓の先頭からスペースを削除し、各名前の末尾にスペースを 1 つ連結してから、姓を連結します。

```
LTRIM( RTRIM( EMP_FIRST ) || ' ' || LTRIM( EMP_LAST ))
```

EMP_FIRST	EMP_LAST	RETURN VALUE
' Alfred'	' Rice '	Alfred Rice
' Bernice'	' Kersins'	Bernice Kersins
NULL	' Proud'	Proud
' Curt'	NULL	Curt
NULL	NULL	NULL

注: CONCAT 関数を使って 2 つの文字列を連結することもできます。ただし、演算子を使うと短い時間で同じ結果を得られます。

比較演算子

比較演算子を使用すると、文字列または数値を比較し、データを操作して、TRUE（1）または FALSE（0）の値を返すことができます。

以下の表に、トランスフォーメーション言語の比較演算子を示します。

演算子	意味
=	等しい。
>	より大きい。
<	より小さい。
>=	以上。
<=	以下。
<>	次に等しくない。
! =	次に等しくない。
^=	次に等しくない。

数値を比較する場合、または特定のフィールドのプライマリキーに対するソート順に基づく範囲の行を返す場合、より大きい（>）演算子および、より小さい（<）演算子を使用します。

式中で比較演算子を使う場合は、オペランドは同じデータ型でなければなりません。例えば、123.4 > '123'という式は無効です。この式では小数と文字列を比較しているからです。式 123.4 > 123 と式'a' != 'b'は、オペランドが同じデータ型なので、有効です。

値を NULL 値と比較すると、結果は NULL になります。

フィルタ条件が NULL と評価された場合、データ統合は NULL を返します。

階層の比較

等しい（=）と等しくない（!=）の演算子を使用して、2 つの配列または 2 つの構造体を比較できます。

配列の比較

次の条件に該当する場合、2 つの配列は同等です。

- 配列要素が同じデータ型である。
- 配列が同じサイズである。
- 各インデックスの要素が同じである。

例えば、次のような配列があるとします。

```
A = [1, 2, 3]
B = [1, 2, 3]
```

次の比較を行うことができます。

```
A = B
```

どちらの配列も整数を含み、同じサイズで、各インデックスの要素は、A[0]=B[0]、A[1]=B[1]、および A[2]=B[2] のように同じです。戻り値は TRUE (1)です。

構造体の比較

2 つの構造体は、対応する構造体要素が次の条件に該当する場合、同等です。

- 要素が同じデータ型である。
- 要素は同じデータを保持している。

注: 2 つの構造体は、要素の名前が異なっていても同等である。

例えば、次のような構造体があるとします。

```
struct1 {
    name:'Paul'
    zip:10004
}

struct2 {
    firstname:'Paul'
    zip1:10004
}
```

次の比較を行うことができます。

```
struct1 = struct2
```

対応する要素は同じデータ型であり、同じデータを保持するため、戻り値は TRUE (1)です。

論理演算子

論理演算子は、数値データの操作に使用します。数値を返す式は、ゼロ以外の値であれば TRUE、ゼロの場合は FALSE、NULL の場合は NULL と評価されます。

以下の表に、トランスフォーメーション言語の論理演算子を示します。

演算子	意味
NOT	式の結果を否定します。たとえば、式の評価結果が TRUE であれば、演算子 NOT は FALSE を返します。式の評価結果が FALSE であれば、NOT は TRUE を返します。
AND	2 つの条件を結合して、両方の評価結果が TRUE であれば TRUE を返します。いずれか一方の条件が TRUE でない場合は、FALSE を返します。
OR	2 つの条件を結合して、いずれか一方の評価結果が TRUE であれば TRUE を返します。条件が両方とも TRUE でない場合は、FALSE を返します。

NULL

NULL 値とブール式を結合する式は ANSI に準拠した結果を生成します。

例えば、データ統合は次の結果を生成します。

- NULL AND TRUE = NULL
- NULL AND FALSE = FALSE

第 4 章

日付

この章では、以下の項目について説明します。

日付の概要

日付関数を使用することで、日付を丸める、切り詰める、比較する、日付の一部を抽出する、日付に算術演算を行う、などの操作を実行できます。日付関数には、日付データ型を持つ任意の値を渡すことができます。

日付変数を使用して、データ統合をホストするコンピュータの現在の日付またはセッションの開始時刻を取得します。

また、トランスフォーメーション言語には次の 3 種類の形式文字列があります。

日付フォーマット文字列

日付関数で日付の要素を指定するために使用します。

TO_CHAR **フォーマット文字列**

戻り文字列の形式を指定するために使用します。

TO_DATE **および** IS_DATE **フォーマット文字列**

日付に変換する文字列やテストする文字列の形式を指定するために使用します。

Date/Time データ型

トランスフォーメーション言語では、汎用データ型を使って各種のソースからのデータを変換します。そのようなトランスフォーメーションのデータ型の 1 つとして、Date/Time データ型があります。データ統合は、日付をバイナリ形式で内部的に格納します。

日付関数は、Date/Time 値のみを取ります。日付関数に文字列を渡すには、まず TO_DATE を使って文字を Date/Time 値に変換します。例えば、次の式は文字列フィールドを Date/Time 値に変換してから、各日付の月の数値に 1 を加えます。

```
ADD_TO_DATE( TO_DATE( STRING_PORT, 'MM/DD/RR'), 'MM', 1 )
```

注: データ統合は、西暦 1753 年から西暦 9999 年までの日付をサポートします。

ミリ秒

データ統合は、秒までの日時値をサポートします。

ミリ秒を含む日時値をインポートすると、データ統合では秒単位で切り詰められます。ミリ秒をサポートするターゲットカラムに日時値を書き込んだ場合、データ統合では日付のミリ秒の部分に 0 が挿入されます。

ユリウス日、修正ユリウス日、およびグレゴリオ暦

データ統合は、グレゴリオ暦システムの日付のみをサポートします。それ以外の方式で表現された日付はサポートされていません。

注: ユリウス暦の日付はユリウス*日*と呼ばれ、データ統合ではサポートされていません。この用語をユリウス*日*や修正ユリウス日と混同しないように注意する必要があります。

トランスフォーメーション言語には、J フォーマット文字列を使用して修正ユリウス日（MJD）形式を処理する機能があります。

ある日付に対する MJD は、紀元前 4713 年 1 月 1 日深夜 00 時 00 分 00 秒を起点とした日数で表されます。定義により、MJD の時間部分は 24 時間の一部を表す 10 進数として表現されます。J フォーマット文字列では、この時間部分は変換されません。

例えば、次の TO_DATE 式は、SHIP_DATE_MJD_STRING フィールドの文字列をデフォルト日付形式の値に変換します。

```
TO_DATE (SHIP_DATE_MJD_STR, 'J')
```

SHIP_DATE_MJD_STR	RETURN_VALUE
2451544	Dec 31 1999 00:00:00
2415021	Jan 1 1900 00:00:00

J フォーマット文字列には日付の時間部分が含まれないため、戻り値では時間が 00:00:00 に設定されています。

J フォーマット文字列を TO_CHAR 式で使用することもできます。たとえば、TO_CHAR 式で J フォーマット文字列を使用して、日付値を MJD 値の文字列に変換できます。以下に例を示します。

```
TO_CHAR(SHIP_DATE, 'J')
```

SHIP_DATE	RETURN_VALUE
Dec 31 1999 23:59:59	2451544
Jan 1 1900 01:02:03	2415021

注: データ統合は、TO_CHAR 式の日付の時刻部分を無視します。

2000 年の日付

トランスフォーメーション言語の日付関数は、すべて西暦 2000 年問題に対応しています。データ統合は、西暦 1753 年から西暦 9999 年までの日付をサポートします。

RR 形式文字列

トランスフォーメーション言語では、2 桁の年を含む文字列を日付に変換するための RR フォーマット文字列が提供されています。TO_DATE で RR フォーマット文字列を使用することにより、MM/DD/RR 形式の文字列を

日付に変換できます。RR フォーマット文字列による変換結果は、現在の年が何年であるかによって異なります。

現在の年が 0〜49 の場合

現在の年が 0〜49 の間（例: 2003 年）で、変換元の文字列の年が 0〜49 である場合、データ統合は、変換元の文字列の 2 桁の年を現在の世紀に足した値を返します。変換元の文字列の年が 50〜99 の間である場合、データ統合は、変換元の文字列の 2 桁の年を前の世紀に足した値を返します。

現在の年が 50〜99 の場合

現在の年が 50〜99 の間（例: 1998 年）で、変換元の文字列の年が 0〜49 である場合、データ統合は、変換元の文字列の 2 桁の年を次の世紀に足した値を返します。元の文字列の年が 50〜99 の間である場合、データ統合は、指定された 2 桁の年を現在の世紀に足した値を返します。

以下の表に、RR フォーマット文字列による日付への変換方法をまとめて示します。

現在の年	変換元の年	RR フォーマット文字列の戻り値
0〜49	0〜49	現在の世紀
0〜49	50〜99	前の世紀
50〜99	0〜49	次の世紀
50〜99	50〜99	現在の世紀

RR の例

次の式は、現在の年が 1950 年から 2049 年までの間は同じ結果を返します。

```
TO_DATE( ORDER_DATE, 'MM/DD/RR' )
```

ORDER_DATE	RETURN_VALUE
'04/12/98'	04/12/1998 00:00:00
'11/09/01'	11/09/2001 00:00:00

フォーマット文字列 YY と RR の違い

トランスフォーメーション言語には YY フォーマット文字列もあります。RR も YY もともに 2 桁の年を指定するフォーマット文字列です。YY と RR は、TO_DATE 以外の日付関数で使用した場合はすべて同じ結果を返します。

TO_DATE 式では、RR の場合と YY の場合で結果が異なります。

以下の表に、各フォーマット文字列が返す結果の違いを示します。

文字列	現在の年	TO_DATE(String、'MM/DD/RR')	TO_DATE(String、'MM/DD/YY')
04/12/98	1998	04/12/1998 00:00:00	04/12/1998 00:00:00
11/09/01	1998	11/09/2001 00:00:00	11/09/1901 00:00:00

文字列	現在の年	TO_DATE(String、'MM/DD/RR')	TO_DATE(String、'MM/DD/YY')
04/12/98	2003	04/12/1998 00:00:00	04/12/2098 00:00:00
11/09/01	2003	11/09/2001 00:00:00	11/09/2001 00:00:00

2000 年以降の日付については、YY を使用した場合に得られる値は RR の場合に比べて有用ではありません。21 世紀の日付については、RR フォーマット文字列を使用することを推奨します。

データベースの日付

日付の形式はデータベースごと、あるいはアプリケーションごとに異なりますが、データ統合は、Date データ型を持つすべての日付を読み込むことができます。

一般に、データベースに格納される日付には日付と時刻の値が含まれています。日付には月、日、年が含まれ、時間には場合により時、分、秒が含まれます。これらの Date/Time データはどの日付関数にも渡すことができます。

フラットファイルの日付

トランスフォーメーション言語では、TO_DATE 関数を使って文字列を Date/Time 値に変換することができます。また、TO_DATE で変換する前に、IS_DATE 関数を使って文字列が正しい日付かどうかを確認することもできます。

注: トランスフォーメーション言語の日付関数は、日付値のみを取ります。日付関数に文字列を渡したい場合は、まず TO_DATE 関数を使って文字列をトランスフォーメーションの Date/Time データ型に変更します。

デフォルトの日付形式

このアプリケーションでは、日付を表す文字列を格納および操作する場合、デフォルトの日付形式を使用します。データ統合は日付をバイナリ形式で格納するため、データ統合は特定の状況ではデフォルトの日付形式のみを使用します。

次の操作を実行した場合、データ統合では、デフォルトの日付形式のみが使用されます。

Date/Time **フィールドを文字列フィールドに接続して日付を文字列に変換した場合。**

アプリケーションでは、日付がデフォルトの形式（MM/DD/YYYY HH24:MI:SS）の文字列に変換されます。

文字列フィールドを Date/Time **フィールドに接続して文字列を日付に変換した場合。**

アプリケーションが予期する文字列の値は、デフォルトの日付形式（MM/DD/YYYY HH24:MI:SS）です。入力値がこの形式に一致しない場合、または無効な日付の場合、データ統合はその行をスキップします。文字列がデフォルトの日付形式である場合、データ統合は文字列を日付値に変換します。

TO_CHAR(date, [format_string])**を使用して日付を文字列に変換した場合。**

フォーマット文字列を省略すると、データ統合はデフォルトの日付形式（MM/DD/YYYY HH24:MI:SS）の文字列を返します。フォーマット文字列を指定した場合、データ統合は指定された形式で文字列を返します。

TO_DATE(date, [format_string])**を使用して文字列を日付に変換した場合。**

フォーマット文字列が省略されている場合、データ統合が予期する文字列は、デフォルトの日付形式（MM/DD/YYYY HH24:MI:SS）です。フォーマット文字列が指定されている場合、データ統合が予期する文字列は、指定された形式です。

デフォルト日付形式の MM/DD/YYYY HH24:MI:SS には、以下のものが含まれます。

- 月（1 月 = 01、9 月 = 09）
- 日
- 年（1998 のような 4 桁表記）
- 時（24 時間形式、たとえば 12:00:00AM = 0, 1:00:00AM = 1, 12:00:00PM = 12, 11:00:00PM = 23）
- 分
- 秒

日付フォーマット文字列

入力された日付は、フォーマット文字列と日付関数の組み合わせによって評価できます。日付フォーマット文字列は国際化されていないため、下の表に示す定義済みの形式で入力しなければなりません。

以下の表に、日付の一部を指定するフォーマット文字列のリストを示します。

フォーマット文字列	説明
D, DD, DDD, DAY, DY, J	日（01〜31）。これらのフォーマット文字列は、日付の日の部分全体を指定するために使用します。たとえば、日付関数に「12-APR-1997」を渡す場合、これらのフォーマット文字列を使って「12」の部分を指定します。
HH, HH12, HH24	時（0〜23）。ゼロは午前（深夜）12 時を表します。これらのフォーマット文字列は、日付の時の部分全体を指定するために使用します。たとえば、「12-APR-1997 2:01:32 PM」の日付を渡す場合、HH、HH12、または HH24 を使用して日付の時の部分を指定します。
MI	分（0〜59）。
MM, MON, MONTH	月（01〜12）。これらのフォーマット文字列は、日付の月の部分全体を指定するために使用します。たとえば、日付関数に「12-APR-1997」を渡す場合、MM、MON、または MONTH を使って「APR」の部分を指定します。
MS	ミリ秒（0-999）
NS	ナノ秒（0-999999999）
SS , SSSS	秒（0〜59）。
US	マイクロ秒（0-999999）
Y, YY, YYY, YYYY , RR	年（1753〜9999）。これらのフォーマット文字列は、日付の年の部分全体を指定するために使用します。たとえば、日付関数に「12-APR-1997」を渡す場合、Y、YY、YYY、または YYYY を使って「1997」の部分を指定します。

注: フォーマット文字列は大文字と小文字を区別しません。フォーマット文字列は必ず一重引用符で囲んで指定します。

以下の表に、入力日付を評価する場合に日付形式文字列を使用する日付関数を示します。

機能	説明
ADD_TO_DATE	日付の中で変更したい部分。
DATE_DIFF	日付の中で、2 つの日付の差を計算するために使用したい部分。
GET_DATE_PART	日付の中で返したい部分。この関数は、デフォルトの日付形式に基づいて整数値を返します。
IS_DATE	確認対象の日付。
ROUND (Dates)	日付の中で丸めたい部分。
SET_DATE_PART	日付の中で変更したい部分。
Systimestamp	タイムスタンプ精度。
TO_CHAR (Dates)	文字列。
TO_DATE	文字列。
TRUNC (Dates)	日付の中で切り捨てたい部分。

TO_CHAR フォーマット文字列

TO_CHAR 関数は、Date/Time データ型を、指定した形式の文字列に変換します。日付の全体、または日付の一部を文字列に変換できます。TO_CHAR を使って日付を文字列に変換することにより、レポート用に日付の形式を変更することができます。

一般に、TO_CHAR はターゲットがフラットファイルの場合か、Date/Time データ型をサポートしないデータベースの場合に使用します。

以下の表に、TO_CHAR 関数で使用される日付のフォーマット文字列のリストを示します。

フォーマット文字列	説明
AM、A.M.、PM、P.M.	午前または午後。これらのいずれかの形式文字列を使用して、時刻が午前か午後かを指定します。AM と A.M.、PM と P.M.はそれぞれ同じ値を返します。
D	曜日（1～7）。日曜日が 1 となります。
DD	日（01～31）。
DDD	1 年における通算日（001～366）。366 はうるう年の場合です。
DAY	曜日の英語名（9 文字まで）。例：Wednesday。
DY	曜日の英語名の 3 文字の略称。例：Wed。

フォーマット文字列	説明
HH, HH12	時（01〜12）。
HH24	時（00〜23）。00 は午前（深夜）12 時を表します。
J	修正ユリウス日。西暦の日付を修正ユリウス日の値に等価な文字列に変換します。修正ユリウス日は紀元前 4713 年 1 月 1 日 00 時 00 分 00 秒から数えた日付です。日付の時刻部分は無視されます。 例えば、TO_CHAR(SHIP_DATE, 'J')の式は、「Dec 31 1999 23:59:59」を「2451544」の文字列に変換します。
MI	分（00〜59）。
MM	月（01〜12）。
MONTH	月の英語名（9 文字まで）。例：January。
MON	月の英語名の 3 文字の略称。例：Jan。
Q	四半期（1〜4）。1 月から 3 月までが 1 となります。
RR	年の下 2 桁。関数はその前の桁を削除します。たとえば、'RR'を使用して年 1997 を渡すと、TO_CHAR は 97 を返します。 TO_CHAR と組み合わせて使用すると、'RR'は'YY'と同じ結果を生成し、'YY'を置き換えることもできます。ただし、TO_DATE と組み合わせて使用した場合、'RR'は直近の世紀を計算し、年の最初の 2 桁を表示します。
SS	秒（00〜59）。
SSSSS	深夜 0 時からの通算秒（00000〜86399）。TO_CHAR 式で SSSSS が使用されている場合、データ統合は、日付の時刻部分のみを評価します。 例えば、式 TO_CHAR(SHIP_DATE, 'MM/DD/YYYY SSSSS')は「12/31/1999 01:02:03」を「12/31/1999 03783」に変換します。
Y	年の下 1 桁。関数はその前の桁を削除します。 たとえば、'Y'を使用して年 1997 を渡すと、TO_CHAR は 7 を返します。
YY	年の下 2 桁。関数はその前の桁を削除します。 例えば、'YY'を使用して年 1997 を渡すと、TO_CHAR は 97 を返します。
YYY	年の下 3 桁。関数はその前の桁を削除します。 例えば、'YYY'を使用して年 1997 を渡すと、TO_CHAR は 997 を返します。
YYYY	年の部分全体。たとえば、'YYYY'を使用して年 1997 を渡すと、TO_CHAR は 1997 を返します。
W	月の何週目か（1〜5）。週 1 は月の 1 日から 7 日まで、週 2 は 8 日から 14 日までです。たとえば、Feb 1 は 2 月の第 1 週を示します。
WW	年の何週目か（01〜53）。週 01 は 1 月 1 日から 1 月 7 日まで、週 02 は 1 月 8 日から 1 月 14 日まで、というようになります。

フォーマット文字列	説明
- / .; :	出力に表示する区切り文字。これらの記号を使って日付の各要素間を区切ることができます。たとえば、次のようにピリオドを使って日付の要素を区切る式を作成できます。TO_CHAR(DATES, 'MM.DD.YYYY')
"文字列"	出力に表示する文字列。例えば、TO_CHAR(DATES, 'MM/DD/YYYY "Sales Were Up"')という式を使用して、日付として 1997 年 4 月 1 日を渡す場合、関数は文字列'04/01/1997 Sales Were Up'を返します。リポジトリのコードページで有効なマルチバイト文字も入力できます。
“ “	二重引用符は、D"D""DDD のように、意味が不明確なフォーマット文字列を区切るために使用します。空の二重引用符は出力には表示されません。

注: フォーマット文字列は大文字と小文字を区別しません。フォーマット文字列は必ず一重引用符で囲んで指定します。

例

J、SSSSS、RR、および YY のフォーマット文字列を使った例を以下に示します。これ以外の例については、各関数の説明を参照してください。

注: このアプリケーションは、TO_CHAR 式の中の日付の時刻部分を無視します。

J 形式文字列

TO_CHAR 式で J フォーマット文字列を使用して、日付値を MJD 値の文字列に変換します。以下に例を示します。

```
TO_CHAR(SHIP_DATE, 'J')
```

SHIP_DATE	RETURN_VALUE
Dec 31 1999 23:59:59	2451544
Jan 1 1900 01:02:03	2415021

SSSSS 形式文字列

SSSSS フォーマット文字列を TO_CHAR 式で使用することもできます。たとえば、次の式は SHIP_DATE ポートの日付を深夜 0 時からの通算秒を示す文字列に変換します。

```
TO_CHAR( SHIP_DATE, 'SSSSS')
```

SHIP_DATE	RETURN_VALUE
12/31/1999 01:02:03	3783
09/15/1996 23:59:59	86399

RR 形式文字列

次の式は、日付を MM/DD/YY 形式の文字列に変換します。

```
TO_CHAR( SHIP_DATE, 'MM/DD/RR')
```

SHIP_DATE	RETURN_VALUE
12/31/1999 01:02:03	12/31/99
09/15/1996 23:59:59	09/15/96
05/17/2003 12:13:14	05/17/03

YY 形式文字列

TO_CHAR 式では、YY フォーマット文字列は RR フォーマット文字列と同じ結果を返します。次の式は、日付を MM/DD/YY 形式の文字列に変換します。

```
TO_CHAR( SHIP_DATE, 'MM/DD/YY')
```

SHIP_DATE	RETURN_VALUE
12/31/1999 01:02:03	12/31/99
09/15/1996 23:59:59	09/15/96
05/17/2003 12:13:14	05/17/03

TO_DATE および IS_DATE フォーマット文字列

TO_DATE 関数は、指定した形式の文字列を Date/Time 値に変換します。一般に、TO_DATE はフラットファイルからの文字列を Date/Time 値に変換するために使用します。TO_DATE フォーマット文字列は国際化されていないため、下の表に示す定義済みの形式で入力しなければなりません。

注: TO_DATE と IS_DATE では、同じ種類のフォーマット文字列を使用します。

TO_DATE 式を作成する場合、変換元の文字列の日付の各部分に対してフォーマット文字列を使用します。ソース文字列フォーマットおよびフォーマット文字列は、日付区切り文字を含め、必ず一致している必要があります。一致しない部分がある場合、データ統合は、文字列を変換せずにその行をスキップします。フォーマット文字列を省略する場合は、変換元の文字列がデフォルトの日付形式（MM/DD/YYYY HH24:MI:SS）になっていなければなりません。

IS_DATE は、値が正しい日付であるかどうかを検査する関数です。正しい日付とは、デフォルトの日付形式（MM/DD/YYYY HH24:MI:SS）で有効な日付を表している文字列を意味します。テストする文字列がこの日付形式ではない場合、以下の表に記載されたフォーマット文字列を使用して日付の形式を指定します。文字列が指定されたフォーマット文字列に一致しない場合、または正しい日付でない場合には、関数は FALSE（0）を返します。文字列がフォーマット文字列に一致し、有効な日付である場合には、関数は TRUE（1）を返します。IS_DATE フォーマット文字列は国際化されていないため、下の表に示す定義済みの形式で入力しなければなりません。

以下の表に、TO_DATE および IS_DATE 関数で使用されるフォーマット文字列のリストを示します。

フォーマット文字列	説明
AM, a.m., PM, p.m.	午前または午後。これらのいずれかの形式文字列を使用して、時刻が午前か午後かを指定します。AM と A.M.、PM と P.M.はそれぞれ同じ値を返します。
DAY	曜日の英語名（9 文字まで）。例：Wednesday。DAY 形式文字列は大文字と小文字を区別しません。
DD	日（1〜31）。
DDD	1 年における通算日（001〜366）。366 はうるう年の場合です。
DY	曜日の英語名の 3 文字の略称。例：Wed。DY 形式文字列は大文字と小文字を区別しません。
HH, HH12	時（1〜12）。
HH24	時（0〜23）。ゼロは午前（深夜）12 時を表します。
J	修正ユリウス日。MJD 形式の文字列を日付値に変換します。元の文字列の時刻部分を無視し、すべての日付に 00:00:00 の時刻を割り当てます。 例えば、TO_DATE('2451544', 'J')の式は「2451544」を「Dec 31 1999 00:00:00」に変換します。
MI	分（0〜59）。
MM	月（1〜12）。
MONTH	月の英語名（9 文字まで）。例：August。大文字と小文字は区別されません。
MON	月の英語名の 3 文字の略称。例：Aug。大文字と小文字は区別されません。
MS	ミリ秒（0-999）
NS	ナノ秒。TO_DATE および IS_DATE は、'NS'のフォーマットトークンを使用してサブ秒をサポートできます。単位はナノ秒になります。サブ秒の部分がミリ秒の場合でも、次の例の場合は、ゼロを３つ追加して使用できます。 TO_DATE('2005-05-02 09:23:34 123000', 'YYYY-MM-DD HH24:MI:SS NS') TO_DATE('2005-05-02 09:23:34.123' || '000', 'YYYY-MM-DD HH24:MI:SS.NS') TO_DATE('2005-05-02 09:23:34123000', 'YYYY-MM-DD HH24:MI:SSNS')
RR	4 桁の年（1998、2034 など）。変換元の文字列が 2 桁の年を含む場合に使用します。 TO_DATE で使用すると、2 桁の年が 4 桁の年に変換されます。 - 現在の年が 50 - 99 の場合。現在の年が 50〜99 の間（例: 1998 年）で、変換元の文字列に含まれる年の値が 0〜49 の間である場合、データ統合は、次の世紀に変換元の文字列の 2 桁の年を足した値を返します。変換元の文字列に含まれる年の値が 50〜99 の間である場合、データ統合は、現在の世紀に指定された 2 桁の年を足した値を返します。 - 現在の年が 0 - 49 の場合。現在の年が 0〜49 の間（例: 2003 年）で、変換元の文字列の年が 0〜49 である場合、データ統合は、変換元の文字列の 2 桁の年を現在の世紀に足した値を返します。変換元の文字列の年が 50〜99 の間である場合、データ統合は、変換元の文字列の 2 桁の年を前の世紀に足した値を返します。
SS	秒（0〜59）。

フォーマット文字列	説明
SSSSS	深夜 0 時からの通算秒。TO_DATE 式で SSSSS が使用されている場合、データ統合は、日付の時刻部分のみを評価します。 例えば、式 TO_DATE(DATE_STR, 'MM/DD/YYYY SSSSS')は「12/31/1999 3783」を「12/31/1999 01:02:03」に変換します。
US	マイクロ秒（0-999999）
Y	Secure Agent が動作しているマシン上での現在の年の下 1 桁を、文字列値で置き換えます。
YY	Secure Agent が動作しているマシン上での現在の年の下 2 桁を、文字列値で置き換えます。
YYY	Secure Agent が動作しているマシン上での現在の年の下 3 桁を、文字列値で置き換えます。
YYYY	年の全 4 桁。2 桁の年を渡すときには、この形式文字列は使用しないでください。代わりに、RR または YY 形式を使用します。

要件

データ統合が予期する形式は、次の条件を満たす TO_DATE 文字列です。

- フォーマット文字列は、日付区切り文字を含め、TO_DATE 文字列の形式と一致しなければなりません。一致しない場合、データ統合は不正確な値を返すか行をスキップする可能性があります。例えば、2020 年 5 月 12 日を表す文字列'20200512'を TO_DATE に渡す場合は、YYYYMMDD フォーマット文字列を指定する必要があります。フォーマット文字列が指定されていない場合、データ統合が予期する文字列は、デフォルトの日付形式（MM/DD/YYYY HH24:MI:SS）です。同様に、フォーマット文字列に一致しない文字列を渡すと、データ統合はエラーを返して行をスキップします。例えば、TO_DATE に 2020120 の文字列が渡され、YYYYMMDD フォーマット文字列が指定されていると、文字列がフォーマット文字列と一致しないため、データ統合はエラーを返して行をスキップします。
- フォーマット文字列は必ず一重引用符で囲んで指定します。

  ヒント: デフォルトでは、データ統合はフォーマット文字列 MM/DD/YYYY HH24:MI:SS を使用します。フォーマット文字列は大文字と小文字を区別しません。

例

J、RR、および SSSSS のフォーマット文字列を使った例を以下に示します。これ以外の例については、各関数の説明を参照してください。

J 形式文字列

次の式は、SHIP_DATE_MJD_STRING フィールドの文字列をデフォルト日付形式の値に変換します。

```
TO_DATE (SHIP_DATE_MJD_STR, 'J')
```

SHIP_DATE_MJD_STR	RETURN_VALUE
2451544	Dec 31 1999 00:00:00
2415021	Jan 1 1900 00:00:00

J フォーマット文字列には日付の時間部分が含まれないため、戻り値では時間が 00:00:00 に設定されています。

RR 形式文字列

次の式は、文字列を 4 桁形式の年に変換します。現在の年は 1998 です。

```
TO_DATE( DATE_STR, 'MM/DD/RR')
```

DATE_STR	RETURN VALUE
04/01/98	04/01/1998 00:00:00
08/17/05	08/17/2005 00:00:00

YY 形式文字列

次の式は、文字列を 4 桁形式の年に変換します。現在の年は 1998 です。

```
TO_DATE( DATE_STR, 'MM/DD/YY')
```

DATE_STR	RETURN VALUE
04/01/98	04/01/1998 00:00:00
08/17/05	08/17/1905 00:00:00

注: 2 行目については、RR は 2005 年を返しますが、YY は 1905 年を返します。

SSSSS 形式文字列

次の式は、深夜からの通算秒を含む文字列を日付値に変換します。

```
TO_DATE( DATE_STR, 'MM/DD/YYYY SSSSS')
```

DATE_STR	RETURN_VALUE
12/31/1999 3783	12/31/1999 01:02:03
09/15/1996 86399	09/15/1996 23:59:59

日付の算術演算について

トランスフォーメーション言語には以下の組込み日付関数が用意されており、Date/Time 値に対して算術演算を実行できます。

ADD_TO_DATE

日付の中の特定の部分を加算または減算します。

DATE_DIFF

2 つの日付の差を求めます。

SET_DATE_PART

日付の一部分を変更します。

数値算術演算子（+および-など）を使って日付の加算や減算を行うことはできません。

データ統合は、うるう年を認識し、西暦 1753 年 1 月 1 日 00:00:00〜西暦 9999 年 12 月 31 日 23:59:59 の間の日付を受け入れます。

注: データ統合は、Date/Time 型を使用して日付値を指定します。Date/Time 値に対しては日付関数のみが使用できます。

第 5 章

関数

この章では、トランスフォーメーション言語の関数をアルファベット順に説明します。各関数の説明は、以下の項目からなっています。

- 構文
- 戻り値
- 例

関数の概要

集計関数はマッピングタスクで使用できます。マッピングタスクと同期タスクでは、他のすべての関数を使用できます。

トランスフォーメーション言語には次の関数カテゴリが用意されています。

- 集計
- 変換
- データクレンジング
- 日付
- エンコーディング
- 財務
- 水平拡張
- 数値
- 科学
- 特殊
- 文字列
- テスト
- Window

Aggregate 関数

集計関数は、選択した複数のフィールドの非 NULL 値を要約した値を返します。

集計関数を使用すると、次のタスクを実行できます。

- グループ内のすべての行に関して、ある 1 つの値を計算する。
- アグリゲータオブジェクトで各グループに対して 1 つの値を返す。
- 選択したフィールドの特定の行に対して値を計算するようなフィルタを適用する。
- 演算子を使って関数内で算術演算を実行する。
- 同じソース列から得られた複数の集計値を 1 回のパスで計算する。

マッピングタスクのみで集計関数を使用する。

トランスフォーメーション言語には、以下の集計関数が用意されています。

- AVG
- COUNT
- FIRST
- LAST
- MAX (Date)
- MAX (Number)
- MAX (String)
- MEDIAN
- MIN (Date)
- MIN (Number)
- MIN (String)
- PERCENTILE
- STDDEV
- SUM
- VARIANCE

集計関数は、アグリゲータオブジェクトでのみ使用できます。他の集計関数の中にネストできる集計関数は 1 つだけです。データ統合は、最も内側の集計関数式を評価し、その結果を外側の集計関数式の評価に使用します。詳細モードでは集計関数をネストできません。

例えば、次のように、ID でグループ分けして、2 つの集計関数をネストしたアグリゲータオブジェクトを設定することができます。

```
SUM( AVG( earnings ) )
```

このとき、データセットには以下の値が格納されています：

ID	EARNINGS
1	32
1	45
1	100

ID	EARNINGS
2	65
2	75
2	76
3	21
3	45
3	99

戻り値は 186 です。データ統合は、ID でグループ分けを行い、AVG 式を評価して 3 つの値を返します。次に、その値を SUM 関数で追加して結果を出します。

フィルタ条件

フィルタ条件を使用して、検索によって返される行を制限します。

フィルタは、検索によって返される行を制限します。すべての集計関数および、MOVINGAVG、MOVINGSUM の関数に、フィルタ条件を適用できます。フィルタ条件の値は TRUE、FALSE、または NULL でなければなりません。フィルタ条件の値が NULL または FALSE である場合、データ統合はその行を選択しません。

有効なトランスフォーメーション式を必要に応じて入力できます。たとえば、次の式は給与が$50,000 を超える従業員すべてに対して、給与のメジアンを計算します。

```
MEDIAN( SALARY, SALARY > 50000 )
```

他の数値をフィルタ条件として使用することもできます。例えば、MEDIAN 関数の完全な構文として、次のように入力することができます。これには、数値フィールドが含まれています。

```
MEDIAN( PRICE, QUANTITY > 0 )
```

いずれの場合も、データ統合はフィルタ条件に対して小数値を整数に丸めます（例えば、1.5 は 2、1.2 は 1、0.35 は 0 になります）。値が 0 に丸められた場合、フィルタ条件は FALSE を返します。値を丸めたくない場合には、TRUNC 関数で値を切り捨てて整数にします。

```
MEDIAN( PRICE, TRUNC( QUANTITY ) > 0 )
```

フィルタ条件を省略すると、関数はフィールド内のすべての行を選択します。

変換関数

トランスフォーメーション言語には、以下の変換関数が用意されています。

- TO_BIGINT
- TO_CHAR(Date)
- TO_CHAR(Number)
- TO_DATE
- TO_DECIMAL
- TO_FLOAT
- TO_INTEGER

データクレンジング関数

トランスフォーメーション言語には、データエラーをなくすための関数群があります。データクレンジング関数を使用して、以下のタスクを完了できます。

- ソース値のテスト。
- ソース値のデータ型の変換。
- 文字列値の切り詰め。
- 文字列内の文字の上書き。
- 文字列のエンコード。
- 正規表現パターンのマッチ。

トランスフォーメーション言語には、以下のデータクレンジング関数が用意されています。

- BETWEEN
- GREATEST
- IN
- INSTR
- IS_DATE
- IS_NUMBER
- IS_SPACES
- ISNULL
- LEAST
- LTRIM
- METAPHONE
- REG_EXTRACT
- REG_MATCH
- REG_REPLACE
- REPLACECHR
- REPLACESTR
- RTRIM
- SOUNDEX
- SUBSTR
- TO_BIGINT
- TO_CHAR
- TO_DATE
- TO_DECIMAL
- TO_FLOAT
- TO_INTEGER

日付関数

トランスフォーメーション言語にはいくつかの日付関数が用意され、日付を丸めたり、切り捨てたり、比較したり、日付の一部を抽出したり、日付に算術演算を行ったりできます。

どの日付関数にも、日付データ型を持つ任意の値を渡すことができます。ただし、日付関数に文字列を渡したい場合は、まず TO_DATE 関数を使って文字列をトランスフォーメーションの Date/Time データ型に変換する必要があります。

トランスフォーメーション言語には、以下の日付関数が用意されています。

- ADD_TO_DATE
- DATE_COMPARE
- DATE_DIFF
- GET_DATE_PART
- LAST_DAY
- MAKE_DATE_TIME
- ROUND
- SET_DATE_PART
- Systimestamp
- TRUNC

一部の日付関数には *format* 引数が含まれています。この引数には、トランスフォーメーション言語のフォーマット文字列のいずれかを指定します。日付フォーマット文字列は国際化されていません。

トランスフォーメーションの Date/Time データ型では、ミリ秒をサポートしていません。したがって、ミリ秒を含む日付を渡した場合、データ統合では、日付のミリ秒部分が切り詰められます。

エンコード関数

トランスフォーメーション言語には、データのエンコード、暗号化、圧縮、およびチェックサムのための以下の関数が用意されています。

- AES_DECRYPT
- AES_ENCRYPT
- AES_GCM_DECRYPT
- AES_GCM_ENCRYPT
- COMPRESS
- CRC32
- DEC_BASE64
- DECOMPRESS
- ENC_BASE64
- MD5

財務関数

トランスフォーメーション言語には、以下の財務関数が用意されています。

- FV

- NPER
- PMT
- PV
- RATE

水平拡張関数

水平拡張関数を使用して、水平マクロ式を作成します。

水平拡張関数には、%OPR_<function_type>%の命名規則を使用します。

水平拡張関数では、丸カッコではなく角かっこ（[ ]）を使用します。

トランスフォーメーション言語には、次の水平拡張関数が用意されています。

- %OPR_CONCAT%
- %OPR_CONCATDELIM%
- %OPR_IIF%
- %OPR_SUM%

数値関数

トランスフォーメーション言語には、以下の数値関数が用意されています。

- ABS
- CEIL
- CONV
- CUME
- EXP
- FLOOR
- LN
- LOG
- MOD
- MOVINGAVG
- MOVINGSUM
- POWER
- RAND
- ROUND
- SIGN
- SQRT
- TRUNC

科学関数

トランスフォーメーション言語には、次の科学関数が用意されています。

- COS
- COSH
- SIN
- SINH
- TAN
- TANH

特殊関数

トランスフォーメーション言語には、次の特殊関数が用意されています。

- ABORT
- DECODE
- ERROR
- IIF
- SETCOUNTVARIABLE
- SETMAXVARIABLE
- SETMINVARIABLE
- SETVARIABLE

特殊関数の中に他の関数をネストすることもできます。

文字列関数

トランスフォーメーション言語には、次の文字列関数が用意されています。

- ASCII
- CHOOSE
- CHR
- CHRCODE
- CONCAT
- INDEXOF
- INITCAP
- INSTR
- LENGTH
- LOWER
- LPAD
- LTRIM
- REPLACECHR
- REPLACESTR

- REVERSE
- RPAD
- RTRIM
- SUBSTR
- UPPER

文字データを評価するため、文字列関数 LOWER、UPPER、および INITCAP では、そのタスクを実行する Secure Agent のコードページを使用します。

テスト関数

トランスフォーメーション言語には、次のテスト関数が用意されています。

- ISNULL
- IS_DATE
- IS_NUMBER
- IS_SPACES

ウィンドウ関数

詳細モードでは、現在の行に関連した一連の行で計算を実行するウィンドウ関数のグループがトランスフォーメーション言語に含まれます。この関数は、入力行ごとに 1 つの戻り値を計算します。

トランスフォーメーション言語は、以下のウィンドウ関数を生成します。

- LAG
- LEAD

ウィンドウプロパティを設定すると、式トランスフォーメーションでウィンドウ関数を使用できるようになります。ウィンドウプロパティを設定した場合、集計関数をウィンドウ関数として使用することもできます。集計関数は、ウィンドウ関数として単一の出力行に行をグループ化することはありませんが、個々の行ごとに出力値を返します。

ウィンドウ関数としての集計関数

LEAD と LAG に加えて、ウィンドウ関数として集計関数を使用することもできます。SUM や AVG などの集計関数をウィンドウ関数として使用すると、計算を実行できます。ウィンドウ関数には、特定の終了オフセットを設定できるため、ステートフル関数よりも柔軟です。

集計関数をウィンドウ関数として使用するには、ウィンドウプロパティでフレームを定義して、計算範囲を制限する必要があります。集計関数は、フレームを超えて計算を実行し、各行に 1 つの値を生成します。

例

あなたは過去 2 年間、さまざまな数量の木材を販売してきた材木営業担当者であるとします。販売数量の現在までの累計を計算します。

次の表に、各販売 ID、日付、販売数量の一覧を示します。

Sale_ID	Date	Quantity
30001	2016-08-02	10

Sale_ID	Date	Quantity
10001	2016-12-24	10
10005	2016-12-24	30
40001	2017-01-09	40
10006	2017-01-18	10
20001	2017-02-12	20

SUM 関数によって、すべての値が合計され、1 つの出力値が返されます。各行の累計を得るため、関数の境界に対してフレームを定義できます。

[ウィンドウ] タブで、次のプロパティを設定します。

- 開始オフセット: 前のすべての行
- 終了オフセット: 0
- オーダーキー: 昇順の日付

次の集計関数を定義します。

```
SUM (Quantity)
```

SUM は、現在の行の数量を、現在の行の前のすべての行の数量に加算します。この関数は、各行の累計を返します。

次の表に、各日付の累計を示します。

Sale_ID	Date	Quantity	Total
30001	2016-08-02	10	10
10001	2016-12-24	10	20
10005	2016-12-24	30	50
40001	2017-01-09	40	90
10006	2017-01-18	10	100
20001	2017-02-12	20	120

ウィンドウ関数としてのネストされた集計関数

ウィンドウ関数内にネストされた集計関数は、パーティションごとに個別の計算を実行します。

ネストされた集計関数を式トランスフォーメーションに含め、そのトランスフォーメーションをウィンドウ関数用に設定した場合、関数はパーティションごとに個別の計算を実行します。

データを P2 でパーティション化し、先行する行および後続のすべての行のフレームを指定します。ウィンドウ関数は次の計算を実行します。

1. COUNT (P1)により、各行に 1 つの値を生成します。COUNT は、パーティションの行のうち、NULL 以外の値を持つ行の数を返します。

2. その値の MEDIAN は、COUNT によって生成された値のウィンドウのメジアンを生成します。

ウィンドウ関数は次の出力を生成します。

P1	P2	Output
10	1	3
7	1	3
12	1	3
11	2	4
13	2	4
8	2	4
10	2	4

複数のウィンドウ関数を使用して、集計関数をネストできます。以下に例を示します。

```
LAG ( LEAD( MAX( FIRST ( p1 )))
```

注: 複数のウィンドウ関数 LEAD と LAG をネストすることはできますが、複数の集計関数を 1 つの集計関数内にネストすることはできません。

関数のクイックリファレンス

以下の表には、フィールド式で使用可能な関数の構文と簡単な説明が含まれます。

注: 使用できる関数はマッピングタイプによって異なります。

関数	関数タイプ	構文	説明
%OPR_CONCAT%	水平拡張	%OPR_CONCAT[*macro_input_field*]%	CONCAT 関数を使用して、式マクロ内の式を拡張し、複数のフィールドを連結します。 詳細については、「%OPR_CONCAT%」 (ページ 65)を参照してください。
%OPR_CONCATDELIM%	水平拡張	%OPR_CONCATDELIM[*macro_input_field*]%	CONCAT 関数を使用して、式マクロ内の式を拡張し、複数のフィールドを連結してカンマ区切り文字を追加します。 詳細については、「%OPR_CONCATDELIM%」 (ページ 66)を参照してください。
%OPR_IIF%	水平拡張	%OPR_IIF[*condition*, *macro_input_field* [,*value*]]%	IIF 関数を使用して、式マクロ内の式を拡張し、IIF 文のセットを評価します。 詳細については、「%OPR_IIF%」 (ページ 67)を参照してください。

関数	関数タイプ	構文	説明
%OPR_SUM%	水平拡張	%OPR_SUM[*macro_input_field* [, *filter_condition*]]%	SUM 関数を使用して、式マクロ内の式を拡張し、すべてのフィールドの合計を返します。 詳細については、「%OPR_SUM%」 (ページ 69)を参照してください。
ABORT	特殊	ABORT(*string*)	セッションを停止して、指定されたエラーメッセージを発行します。 詳細については、「ABORT」 (ページ 70)を参照してください。
ABS	数値	ABS(*numeric_value*)	数値の絶対値を返します。 詳細については、「ABS」 (ページ 70)を参照してください。
ADD_TO_DATE	データクレンジング、日付	ADD_TO_DATE(*date*, *format*, *amount*)	Date/Time 値の一部に指定された量を加算し、指定された日付と同じ形式の日付を返します。 年が YYYY の形式で指定されていない場合、データ統合は日付を現在の世紀とみなします。 詳細については、「ADD_TO_DATE」 (ページ 71)を参照してください。
AES_DECRYPT	エンコード	AES_DECRYPT(*value*, *key*)	入力値を AES-ECB 復号化した後に、復号化された値を文字列の形式で返します。 詳細については、「AES_DECRYPT」 (ページ 74)を参照してください。
AES_ENCRYPT	エンコード	AES_ENCRYPT (*value*, *key*)	入力値を AES-ECB 暗号化した後に、バイナリデータを暗号化された形式で返します。 詳細については、「AES_ENCRYPT」 (ページ 75)を参照してください。
AES_GCM_DECRYPT	エンコード	AES_GCM_DECRYPT (*value, init_vector, key* [, *keysize*])	指定された initialization vector、key で入力値を AES-GCM 復号化した後に、プレーンテキストの復号化された値を文字列として返します。 詳細については、「AES_GCM_DECRYPT」 (ページ 76)を参照してください。
AES_GCM_ENCRYPT	エンコード	AES_GCM_ENCRYPT (*value, init_vector, key* [, *keysize*])	指定された initialization vector、key で入力値を AES-GCM 暗号化した後に、ciphertext をバイナリ値で返します。ciphertext は、暗号化されたプレーンテキストです。 詳細については、「AES_GCM_ENCRYPT」 (ページ 77)を参照してください。

関数	関数タイプ	構文	説明
ASCII	文字列	ASCII (*string*)	関数に渡された文字列の最初の文字に対応する ASCII 数値を返します。 この関数の動作は、CHRCODE 関数と同じです。既存の式で ASCII 関数を使用する場合でも正常に機能します。ただし、新しい式を作成する場合は、ASCII 関数ではなく CHRCODE 関数を使用します。 詳細については、「ASCII」 (ページ 78)を参照してください。
AVG	集計	AVG(*numeric_value* [, *filter_condition*])	行のグループのすべての値の平均を返します。 詳細については、「AVG」 (ページ 79)を参照してください。
CEIL	数値	CEIL(*numeric_value*)	指定された数値に等しいかそれより大きい最小の整数を返します。 詳細については、「CEIL」 (ページ 80)を参照してください。
CHOOSE	文字列	CHOOSE(*index, string1*, [*string2*, ..., *stringN*])	文字列のリストから指定された位置の文字列を選択します。 詳細については、「CHOOSE」 (ページ 81)を参照してください。
CHR	文字列	CHR(*numeric_value*)	指定された数値に対応する ASCII 文字を返します。 詳細については、「CHR」 (ページ 82)を参照してください。
CHRCODE	文字列	CHRCODE(*string*)	関数に渡された文字列の最初の文字に対応する ASCII 数値を返します。 この関数の動作は、ASCII 関数と同じです。 詳細については、「CHRCODE」 (ページ 83)を参照してください。
COMPRESS	エンコード	COMPRESS(*value*)	zlib 圧縮アルゴリズムを使用してデータを圧縮します。 詳細については、「COMPRESS」 (ページ 84)を参照してください。
CONCAT	文字列	CONCAT(*first_string, second_string*)	2 つの文字列を連結します。 詳細については、「CONCAT」 (ページ 84)を参照してください。
CONVERT_BASE	数値	CONVERT_BASE(*value, source_base, dest_base*)	ある基準価格の数値を別の基準価格へ変換します。 詳細については、「CONVERT_BASE」 (ページ 86)を参照してください。

関数	関数タイプ	構文	説明
COS	科学	COS(*numeric_value*)	数値（ラジアン単位）の余弦を返します。 詳細については、「COS」 (ページ 86)を参照してください。
COSH	科学	COSH(*numeric_value*)	数値（ラジアン単位）の双曲余弦を返します。 詳細については、「COSH」 (ページ 87)を参照してください。
COUNT	集計	COUNT(*value*[, *filter_condition*]) または COUNT(*[, *filter_condition*])	グループ内で NULL 以外の値を持つ行の数を返します。 詳細については、「COUNT」 (ページ 88)を参照してください。
CRC32	エンコード	CRC32(*value*)	32 ビット Cyclic Redundancy Check (CRC32)の値を返します。 詳細については、「CRC32」 (ページ 90)を参照してください。
CUME	数値	CUME(*numeric_value*[, *filter_condition*])	現在の合計を返します。 詳細については、「CUME」 (ページ 91)を参照してください。
DATE_COMPARE	データクレンジング、日付	DATE_COMPARE(*date1*, *date2*)	2 つの日付のうちどちらが早いかを示す値を返します。 詳細については、「DATE_COMPARE」 (ページ 92)を参照してください。
DATE_DIFF	データクレンジング、日付	DATE_DIFF(*date1, date2, format*)	2 つの日付の間の時間の長さを、指定した単位（年、月、日、時間、分、または秒）で返します。 詳細については、「DATE_DIFF」 (ページ 94)を参照してください。
DEC_BASE64	エンコード	DEC_BASE64(*value*)	値をデコードし、そのデータを表す Binary データの文字列を返します。 詳細については、「DEC_BASE64」 (ページ 96)を参照してください。
DECODE	特殊	DECODE(*value, first_search, first_result*[, *second_search, second_result*]…[, *default*])	指定された値をカラム内で検索します。 詳細については、「DECODE」 (ページ 96)を参照してください。
DECOMPRESS	エンコード	DECOMPRESS(*value, precision*)	zlib 圧縮アルゴリズムを使用してデータを解凍します。 詳細については、「DECOMPRESS」 (ページ 98)を参照してください。

関数	関数タイプ	構文	説明
ENC_BASE64	エンコード	ENC_BASE64(*value*)	データのエンコードは、MIME (Multipurpose Internet Mail Extensions) エンコードを使用して Binary データを文字列データに変換します。 詳細については、「ENC_BASE64」 (ページ 99)を参照してください。
ERROR	特殊	ERROR(*string*)	データ統合は行をスキップします。指定されたエラーメッセージを含む行をエラー行ファイルに書き込みます。 詳細については、「ERROR」 (ページ 100)を参照してください。
EXP	数値	EXP(*exponent*)	指定された指数に対する e (=2.71828183) の累乗を返します。 詳細については、「EXP」 (ページ 100)を参照してください。
FIRST	集計	FIRST(*value* [, *filter_condition*])	フィールドまたはグループ内で見つかった最初の値を返します。 詳細については、「FIRST」 (ページ 101)を参照してください。
FLOOR	数値	FLOOR(*numeric_value*)	指定された数値に等しいかそれより小さい最大の整数を返します。 詳細については、「FLOOR」 (ページ 103)を参照してください。
FV	財務	FV(*rate, terms, payment* [, *present value, type*])	定期的に一定額を支払い、一定の利率で利息が付く投資の将来価値を返します。 詳細については、「FV」 (ページ 104)を参照してください。
GET_DATE_PART	日付、データクレンジング	GET_DATE_PART(*date, format*)	デフォルト日付形式 MM/DD/YYYY HH24:MI:SS に基づいて、日付の中の指定された部分を整数として返します。 詳細については、「GET_DATE_PART」 (ページ 104)を参照してください。
GREATEST	データクレンジング	GREATEST(*value1*, [*value2*, ..., *valueN*,] *CaseFlag*)	入力値のリストから最大値を返します。 詳細については、「GREATEST」 (ページ 106)を参照してください。
IIF	特殊	IIF(*condition, value2* [, *value2*])	条件の結果に基づいて、指定した 2 つの値のうちの 1 つを返します。 詳細については、「IIF」 (ページ 107)を参照してください。

関数	関数タイプ	構文	説明
IN	データクレンジング	IN(*valueToSearch*, *value1*, [*value2*, ..., *valueN*,] *CaseFlag*)	入力値を値のリストとマッチングします。 詳細については、「IN 」 (ページ 109)を参照してください。
INDEXOF	文字列	INDEXOF(*valueToSearch*, *string1*, [*string2*, ..., *stringN*,] *CaseFlag*)	文字列のリストから文字列のインデックスを検索します。 詳細については、「INDEXOF」 (ページ 110)を参照してください。
INITCAP	文字列	INITCAP(*string*)	文字列の各語の最初の文字を大文字に変換して、他の文字をすべて小文字に変換します。 詳細については、「INITCAP」 (ページ 111)を参照してください。
INSTR	文字列、データクレンジング	INSTR(*string*, *search_value* [, *start* [, *occurrence*]])	文字列の中で、指定した文字が左から数えて何文字目にあるかを返します。 詳細については、「INSTR」 (ページ 112)を参照してください。
IS_DATE	データクレンジング、テスト	IS_DATE(*value*)	値が正しい日付であるかどうかを返します。 詳細については、「IS_DATE」 (ページ 114)を参照してください。
IS_NUMBER	データクレンジング、テスト	IS_NUMBER(*value*)	文字列が正しい数値であるかどうかを返します。 詳細については、「IS_NUMBER」 (ページ 116)を参照してください。
IS_SPACES	データクレンジング、テスト	IS_SPACES(*value*)	値がスペースだけで構成されているかどうかを返します。 詳細については、「IS_SPACES」 (ページ 118)を参照してください。
ISNULL	データクレンジング、テスト	ISNULL(*value*)	値が NULL であるかどうかを返します。 詳細については、「ISNULL」 (ページ 119)を参照してください。
LAG	Window	LAG(field_name, offset, default_value)	前の行の値を返します。 詳細については、「LAG」 (ページ 120)を参照してください。
LAST	集計	LAST(*value* [, *filter_condition*])	選択したフィールドの最後の行を返します。 詳細については、「LAST」 (ページ 122)を参照してください。

関数	関数タイプ	構文	説明
LAST_DAY	データクレンジング、日付	LAST_DAY(*date*)	カラム内の各日付に対して、その月の最後の日の日付を返します。 詳細については、「LAST_DAY」 (ページ 123)を参照してください。
LEAD	Window	LEAD(field_name, offset, default_value)	次の行の値を返します。 詳細については、「LEAD」 (ページ 124)を参照してください。
LEAST	データクレンジング	LEAST(*value1*, [*value2*, ..., *valueN*,] *CaseFlag*)	入力値のリストから最小値を返します。 詳細については、「LEAST」 (ページ 126)を参照してください。
LENGTH	文字列	LENGTH(*string*)	文字列内の文字数を返します。文字列の末尾の空白も含めます。 詳細については、「LENGTH」 (ページ 127)を参照してください。
LN	数値	LN(*numeric_value*)	数値の 自然対数を返します。 詳細については、「LN」 (ページ 127)を参照してください。
LOG	数値	LOG(*base, exponent*)	数値の対数を返します。 詳細については、「LOG」 (ページ 128)を参照してください。
LOWER	文字列	LOWER(*string*)	大文字の文字列を小文字に変換します。 詳細については、「LOWER」 (ページ 130)を参照してください。
LPAD	文字列	LPAD(*first_string, length* [, *second_string*])	文字列の先頭にいくつかの空白または文字を追加して、文字列を指定した長さにします。 詳細については、「LPAD」 (ページ 131)を参照してください。
LTRIM	文字列、データクレンジング	LTRIM(*string* [, *trim_set*])	文字列の先頭から空白または文字を削除します。 詳細については、「LTRIM」 (ページ 132)を参照してください。
MAKE_DATE_TIME	データクレンジング、日付	MAKE_DATE_TIME(*year, month, day, hour, minute, second*)	入力値に基づく日付と時間を返します。 詳細については、「MAKE_DATE_TIME」 (ページ 133)を参照してください。
MAX (Dates)	集計	MAX(*date* ,*filter_condition*)	フィールドまたはグループ内で見つかった最新の日付を返します。 詳細については、「MAX (Dates)」 (ページ 134)を参照してください。

関数	関数タイプ	構文	説明
MAX (Numbers)	集計	MAX(*numeric_value* [, *filter_condition*])	フィールドまたはグループ内の最大の数値を返します。 詳細については、「MAX (Numbers)」 (ページ 135)を参照してください。
MAX (String)	集計	MAX(*string* [, *filter_condition*])	フィールドまたはグループ内における最大の文字列値を返します。 詳細については、「MAX (String)」 (ページ 137)を参照してください。
MD5	エンコード	MD5(*value*)	入力値のチェックサムを計算します。この関数は、MD5（Message-Digest アルゴリズム 5）を使用しています。 詳細については、「MD5」 (ページ 138)を参照してください。
MEDIAN	集計	MEDIAN(*numeric_value* [, *filter_condition*])	選択されたフィールドのすべての値の中央値を返します。 詳細については、「MEDIAN」 (ページ 139)を参照してください。
METAPHONE	データクレンジング	METAPHONE(*string* [, *length*])	英字（A - Z）をエンコードします。 詳細については、「METAPHONE」 (ページ 140)を参照してください。
MIN (Dates)	集計	MIN(*date* ,*filter_condition*)	フィールドまたはグループ内で最も古い日付を返します。 詳細については、「MIN (Dates)」 (ページ 144)を参照してください。
MIN (numbers)	集計	MIN(*date* ,*filter_condition*)	フィールドまたはグループ内の最小の数値を返します。 詳細については、「MIN (numbers)」 (ページ 145)を参照してください。
MIN (String)	集計	MIN(*string* [, *filter_condition*])	フィールドまたはグループ内の最低の文字列値を返します。 詳細については、「MIN (String)」 (ページ 146)を参照してください。
MOD	数値	MOD(*numeric_value, divisor*)	除算の余りを返します。 詳細については、「MOD」 (ページ 148)を参照してください。
MOVINGAVG	数値	MOVINGAVG(*numeric_value, rowset* [, *filter_condition*])	指定された行のセットについて、行ごとの平均を返します。 詳細については、「MOVINGAVG」 (ページ 149)を参照してください。
MOVINGSUM	数値	MOVINGSUM(*numeric_value, rowset* [, *filter_condition*])	指定された行のセットについて、行ごとの合計を返します。 詳細については、「MOVINGSUM」 (ページ 150)を参照してください。

関数	関数タイプ	構文	説明
NPER	財務	NPER(*rate, present value, payment* [, *future value, type*])	一定の利率、支払周期、支払額に基づいて、投資の期間数を返します。 詳細については、「NPER」 (ページ 151)を参照してください。
PERCENTILE	集計	PERCENTILE(*numeric_value, percentile* [, *filter_condition*])	数値のグループ内で、与えられたパーセンタイルに入る値を計算します。 詳細については、「PERCENTILE」 (ページ 152)を参照してください。
PMT	財務	PMT(*Rate, terms, present value* [, *future value, type*])	一定の利率で定額を支払う場合の貸付の支払額を返します。 詳細については、「PMT」 (ページ 154)を参照してください。
POWER	数値	POWER(*base, exponent*)	指定された指数に基づく値の累乗を返します。 詳細については、「POWER」 (ページ 155)を参照してください。
PV	財務	PV(*Rate, terms, payment* [, *future value, type*])	投資の現在価値を返します。 詳細については、「PV」 (ページ 156)を参照してください。
RAND	数値	RAND(*seed*)	0～1 の範囲の乱数を返します。 詳細については、「RAND」 (ページ 157)を参照してください。
RATE	財務	RATE(*terms, payment, present value*[, *future value, type*])	証券ごとに期間別に得た金利を返します。 詳細については、「RATE」 (ページ 158)を参照してください。
REG_EXTRACT	データクレンジング	REG_EXTRACT(*subject, pattern, subPatternNum*)	入力値から正規表現のサブパターンを抽出します。 詳細については、「REG_EXTRACT」 (ページ 158)を参照してください。
REG_MATCH	データクレンジング	REG_MATCH(*subject, pattern*)	値が正規表現のパターンに一致するかどうかを返します。 詳細については、「REG_MATCH」 (ページ 161)を参照してください。
REG_REPLACE	データクレンジング	REG_REPLACE(subject, pattern, replace, numReplacements)	文字列内の文字を新しい文字パターンに置換します。 詳細については、「REG_REPLACE」 (ページ 162)を参照してください。
REPLACECHR	文字列、データクレンジング	REPLACECHR(*CaseFlag, InputString, OldCharSet, NewChar*)	文字列内の文字を 1 文字または文字なしに置換します。 詳細については、「REPLACECHR」 (ページ 163)を参照してください。

関数	関数タイプ	構文	説明
REPLACESTR	文字列、データクレンジング	REPLACESTR (*InputString*, *OldString1*, [OldString2, ... OldStringN,] *NewString*)	文字列内の文字を 1 文字、複数の文字または文字なしに置換します。 詳細については、「REPLACESTR」 (ページ 166)を参照してください。
REVERSE	文字列	REVERSE(*string*)	入力文字列を逆順にします。 詳細については、「REVERSE」 (ページ 169)を参照してください。
ROUND	データクレンジング、日付、数値	ROUND(*date* [, *format*]) または ROUND(*numeric_value* [, *precision*])	データクレンジングの場合、日付の一部を丸めます。数値の場合、数値を指定された桁数に丸めます。 詳細については、「ROUND (Dates)」 (ページ 170)または「ROUND (Numbers)」 (ページ 172)を参照してください。
RPAD	文字列	RPAD(*first_string*, *length* [, *second_string*])	文字列の末尾に空白または文字を追加して、文字列を指定した長さに変換します。 詳細については、「RPAD」 (ページ 174)を参照してください。
RTRIM	文字列、データクレンジング	RTRIM(*string* [, *trim_set*])	文字列の末尾から空白または文字を削除します。 詳細については、「RTRIM」 (ページ 175)を参照してください。
SET_DATE_PART	データクレンジング、日付	SET_DATE_PART(*date*, *format*, *value*)	Date/Time 値の一部を指定された値に設定します。 詳細については、「SET_DATE_PART」 (ページ 176)を参照してください。
SETCOUNTVARIABLE	特殊	SETCOUNTVARIABLE(*$$Variable*)	関数が評価した行を数えて、そのカウントに基づいて入出力パラメータの現在の値を増やします。 詳細については、「SETCOUNTVARIABLE」 (ページ 179)を参照してください。
SETMAXVARIABLE	特殊	SETMAXVARIABLE($$*Variable*,)	入出力パラメータの現在の値を、パラメータの現在の値と指定値のいずれか高い方に設定します。 詳細については、「SETMAXVARIABLE」 (ページ 179)を参照してください。
SETMINVARIABLE	特殊	SETMINVARIABLE($$*Variable*, *value*)	入出力パラメータの現在の値を、パラメータの現在の値と指定値のいずれか低い方に設定します。 詳細については、「SETMINVARIABLE」 (ページ 180)を参照してください。

関数	関数タイプ	構文	説明
SETVARIABLE	特殊	SETVARIABLE($$*Variable*, *value*)	入力パラメータの現在の値を指定した値に設定します。 詳細については、「SETVARIABLE」 (ページ 181)を参照してください。
SIGN	数値	SIGN(*numeric_value*)	数値が正の数、負の数、または 0 のいずれであるかを示します。 詳細については、「SIGN」 (ページ 182)を参照してください。
SIN	科学	SIN(*numeric_value*)	数値（ラジアン単位）の正弦を返します。 詳細については、「SIN」 (ページ 183)を参照してください。
SINH	科学	SINH(*numeric_value*)	数値（ラジアン単位）の双曲正弦を返します。 詳細については、「SINH」 (ページ 184)を参照してください。
SOUNDEX	データクレンジング	SOUNDEX(*string*)	文字列値を 4 字の文字列にエンコードします。 詳細については、「SOUNDEX」 (ページ 185)を参照してください。
SQRT	数値	SQRT(*numeric_value*)	正の数値の平方根を返します。 詳細については、「SQRT」 (ページ 186)を参照してください。
STDDEV	集計	STDDEV(*numeric_value* [, *filter_condition*])	関数に渡された数値の標準偏差を返します。 詳細については、「STDDEV」 (ページ 187)を参照してください。
SUBSTR	文字列、データクレンジング	SUBSTR(*string*, *start* [, *length*])	文字列の一部を返します。 詳細については、「SUBSTR」 (ページ 189)を参照してください。
SUM	集計	SUM(*numeric_value* [, *filter_condition*])	選択されたフィールドのすべての値の合計を返します。 詳細については、「SUM」 (ページ 191)を参照してください。
SYSTIMESTAMP	日付	SYSTIMESTAMP([*format*])	タスクを開始する Secure Agent のホストシステムの現在の日時をナノ秒までの精度で返します。 詳細については、「Systimestamp」 (ページ 192)を参照してください。

関数	関数タイプ	構文	説明
TAN	科学	TAN(*numeric_value*)	数値（ラジアン単位）の正接を返します。 詳細については、「TAN」 (ページ 193)を参照してください。
TANH	科学	TANH(*numeric_value*)	数値（ラジアン単位）の双曲正接を返します。 詳細については、「TANH」 (ページ 194)を参照してください。
TO_BIGINT	変換、データクレンジング	TO_BIGINT(*value* [, *flag*])	文字列または数値を Bigint 値に変換します。 詳細については、「TO_BIGINT」 (ページ 195)を参照してください。
TO_CHAR	変換、データクレンジング	TO_CHAR(*date* [, *format*]) または TO_CHAR(*numeric_value*)	日付または数値をテキスト文字列に変換します。 詳細については、「TO_CHAR (Dates)」 (ページ 196)または「TO_CHAR（数値）」 (ページ 200)を参照してください。
TO_DATE	変換、データクレンジング	TO_DATE(*string* [, *format*])	文字列をそれと同じ形式で日付データ型に変換します。変換の場合、文字列が mm/dd/yyyy hh:mi:ss 形式でなければ日付形式を指定する必要があります。 詳細については、「TO_DATE」 (ページ 201)を参照してください。日付形式の詳細については、第 4 章, 「日付」 (ページ 27)を参照してください。
TO_DECIMAL	変換、データクレンジング	TO_DECIMAL(*value* [, *scale*])	任意の値（Binary を除く）を Decimal に変換します。 詳細については、「TO_DECIMAL」 (ページ 204)を参照してください。
TO_FLOAT	変換、データクレンジング	TO_FLOAT(*value*)	任意の値（Binary を除く）を倍精度浮動小数点数（Double データ型）に変換します。 詳細については、「TO_FLOAT」 (ページ 205)を参照してください。
TO_INTEGER	変換、データクレンジング	TO_INTEGER(*value*)	任意の値（Binary を除く）の小数点以下を丸めて、整数に変換します。 詳細については、「TO_INTEGER」 (ページ 205)を参照してください。
TRUNC	データクレンジング、日付、数値	TRUNC(*date* [, *format*]) または TRUNC(*numeric_value* [, *precision*])	日付を特定の年、月、日、時、または分に切り詰めます。 数値を特定の桁数に切り詰めます。 詳細については、「TRUNC (Dates)」 (ページ 207)または「TRUNC（Numbers）」 (ページ 209)を参照してください。

関数	関数タイプ	構文	説明
UPPER	文字列	UPPER(*string*)	小文字の文字列を大文字に変換します。 詳細については、「UPPER」 (ページ 211)を参照してください。
VARIANCE	集計	VARIANCE(*numeric_value* [, *filter_condition*])	渡された値の分散を返します。 詳細については、「VARIANCE」 (ページ 211)を参照してください。

詳細モードの機能

詳細モードの式エディタには、高度な機能を有効にする関数が含まれています。マッピングを詳細モードにコピーする際には、利用可能な関数を使用するように式を更新しなければならないことがあります。

詳細モードで使用できない関数を含むマッピングを詳細モードにコピーすると、**[検証]** パネルに検証エラーが表示されます。このエラーを解決するには、詳細モードで利用可能な関数のみを使用するように、マッピングのデータロジックを更新します。詳細モードでは、関数が異なる動作をする場合があります。

次の表に、関数と使用できるマッピングを示します。

関数	使用できるマッピング
%OPR_CONCAT%	すべてのマッピング
%OPR_CONCATDELIM%	すべてのマッピング
%OPR_IIF%	すべてのマッピング
%OPR_SUM%	すべてのマッピング
ABORT	詳細モード以外
ABS	すべてのマッピング
ADD_TO_DATE	すべてのマッピング
AES_DECRYPT	すべてのマッピング
AES_ENCRYPT	すべてのマッピング
AES_GCM_DECRYPT	すべてのマッピング
AES_GCM_ENCRYPT	すべてのマッピング
ASCII	すべてのマッピング
AVG	すべてのマッピング
CEIL	すべてのマッピング

関数	使用できるマッピング
CHOOSE	すべてのマッピング
CHR	すべてのマッピング
CHRCODE	すべてのマッピング
COMPRESS	すべてのマッピング
CONCAT	すべてのマッピング
CONVERT_BASE	すべてのマッピング
COS	すべてのマッピング
COSH	すべてのマッピング
COUNT	すべてのマッピング
CRC32	すべてのマッピング
CUME	詳細モード以外
DATE_COMPARE	すべてのマッピング
DATE_DIFF	すべてのマッピング
DEC_BASE64	すべてのマッピング
DECODE	すべてのマッピング
DECOMPRESS	すべてのマッピング
ENC_BASE64	すべてのマッピング
ERROR	詳細モード以外
EXP	すべてのマッピング
FIRST	詳細モード以外 詳細モードではウィンドウ関数としてのみ使用可能
FLOOR	すべてのマッピング
FV	すべてのマッピング
GET_DATE_PART	すべてのマッピング
GREATEST	すべてのマッピング
IIF	すべてのマッピング
IN	すべてのマッピング

関数	使用できるマッピング
INDEXOF	すべてのマッピング
INITCAP	すべてのマッピング
INSTR	すべてのマッピング
IS_DATE	すべてのマッピング
IS_NUMBER	すべてのマッピング
IS_SPACES	すべてのマッピング
ISNULL	すべてのマッピング
LAG	詳細モード
LAST	詳細モード以外 詳細モードではウィンドウ関数としてのみ使用可能
LAST_DAY	すべてのマッピング
LEAD	詳細モード
LEAST	すべてのマッピング
LENGTH	すべてのマッピング
LN	すべてのマッピング
LOG	すべてのマッピング
LOWER	すべてのマッピング
LPAD	すべてのマッピング
LTRIM	すべてのマッピング
MAKE_DATE_TIME	すべてのマッピング
MAX (Dates)	すべてのマッピング
MAX (Numbers)	すべてのマッピング
MAX (String)	すべてのマッピング
MD5	すべてのマッピング
MEDIAN	すべてのマッピング
METAPHONE	すべてのマッピング
MIN (Dates)	すべてのマッピング

関数	使用できるマッピング
MIN (numbers)	すべてのマッピング
MIN (String)	すべてのマッピング
MOD	すべてのマッピング
MOVINGAVG	詳細モード以外
MOVINGSUM	詳細モード以外
NPER	すべてのマッピング
PERCENTILE	すべてのマッピング
PMT	すべてのマッピング
POWER	すべてのマッピング
PV	すべてのマッピング
RAND	すべてのマッピング
RATE	すべてのマッピング
REG_EXTRACT	すべてのマッピング
REG_MATCH	すべてのマッピング
REG_REPLACE	すべてのマッピング
REPLACECHR	すべてのマッピング
REPLACESTR	すべてのマッピング
REVERSE	すべてのマッピング
ROUND	すべてのマッピング
RPAD	すべてのマッピング
RTRIM	すべてのマッピング
SET_DATE_PART	すべてのマッピング
SETCOUNTVARIABLE	すべてのマッピング
SETMAXVARIABLE	すべてのマッピング
SETMINVARIABLE	すべてのマッピング
SETVARIABLE	詳細モード以外
SIGN	すべてのマッピング

関数	使用できるマッピング
SIN	すべてのマッピング
SINH	すべてのマッピング
SOUNDEX	すべてのマッピング
SQRT	すべてのマッピング
STDDEV	すべてのマッピング
SUBSTR	すべてのマッピング
SUM	すべてのマッピング
SYSTIMESTAMP	すべてのマッピング
TAN	すべてのマッピング
TANH	すべてのマッピング
TO_BIGINT	すべてのマッピング
TO_CHAR	すべてのマッピング
TO_DATE	すべてのマッピング
TO_DECIMAL	すべてのマッピング
TO_FLOAT	すべてのマッピング
TO_INTEGER	すべてのマッピング
TRUNC	すべてのマッピング
UPPER	すべてのマッピング
VARIANCE	すべてのマッピング

%OPR_CONCAT%

CONCAT 関数を使用して、式マクロ内の式を拡張し、複数のフィールドを連結します。 %OPR_CONCAT%は、すべてのデータをテキストに変換してから、フィールドを連結します。

構文

%OPR_CONCAT[*macro_input_field*]%

以下の表に、この関数の引数を示します。

引数	必須/オプション	説明
macro_input_field	必須	任意のデータ型（Binary を除く）。受信フィールドのセットまたは定数のセットを使用できます。マクロ入力フィールド、または少なくとも 1 つのマクロ入力フィールドを含む式を入力できます。

戻り値

文字列。

一連のフィールドのすべての文字列が NULL の場合は、NULL になります。

すべての文字列が NULL の場合、NULL を返します。それ以外の場合、NULL 値を無視して NULL 以外の文字列を連結します。

例

1 つのシステムにマージされた 2 つの従来のシステムがあります。両方のシステムの ID 番号を別のシステムに連結する必要があります。マクロ入力フィールド%ClientID%に、次のフィールドが含まれるとします。

```
ID1, ID2
```

次の式を使用して、%ClientID%内のフィールドを連結します。

```
 %OPR_CONCAT[ %ClientID% ]%
```

%OPR_CONCAT%は、次の ID を 1 つの戻り値に連結します。

ID1	ID2	RETURN VALUE
A6JU4199	7021	A6JU41997021
NULL	7022	7022
T7QX9018	7023	T7QX90187023
C2CL5421	7024	C2CL54217024
NULL	NULL	NULL

%OPR_CONCATDELIM%

CONCAT 関数を使用して、式マクロ内の式を拡張し、複数のフィールドを連結してカンマ区切り文字を追加します。 %OPR_CONCATDELIM%は、すべてのデータをテキストに変換してから、フィールドを連結します。

構文

```
%OPR_CONCATDELIM[ macro_input_field ]%
```

以下の表に、この関数の引数を示します。

引数	必須/オプション	説明
macro_input_field	必須	任意のデータ型（Binary を除く）。受信フィールドのセットまたは定数のセットを使用できます。マクロ入力フィールド、または少なくとも 1 つのマクロ入力フィールドを含む式を入力できます。

戻り値

文字列。

すべての文字列が NULL の場合、NULL を返します。それ以外の場合、NULL 値を無視して NULL 以外の文字列を連結します。

セット内のすべての文字列が NULL の場合、%OPR_CONCATDELIM%は NULL を返します。

例

マクロ入力フィールド%Address%に次のフィールドが含まれるとします。

```
City, State, Zip
```

次の式は、%Address%のフィールドを連結し、フィールド間にカンマを追加します。

```
 %OPR_CONCATDELIM[ %Address% ]%
```

%OPR_CONCAT_DELIM%は、次の住所フィールドを 1 つの戻り値に連結します。

CITY	STATE	ZIP	RETURN VALUE
Reston	VA	20190	Reston,VA,20190
Maywood	IL	NULL	Maywood,IL
Eagle River	AK	99577	Eagle River,AK,99577
St. Louis	MO	63110	St. Louis,MO,63110
NULL	NULL	NULL	NULL

%OPR_IIF%

IIF 関数を使用して、式マクロ内の式を拡張し、IIF 文のセットを評価します。

構文

%OPR_IIF[*condition*, *macro_input_field* [,*value*]]%

以下の表に、この関数の引数を示します。

引数	必須/ オプション	説明
condition	必須	値を求める条件。TRUE または FALSE になる有効な式を必要に応じて入力できます。マクロ入力フィールド、または少なくとも 1 つのマクロ入力フィールドを含む式を入力できます。
macro_input _field	必須	条件が TRUE のときに返したい値。バイナリ以外の任意のデータ型。戻り値はこの引数で指定したデータ型になります。マクロ入力フィールド、または少なくとも 1 つのマクロ入力フィールドを含む式を入力できます。
value	オプション	条件が FALSE のときに返したい値。バイナリ以外の任意のデータ型。水平マクロまたは垂直マクロには、有効な式を必要に応じて入力できます。ハイブリッドマクロには、出力マクロフィールドを定義するために使用する入力マクロフィールドを入力します。

一部のシステムで使用される条件関数と異なり、%OPR_IIF%関数では FALSE（*value*）条件は必須ではありません。*value* を省略すると、関数は条件が FALSE のときに以下の値を返します。

- *macro_input_field* が Numeric データ型の場合はゼロ。
- *macro_input_field* が String データ型の場合は空の文字列。
- *macro_input_field* が Date/Time データ型の場合は NULL。

戻り値

条件が TRUE の場合は、*macro_input_field* になります。

条件が FALSE の場合は *value*。

データにマルチバイト文字が含まれ、条件引数で文字列データを比較する場合、タスクを実行する Secure Agent のコードページに応じた戻り値が返されます。

%OPR_IIF%およびデータ型

%OPR_IIF%を使用した場合、戻り値のデータ型は最大の精度を持つ結果のデータ型と同じものとなります。

1 つ以上の結果が Double である場合、戻り値のデータ型は Double となります。

例

次に示すマクロ入力フィールド%tmin%および%tmax%では、次に示すマクロ入力フィールド%type%に対応する範囲を作成します。

```
%tmin%          %tmax%          %type%
0               30              very cold
31              60              cold
61              90              warm
91              100             hot
101             150             very hot
```

次の式では、温度データに基づいてタイプを割り当てます。

```
%OPR_IIF [ (TEMP >= %tmin%) AND (TEMP <= %tmax%), '%type%', 'out of range' ]%
```

%OPR_IIF%は次の温度を評価して、タイプを返します。

TEMP	RETURN VALUE
79	warm
21	very cold
170	out of range
99	hot
90	warm

%OPR_SUM%

SUM 関数を使用して、式マクロ内の式を拡張し、すべてのフィールドの合計を返します。 オプションとして、合計を計算するために読み込む行を制限するフィルタを適用できます。SUM で使用できる他の集計関数は 1 つのみです。またネストされた関数は Numeric データ型を返さなければなりません。

構文

```
%OPR_SUM[ macro_input_field [, filter_condition ] ]%
```

以下の表に、この関数の引数を示します。

引数	必須/オプション	説明
macro_input_field	必須	Numeric データ型。追加したい値を渡します。マクロ入力フィールド、または少なくとも 1 つのマクロ入力フィールドを含む式を入力できます。
filter_condition	オプション	検索される行を制限します。フィルタ条件は数値であるか、TRUE、FALSE、または NULL の値が求められなければなりません。有効なトランスフォーメーション式を必要に応じて入力できます。

戻り値

数値。

関数に渡されたフィールドがすべて NULL である場合、または行が 1 つも選択されていない場合には、NULL となります。例えば、すべての行でフィルタ条件が FALSE または NULL の場合、%OPR_SUM%は NULL を返します。

値の 1 つが NULL であると、%OPR_SUM%はその値を無視します。すべての値が NULL の場合、%OPR_SUM%は NULL を返します。

Group By

%OPR_SUM%は、トランスフォーメーションで定義したグループ化フィールドに基づいて値をグループ分けし、各グループについて 1 つの結果を返します。

グループ化フィールドがない場合、%OPR_SUM%はすべての行を 1 つのグループとして扱い、1 つの値を返します。

例

%Sales%マクロ入力フィールドは、次のフィールドを表します。

```
Q1, Q2, Q3, Q4
```

次の式は、%Sales%が表すフィールドの合計を返します。

```
%OPR_SUM[ %Sales% ]%
```

%OPR_SUM%は、次に示す各四半期の売上高を合計し、1 つの値を返します。

Q1	Q2	Q3	Q4	RETURN VALUE
50	200	50	100	400
100	NULL	100	250	450
500	200	200	200	1100
NULL	NULL	NULL	NULL	NULL
400	NULL	NULL	400	800

ABORT

タスクを停止し、ジョブの詳細にエラーメッセージを発行します。データ統合が ABORT 関数を検出すると、その行のデータの変換が停止します。

データを検査するには、ABORT を使用します。一般に ABORT は、IIF または DECODE 関数内で、タスクを強制終了する規則を設定するために使用されます。ABORT を使用してターゲットに NULL 値を書き込むのを回避することもできます。

構文

```
ABORT( string )
```

引数	必須/オプション	説明
string	必須	文字列。セッションの停止時にジョブ詳細に表示するメッセージを指定します。文字列の長さに制限はありません。有効な式を必要に応じて入力できます。

戻り値

NULL。

ABS

数値の絶対値を返します。

構文

ABS(*numeric_value*)

引数	必須/オプション	説明
numeric_value	必須	数値データ型。指定された数値の絶対値を返します。有効な式を必要に応じて入力できます。

戻り値

正の数値。ABS は、引数として渡された数値と同じデータ型を返します。Double を渡した場合は、Double を返します。同様に、整数を渡した場合は、整数を返します。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、2 つの数値の差を、どちらが大きいかに関係なく正の値として返します。

```
ABS( PRICE - COST )
```

PRICE	COST	RETURN VALUE
250	150	100
52	48	4
169.95	69.95	100
59.95	NULL	NULL
70	30	40
430	330	100
100	200	100

ADD_TO_DATE

指定された量を Date/Time 値の一部に加算し、関数に渡した日付と同じ形式の日付を返します。

ADD_TO_DATE には、正または負の整数値を入力できます。ADD_TO_DATE を使用して、日付の以下の部分を変更することができます。

年

amount 引数に正または負の整数を入力します。年のフォーマット文字列として、Y、YY、YYY、YYYY のいずれかを使用できます。例えば、SHIP_DATE カラムのすべての日付に 10 年を加えるには、式 `ADD_TO_DATE( SHIP_DATE, 'YY', 10 )`を使用します。

月

amount 引数に正または負の整数を入力します。月のフォーマット文字列として、MM、MON、MONTH のいずれかを使用できます。例えば、SHIP_DATE カラムのすべての日付から 10 ヶ月を引くには、式 ADD_TO_DATE(SHIP_DATE, 'MONTH', -10)を使用します。

日

amount 引数に正または負の整数を入力します。日のフォーマット文字列として、D、DD、DDD、DY、DAY のいずれかを使用できます。例えば、SHIP_DATE カラムのすべての日付に 10 日を加えるには、式 ADD_TO_DATE(SHIP_DATE, 'DD', 10)を使用します。

時間

amount 引数に正または負の整数を入力します。時間のフォーマット文字列として、HH、HH12、HH24 のいずれかを使用できます。例えば、SHIP_DATE カラムのすべての日付に 14 時間を加えるには、式 ADD_TO_DATE(SHIP_DATE, 'HH', 14)を使用します。

分

amount 引数に正または負の整数を入力します。分の設定にはフォーマット文字列 MI を使用します。例えば、SHIP_DATE カラムのすべての日付に 25 分を加えるには、式 ADD_TO_DATE(SHIP_DATE, 'MI', 25)を使用します。

秒

amount 引数に正または負の整数を入力します。秒の設定には SS フォーマット文字列を使用します。例えば、SHIP_DATE カラムのすべての日付に 59 秒を加えるには、以下の式を使用します。

```
ADD_TO_DATE( SHIP_DATE, 'SS', 59 )
```

構文

```
ADD_TO_DATE( date, format, amount )
```

引数	必須/オプション	説明
date	必須	Date/Time データ型。変更したい値を渡します。有効な式を必要に応じて入力できます。
format	必須	日付値の中で変更したい場所を指定するフォーマット文字列。フォーマット文字列は'mm'のように一重引用符で囲みます。フォーマット文字列は大文字と小文字を区別しません。
amount	必須	日付値を変更する値として、年数、月数、日数、時間数などを指定する整数値。整数を求める有効な式を必要に応じて入力できます。

戻り値

関数に渡した日付と同じ形式の日付。

関数に引数として NULL 値が渡された場合には、NULL です。

例

以下の式はすべて、DATE_SHIPPED カラムの各日付に 1 か月を加算します。特定の月に存在しない日を作成する値を渡すと、データ統合ではその月の最後の日が返されます。例えば、1998 年 1 月 31 日に 1 か月を追加すると、データ統合では 1998 年 2 月 28 日が返されます。

また、ADD_TO_DATE はうるう年を認識し、Jan 29 2000 に 1 か月を加えます。

```
ADD_TO_DATE( DATE_SHIPPED, 'MM', 1 )
ADD_TO_DATE( DATE_SHIPPED, 'MON', 1 )
ADD_TO_DATE( DATE_SHIPPED, 'MONTH', 1 )
```

DATE_SHIPPED	RETURN VALUE
Jan 12 1998 12:00:30AM	Feb 12 1998 12:00:30AM
Jan 31 1998 6:24:45PM	Feb 28 1998 6:24:45PM
Jan 29 2000 5:32:12AM	Feb 29 2000 5:32:12AM *(Leap Year)*
Oct 9 1998 2:30:12PM	Nov 9 1998 2:30:12PM
NULL	NULL

以下の式はすべて、DATE_SHIPPED カラムの各日付から 10 日を減算します。

```
ADD_TO_DATE( DATE_SHIPPED, 'D', -10 )
ADD_TO_DATE( DATE_SHIPPED, 'DD', -10 )
ADD_TO_DATE( DATE_SHIPPED, 'DDD', -10 )
ADD_TO_DATE( DATE_SHIPPED, 'DY', -10 )
ADD_TO_DATE( DATE_SHIPPED, 'DAY', -10 )
```

DATE_SHIPPED	RETURN VALUE
Jan 1 1997 12:00:30AM	Dec 22 1996 12:00AM
Jan 31 1997 6:24:45PM	Jan 21 1997 6:24:45PM
Mar 9 1996 5:32:12AM	Feb 29 1996 5:32:12AM *(Leap Year)*
Oct 9 1997 2:30:12PM	Sep 30 1997 2:30:12PM
Mar 3 1996 5:12:20AM	Feb 22 1996 5:12:20AM
NULL	NULL

以下の式はすべて、DATE_SHIPPED カラムの各日付から 15 時間を減算します。

```
ADD_TO_DATE( DATE_SHIPPED, 'HH', -15 )
ADD_TO_DATE( DATE_SHIPPED, 'HH12', -15 )
ADD_TO_DATE( DATE_SHIPPED, 'HH24', -15 )
```

DATE_SHIPPED	RETURN VALUE
Jan 1 1997 12:00:30AM	Dec 31 1996 9:00:30AM
Jan 31 1997 6:24:45PM	Jan 31 1997 3:24:45AM
Oct 9 1997 2:30:12PM	Oct 8 1997 11:30:12PM
Mar 3 1996 5:12:20AM	Mar 2 1996 2:12:20PM
Mar 1 1996 5:32:12AM	Feb 29 1996 2:32:12PM *(Leap Year)*
NULL	NULL

日付の使用について

ADD_TO_DATE の使用時には、下記のヒントを参照してください。

- フォーマット文字列を指定して、*amount* 引数を正または負の整数にすることにより、日付のどの部分でも加算または減算することができます。
- 特定の月に存在しない日を作成する値を渡すと、データ統合ではその月の最後の日が返されます。例えば、1998 年 1 月 31 日に 1 か月を追加すると、データ統合では 1998 年 2 月 28 日が返されます。
- 日付の操作を容易にするために、TRUNC や ROUND をネストすることができます。
- TO_DATE をネストして文字を日付に変換できます。
- ADD_TO_DATE は、日付の中で指定された 1 か所のみを変更します。日付の標準時刻をサマータイム時刻に変更したい場合には、日付の「時」部分を変更する必要があります。

AES_DECRYPT

復号化された値を文字列形式で返します。データ統合は、電子符号表（ECB）モードの動作で Advanced Encryption Standard（AES）アルゴリズムを使用します。AES アルゴリズムは、128 ビット暗号化を使用する FIPS 承認済みの暗号化アルゴリズムです。

構文

AES_DECRYPT (*value, key*)

引数	必須/オプション	説明
value	必須	バイナリデータ型。解読する値。
key	必須	文字列データ型。精度 16 文字以下。値を復号化するには、暗号化に使用したキーを使用します。

戻り値

復号化された文字列値。

NULL 値を入力した場合は、NULL です。

例

以下の例では、復号化された社会保険番号が返されます。この例では、データ統合は、SUBSRT 関数を使用して社会保険番号の始めの 3 桁の数字からキーを引き出します。

```
AES_DECRYPT( SSN_ENCRYPT, SUBSTR( SSN,1,3 ))
```

SSN_ENCRYPT	DECRYPTED VALUE
07FB945926849D2B1641E708C85E4390	832-17-1672
9153ACAB89D65A4B81AD2ABF151B099D	832-92-4731
AF6B5E4E39F974B3F3FB0F22320CC60B	832-46-7552
992D6A5D91E7F59D03B940A4B1CBBCBE	832-53-6194

SSN_ENCRYPT	DECRYPTED VALUE
992D6A5D91E7F59D03B940A4B1CBBCBE	832-81-9528

AES_ENCRYPT

暗号化された形式でバイナリデータを返します。データ統合は、電子符号表（ECB）モードの動作で Advanced Encryption Standard（AES）アルゴリズムを使用します。AES アルゴリズムは、128 ビット暗号化を使用する FIPS 承認済みの暗号化アルゴリズムです。

この関数を使用して、機密データが必要以上に露出されることを回避します。例えばデータベースに社会保険番号を格納する場合、AES_ENCRYPT 関数を使用して社会保険番号を暗号化し、機密性を確保します。

構文

AES_ENCRYPT (*value, key*)

引数	必須/ オプション	説明
value	必須	文字列データ型。暗号化する値。
key	必須	文字列データ型。精度 16 文字以下。

戻り値

暗号化された Binary 値。

NULL 値を入力した場合は、NULL です。

例

以下の例では、社会保険番号の暗号化された値が返されます。この例では、データ統合は、SUBSTR 関数を使用して社会保険番号の始めの 3 桁の数字からキーを引き出します。

AES_ENCRYPT(SSN, SUBSTR(SSN,1,3))

SSN	ENCRYPTED VALUE
832-17-1672	07FB945926849D2B1641E708C85E4390
832-92-4731	9153ACAB89D65A4B81AD2ABF151B099D
832-46-7552	AF6B5E4E39F974B3F3FB0F22320CC60B
832-53-6194	992D6A5D91E7F59D03B940A4B1CBBCBE
832-81-9528	992D6A5D91E7F59D03B940A4B1CBBCBE

ヒント

ターゲットでバイナリデータがサポートされていない場合、ENC_BASE64 関数で AES_ENCRYPT を使用して、データベースと互換性のある形式でデータを保存します。

AES_GCM_DECRYPT

プレーンテキストを文字列として返します。データ統合は、ガロア/カウンタモード（GCM）の動作で Advanced Encryption Standard（AES）アルゴリズムを使用します。AES アルゴリズムは、128、192、または 256 ビットキーを使用する FIPS 承認済みの暗号化アルゴリズムです。

注: ［検証］ボタンを使用してこの関数を検証することはできません。構文と引数の規則を確認して、引数が有効であることを確認してください。

構文

AES_GCM_DECRYPT (*value, init_vector, key [, keysize]*)

引数	必須/ オプション	説明
value	必須	バイナリデータ型。プレーンテキストに復号化される暗号文の値。
init_vector	必須	String データ型。値を復号化するには、暗号文の暗号化に使用した初期化ベクトル（IV）を使用します。IV は 96 ビットで、ランダムに生成されたものである必要があります。
key	必須	サイズが 128、192、または 256 ビットの文字列データ型。値を復号化するには、暗号化に使用したキーを使用します。
keysize	key 引数のサイズ（ビット単位）により、keysize 引数がオプションであるか必須であるかが決まります。 - key 引数のサイズが 128 ビット以下の場合、keysize はオプションです。 - key 引数のサイズが 128 ビットより大きく、かつ 192 ビット以下の場合、keysize は 192 ビットであり、必須です。 - key 引数のサイズが 192 ビットより大きく、かつ 256 ビット以下の場合、keysize は 256 ビットであり、必須です。	整数データ型。指定された key のサイズ。 可能な値: 128、192、256 ビット。デフォルト値は 128 ビットです。

戻り値

復号化された文字列プレーンテキスト。

NULL 値を入力した場合は、NULL です。

例

以下の例では、復号化された社会保険番号が返されます。*init_vector* と *key* は、社会保障番号を暗号化するのに使用されたものと同じ値です。

この例では、*init_vector* は 12 文字（96 ビット）で、*key* は 16 文字（128 ビット）です。key がデフォルト値の 128 ビットであるため、*keysize* はオプションです。

```
AES_GCM_DECRYPT(SSN_ENCRYPT, '012345678901', '1234567890123456', 128)
```

この例では、 *init_vector* は 12 文字（96 ビット）で、*key* は 17 文字（136 ビット）です。key が 128 ビットを超えるため、*keysize* は必須です。key のサイズが 192 ビットより小さい場合、key は null でパディングされます。

```
AES_GCM_DECRYPT(SSN_ENCRYPT, '123456789012', '12345678901234567', 192)
```

AES_GCM_ENCRYPT

バイナリ暗号文を返します。データ統合は、ガロア/カウンタモード（GCM）の動作で Advanced Encryption Standard（AES）アルゴリズムを使用します。AES アルゴリズムは、128、192、または 256 ビットキーを使用する FIPS 承認済みの暗号化アルゴリズムです。

この関数を使用して、機密データが必要以上に露出されることを回避します。例えばデータベースに社会保険番号を格納する場合、AES_GCM_ENCRYPT 関数を使用して社会保険番号を暗号化し、機密性を確保します。

AES-GCM は、128 ビットの認証タグを作成し、それを暗号化された暗号文に追加します。復号化は、認証タグを検証して削除します。認証タグは、データの暗号化チェックサムであり、偶発的なエラーとデータの意図的な変更の両方を明らかにします。

注: ［検証］ボタンを使用してこの関数を検証することはできません。構文と引数の規則を確認して、引数が有効であることを確認してください。

構文

AES_GCM_ENCRYPT (*value, init_vector, key [, keysize]*)

引数	必須/ オプション	説明
value	必須	String データ型。暗号文に暗号化されるプレーンテキストの値。
init_vector	必須	String データ型。プレーンテキストを暗号化するには、初期化ベクトル（IV）を使用します。IV は 96 ビットで、ランダムに生成されたものである必要があります。IV は、暗号化プロセスの開始にランダム性を付加するために、暗号化中にキーとともに使用されるビットのブロックです。 **注**: キーが同じで暗号化文字列が異なる IV を再利用しないでください。
key	必須	サイズが 128、192、または 256 ビットの文字列データ型。key のサイズが keysize より小さい場合、残りは null としてパディングされます。
keysize	key 引数のサイズ（ビット単位）により、keysize 引数がオプションであるか必須であるかが決まります。 - key 引数のサイズが 128 ビット以下の場合、keysize はオプションです。 - key 引数のサイズが 128 ビットより大きく、かつ 192 ビット以下の場合、keysize は 192 ビットであり、必須です。 - key 引数のサイズが 192 ビットより大きく、かつ 256 ビット以下の場合、keysize は 256 ビットであり、必須です。	整数データ型。指定された key のサイズ。 可能な値: 128、192、または 256 ビット。デフォルト値は 128 ビットです。

戻り値

暗号化されたバイナリ暗号文。

NULL 値を入力した場合は、NULL です。

例

以下の例では、社会保険番号の暗号化された値が返されます。*init_vector* と *key* 値は、後で社会保障番号を復号化するために使用されます。

この例では、*init_vector* は 12 文字（96 ビット）で、*key* は 16 文字（128 ビット）です。key がデフォルト値の 128 ビットであるため、*keysize* はオプションです。

```
AES_GCM_ENCRYPT('832-17-1672', '012345678901', '1234567890123456', 128)
```

この例では、*init_vector* は 12 文字（96 ビット）で、*key* は 17 文字（136 ビット）であるため、*keysize* は必須です。key のサイズが 192 ビットより小さい場合、key は null でパディングされます。

```
AES_GCM_ENCRYPT('832-17-1672', '123456789012', '12345678901234567', 192)
```

ヒント

ターゲットでバイナリデータがサポートされていない場合、ENC_BASE64 関数で AES_GCM_ENCRYPT を使用して、データベースと互換性のある形式でデータを保存します。

ASCII

ASCII 関数では、関数に渡された文字列の最初の文字に対応する UNICODE 数値を返します。

ASCII にはどのようなサイズの文字列でも渡せますが、求められるのは文字列の最初の文字だけです。ASCII に文字列値を渡す前に、UNICODE 値に変換する特定の文字を取り出します。例えば RTRIM などの文字列操作関数を使うことができます。数値を渡した場合、ASCII はそれを文字列に変換して、その文字列の最初の文字の UNICODE 値を返します。

注: この関数の動作は、CHRCODE 関数と同じです。既存の式で ASCII を使用している場合でも正常に機能します。ただし、新しい式を作成する場合は、ASCII 関数ではなく CHRCODE 関数を使用します。

構文

```
ASCII ( string )
```

引数	必須/オプション	説明
string	必須	文字列。UNICODE 値として返したい値を渡します。有効な式を必要に応じて入力できます。

戻り値

整数。文字列の最初の文字の UNICODE 値。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、ITEMS カラムの各値の最初の文字に対応する UNICODE 値を返します。

```
ASCII( ITEMS )
```

ITEMS	RETURN VALUE
Flashlight	70
Compass	67

ITEMS	RETURN VALUE
Safety Knife	83
Depth/Pressure Gauge	68
Regulator System	82

AVG

行のグループのすべての値の平均を返します。オプションとして、平均を計算するために読み込む行を制限するフィルタを適用できます。

AVG には他の集計関数は 1 つしかネストできません。また、ネストされた関数は数値データ型を返す必要があります。詳細モードでは集計関数をネストできません。

マッピングタスクでのみ使用します。

構文

AVG(*numeric_value* [, *filter_condition*])

引数	必須/オプション	説明
numeric_value	必須	数値データタイプ。平均を計算したい値を渡します。有効なトランスフォーメーション式を必要に応じて入力できます。
filter_condition	オプション	検索される行を制限します。フィルタ条件は数値であるか、TRUE、FALSE、または NULL の値が求められなければなりません。有効なトランスフォーメーション式を必要に応じて入力できます。

戻り値

数値。

関数に渡された値がすべて NULL である場合、または行が 1 つも選択されていない場合（たとえば、フィルタ条件の値がすべての行に対して FALSE または NULL であった場合）には、NULL です。

NULL

値が NULL であると、AVG はその行を無視します。ただし、渡されたすべての値が NULL である場合には、NULL を返します。

Group By

AVG は、トランスフォーメーションで定義したグループ化フィールドに基づいて値をグループ分けし、各グループについて 1 つの結果を返します。

グループ化フィールドがない場合には、AVG はすべての行を 1 つのグループとして扱い、1 つの値を返します。

例

次の式は、懐中電灯の平均卸売り原価を返します。

```
AVG( WHOLESALE_COST, ITEM_NAME='Flashlight' )
```

ITEM_NAME	WHOLESALE_COST
Flashlight	35.00
Navigation Compass	8.05
Regulator System	150.00
Flashlight	29.00
Depth/Pressure Gauge	88.00
Flashlight	31.00

RETURN VALUE: 31.66

ヒント

AVG 関数で平均を計算する前に、AVG に渡す値に算術演算を実行することができます。以下に例を示します。

```
AVG( QTY * PRICE - DISCOUNT )
```

CEIL

この関数に渡された数値以上の最小の整数を返します。 たとえば、CEIL に 3.14 を渡すと、4 を返します。CEIL に 3.98 を渡すと、4 を返します。同様に、CEIL に-3.17 を渡すと、-3 を返します。

構文

CEIL(*numeric_value*)

引数	必須/オプション	説明
numeric_value	必須	数値データ型。有効な式を必要に応じて入力できます。

戻り値

0～28 の宣言された精度を持つ数値を渡した場合は、整数です。

28 を超える宣言された精度を持つ数値を渡した場合は、Double 値です。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、価格を次の整数に丸めて返します。

```
CEIL( PRICE )
```

PRICE	RETURN VALUE
39.79	40
125.12	126
74.24	75
NULL	NULL
-100.99	-100

ヒント

CEIL 関数で整数を返す前に、CEIL に渡す値に算術演算を実行することができます。たとえば、数値に 10 を掛けてからその結果の値より小さい最小の整数を計算したい場合は、関数を次のように記述します。

```
CEIL( PRICE * 10 )
```

CHOOSE

文字列のリストから指定された位置の文字列を選択します。位置と値を指定します。値が位置に一致する場合、データ統合は値を返します。

構文

CHOOSE(*index, string1*, [*string2*, ..., *stringN*])

引数	必須/オプション	説明
index	必須	数値データ型。一致させる値の位置に基づき、数字を入力します。
string	必須	任意の文字値。

戻り値

インデックス値の位置に一致する文字列。

インデックス値の位置に一致する文字列が存在しない場合は NULL です。

例

以下の式は、インデックス値である 2 に基づき、'flashlight'の文字列を返します。

```
CHOOSE( 2, 'knife', 'flashlight', 'diving hood' )
```

以下の式は、インデックス値である 4 に基づき、NULL を返します。

```
CHOOSE( 4, 'knife', 'flashlight', 'diving hood' )
```

式には 4 つ目の引数が含まれていないため、CHOOSE は NULL を返します。

CHR

CHR は、この関数に渡す数値に対応する ASCII 文字を返します。ASCII 値は 0-255 の範囲になります。CHR には任意の整数を渡せますが、印刷可能な文字は ASCII コード 32-126 だけです。

構文

CHR(*numeric_value*)

引数	必須/オプション	説明
numeric_value	必須	数値データ型。ASCII 文字として返す値を渡します。有効な式を必要に応じて入力できます。

戻り値

ASCII 文字。1 文字を含む文字列。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、ITEM_ID カラムの各数値に対応する ASCII 文字を返します。

```
CHR( ITEM_ID )
```

ITEM_ID	RETURN VALUE
65	A
122	z
NULL	NULL
88	X
100	d
71	G

CHR 関数を使用して、一重引用符を文字列に連結することができます。一重引用符は、文字列リテラル内で使用できない唯一の文字です。次の例を検討します。

```
'Joan' || CHR(39) || 's car'
```

戻り値は次のとおりです。

```
Joan's car
```

CHRCODE

CHRCODE では、関数に渡された文字列の最初の文字に対応する UNICODE 数値を返します。

通常は、CHRCODE に文字列値を渡す前に、UNICODE 値に変換したい特定の文字を取り出します。例えば RTRIM などの文字列操作関数を使うことができます。数値を渡した場合、CHRCODE はそれを文字列に変換して、その文字列の最初の文字の UNICODE 値を返します。

注: この関数の動作は、ASCII 関数と同じです。式で現在、ASCII を使用している場合でも正しく動作します。ただし、新しい式を作成する場合は、ASCII 関数ではなく CHRCODE 関数を使用します。

構文

CHRCODE (*string*)

引数	必須/オプション	説明
string	必須	文字列。UNICODE 値として返したい値を渡します。有効な式を必要に応じて入力できます。

戻り値

整数。文字列の最初の文字の UNICODE 値。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、ITEMS カラムの各値の最初の文字に対応する UNICODE 値を返します。

```
CHRCODE( ITEMS )
```

ITEMS	RETURN VALUE
Flashlight	70
Compass	67
Safety Knife	83
Depth/Pressure Gauge	68
Regulator System	82

COMPRESS

zlib 圧縮アルゴリズムを使用してデータを圧縮します。zlib 圧縮アルゴリズムは、WinZip との互換性があります。広域ネットワークに大量のデータを送信する前に、COMPRESS 関数を使用します。

構文

COMPRESS(*value*)

引数	必須/ オプション	説明
value	必須	文字列データ型。圧縮するデータ。

戻り値

入力値の圧縮された Binary 値。

NULL 値を入力した場合は、NULL です。

例

組織にはオンライン注文サービスがあります。顧客の注文データは、広域ネットワーク上で送信します。ソースには 10MB の行が格納されています。この行のデータは、COMPRESS を使用して圧縮できます。データを圧縮すると、データ統合がネットワークに書き込むデータの量が減少します。そのため、タスクのパフォーマンスが向上します。

CONCAT

2 つの文字列を連結します。CONCAT は、文字列を連結する前にすべてのデータを文字に変換します。または、文字列演算子 || を使用して文字列を連結します。CONCAT の代わりに文字列演算子 || を使用すると、タスク実行時のパフォーマンスが向上します。

構文

CONCAT(*first_string, second_string*)

引数	必須/ オプション	説明
first_string	必須	任意のデータ型（Binary を除く）。連結したい 1 番目の文字列です。有効な式を必要に応じて入力できます。
second_string	必須	任意のデータ型（Binary を除く）。連結したい 2 番目の文字列です。有効な式を必要に応じて入力できます。

戻り値

文字列。

文字列が両方とも NULL である場合には、NULL です。

NULL

いずれか一方の文字列が NULL である場合には、CONCAT はその文字列を無視して、もう一方の文字列を返します。

両方の文字列が NULL である場合には、CONCAT は NULL を返します。

例

次の式は、FIRST_NAM カラムと LAST_NAME カラムの名前を連結します。

```
CONCAT( FIRST_NAME, LAST_NAME )
```

FIRST_NAME	LAST_NAME	RETURN VALUE
John	Baer	JohnBaer
NULL	Campbell	Campbell
Bobbi	Apperley	BobbiApperley
Jason	Wood	JasonWood
Dan	Covington	DanCovington
Greg	NULL	Greg
NULL	NULL	NULL
100	200	100200

CONCAT では文字列の間にスペースが追加されません。2 つの文字列の間にスペースを追加したい場合には、2 つの CONCAT 関数をネストした式を記述することができます。たとえば、次の式はまず名前の末尾にスペースを連結してから、姓を連結します。

```
CONCAT( CONCAT( FIRST_NAME, ' ' ), LAST_NAME )
```

FIRST_NAME	LAST_NAME	RETURN VALUE
John	Baer	John Baer
NULL	Campbell	Campbell *(includes leading space)*
Bobbi	Apperley	Bobbi Apperley
Jason	Wood	Jason Wood
Dan	Covington	Dan Covington
Greg	NULL	Greg
NULL	NULL	NULL

CHR および CONCAT 関数を使って、一重引用符を文字列に連結することができます。一重引用符は、文字列リテラル内で使用できない唯一の文字です。次の例を検討します。

```
CONCAT( 'Joan', CONCAT( CHR(39), 's car' ))
```

戻り値は次のとおりです。

```
Joan's car
```

CONVERT_BASE

ある基準価格の数値を別の基準価格へ変換します。

構文

CONVERT_BASE(*value, source_base, dest_base*)

引数	必須/オプション	説明
value	必須	文字列データ型。他のベースに変換する値。
source_base	必須	数値データ型。変換するデータの現在のベース値。ベースの最小値は 2 です。ベースの最大値は 36 です。
dest_base	必須	数値データ型。データの変換後のベース値。ベースの最小値は 2 です。ベースの最大値は 36 です。

戻り値

数値。

例

以下の例では、2222 を小数点ベース値の 10 からバイナリベース値の 2 に変換します。

```
CONVERT_BASE( 2222, 10, 2 )
```

データ統合は、100010101110 を返します。

COS

数値（ラジアン単位）の余弦を返します。

構文

COS(*numeric_value*)

引数	必須/オプション	説明
numeric_value	必須	数値データ型。ラジアンで表された数値データ（角度に π を掛け、180 で割ったもの）。余弦を計算したい値を渡します。有効な式を必要に応じて入力できます。

戻り値

Double 値。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、DEGREES カラムのすべての値に対して余弦を返します。

```
COS( DEGREES * 3.14159265359 / 180 )
```

DEGREES	RETURN VALUE
0	1.0
90	0.0
70	0.342020143325593
30	0.866025403784421
5	0.996194698091745
18	0.951056516295147
89	0.0174524064371813
NULL	NULL

ヒント

COS 関数で余弦を計算する前に、COS に渡す値に算術演算を実行することができます。例えば、次のように、カラム内の値をラジアンに変換してから余弦を計算することができます。

```
COS( ARCS * 3.14159265359 / 180 )
```

COSH

数値（ラジアン単位）の双曲余弦を返します。

構文

COSH(*numeric_value*)

引数	必須/オプション	説明
numeric_value	必須	数値データ型。ラジアンで表された数値データ（角度に π を掛け、180 で割ったもの)。双曲余弦を計算したい値を渡します。有効な式を必要に応じて入力できます。

戻り値

Double 値。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、ANGLES カラムの値に対して双曲余弦を返します。

```
COSH( ANGLES )
```

ANGLES	RETURN VALUE
1.0	1.54308063481524
2.897	9.0874465864177
3.66	19.4435376920294
5.45	116.381231106176
0	1.0
0.345	1.06010513656773
NULL	NULL

ヒント

COSH 関数で双曲余弦を計算する前に、COSH に渡す値に算術演算を実行することができます。以下に例を示します。

```
COSH( MEASURES.ARCS / 360 )
```

COUNT

グループ内で NULL 以外の値を持つ行の数を返します。オプションで、アスタリスク（*）引数を指定すれば、トランスフォーメーション内のすべての入力値の数を数えることもできます。行数を数える前に、条件に基づいて行をフィルタリングすることができます。

COUNT の中にネストできる他の集計関数は 1 つだけです。詳細モードでは集計関数をネストできません。

マッピングタスクでのみ使用します。

構文

```
COUNT( value [, filter_condition] )
```

または

COUNT(* [, *filter_condition*])

引数	必須/オプション	説明
value	必須	Binary 以外の任意のデータ型。数えたい値を渡します。有効なトランスフォーメーション式を必要に応じて入力できます。
*	オプション	*を指定すると、トランスフォーメーション内の *all rows* を数えられます。
filter_condition	オプション	検索される行を制限します。フィルタ条件は数値であるか、TRUE、FALSE、または NULL の値が求められなければなりません。有効なトランスフォーメーション式を必要に応じて入力できます。

戻り値

整数。

関数に渡された値がすべて NULL である場合は、0 です（アスタリスク（*）引数を指定した場合は除きます）。

NULL

値がすべて NULL である場合、関数は 0 を返します。

アスタリスク引数を指定した場合には、関数は、行のカラムに NULL 値があるかどうかに関係なく、すべての行を数えます。

value 引数を指定した場合には、関数は NULL 値の列を無視します。

Group By

COUNT は、トランスフォーメーションで定義したグループ化フィールドに基づいて値をグループ分けし、各グループについて 1 つの結果を返します。グループ化フィールドがない場合には、COUNT はすべての行を 1 つのグループとして扱い、1 つの値を返します。

例

次の式は、在庫が 5 個未満（NULL を除く）の品物がいくつあるかを数えます。

```
COUNT( ITEM_NAME, IN_STOCK < 5 )
```

ITEM_NAME	IN_STOCK
Flashlight	10
NULL	2
Compass	NULL
Regulator System	5
Safety Knife	8
Halogen Flashlight	1

RETURN VALUE: 1

この例では、「Halogen flashlight」は数えられますが、NULL の項目は数えられません。次の例の場合は、NULL 値を含めてトランスフォーメーション内のすべての行数を数えます。

```
COUNT( *, QTY < 5 )
```

ITEM_NAME	QTY
Flashlight	10
NULL	2
Compass	NULL
Regulator System	5
Safety Knife	8
Halogen Flashlight	1

RETURN VALUE: 2

この例では、NULL 項目と「Halogen flashlight」が数えられます。アスタリスク引数を指定して、フィルタを使用しない場合は、トランスフォーメーションに渡される行がすべて数えられます。以下に例を示します。

```
COUNT( * )
```

ITEM_NAME	QTY
Flashlight	10
NULL	2
Compass	NULL
Regulator System	5
Safety Knife	8
Halogen Flashlight	1

RETURN VALUE: 6

CRC32

32 ビット Cyclic Redundancy Check (CRC32)の値を返します。CRC32 を使用して、データ転送エラーを検索します。ファイルに保存されているデータが変更されていないか確認するためにも、CRC32 を使用できます。

注: CRC32 は、異なる入力文字列に対しても同じ出力結果を返すことができます。CRC32 を使用してキーを生成した場合、予期せぬ結果が出ることもあります。

構文

CRC32(*value*)

引数	必須/オプション	説明
value	必須	データ型は String または Binary。冗長性チェックを実行する値を渡します。入力値では、大文字と小文字が区別されます。入力値の大文字と小文字が区別されると、戻り値に影響します。たとえば、CRC32(informatica)と CRC32 (Informatica)では、異なる戻り値が返されます。

戻り値

32 ビットの整数値。

例

広域ネットワーク全体のソースからデータを読み込む必要があるとします。データは、転送中に変更しなければなりません。ファイル内のデータのチェックサムを計算し、ファイルと一緒に保存できます。データ統合がソースデータを読み込むと、データ統合は CRC32 を使用してチェックサムを計算し、格納されている値と比較することができます。2 つの値が一致する場合、データは変更されていません。

CUME

現在の合計を返します。 つまり、CUME は値を 1 つ加算するたびに合計を返します。 現在の合計を計算する前に、一部の行を除外するフィルタ条件を追加することもできます。

CUME および同種の関数（MOVINGAVG、MOVINGSUM など）を使用すると、現在の値を計算することによりレポート作成が簡単になります。

構文

CUME(*numeric_value* [, *filter_condition*])

引数	必須/オプション	説明
numeric_value	必須	数値データ型。現在の合計を計算したい値を渡します。有効な式を必要に応じて入力できます。ネストした式を作成することにより、関数の結果に基づいて（結果が数値である限り）現在の合計を計算することができます。
filter_condition	オプション	検索される行を制限します。フィルタ条件は数値であるか、TRUE、FALSE、または NULL の値が求められなければなりません。有効な式を必要に応じて入力できます。

戻り値

数値。

関数に渡された値がすべて NULL である場合、または行が 1 つも選択されていない場合（たとえば、フィルタ条件の値がすべての行に対して FALSE または NULL であった場合）には、NULL です。

NULL

値が NULL であると、CUME は前の行での現在の合計を返します。ただし、選択されたカラムの値がすべて NULL である場合には、NULL を返します。

例

以下の例では、PERSONAL_SALES カラムの現在までの合計を返します。

```
CUME( PERSONAL_SALES )
```

PERSONAL_SALES	RETURN VALUE
40000	40000
80000	120000
40000	160000
60000	220000
NULL	220000
50000	270000

同様に、現在の合計を計算する前に値を加算することもできます。

```
CUME( CA_SALES + OR_SALES )
```

CA_SALES	OR_SALES	RETURN VALUE
40000	10000	50000
80000	50000	180000
40000	2000	222000
60000	NULL	222000
NULL	NULL	222000
50000	3000	275000

DATE_COMPARE

2 つの日付のうちどちらが早いかを示す整数を返します。DATE_COMPARE は、日付値ではなく整数値を返します。

構文

DATE_COMPARE(*date1*, *date2*)

引数	必須/オプション	説明
date1	必須	Date/Time データ型。比較したい 1 つ目の日付です。値が日付となる有効な式を必要に応じて入力できます。
date2	必須	Date/Time データ型。比較したい 2 つ目の日付です。値が日付となる有効な式を必要に応じて入力できます。

戻り値

1 つ目の日付のほうが早い場合は、-1。

2 つの日付が同じである場合は、0。

2 つ目の日付のほうが早い場合は、1。

一方の日付が NULL である場合は、NULL。

例

次の式は、DATE_PROMISED カラムと DATE_SHIPPED カラムの各日付を比較して、どちらが早いかを示す整数を返します。

DATE_COMPARE(DATE_PROMISED, DATE_SHIPPED)

DATE_PROMISED	DATE_SHIPPED	RETURN VALUE
Jan 1 1997	Jan 13 1997	-1
Feb 1 1997	Feb 1 1997	0
Dec 22 1997	Dec 15 1997	1
Feb 29 1996	Apr 12 1996	-1 *(Leap year)*
NULL	Jan 6 1997	NULL
Jan 13 1997	NULL	NULL

DATE_DIFF

2 つの日付の間の時間の長さを返します。要求できるフォーマットは年、月、日、時間、分、または秒です。データ統合は、1 つ目の日付から 2 つ目の日付を減算して、その差を返します。

構文

DATE_DIFF(*date1*, *date2*, *format*)

引数	必須/オプション	説明
date1	必須	Date/Time データ型。比較したい 1 つ目の日付の値を渡します。有効な式を必要に応じて入力できます。
date2	必須	Date/Time データ型。比較したい 2 つ目の日付の値を渡します。有効な式を必要に応じて入力できます。
format	必須	日付または時間計測の単位を指定するフォーマット文字列。年、月、日、時間、分、または秒が指定できます。'mm'など、日付の中の 1 つの部分のみを指定できます。フォーマット文字列は一重引用符で囲んでください。フォーマット文字列は大文字と小文字を区別しません。（たとえば、フォーマット文字列'mm'は'MM'、'Mm'、'mM'と同じです。

戻り値

Double 値。*date1* が *date2* より遅い日付である場合には、戻り値は正の数です。*date1* が *date2* より早い日付である場合には、戻り値は負の数です。

2 つの日付が同じである場合は、ゼロです。

一方（または両方）の日付が NULL である場合は、NULL です。

例

次の式は、DATE_PROMISED カラムと DATE_SHIPPED カラムの日付の差が何時間あるかを返します。

```
DATE_DIFF( DATE_PROMISED, DATE_SHIPPED, 'HH' )
DATE_DIFF( DATE_PROMISED, DATE_SHIPPED, 'HH12' )
DATE_DIFF( DATE_PROMISED, DATE_SHIPPED, 'HH24' )
```

DATE_PROMISED	DATE_SHIPPED	RETURN VALUE
Jan 1 1997 12:00:00AM	Mar 29 1997 12:00:00PM	-2100
Mar 29 1997 12:00:00PM	Jan 1 1997 12:00:00AM	2100
NULL	Dec 10 1997 5:55:10PM	NULL
Dec 10 1997 5:55:10PM	NULL	NULL
Jun 3 1997 1:13:46PM	Aug 23 1996 4:20:16PM	6812.89166666667
Feb 19 2004 12:00:00PM	Feb 19 2005 12:00:00PM	-8784

次の式は、DATE_PROMISED カラムと DATE_SHIPPED カラムの日付の差を日数で返します。

```
DATE_DIFF( DATE_PROMISED, DATE_SHIPPED, 'D' )
DATE_DIFF( DATE_PROMISED, DATE_SHIPPED, 'DD' )
DATE_DIFF( DATE_PROMISED, DATE_SHIPPED, 'DDD' )
```

```
DATE_DIFF( DATE_PROMISED, DATE_SHIPPED, 'DY' )
DATE_DIFF( DATE_PROMISED, DATE_SHIPPED, 'DAY' )
```

DATE_PROMISED	DATE_SHIPPED	RETURN VALUE
Jan 1 1997 12:00:00AM	Mar 29 1997 12:00:00PM	-87.5
Mar 29 1997 12:00:00PM	Jan 1 1997 12:00:00AM	87.5
NULL	Dec 10 1997 5:55:10PM	NULL
Dec 10 1997 5:55:10PM	NULL	NULL
Jun 3 1997 1:13:46PM	Aug 23 1996 4:20:16PM	283.870486111111
Feb 19 2004 12:00:00PM	Feb 19 2005 12:00:00PM	-366

次の式は、DATE_PROMISED カラムと DATE_SHIPPED カラムの日付の差を月数で返します。

```
DATE_DIFF( DATE_PROMISED, DATE_SHIPPED, 'MM' )
DATE_DIFF( DATE_PROMISED, DATE_SHIPPED, 'MON' )
DATE_DIFF( DATE_PROMISED, DATE_SHIPPED, 'MONTH' )
```

DATE_PROMISED	DATE_SHIPPED	RETURN VALUE
Jan 1 1997 12:00:00AM	Mar 29 1997 12:00:00PM	-2.91935483870968
Mar 29 1997 12:00:00PM	Jan 1 1997 12:00:00AM	2.91935483870968
NULL	Dec 10 1997 5:55:10PM	NULL
Dec 10 1997 5:55:10PM	NULL	NULL
Jun 3 1997 1:13:46PM	Aug 23 1996 4:20:16PM	9.3290162037037
Feb 19 2004 12:00:00PM	Feb 19 2005 12:00:00PM	-12

次の式は、DATE_PROMISED カラムと DATE_SHIPPED カラムの日付の差を年数で返します。

```
DATE_DIFF( DATE_PROMISED, DATE_SHIPPED, 'Y' )
DATE_DIFF( DATE_PROMISED, DATE_SHIPPED, 'YY' )
DATE_DIFF( DATE_PROMISED, DATE_SHIPPED, 'YYY' )
DATE_DIFF( DATE_PROMISED, DATE_SHIPPED, 'YYYY' )
```

DATE_PROMISED	DATE_SHIPPED	RETURN VALUE
Jan 1 1997 12:00:00AM	Mar 29 1997 12:00:00PM	-0.24327956989247
Mar 29 1997 12:00:00PM	Jan 1 1997 12:00:00AM	0.24327956989247
NULL	Dec 10 1997 5:55:10PM	NULL
Dec 10 1997 5:55:10PM	NULL	NULL
Jun 3 1997 1:13:46PM	Aug 23 1996 4:20:16PM	0.77741801697531
Feb 19 2004 12:00:00PM	Feb 19 2005 12:00:00PM	-1

DEC_BASE64

Base64 のエンコードされた値をデコードし、データを表すバイナリデータと文字列を返します。

構文

DEC_BASE64(*value*)

引数	必須/オプション	説明
値	必須	文字列データ型。デコードするデータ。

戻り値

デコードした Binary 値。

NULL 値を入力した場合は、NULL です。

例

MQSeries メッセージ ID をエンコードしてフラットファイルに書き込みました。フラットファイルソースから、MQSeries メッセージ ID を含むデータを読み込む必要があります。この場合、DEC_BASE64 を使用して ID をデコードし、元の Binary 値に変換できます。

DECODE

指定した値をカラムで検索します。値を見つけると、定義された結果値を返します。1 つの DECODE 関数内には、いくつでも検索を指定できます。

DECODE を使って文字列カラムの値を検索する場合には、RTRIM 関数で末尾の空白を削除することも、検索文字列内に空白を含めることもできます。

構文

DECODE(*value*, *first_search*, *first_result* [, *second_search*, *second_result*]...[,*default*])

引数	必須/オプション	説明
value	必須	任意のデータ型（Binary を除く）。検索対象となる値を渡します。有効な式を必要に応じて入力できます。
search	必須	value 引数と同じデータ型を持つ任意の値。検索したい値を渡します。検索する値と値引数は、必ず一致させる必要があります。値の一部を検索することはできません。また、検索する値では大文字と小文字が区別されます。 例えば、特定のカラムで文字列'Halogen Flashlight'を検索したい場合には、'Halogen'だけでなく'Halogen Flashlight'と入力する必要があります。'Halogen'と入力すると、一致する値が見つかりません。有効な式を必要に応じて入力できます。

引数	必須/オプション	説明
result	必須	任意のデータ型（Binary を除く）。検索で一致した値が見つかったときに返す値です。有効な式を必要に応じて入力できます。
default	オプション	任意のデータ型（Binary を除く）。検索で一致した値が見つからなかったときに返す値です。有効な式を必要に応じて入力できます。

戻り値

一致した値が見つかった場合は、*First_result*。

一致した値が見つからなかった場合は、デフォルト値。

一致した値が見つからなかった場合、デフォルト値を指定していなければ NULL。

複数の条件が満たされている場合でも、データ統合は最初に一致した結果を返します。

データにマルチバイト文字が含まれ、DECODE 式で文字列データを比較する場合、タスクを実行する Secure Agent のコードページに応じた戻り値が返されます。

DECODE とデータ型

DECODE を使う場合、戻り値のデータ型は常に最大の精度を持つ結果のデータ型と同じです。

たとえば、次のような式があるとします。

```
DECODE ( CONST_NAME,
         'Five', 5,
         'Pythagoras', 1.414213562,
         'Archimedes', 3.141592654,
         'Pi', 3.141592654 )
```

この式の戻り値は 5、1.414213562、3.141592654 です。最初の結果は整数で、残りの結果は小数です。Decimal データ型は整数データ型よりも精度が高くなります。この式では、結果は常に Decimal データ型として記述します。

少なくとも 1 つの結果が Double の場合、戻り値のデータ型は Double になります。

文字列戻り値と数値戻り値の両方を備える DECODE 関数は作成できません。

例えば、次の式は、戻り値に文字列と数値の両方が含まれるため無効です。

```
DECODE ( CONST_NAME,
         'Five', 5,
         'Pythagoras', '1.414213562',
         'Archimedes', '3.141592654',
         'Pi', 3.141592654 )
```

例

次の式は、DECODE を使用し、特定の ITEM_ID を検索して ITEM_NAME を返します。

```
DECODE( ITEM_ID, 10, 'Flashlight',
                 14, 'Regulator',
                 20, 'Knife',
                 40, 'Tank',
                 'NONE' )
```

ITEM_ID	RETURN VALUE
10	Flashlight

ITEM_ID	RETURN VALUE
14	Regulator
17	NONE
20	Knife
25	NONE
NULL	NONE
40	Tank

検索値が ITEM_ID に一致しないため、DECODE は項目 17 および 25 に対してデフォルト値の NONE を返します。また、NULL の ITEM_ID に対しても NONE を返しています。

次の式は、複数の列および条件をテストして、上から順に'TRUE'であるか'FALSE'であるかを求めます。

```
DECODE( TRUE,
        Var1 = 22, 'Variable 1 was 22!',
        Var2 = 49, 'Variable 2 was 49!',
        Var1 < 23, 'Variable 1 was less than 23.',
        Var2 > 30, 'Variable 2 was more than 30.',
        'Variables were out of desired ranges.')
```

Var1	Var2	RETURN VALUE
21	47	Variable 1 was less than 23.
22	49	Variable 1 was 22!
23	49	Variable 2 was 49!
24	27	Variables were out of desired ranges.
25	50	Variable 2 was more than 30.

DECOMPRESS

zlib 圧縮アルゴリズムを使用してデータを解凍します。zlib 圧縮アルゴリズムは、WinZip との互換性があります。広域ネットワークからデータを受け取る場合、DECOMPRESS 関数を使用します。

構文

DECOMPRESS(*value, precision*)

引数	必須/ オプション	説明
value	必須	バイナリデータ型。解凍するデータ。
precision	オプション	Integer データ型。

戻り値

入力値の解凍された Binary 値。

NULL 値を入力した場合は、NULL です。

例

組織にはオンライン注文サービスがあります。広域ネットワークから、圧縮された顧客注文データを受け取りました。データ統合を使用してこのデータを読み取り、データウェアハウスにロードする必要があります。この場合、その行の DECOMPRESS を使用してデータの各行を解凍できます。データ統合は、その後、解凍されたデータをターゲットにロードできるようになります。

ENC_BASE64

データのエンコードは、MIME（Multipurpose Internet Mail Extensions）エンコードを使用してバイナリデータを文字列データに変換します。

Binary データをサポートしていないデータベースまたはファイルにデータを保存する場合、データをエンコードします。また、文字列形式の Binary データを渡すためにも、データをエンコードできます。エンコードされたデータは、元のデータより約 33%長くなります。データは、ランダムに並べられた文字セットとして表示されます。

構文

ENC_BASE64(*value*)

引数	必須/ オプション	説明
value	必須	バイナリデータ型または文字列データ型。エンコードするデータ。

戻り値

エンコードされた値。

NULL 値を入力した場合は、NULL です。

例

MQSeries ソースからメッセージを読み込み、そのデータをフラットファイルターゲットに書き込む必要があります。このとき、ターゲットデータの一部として MQSeries メッセージ ID を格納しなければいけません。しかし、MsgID フィールドは Binary で、フラットファイルターゲットでは Binary データはサポートされていません。この場合は、データ統合がデータをターゲットに書き込む前に、ENC_BASE64 を使用して MsgID をエンコードします。

ERROR

データ統合は、行をスキップし、定義されたエラーメッセージを表示します。データ統合は、スキップした行とエラーメッセージをエラー行ファイルに書き込みます。

式で ERROR を使用して、データを検査します。一般に ERROR は、IIF または DECODE 関数内で、行をスキップする規則を設定するために使用されます。ERROR を使用することにより、NULL 値を渡さないようにすることができます。

式に ERROR を追加してトランスフォーメーションエラーを処理することもできます。

構文

ERROR(*string*)

引数	必須/オプション	説明
string	必須	文字列値。データ統合が、ERROR 関数を含む式に基づいて行をスキップした場合に表示するメッセージ。文字列の長さに制限はありません。

戻り値

文字列。

例

以下の例は、組織内における全部署の従業員の平均給与を計算するマッピングを参照する場合に、負の値をスキップする方法を示しています。次に示す式は、データ統合で Salary カラムに負の給与が見つかった場合にその行をスキップしてエラーを表示するように、IIF 式内に ERROR 関数をネストしています。

```
IIF( SALARY < 0, ERROR ('Error. Negative salary found. Row skipped.', EMP_SALARY )
```

SALARY	RETURN VALUE
10000	10000
-15000	'Error. Negative salary found. Row skipped.'
NULL	NULL
150000	150000
1005	1005

EXP

E の指定乗（指数）を返します（E=2.71828183）。例えば、EXP(2)は 7.38905609893065 を返します。通常、この関数は科学データや技術データの分析に使用します。EXP は LN 関数の逆数です。LN は数値の自然対数を返します。

構文

EXP(*exponent*)

引数	必須/オプション	説明
exponent	必須	数値データ型。e の累乗に対する指数の値有効な式を必要に応じて入力できます。

戻り値

Double 値。

関数に引数として NULL 値が渡された場合には、NULL。

例

次の式では、Numbers カラムに格納されている値を指数値として使っています。

EXP(NUMBERS)

NUMBERS	RETURN VALUE
10	22026.4657948067
-2	0.135335283236613
8.55	5166.754427176
NULL	NULL

FIRST

フィールドまたはグループ内で見つかった最初の値を返します。オプションとして、読み込む行を制限するフィルタを適用できます。FIRST の中にネストできる他の集計関数は 1 つだけです。

マッピングタスクでのみ使用します。

構文

FIRST(*value* [, *filter_condition*])

引数	必須/オプション	説明
value	必須	Binary 以外の任意のデータ型。最初の値を返したい値を渡します。有効なトランスフォーメーション式を必要に応じて入力できます。
filter_condition	オプション	検索される行を制限します。フィルタ条件は数値であるか、TRUE、FALSE、または NULL の値が求められなければなりません。有効なトランスフォーメーション式を必要に応じて入力できます。

戻り値

グループの最初の値。

関数に渡された値がすべて NULL である場合、または行が 1 つも選択されていない場合（たとえば、フィルタ条件の値がすべての行に対して FALSE または NULL であった場合）には、NULL です。

NULL

値が NULL であると、FIRST はその行を無視します。ただし、フィールドから渡された値がすべて NULL である場合には、NULL を返します。

Group By

FIRST は、トランスフォーメーションで定義したグループ化フィールドに基づいて値をグループ分けし、各グループについて 1 つの結果を返します。

グループ化フィールドがない場合には、FIRST はすべての行を 1 つのグループとして扱い、1 つの値を返します。

例

次の式は、ITEM_NAME フィールド内で価格が$10.00 を超える最初の値を返します。

```
FIRST( ITEM_NAME, ITEM_PRICE > 10 )
```

ITEM_NAME	ITEM_PRICE
Flashlight	35.00
Navigation Compass	8.05
Regulator System	150.00
Flashlight	29.00
Depth/Pressure Gauge	88.00
Flashlight	31.00

RETURN VALUE: Flashlight

次の式は、ITEM_NAME フィールド内で価格が$40.00 を超える最初の値を返します。

```
FIRST( ITEM_NAME, ITEM_PRICE > 40 )
```

ITEM_NAME	ITEM_PRICE
Flashlight	35.00
Navigation Compass	8.05
Regulator System	150.00
Flashlight	29.00
Depth/Pressure Gauge	88.00
Flashlight	31.00

RETURN VALUE: Regulator System

FLOOR

渡された数値以下の最大の整数を返します。たとえば、FLOOR に 3.14 を渡すと、3 を返します。3.98 を渡すと、3 を返します。同様に、-3.17 を渡すと-4 を返します。

構文

FLOOR(*numeric_value*)

引数	必須/オプション	説明
numeric_value	必須	数値データ型。数値データを求める有効な式を必要に応じて入力できます。

戻り値

0～28 の宣言された精度を持つ数値を渡した場合は、整数です。

28 を超える宣言された精度を持つ数値を渡した場合は、Double 値です。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、PRICE カラムの値に等しいかそれより小さい最大の整数を返します。

FLOOR(PRICE)

PRICE	RETURN VALUE
39.79	39
125.12	125
74.24	74
NULL	NULL
-100.99	-101

ヒント

FLOOR に渡す値に算術演算を実行することができます。たとえば数値に 10 を掛け、その結果より小さい最大整数を計算するには、以下の関数を書き込みます。

FLOOR(UNIT_PRICE * 10)

FV

定期的に一定額を支払い、一定の利率で利息が付く投資の将来価値を返します。

構文

FV(*rate, terms, payment* [, *present value, type*])

引数	必須/オプション	説明
rate	必須	数値。各期間で得た金利。十進数で表示されます。率のパーセント値を 100 で割った小数を指定します。0 以上を指定する必要があります。
terms	必須	数値。期間または支払の数値。0 より大きな値を指定する必要があります。
payment	必須	数値。期間ごとの支払額。負の数を指定する必要があります。
present value	オプション	数値。投資のカレント値。この引数を省略すると、FV はゼロを使用します。
type	オプション	整数。支払時期。期首支払の場合は、1 を入力します。期末支払の場合は、0 を入力します。デフォルトは 0 です。0 または 1 以外の値を入力すると、データ統合ではその値が 1 として処理されます。

戻り値

数値。

例

月 1 回の複利計算で年利 9%の金利が（月利 9%/12、または 0.75%）支払われる口座に、$2,000 預金するとします。今後 12 ヶ月間は、毎月月初に 250 ドルを預金する予定です。以下の式では、この場合の 1 年後の口座残高は$5,337.96 になります。

```
FV(0.0075, 12, -250, -2000, TRUE)
```

注意事項

期間ごとに得た金利を計算するには、年利を 1 年間の支払回数で除算します。支払値と現在価値は支払額を指すため、負の値になります。

GET_DATE_PART

日付の中の指定した部分を整数値として返します。例えば、日付の月の部分を返す式を作成して「Apr 1 1997 00:00:00」のような日付を渡すと、GET_DATE_PART は 4 を返します。

構文

GET_DATE_PART(*date*, *format*)

引数	必須/オプション	説明
date	必須	Date/Time データ型。有効な式を必要に応じて入力できます。
format	必須	日付値の中で変更したい場所を指定するフォーマット文字列。フォーマット文字列は'mm'のように一重引用符で囲みます。フォーマット文字列は大文字と小文字を区別しません。各フォーマット文字列は、デフォルト形式 MM/DD/YYYY HH24:MI:SS に基づいて日付の各部分の全体を返します。 たとえば、日付「Apr 1 1997」を GET_DATE_PART に渡す場合、フォーマット文字列'Y'、'YY'、'YYY'、'YYYY'はすべて 1997 を返します。

戻り値

日付の中の指定された部分を示す整数。

関数に NULL 値を渡した場合は NULL です。

例

以下の式は、DATE_SHIPPED カラムの各日付の時の部分を返します。デフォルトの日付形式は 24 時間方式に基づくため、12:00:00AM は 0 を返します。

```
GET_DATE_PART( DATE_SHIPPED, 'HH' )
GET_DATE_PART( DATE_SHIPPED, 'HH12' )
GET_DATE_PART( DATE_SHIPPED, 'HH24' )
```

DATE_SHIPPED	RETURN VALUE
Mar 13 1997 12:00:00AM	0
Sep 2 1997 2:00:01AM	2
Aug 22 1997 12:00:00PM	12
June 3 1997 11:30:44PM	23
NULL	NULL

以下の式は、DATE_SHIPPED カラムの各日付の日の部分を返します。

```
GET_DATE_PART( DATE_SHIPPED, 'D' )
GET_DATE_PART( DATE_SHIPPED, 'DD' )
GET_DATE_PART( DATE_SHIPPED, 'DDD' )
GET_DATE_PART( DATE_SHIPPED, 'DY' )
GET_DATE_PART( DATE_SHIPPED, 'DAY' )
```

DATE_SHIPPED	RETURN VALUE
Mar 13 1997 12:00:00AM	13
June 3 1997 11:30:44PM	3
Aug 22 1997 12:00:00PM	22
NULL	NULL

以下の式は、DATE_SHIPPED カラムの各日付の月の部分を返します。

```
GET_DATE_PART( DATE_SHIPPED, 'MM' )
GET_DATE_PART( DATE_SHIPPED, 'MON' )
GET_DATE_PART( DATE_SHIPPED, 'MONTH' )
```

DATE_SHIPPED	RETURN VALUE
Mar 13 1997 12:00:00AM	3
June 3 1997 11:30:44PM	6
NULL	NULL

以下の式は、DATE_SHIPPED カラムの各日付の年の部分を返します。

```
GET_DATE_PART( DATE_SHIPPED, 'Y' )
GET_DATE_PART( DATE_SHIPPED, 'YY' )
GET_DATE_PART( DATE_SHIPPED, 'YYY' )
GET_DATE_PART( DATE_SHIPPED, 'YYYY' )
```

DATE_SHIPPED	RETURN VALUE
Mar 13 1997 12:00:00AM	1997
June 3 1997 11:30:44PM	1997
NULL	NULL

GREATEST

入力値のリストから最大値を返します。この関数を使用すると、文字列、日付、または数字の最大値が返されます。デフォルトでは、大文字と小文字を区別します。

構文

GREATEST(*value1*, [*value2*, ..., *valueN*,] *CaseFlag*)

引数	必須/オプション	説明
value	必須	任意のデータ型（Binary を除く）。データ型は他の値との互換性が必要です。他の値と比較しなければならない値。最低 1 つの値引数を入力する必要があります。 この値および他の入力値が数字の場合、すべての値で可能な限り高い精度が適用されます。例えば、一部の値が Integer データ型で他の値が Double データ型である場合、データ統合はその値を Double データ型に変換します。
CaseFlag	オプション	整数でなければなりません。この関数の引数の大文字と小文字を区別するかどうかを指定します。有効な式を必要に応じて入力できます。 *>CaseFlag* が 0 以外の数値の場合、大文字と小文字を区別します。 *>CaseFlag* が NULL 値または 0 の場合、大文字と小文字を区別しません。

戻り値

入力値の中で最大である場合は *value1*、入力値の中で最大である場合は *value2* など。

すべての引数が NULL の場合は、NULL です。

例

次の式は、注文した項目の最大数を返します。

```
GREATEST( QUANTITY1, QUANTITY2, QUANTITY3 )
```

QUANTITIY1	QUANTITY2	QUANTITY3	RETURN VALUE
150	756	27	756
			NULL
5000	97	17	5000
120	1724	965	1724

IIF

条件の結果に基づいて、指定した 2 つの値のうちの 1 つを返します。

構文

```
IIF( condition, value1 [,value2] )
```

引数	必須/オプション	説明
condition	必須	値を求める条件。TRUE または FALSE になる有効な式を必要に応じて入力できます。
value1	必須	任意のデータ型（Binary を除く）。条件が TRUE のときに返したい値。戻り値は常にこの引数で指定したデータ型になります。有効な式（別の IIF 式を含む）を必要に応じて入力できます。
value2	オプション	任意のデータ型（Binary を除く）。条件が FALSE のときに返したい値。有効な式（別の IIF 式を含む）を必要に応じて入力できます。

一部のシステムで使用される条件関数と異なり、IIF 関数では FALSE 条件（*value2*）は必須ではありません。*value2* を省略すると、関数は条件が FALSE のときに以下の値を返します。

- *value1* が数値データ型のときは、0。
- *value1* が文字列データ型のときは、空文字列。
- *value1* が Date/Time データ型のときは、NULL。

例えば、下記の式には FALSE 条件が含まれておられず、*value1* は文字列データ型であるため、データ統合は FALSE となる各行に対して空の文字列を返します。

```
IIF( SALES > 100, EMP_NAME )
```

SALES	EMP_NAME	RETURN VALUE
150	John Smith	John Smith

SALES	EMP_NAME	RETURN VALUE
50	Pierre Bleu	'' *(empty string)*
120	Sally Green	Sally Green
NULL	Greg Jones	'' *(empty string)*

戻り値

条件が TRUE の場合は *value1*。

条件が FALSE の場合は *value2*。

例えば、下記の式には FALSE 条件である NULL が含まれているため、データ統合は FALSE となる各行に対して NULL を返します。

```
IIF( SALES > 100, EMP_NAME, NULL )
```

SALES	EMP_NAME	RETURN VALUE
150	John Smith	John Smith
50	Pierre Bleu	NULL
120	Sally Green	Sally Green
NULL	Greg Jones	NULL

データにマルチバイト文字が含まれ、条件引数で文字列データを比較する場合、タスクを実行する Secure Agent のコードページに応じた戻り値が返されます。

IIF とデータ型

IIF を使う場合、戻り値のデータ型は常に最大の精度を持つ結果のデータ型と同じです。

例えば、次のような式があるとします。

```
IIF( SALES < 100, 1, .3333 )
```

TRUE の結果（1）は整数であり、FALSE の結果（.3333）は小数です。Decimal データ型は整数データ型よりも精度が高くなります。したがって、戻り値のデータ型は常に 10 進となります。

少なくとも 1 つの結果が Double の場合、戻り値のデータ型は Double になります。

IIF の特殊な使用法

ネストした IIF 文を使用して、複数の条件をテストできます。以下の例は、各種条件をテストし、販売額がゼロまたは負の場合には 0 を返します。

```
IIF( SALES > 0, IIF( SALES < 50, SALARY1, IIF( SALES < 100, SALARY2, IIF( SALES < 200, SALARY3, BONUS))), 0 )
```

このロジックは、次のようにコメントを加えると読みやすくなります。

```
IIF( SALES > 0,
   --then test to see if sales is between 1 and 49:
      IIF( SALES < 50,
         --then return SALARY1
            SALARY1,
         --else test to see if sales is between 50 and 99:
            IIF( SALES < 100,
               --then return
                  SALARY2,
               --else test to see if sales is between 100 and 199:
                  IIF( SALES < 200,
```

```
                  --then return
                     SALARY3,
                  --else for sales over 199, return
                     BONUS)
               )
         ),
   --else for sales less than or equal to zero, return
      0)
```

更新方式で IIF を使用します。以下に例を示します。

```
IIF( ISNULL( ITEM_NAME ), DD_REJECT, DD_INSERT)
```

IIF の代替手段

多くの場合、IIF の代わりに「DECODE」 (ページ 96)を使用します。DECODE を使うとコードが読みやすくなる場合があります。前節の最初の例で IIF の代わりに DECODE を使用すると、以下のようになります。

```
DECODE( TRUE,
   SALES > 0 and SALES < 50, SALARY1,
   SALES > 49 AND SALES < 100, SALARY2,
   SALES > 99 AND SALES < 200, SALARY3,
   SALES > 199, BONUS)
```

IN

入力値を値のリストとマッチングします。デフォルトでは、大文字と小文字を区別します。

構文

IN(*valueToSearch*, *value1*, [*value2*, ..., *valueN*,] *CaseFlag*)

引数	必須/オプション	説明
valueToSearch	必須	文字列、日付または数値。値のカンマ区切りリストに一致させる入力値。
value	必須	文字列、日付または数値。検索する値のカンマ区切りリスト。値にはカラムを使用できます。一覧表示できる値の最大表示数は指定されていません。
CaseFlag	オプション	整数でなければなりません。この関数の引数の大文字と小文字を区別するかどうかを指定します。有効な式を必要に応じて入力できます。 *CaseFlag* が 0 以外の数値の場合、大文字と小文字を区別します。 *CaseFlag* が NULL 値または 0 の場合、大文字と小文字を区別しません。

戻り値

入力値が値リストに一致する場合は TRUE (1)。

入力値が値リストに一致しない場合は FALSE (0)。

NULL 値を入力した場合は、NULL です。

例

以下の式は、入力値が safety Knife、Chisel Point Knife、Medium Titanium Knife のいずれであるかを決定します。入力値は、必ずしもカンマ区切りリストの値の表記（大文字/小文字）に一致させる必要はありません。

```
IN( ITEM_NAME, 'Chisel Point Knife', 'Medium Titanium Knife', 'Safety Knife', 0 )
```

ITEM_NAME	RETURN VALUE
Stabilizing Vest	0 (FALSE)
Safety knife	1 (TRUE)
Medium Titanium knife	1 (TRUE)
	NULL

INDEXOF

値リストから値のインデックスを検索します。デフォルトでは、大文字と小文字を区別します。

構文

INDEXOF(*valueToSearch, string1*, [*string2*, ..., *stringN*,] *CaseFlag*)

引数	必須/オプション	説明
valueToSearch	必須	文字列データ型。文字列リストで検索する値。
string	必須	文字列データ型。検索する値のカンマ区切りリスト。値は文字列形式で指定できます。リスト長に制限はありません。MatchCase をゼロに設定しなければ、値の大文字と小文字は区別されません。
CaseFlag	必須	整数でなければなりません。この関数の引数の大文字と小文字を区別するかどうかを指定します。有効な式を必要に応じて入力できます。 *CaseFlag* が 0 以外の数値の場合、大文字と小文字を区別します。 *CaseFlag* が NULL 値または 0 の場合、大文字と小文字を区別しません。

戻り値

入力値が *string1* に一致する場合は 1、入力値が *string2* に一致する場合は 2、以下同じ。

入力値が見つからない場合は 0 です。

NULL 値を入力した場合は、NULL です。

例

以下の式は、ITEM_NAME カラムの値が 1 番目、2 番目、または 3 番目の文字列に一致するかどうかを決定します。

```
INDEXOF( ITEM_NAME, 'diving hood', 'flashlight', 'safety knife')
```

ITEM_NAME	RETURN VALUE
Safety Knife	0
diving hood	1
Compass	0
safety knife	3
flashlight	2

Safety Knife は入力値の表記（大文字/小文字）に一致しないため、ゼロの値を返します。

INITCAP

文字列の各語の最初の文字を大文字に変換して、他の文字をすべて小文字に変換します。語と語の区切りは、スペース（空白、改ページ、改行、復帰、タブ、垂直タブ）、または英数字以外の文字です。例えば、文字列「...THOMAS」を渡すと、関数は Thomas を返します。

構文

```
INITCAP( string )
```

引数	必須/オプション	説明
string	必須	任意のデータ型（Binary を除く）。有効な式を必要に応じて入力できます。

戻り値

文字列。データにマルチバイト文字が含まれる場合、タスクを実行する Secure Agent のコードページに応じた戻り値が返されます。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、FIRST_NAME カラム内のすべての名前について、最初の文字を大文字にします。

```
INITCAP( FIRST_NAME )
```

FIRST_NAME	RETURN VALUE
ramona	Ramona
18-albert	18-Albert
NULL	NULL

FIRST_NAME	RETURN VALUE
?!SAM	?!Sam
THOMAS	Thomas
PierRe	Pierre

INSTR

文字列の中で、指定した文字が左から数えて何文字目にあるかを返します。

構文

INSTR(*string*, *search_value* [,*start* [,*occurrence*]])

引数	必須/オプション	説明
string	必須	文字列はすべて文字からなっていなければなりません。評価したい値を渡します。有効な式を必要に応じて入力できます。式の結果は文字列でなければなりません。文字列ではない場合、評価する前に INSTR が値を文字列に変換します。
search_value	必須	任意の値。検索値では、大文字と小文字が区別されます。検索したい文字または文字列を指定します。search_value は、文字列の一部に一致する必要があります。たとえば、INSTR('Alfred Pope', 'Alfred Smith')と記述すると、関数はゼロを返します。 有効な式を必要に応じて入力できます。文字列を検索したい場合は、'abc'のように検索したい文字列を一重引用符で囲みます。
start	オプション	整数値でなければなりません。文字列内で検索を開始する位置を示します。有効な式を必要に応じて入力できます。 デフォルト値は 1 です。つまり、INSTR は文字列の最初の文字から検索を開始します。 開始位置がゼロの場合、INSTR は文字列の最初の文字から検索を開始します。開始位置が正の数である場合、INSTR は文字列の先頭からその数だけ数えた位置から検索を開始します。開始位置が負の数である場合、INSTR は文字列の最後からその数だけ数えた位置から検索を開始します。この引数を省略すると、関数はデフォルト値の 1 を使用します。
occurrence	オプション	ゼロより大きい正の整数。有効な式を必要に応じて入力できます。検索値が文字列内に複数回出現する場合、何回目の出現を検索するかを指定できます。たとえば、開始位置から 2 回目に出現する検索値を検索する場合は、2 を入力します。 この引数を省略すると、関数はデフォルト値の 1 を使用します。この場合、INSTR は最初に出現する検索値を検索します。10 進数を渡すと、データ統合はその値を近似値の整数に丸めます。負の整数またはゼロを渡すと、ワークフローを実行したときにマッピングに失敗します。

戻り値

検索が成功した場合は、整数。この整数は、*search_value* の最初の文字が文字列内で左から数えて何文字目にあるかを示します。

検索に失敗した場合は、ゼロです。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、各会社名の先頭から数えて、文字‘a’ が最初に出現する位置を返します。*search_value* 引数では大文字と小文字が区別されるため、‘Blue Fin Aqua Center’の‘A’はスキップし、‘Aqua’の‘a’の位置を返します。

```
INSTR( COMPANY, 'a' )
```

COMPANY	RETURN VALUE
Blue Fin Aqua Center	13
Maco Shark Shop	2
Scuba Gear	5
Frank's Dive Shop	3
VIP Diving Club	0

次の式は、各会社名の先頭から数えて、文字‘a’が 2 回目に出現する位置を返します。*search_value* 引数では大文字と小文字が区別されるため、‘Blue Fin Aqua Center’の‘A’はスキップし、0 を返します。

```
INSTR( COMPANY, 'a', 1, 2 )
```

COMPANY	RETURN VALUE
Blue Fin Aqua Center	0
Maco Shark Shop	8
Scuba Gear	9
Frank's Dive Shop	0
VIP Diving Club	0

次の式は、各会社名の最後の文字から数えて、文字‘a’が 2 回目に出現する位置を返します。*search_value* 引数では大文字と小文字が区別されるため、'Blue Fin Aqua Center'の'A'はスキップし、0 を返します。

```
INSTR( COMPANY, 'a', -1, 2 )
```

COMPANY	RETURN VALUE
Blue Fin Aqua Center	0
Maco Shark Shop	2
Scuba Gear	5
Frank's Dive Shop	0
VIP Diving Club	0

次の式は、各会社名の最後の文字から数えて、文字列'Blue Fin Aqua Center'の最初の文字の位置を返します。

```
INSTR( COMPANY, 'Blue Fin Aqua Center', -1, 1 )
```

COMPANY	RETURN VALUE
Blue Fin Aqua Center	1
Maco Shark Shop	0
Scuba Gear	0
Frank's Dive Shop	0
VIP Diving Club	0

ネストされた INSTR の使用

INSTR 関数を他の関数の中にネストして、複雑な処理を実行できます。

次の式は、文字列を最後の文字から順に評価します。文字列内の最後（一番右側）のスペースを見つけて、それより左側にある文字をすべて返します。

```
SUBSTR( CUST_NAME,1,INSTR( CUST_NAME,' ' ,-1,1 ))
```

CUST_NAME	RETURN VALUE
PATRICIA JONES	PATRICIA
MARY ELLEN SHAH	MARY ELLEN

次の式は、文字列から文字#を削除します。

```
SUBSTR( CUST_ID, 1, INSTR(CUST_ID, '#')-1 ) || SUBSTR( CUST_ID, INSTR(CUST_ID, '#')+1 )
```

CUST_ID	RETURN VALUE
ID#33	ID33
#A3577	A3577
SS #712403399	SS 712403399

IS_DATE

文字列値が正しい日付であるかどうかを返します。

正しい日付とは、デフォルトの日付形式（MM/DD/YYYY HH24:MI:SS）になっている文字列を意味します。テストしたい文字列がデフォルトの日付形式でない場合は、TO_DATE フォーマット文字列を使用して日付の形式を指定します。IS_DATE に渡された文字列が指定されたフォーマット文字列に一致しない場合には、関数は FALSE（0）を返します。文字列がフォーマット文字列に一致する場合には、関数は TRUE（1）を返します。

IS_DATE は、文字列を評価して整数値を返します。

IS_DATE 式のターゲットカラムは、String データ型または Numeric データ型でなければなりません。

IS_DATE を使用して、フラットファイル内のデータをテストまたはフィルタリングしてから、ターゲットに書き込むこともできます。

YY フォーマットの文字列の代わりに、IS_DATE を含む RR フォーマットの文字列を使用します。ほとんどの場合、RR と YY のフォーマット文字列は同じ値を返しますが、場合により、YY が誤った結果を返すことがあります。たとえば、式 IS_DATE('02/29/00', 'YY')は内部で IS_DATE(02/29/1900 00:00:00)として計算され、FALSE を返します。ただし、データ統合は IS_DATE('02/29/00', 'RR')の式を IS_DATE(02/29/2000 00:00:00)と計算し、TRUE を返します。前者の場合、1900 年はうるう年ではないので、2 月 29 日はありません。

注: IS_DATE では、TO_DATE と同じフォーマット文字列を使用します。

構文

IS_DATE(*value* [,*format*])

引数	必須/オプション	説明
value	必須	文字列データ型であること。評価したい行を渡します。有効な式を必要に応じて入力できます。
format	オプション	正しい TO_DATE フォーマット文字列を入力します。フォーマット文字列は、*string* 引数の各部分と一致しなければなりません。たとえば、'Mar 15 1997 12:43:10AM'の文字列を渡す場合、フォーマット文字列'MON DD YYYY HH12:MI:SSAM'を使用する必要があります。フォーマット文字列を省略する場合は、文字列値がデフォルトの日付形式（MM/DD/YYYY HH24:MI:SS）になっていなければなりません。 TO_DATE フォーマット文字列および IS_DATE フォーマット文字列の詳細については、「TO_DATE および IS_DATE フォーマット文字列」 (ページ 35)を参照してください。

戻り値

行が正しい日付の場合は、TRUE（1）。

行が正しい日付でない場合は、FALSE（0）。

式の中に NULL の値がある場合、またはフォーマット文字列が NULL である場合は、NULL。

警告: フォーマット文字列は、日付区切り文字を含め、IS_DATE 文字列の形式と一致しなければなりません。一致しない場合、データ統合は不正確な値を返すか行をスキップする可能性があります。

例

次の式は、INVOICE_DATE カラムの値が正しい日付であるかどうかを確認します。

```
IS_DATE( INVOICE_DATE )
```

この式は、以下のようなデータを返します。

INVOICE_DATE	RETURN VALUE
NULL	NULL
'180'	0 (FALSE)
'04/01/98'	0 (FALSE)
'04/01/1998 00:12:15'	1 (TRUE)
'02/31/1998 12:13:55'	0 (FALSE) (*February does not have 31 days*)
'John Smith'	0 (FALSE)

次の IS_DATE 式では、フォーマット文字列'YYYY/MM/DD'が指定されています。

```
IS_DATE( INVOICE_DATE, 'YYYY/MM/DD' )
```

文字列がこの形式に一致しない場合、IS_DATE は FALSE を返します。

INVOICE_DATE	RETURN VALUE	
NULL	NULL	
'180'	0 (FALSE)	
'04/01/98'	0 (FALSE)	
'1998/01/12'	1 (TRUE)	
'1998/11/21 00:00:13'	0 (FALSE)	
'1998/02/31'	0 (FALSE)	(*February does not have 31 days*)
'John Smith'	0 (FALSE)	

次の例では、IS_DATE を使ってデータをテストしてから、TO_DATE を使って文字列を日付に変換する方法を示しています。この式は、INVOICE_DATE カラムの値を検査して、正しい日付をそれぞれ日付値に変換します。値が正しい日付ではない場合、データ統合は ERROR を返し、その行をスキップします。

この例は Date/Time 値を返します。そのため、式のターゲットカラムは Date/Time 型でなければなりません。

```
IIF( IS_DATE ( INVOICE_DATE, 'YYYY/MM/DD' ), TO_DATE( INVOICE_DATE ), ERROR('Not a valid date' ) )
```

INVOICE_DATE	RETURN VALUE
NULL	NULL
'180'	'Not a valid date'
'04/01/98'	'Not a valid date'
'1998/01/12'	1998/01/12
'1998/11/21 00:00:13'	'Not a valid date'
'1998/02/31'	'Not a valid date'
'John Smith'	'Not a valid date'

IS_NUMBER

文字列が正しい数値であるかどうかを返します。

正しい数値は、以下の要素から構成されています。

- 数値の前のスペース（なくともよい）
- 符号（+/-）（なくともよい）
- 1 つまたは複数の数字、および小数点（なくともよい）

- 科学的記法（なくともよい）。たとえば、文字‘e’または‘E’（Windows では文字‘d’または‘D’）に符号（+/-）が続き、そのあとに 1 つまたは複数の数字が続く（符号はなくともよい）。
- 数値のあとのスペース（なくともよい）

以下のものはすべて正しい数値です。

```
' 100  '
'   +100'
'-100'
'-3.45e+32'
'+3.45E-32'
'+3.45d+32' (Windows only)
'+3.45D-32' (Windows only)
'.6804'
```

IS_NUMBER 式のターゲットカラムは、String データ型または Numeric データ型でなければなりません。

IS_NUMBER を使用して、フラットファイル内のデータをテストまたはフィルタリングしてから、ターゲットに書き込むこともできます。

構文

IS_NUMBER(*value*)

引数	必須/オプション	説明
value	必須	文字列データ型であること。評価したい行を渡します。有効な式を必要に応じて入力できます。

戻り値

行が正しい数値の場合は、TRUE（1）。

行が正しい数値でない場合は、FALSE（0）。

式の中の値が NULL 値である場合は、NULL。

例

次の式は、ITEM_PRICE カラムの値が正しい数値であるかどうかを確認します。

IS_NUMBER(ITEM_PRICE)

ITEM_PRICE	RETURN VALUE
'123.00'	1 (True)
'-3.45e+3'	1 (True)
'-3.45D-3'	1 (True - Windows only)
'-3.45d-3'	0 (False - UNIX only)
'3.45E-'	0 (False) *Incomplete number*
' '	0 (False) Consists entirely of whitespace
''	0 (False) *Empty string*
'+123abc'	0 (False)
' 123'	1 (True) *Leading whitespace*

ITEM_PRICE	RETURN VALUE
'123 '	1 (True) *Trailing whitespace*
'ABC'	0 (False)
'-ABC'	0 (False)
NULL	NULL

TO_FLOAT などの数値変換関数を実行する前に、IS_NUMBER を使用してデータをテストできます。例えば、次の式は ITEM_PRICE カラムの値を検査し、有効な数字を倍精度の浮動小数点値に変換します。値が有効な数値でない場合、データ統合は 0.00 を返します。

```
IIF( IS_NUMBER ( ITEM_PRICE ), TO_FLOAT( ITEM_PRICE ), 0.00 )
```

ITEM_PRICE	RETURN VALUE
'123.00'	123
'-3.45e+3'	-3450
'3.45E-3'	0.00345
' '	0.00 *Consists entirely of whitespace*
''	0.00 *Empty string*
'+123abc'	0.00
'' 123ABC'	0.00
'ABC'	0.00
'-ABC'	0.00
NULL	NULL

IS_SPACES

文字列値がスペースだけで構成されているかどうかを返します。スペースとは、空白、改ページ、改行、復帰、タブ、または垂直タブです。

IS_SPACES は、空の文字列を FALSE と評価します。これは、空の文字列にはスペースが含まれないためです。空文字列かどうかをテストするには、LENGTH を使用します。

構文

IS_SPACES(*value*)

引数	必須/オプション	説明
value	必須	文字列データ型であること。評価したい行を渡します。有効な式を必要に応じて入力できます。

戻り値

行がスペースだけから構成されている場合は、TRUE（1）。

行にデータが含まれている場合は、FALSE（0）。

式の中の値が NULL 値である場合は、NULL。

例

次の式は、ITEM_NAME カラムを検査して、スペースだけから構成されている行を探します。

IS_SPACES(ITEM_NAME)

ITEM_NAME	RETURN VALUE
Flashlight	0 (False)
	1 (True)
Regulator system	0 (False)
NULL	NULL
''	0 (FALSE) (*Empty string does not contain spaces.*)

ヒント

IS_SPACES を使用して、ターゲットテーブル内の文字を取る列にスペースを書き込むことを回避します。例えば、ターゲットテーブル内の固定長 CHAR(5)カラムに顧客名を書き込む場合、スペースの代わりに‘00000’を書き込む場合もあります。その場合、以下のような式を作成できます。

IIF(IS_SPACES(CUST_NAMES), '00000', CUST_NAMES)

ISNULL

値が NULL であるかどうかを返します。ISNULL は空文字列を FALSE として評価します。

注: 空文字列かどうかをテストするには、LENGTH を使用します。

構文

ISNULL(*value*)

引数	必須/オプション	説明
value	必須	任意のデータ型（Binary を除く）。評価したい行を渡します。有効な式を必要に応じて入力できます。

戻り値

値が NULL の場合は、TRUE（1）。

値が NULL でない場合は、FALSE（0）。

例

次の例は、商品テーブル内の NULL 値を確認します。

ISNULL(ITEM_NAME)

ITEM_NAME	RETURN VALUE
Flashlight	0 (FALSE)
NULL	1 (TRUE)
Regulator system	0 (FALSE)
''	0 (FALSE) *Empty string is not NULL*

LAG

詳細モードでは、LAG 関数は式トランスフォーメーションの前の行からのデータを返します。この関数を使用して、現在の行の値と、前の行の値を比較します。

LAG 関数を使用するには、パーティションキーとオーダーキーをウィンドウプロパティとして設定し、式トランスフォーメーションでも設定を行う必要があります。

構文

LAG (*column_name*, *offset*, *default*)

以下の表に、このコマンドの引数を示します。

引数	必須/ オプション	説明
column_name	必須	関数で操作する対象の列または式。
offset	必須	整数データ型。値を取得する、現在の行の前の行数。
default	オプション	オフセットがパーティションまたはテーブルの範囲外にある場合に返されるデフォルト値。デフォルトは NULL です。 入力値と同じデータ型である引数、またはオフセット引数を使用したデフォルトの引数を指定できます。 複合データ型を含むデフォルトの引数、または SYSTIMESTAMP 引数を指定することはできません。

戻り値

指定された *column_name* のデータ型。

戻り値が指定されたパーティションの境界の外側にある場合は *default*。

default が省略されている場合、または NULL に設定されている場合は、NULL。

例 1

次の式は、前の順序が配置された日付を返します。

```
LAG ( ORDER_DATE, 1, NULL )
```

次の表に、このコマンドの順序に関する情報を示します。

ORDER_DATE	ORDER_ID	RETURN VALUE
2017/09/25	1	NULL
2017/09/26	2	2017/09/25
2017/09/27	3	2017/09/26
2017/09/28	4	2017/09/27
2017/09/29	5	2017/09/28
2017/09/30	6	2017/09/29

最初の行の遅延値はパーティションの範囲外であるため、関数はデフォルトの NULL 値を返します。

例 2

あなたの組織はトリップ ID、イベント ID、およびタイムスタンプを含む車両からの GPS ping を受信します。各 ping の時間差を計算するとします。

次の式は、2 つの別々のトリップについて、現在の行と前の行の時間差を計算します。

```
DATE_DIFF( EVENT_TIME, LAG ( EVENT_TIME, 1, NULL ), 'ss' )
```

データをトリップ ID でパーティション化し、イベント ID で順序付けします。

次の表に、このコマンドのトリップに関する情報を示します。

TRIP_ID	EVENT_ID	EVENT_TIME	RETURN VALUE
101	1	2017-05-03 12:00:00	NULL
101	2	2017-05-03 12:00:34	34
101	3	2017-05-03 12:02:00	86
102	1	2017-05-03 12:00:00	NULL
102	2	2017-05-03 12:01:56	116
102	3	2017-05-03 12:02:00	4

最初の行および 4 番目の行の遅延値は指定されたパーティションの範囲外であるため、関数はデフォルトの NULL 値を 2 つ返します。

LAST

選択したフィールドの最後の行を返します。オプションとして、読み込む行を制限するフィルタを適用できます。LAST の中にネストできる他の集計関数は 1 つだけです。

マッピングタスクでのみ使用します。

構文

LAST(*value* [, *filter_condition*])

引数	必須/オプション	説明
value	必須	Binary 以外の任意のデータ型。最後の行を返したい値を渡します。有効なトランスフォーメーション式を必要に応じて入力できます。
filter_condition	オプション	検索される行を制限します。フィルタ条件は数値であるか、TRUE、FALSE、または NULL の値が求められなければなりません。有効なトランスフォーメーション式を必要に応じて入力できます。

戻り値

フィールドの最後の行。

関数に渡された値がすべて NULL である場合、または行が 1 つも選択されていない場合（たとえば、フィルタ条件の値がすべての行に対して FALSE または NULL であった場合）には、NULL です。

例

次の式は、ITEMS_NAME フィールド内で価格が$10.00 を超える最後の行を返します。

```
LAST( ITEM_NAME, ITEM_PRICE > 10 )
```

ITEM_NAME	ITEM_PRICE
Flashlight	35.00
Navigation Compass	8.05
Regulator System	150.00
Flashlight	29.00
Depth/Pressure Gauge	88.00
Vest	31.00

RETURN VALUE: Vest

LAST_DAY

カラム内の各日付に対して、その月の最後の日の日付を返します。

構文

```
LAST_DAY( date )
```

引数	必須/オプション	説明
date	必須	Date/Time データ型。月の最後の日を返したい日付を渡します。日付を求める有効な式を必要に応じて入力できます。

戻り値

日付。関数に渡した日付値に対する月の最後の日。

選択したカラム内の値が NULL である場合は、NULL。

Null

値が NULL であると、LAST_DAY はその行を無視します。ただし、カラムから渡された値がすべて NULL である場合には、NULL を返します。

例

次の式は、ORDER_DATE カラム内の各日付に対して、月の最後の日を返します。

```
LAST_DAY( ORDER_DATE )
```

ORDER_DATE	RETURN VALUE
Apr 1 1998 12:00:00AM	Apr 30 1998 12:00:00AM

ORDER_DATE	RETURN VALUE
Jan 6 1998 12:00:00AM	Jan 31 1998 12:00:00AM
Feb 2 1996 12:00:00AM	Feb 29 1996 12:00:00AM *(Leap year)*
NULL	NULL
Jul 31 1998 12:00:00AM	Jul 31 1998 12:00:00AM

TO_DATE をネストして文字列値を日付に変換できます。TO_DATE には常に時刻の情報があります。時刻値を含まない文字列を渡すと、返される日付には時刻「00:00:00」を含みます。

次の例は、各注文日に対して、月の最後の日を文字列と同じ形式で返します。

```
LAST_DAY( TO_DATE( ORDER_DATE, 'DD-MON-YY' ))
```

ORDER_DATE	RETURN VALUE
'18-NOV-98'	Nov 30 1998 00:00:00
'28-APR-98'	Apr 30 1998 00:00:00
NULL	NULL
'18-FEB-96'	Feb 29 1996 00:00:00 *(Leap year)*

LEAD

詳細モードでは、LEAD 関数は式トランスフォーメーションの次の行からのデータを返します。この関数を使用して、現在の行の値と、次の行の値を比較します。

LEAD 関数を使用するには、パーティションキーとオーダーキーをウィンドウプロパティとして設定し、式トランスフォーメーションでも設定を行う必要があります。

構文

LEAD (*column_name*, *offset*, *default*)

以下の表に、このコマンドの引数を示します。

引数	必須/ オプション	説明
column_name	必須	関数で操作する対象の列または式。
offset	必須	整数データ型。値を取得する、現在の行の前の行数。
default	オプション	オフセットがパーティションまたはテーブルの範囲外にある場合に返されるデフォルト値。デフォルトは NULL です。 入力値と同じデータ型である引数、またはオフセット引数を使用したデフォルトの引数を指定できます。 複合データ型を含むデフォルトの引数、または SYSTIMESTAMP 引数を指定することはできません。

戻り値

指定された *column_name* のデータ型。

戻り値が指定されたパーティションの境界の外側にある場合は *default*。

default が省略されている場合、または NULL に設定されている場合は、NULL。

例 1

この例は、従業員名と雇入日のデータを LEAD 関数で使用した場合を示したものです。

次の表に、従業員に関する情報を示します。

EMPLOYEE	HIRE_DATE	RETURN VALUE
Hynes	2012/12/07	2014/05/18
Williams	2014/05/18	2015/07/24
Pritchard	2015/07/24	2015/12/24
Snyder	2015/12/24	2016/11/15
Troy	2016/11/15	2017/08/10
Randolph	2017/08/10	NULL

それぞれの従業員について、以下の式は次の従業員が雇用された日付を返します。

```
LEAD ( HIRE_DATE, 1, NULL )
```

最後の行には先行する値がないため、関数はデフォルトの NULL 値を返します。

例 2

この例は、2 暦年の販売ノルマの値を LEAD 関数で使用した場合を示したものです。

データを年でパーティション化し、四半期で順序付けします。

次の表に、関数で使用される販売ノルマのデータを示します。

YEAR	QUARTER	SALES_QUOTA	QUOTA_DIFF
2016	1*	300	7700
2016	2*	7000	0
2016	3	8000	0
2017	1	5000	4000
2017	2	400	0
2017	3	9000	0

**第 2 四半期と第 3 四半期の行に先行する値は指定されたパーティションの範囲外であるため、関数は値「0」を返します。*

次の式は、2 つの暦年の第 1 四半期から第 3 四半期までの販売ノルマの値の差を返します。

```
LEAD ( Sales_Quota, 2, 0 ) - Sales_Quota
```

LEAST

入力値のリストから最小値を返します。デフォルトでは、大文字と小文字を区別します。

構文

```
LEAST( value1, [value2, ..., valueN,] CaseFlag )
```

引数	必須/オプション	説明
value	必須	任意のデータ型（Binary を除く）。データ型は他の値との互換性が必要です。他の値と比較しなければならない値。最低 1 つの値引数を入力する必要があります。 この値が Numeric で、他の入力値が他の数値データ型の場合、すべての値では可能な限り高い精度が適用されます。例えば、一部の値が Integer データ型で他の値が Double データ型である場合、データ統合はその値を Double データ型に変換します。
CaseFlag	オプション	整数でなければなりません。この関数の引数の大文字と小文字を区別するかどうかを指定します。有効な式を必要に応じて入力できます。 *>CaseFlag* が 0 以外の数値の場合、大文字と小文字を区別します。 *>CaseFlag* が NULL 値または 0 の場合、大文字と小文字を区別しません。

戻り値

入力値の中で最小である場合は *value1*、入力値の中で最小である場合は *value2* など。

すべての引数が NULL の場合は、NULL です。

例

以下の式は、注文した項目の最小数を返します。

```
LEAST( QUANTITY1, QUANTITY2, QUANTITY3 )
```

QUANTITIY1	QUANTITY2	QUANTITY3	RETURN VALUE
150	756	27	27
			NULL
5000	97	17	17
120	1724	965	120

LENGTH

文字列内の文字数を返します。文字列の末尾の空白も含めます。

構文

LENGTH(*string*)

引数	必須/オプション	説明
string	必須	文字列データ型。評価したい文字列。有効な式を必要に応じて入力できます。

戻り値

文字列の長さを示す整数。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、各顧客名の長さを返します。

```
LENGTH( CUSTOMER_NAME )
```

CUSTOMER_NAME	RETURN VALUE
Bernice Davis	13
NULL	NULL
John Baer	9
Greg Brown	10

ヒント

LENGTH を使用して、空の文字列の条件をテストします。顧客名が空であるフィールドを見つけたい場合には、次のような式を使用します。

```
IIF( LENGTH( CUSTOMER_NAME ) = 0, 'EMPTY STRING' )
```

フィールドが NULL かどうかをテストするには、ISNULL を使用します。スペースについてテストするには、IS_SPACES を使用します。

LN

数値の自然対数を返します。

例えば、LN(3)は 1.098612 を返します。通常、この関数はビジネスデータではなく科学技術データの分析に使用されます。

この関数は、関数 EXP の逆関数です。

構文

LN(*numeric_value*)

引数	必須/オプション	説明
numeric_value	必須	数値データ型。これはゼロより大きい正の数でなければなりません。自然対数を計算したい値を渡します。有効な式を必要に応じて入力できます。

戻り値

Double 値。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、NUMBERS カラムのすべての値に対して自然対数を返します。

```
LN( NUMBERS )
```

NUMBERS	RETURN VALUE
10	2.302585092994
125	4.828313737302
0.96	-0.04082199452026
NULL	NULL
-90	None. (データ統合 writes the row to the error rows file.)
0	None. (データ統合 writes the row to the error rows file.)

注: 負の数または 0 を渡した場合、データ統合ではその行がエラー行ファイルに書き込まれます。*numeric_value* は、ゼロより大きい正の数でなければなりません。

LOG

数値の対数を返します。

一般に、この関数はビジネスデータではなく科学技術データの分析に使用されます。

構文

LOG(*base*, *exponent*)

引数	必須/オプション	説明
base	必須	対数の基数。0 または 1 以外の正の数値でなければなりません。0 または 1 以外の正の数値を求める有効な式。
exponent	必須	対数の指数。0 より大きい正の数値でなければなりません。0 より大きい正の数値を求める有効な式。

戻り値

Double 値。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、NUMBERS カラムのすべての値に対して対数を返します。

LOG(BASE, EXPONENT)

BASE	EXPONENT	RETURN VALUE
15	1	0
.09	10	-0.956244644696599
NULL	18	NULL
35.78	NULL	NULL
-9	18	*None. (データ統合 writes the row to the error rows file.)*
0	5	*None. (データ統合 writes the row to the error rows file.)*
10	-2	*None. (データ統合 writes the row to the error rows file.)*

基数値として負の数、0、または 1 を渡した場合、または指数に負の値を渡した場合、データ統合ではその行がエラー行ファイルに書き込まれます。

LOWER

大文字の文字列を小文字に変換します。

構文

LOWER(*string*)

引数	必須/オプション	説明
string	必須	任意の文字列値。この引数は、小文字として返したい文字列値を渡します。文字列を求める有効な式を必要に応じて入力できます。

戻り値

小文字の文字列。データにマルチバイト文字が含まれる場合、タスクを実行する Secure Agent のコードページに応じた戻り値が返されます。

選択したカラム内の値が NULL である場合は、NULL。

例

次の式は、名前をすべて小文字で返します。

LOWER(FIRST_NAME)

FIRST_NAME	RETURN VALUE
antonia	antonia
NULL	NULL
THOMAS	thomas
PierRe	pierre
BERNICE	bernice

LPAD

文字列の先頭にいくつかの空白または文字を追加して、文字列を指定した長さにします。

構文

LPAD(*first_string*, *length* [,*second_string*])

引数	必須/オプション	説明
first_string	必須	文字列。変更したい文字列を渡します。有効な式を必要に応じて入力できます。
length	必須	正の整数リテラルでなければなりません。この引数は、各文字列の希望の長さを指定します。*length* が負の数である場合、RPAD は NULL を返します。
second_string	オプション	任意の文字列値。*first_string* 値の左側に付加したい文字列。有効な式を必要に応じて入力できます。特定の文字列リテラルを入力できます。ただし、文字列の先頭に追加する文字は、'abc'のように一重引用符で囲みます。この引数は大文字と小文字を区別します。*second_string* を省略すると、関数は最初の文字列の先頭に空白を付加します。

戻り値

指定された長さの文字列。

関数に NULL 値を渡した場合、あるいは *length* が負の数の場合は、NULL です。

例

次の式は、数値の先頭にゼロを付加して、数値を 6 桁に標準化します。

```
LPAD( PART_NUM, 6, '0')
```

PART_NUM	RETURN VALUE
702	000702
1	000001
0553	000553
484834	484834

LPAD は、長さを左から数えます。元の文字列が指定した長さよりも長い場合は、文字列が右側から切り詰められます。たとえば、LPAD('alphabetical', 5, 'x')は文字列'alpha'を返します。

付加する文字列が、指定された長さを返すために必要な文字数よりも長い場合は、その文字列の一部分が使用されます。

```
LPAD( ITEM_NAME, 16, '*..*' )
```

ITEM_NAME	RETURN VALUE
Flashlight	*..**.Flashlight
Compass	*..**..**Compass

ITEM_NAME	RETURN VALUE
Regulator System	Regulator System
Safety Knife	*..*Safety Knife

次の式は、ITEM_NAME カラムの各行について、length 引数が負の数の場合に LPAD がどのように処理するかを示します。

```
LPAD( ITEM_NAME, -5, '.')
```

ITEM_NAME	RETURN VALUE
Flashlight	NULL
Compass	NULL
Regulator System	NULL

LTRIM

文字列の先頭から空白または文字を削除します。LTRIM は、式内で IIF または DECODE とともに使用され、ターゲットテーブルにスペースが入るのを防ぎます。

式で *trim_set* パラメータを指定しない場合、LTRIM はシングルバイトのスペースのみを削除します。

LTRIM を使用して文字列から文字を削除する場合、LTRIM は *trim_set* を *string* 引数内の各文字と左から 1 文字ずつ比較します。文字列内の文字が *trim_set* 内のいずれかの文字と一致した場合、LTRIM はその文字を削除します。LTRIM は、一致する文字が *trim_set* で見つからなくなるまで、文字を比較して削除します。その後、一致する文字が含まれない文字列を返します。

構文

```
LTRIM( string [, trim_set] )
```

引数	必須/オプション	説明
string	必須	任意の文字列値。変更したい文字列を渡します。有効な式を必要に応じて入力できます。文字列の先頭から文字を削除する前に、演算子を使って文字列の比較や連結を実行します。
trim_set	オプション	任意の文字列値。文字列の先頭から削除したい文字を渡します。有効な式を必要に応じて入力できます。文字列も入力できます。ただし、文字列の先頭から削除する文字は、'abc'のように一重引用符で囲む必要があります。削除したい文字を省略すると、関数は文字列の先頭から空白を削除します。 LTRIM では、大文字と小文字が区別されます。たとえば、文字列'Alfredo'から文字'A'を削除したい場合は、必ず'a'ではなく'A'と入力します。

戻り値

文字列。*trim_set* 引数で指定された文字を削除した結果の文字列値。

関数に NULL 値を渡した場合は NULL です。*trim_set* が NULL の場合、関数は NULL を返します。

例

次の式は、LAST_NAME カラムの文字列から文字‘S’と‘.’を削除します。

```
LTRIM( LAST_NAME, 'S.')
```

LAST_NAME	RETURN VALUE
Nelson	Nelson
Osborne	Osborne
NULL	NULL
S. MacDonald	MacDonald
Sawyer	awyer
H. Bender	H. Bender
Steadman	teadman

S. MacDonald から‘S.’を削除し、Sawyer および Steadman から‘S’を削除しますが、H. Bender からはピリオドを削除しません。これは、LTRIM では *trim_set* 引数に指定された文字列を 1 文字ずつ検索していくからです。文字列内の最初の文字が *trim_set* 内の最初の文字と一致した場合、LTRIM はその文字を削除します。その後、LTRIM は文字列内の 2 番目の文字を見ます。それが *trim_set* 内の 2 番目の文字と一致していれば、それを削除します。3 文字目以降も同様に進みます。文字列内の最初の文字が *trim_set* で対応する文字と一致しなかった場合、LTRIM はこの文字列を返して、次の行の評価を行います。

H. Bender の場合は、H が *trim_set* 引数のどの文字とも一致しないため、LTRIM は LAST_NAME カラム内の文字列を返して、次の行に移ります。

ヒント

RTRIM および LTRIM を || または CONCAT とともに使用すると、2 つの文字列を連結したあとで先頭および末尾の空白を削除します。

また、LTRIM をネストして複数組の文字を削除することもできます。たとえば、名前の列から先頭の空白と文字'T'とを削除したい場合は、次のような式を作成します。

```
LTRIM( LTRIM( NAMES ), 'T' )
```

MAKE_DATE_TIME

入力値に基づく日付と時間を返します。

構文

MAKE_DATE_TIME(*year, month, day, hour, minute, second*)

引数	必須/オプション	説明
year	必須	数値データ型。正の整数。
month	必須	数値データ型。1 から 12 までの正の整数（1 月=1、12 月=12）。
day	必須	数値データ型。1 から 31 までの正の整数（日数が 31 日未満の月(2 月、4 月、6 月、9 月、および 11 月)を除く）。
hour	オプション	数値データ型。0 から 24 までの正の整数（0=12AM、12=12PM、24 =12AM）。
minute	オプション	数値データ型。0 から 59 までの正の整数。
second	オプション	数値データ型。0 から 59 までの正の整数。

戻り値

MM/DD/YYYY HH24:MI:SS の日付。

例

以下の式は、ソースカラムから日付と時間を作成します。

MAKE_DATE_TIME(SALE_YEAR, SALE_MONTH, SALE_DAY, SALE_HOUR, SALE_MIN, SALE_SEC)

SALE_YR	SALE_MTH	SALE_DAY	SALE_HR	SALE_MIN	SALE_SEC	RETURN VALUE
2002	10	27	8	36	22	10/27/2002 08:36:22
2000	6	15	15	17		06/15/200 15:17:00
2003	1	3		22	45	01/03/2003 ??:22:45
04	3	30	12	5	10	03/30/2004 12:05:10
99	12	12	5		16	12/12/1999?? 05:??:16

MAX (Dates)

フィールドまたはグループ内で見つかった最新の日付を返します。検索において、行を制限するフィルタを適用できます。MAX を使用して、フィールド内またはグループ内における最大の数値もしくは最高の文字列値を返すこともできます。

MAX の中にネストできる他の集計関数は 1 つだけです。詳細モードでは集計関数をネストできません。

マッピングタスクでのみ使用します。

構文

MAX(*date* [, *filter_condition*])

引数	必須/オプション	説明
date	必須	Date/Time データ型。最大の日付を返したい日付を渡します。有効なトランスフォーメーション式を必要に応じて入力できます。
filter_condition	オプション	検索される行を制限します。フィルタ条件は数値であるか、TRUE、FALSE、または NULL の値が求められなければなりません。有効なトランスフォーメーション式を必要に応じて入力できます。

戻り値

日付。

関数に渡された値がすべて NULL である場合、または行が 1 つも選択されていない場合（たとえば、フィルタ条件の値がすべての行に対して FALSE または NULL であった場合）には、NULL です。

例

フィールドまたはグループに対して、最大の日付を返すことができます。次の式は、懐中電灯の最新の注文日を返します。

MAX(ORDERDATE, ITEM_NAME='Flashlight')

ITEM_NAME	ORDER_DATE
Flashlight	Apr 20 1998
Regulator System	May 15 1998
Flashlight	Sep 21 1998
Diving Hood	Aug 18 1998
Flashlight	NULL

RETURN VALUE: Sep 21 1998

MAX (Numbers)

フィールドまたはグループ内の最大の数値を返します。検索において、行を制限するフィルタを適用できます。MAX を使用して、フィールド内またはグループ内における直近の日付もしくは最高の文字列値を返すこともできます。

MAX の中にネストできる他の集計関数は 1 つだけです。詳細モードでは集計関数をネストできません。

マッピングタスクでのみ使用します。

構文

MAX(*numeric_value* [, *filter_condition*])

引数	必須/オプション	説明
numeric_value	必須	数値データタイプ。最大値を返したい数値を渡します。有効なトランスフォーメーション式を必要に応じて入力できます。
filter_condition	オプション	検索される行を制限します。フィルタ条件は数値であるか、TRUE、FALSE、または NULL の値が求められなければなりません。有効なトランスフォーメーション式を必要に応じて入力できます。

戻り値

数値。

関数に渡された値がすべて NULL である場合、または行が 1 つも選択されていない場合（たとえば、フィルタ条件の値がすべての行に対して FALSE または NULL であった場合）には、NULL です。

NULL

値が NULL であると、MAX はその値を無視します。ただし、フィールドから渡された値がすべて NULL である場合には、NULL を返します。

Group By

MAX は、トランスフォーメーションで定義したグループ化フィールドに基づいて値をグループ分けし、各グループについて 1 つの結果を返します。

グループ化フィールドがない場合には、MAX はすべての行を 1 つのグループとして扱い、1 つの値を返します。

例

次の式は、懐中電灯の最大価格を返します。

```
MAX( PRICE, ITEM_NAME='Flashlight' )
```

ITEM_NAME	PRICE
Flashlight	10.00
Regulator System	360.00
Flashlight	55.00
Diving Hood	79.00
Halogen Flashlight	162.00
Flashlight	85.00
Flashlight	NULL

RETURN VALUE: 85.00

MAX (String)

フィールドまたはグループ内における最大の文字列値を返します。検索において、行を制限するフィルタを適用できます。MAX を使用して、フィールド内またはグループ内における最新の日付もしくは最大の数値を返すこともできます。

注: MAX 関数は、Sorter トランスフォーメーションと同じソート順を使用します。ただし、MAX 関数では大文字と小文字が区別されますが、Sorter トランスフォーメーションでは区別されない場合もあります。

MAX の中にネストできる他の集計関数は 1 つだけです。詳細モードでは集計関数をネストできません。

マッピングタスクでのみ使用します。

構文

MAX(*string* [, *filter_condition*])

引数	必須/オプション	説明
string	必須	String データ型。最大の文字列値を返す文字列値を渡します。有効なトランスフォーメーション式を必要に応じて入力できます。
filter_condition	オプション	検索される行を制限します。フィルタ条件は数値であるか、TRUE、FALSE、または NULL の値が求められなければなりません。有効なトランスフォーメーション式を必要に応じて入力できます。

戻り値

文字列。

関数に渡された値がすべて NULL である場合、または行が 1 つも選択されていない場合（たとえば、フィルタ条件の値がすべての行に対して FALSE または NULL であった場合）には、NULL です。

NULL

値が NULL であると、MAX はその値を無視します。ただし、フィールドから渡された値がすべて NULL である場合には、NULL を返します。

Group By

MAX は、トランスフォーメーションで定義したグループ化フィールドに基づいて値をグループ分けし、各グループについて 1 つの結果を返します。

グループ化フィールドがない場合には、MAX はすべての行を 1 つのグループとして扱い、1 つの値を返します。

例

以下の式は、メーカー ID 104 の最大の項目名を返します。

```
MAX( ITEM_NAME, MANUFACTURER_ID='104' )
```

MANUFACTURER_ID	ITEM_NAME
101	First Stage Regulator
102	Electronic Console
104	Flashlight

MANUFACTURER_ID	ITEM_NAME
104	Battery (9 volt)
104	Rope (20 ft)
104	60.6 cu ft Tank
107	75.4 cu ft Tank
108	Wristband Thermometer

RETURN VALUE: Rope (20 ft)

MD5

入力値のチェックサムを計算します。この関数は、MD5（Message-Digest アルゴリズム 5）を使用しています。MD5 は、ハッシュ値が 128 ビットの一方向暗号ハッシュ関数です。それぞれの入力に対し、一意の値を計算します。MD5 を使用して、データの完全性を確認します。

構文

MD5(*value*)

引数	必須/オプション	説明
value	必須	データ型は String または Binary。チェックサムを計算する値です。入力値の大文字と小文字が区別されると、戻り値に影響します。たとえば、MD5(informatica)と MD5 (Informatica)では異なる戻り値が返されます。

戻り値

0 から 9 および a から f を使用した、16 進数の一意の 32 文字の文字列。

NULL 値を入力した場合は、NULL です。

例

変更したデータをデータベースに書き込む必要があります。ソースからデータを読み込むたびに、MD5 関数を使用して、データ行に一意のチェックサム値を生成できます。同じソースからデータを読み込む新規セッションを実行する場合、前に生成したチェックサム値と新しいチェックサム値を比較できます。その後、新しいチェックサム値が生成された行をターゲットに書き込むことができます。これらの行は、ソース内で変更されたデータを表します。

ヒント

戻り値は、ハッシュキーとして使用できます。

MEDIAN

選択されたフィールドのすべての値の中央値を返します。

フィールド内の値の個数が偶数個である場合、メジアンは、すべての値を数直線上に順番に並べたときに真ん中に位置する 2 つの値の平均となります。フィールド内の値の個数が奇数個である場合は、メジアンは真ん中の数値になります。

MEDIAN には他の集計関数は 1 つしかネストできません。また、ネストされた関数は数値データ型を返す必要があります。詳細モードでは集計関数をネストできません。

データ統合は、データのすべての行を読み取り、中央値の計算を実行します。計算を実行するエレメント数。オプションとして、メジアンを計算するために読み込む行を制限するフィルタを適用できます。

マッピングタスクでのみ使用します。

構文

```
MEDIAN( numeric_value [, filter_condition ] )
```

引数	必須/オプション	説明
numeric_value	必須	数値データタイプ。メジアンを計算したい値を渡します。有効なトランスフォーメーション式を必要に応じて入力できます。
filter_condition	オプション	検索される行を制限します。フィルタ条件は数値であるか、TRUE、FALSE、または NULL の値が求められなければなりません。有効なトランスフォーメーション式を必要に応じて入力できます。

戻り値

数値。

関数に渡された値がすべて NULL である場合、または行が 1 つも選択されていない場合には、NULL となります。たとえば、すべての行に対するフィルタ条件の値が FALSE または NULL です。

NULL

値が NULL であると、MEDIAN はその行を無視します。ただし、フィールドから渡された値がすべて NULL である場合には、NULL を返します。

Group By

MEDIAN は、トランスフォーメーションで定義したグループ化フィールドに基づいて値をグループ分けし、各グループについて 1 つの結果を返します。

グループ化フィールドがない場合には、MEDIAN はすべての行を 1 つのグループとして扱い、1 つの値を返します。

例

すべての部署についての給与のメジアンを計算するには、次の式を指定したフィールドを使って、部署ごとにグループ分けされた Aggregator トランスフォーメーションを作成します。

```
MEDIAN( SALARY )
```

次の式は、固定ベストの注文についてのメジアン値を返します。

```
MEDIAN( SALES, ITEM = 'Stabilizing Vest' )
```

ITEM	SALES
Flashlight	85
Stabilizing Vest	504
Stabilizing Vest	36
Safety Knife	5
Medium Titanium Knife	150
Tank	NULL
Stabilizing Vest	441
Chisel Point Knife	60
Stabilizing Vest	NULL
Stabilizing Vest	1044
Wrist Band Thermometer	110

RETURN VALUE: 472.5

METAPHONE

文字列値をエンコードします。エンコーディングする文字列の長さを指定することができます。

METAPHONE 関数は、英語アルファベット（A～Z）をエンコーディングします。大文字と小文字はどちらも大文字としてエンコーディングします。

METAPHONE は下記の規則に従って文字をエンコードします。

- 入力文字列の最初の文字を除き、母音（A、E、I、O、U）をスキップします。METAPHONE('CAR')は'KR'を返し、METAPHONE('AAR')は'AR'を返します。
- 特別なエンコードガイドラインを使用します。

以下の表に、METAPHONE エンコーディングのガイドラインを示します。

入力	戻り値	条件	例
B	なし	M の後。	METAPHONE ('Lamb')は LM を返します。
B	他の場合。	METAPHONE ('Box')は BKS を返します。	
C	X	IA または H の前。	METAPHONE ('Facial')は FXL を返します。

入力	戻り値	条件	例
S	I、E、または Y の前。	METAPHONE ('Fence')は FNS を返します。	
なし	S の後または I、E、Y の前。	METAPHONE ('Scene')は SN を返します。	
K	他の場合。	METAPHONE ('Cool')は KL を返します。	
D	J	GE、GY、または GI の前。	METAPHONE ('Dodge')は TJ を返します。
T	他の場合。	METAPHONE ('David')は TFT を返します。	
F	F	すべての場合。	METAPHONE ('FOX')は FKS を返します。
G	F	H の前で、入力文字列の最初の文字が B、D、H 以外。	METAPHONE ('Tough')は TF を返します。
なし	H の前で、入力文字列の最初の文字が B、D、または H。	METAPHONE ('Hugh')は HF を返します。	
J	I、E、または Y の前で、繰り返さない場合。	METAPHONE ('Magic')は MJK を返します。	
K	他の場合。	METAPHONE('GUN')は KN を返します。	
H	H	C、G、P、S、T の後でなく、A、E、I、U の前。	METAPHONE ('DHAT')は THT を返します。
なし	他の場合。	METAPHONE ('Chain')は XN を返します。	
J	J	すべての場合。	METAPHONE ('Jen')は JN を返します。
K	なし	C の後。	METAPHONE ('Ckim')は KM を返します。
K	他の場合。	METAPHONE ('Kim')は KM を返します。	
L	L	すべての場合。	METAPHONE ('Laura')は LR を返します。
M	M	すべての場合。	METAPHONE ('Maggi')は MK を返します。
N	N	すべての場合。	METAPHONE ('Nancy')は NNS を返します。
P	F	H の前。	METAPHONE ('Phone')は FN を返します。

入力	戻り値	条件	例
P	他の場合。	METAPHONE (‘Pip’)は PP を返します。	
Q	K	すべての場合。	METAPHONE (‘Queen’)は KN を返します。
R	R	すべての場合。	METAPHONE (‘Ray’)は R を返します。
S	X	H、IO、IA、または CHW の前。	METAPHONE (‘Cash’)は KX を返します。
S	他の場合。	METAPHONE (‘Sing’)は SNK を返します。	
T	X	IA または IO の前。	METAPHONE (‘Patio’)は PX を返します。
0 *	H の前。	METAPHONE (‘Thor’)は 0R を返します。	
なし	CH の前。	METAPHONE (‘Glitch’)は KLTX を返します。	
T	他の場合。	METAPHINE (‘Tim’)は TM を返します。	
インストール要件が満たされていないと、インストールに失敗することがあります。	F	すべての場合。	METAPHONE (‘Vin’)は FN を返します。
W	W	A、E、I、O、または U の前。	METAPHONE (‘Wang’)は WNK を返します。
なし	他の場合。	METAPHONE (‘When’)は HN を返します。	
X	KS	すべての場合。	METAPHONE (‘Six’)は SKS を返します。
Y	Y	A、E、I、O、または U の前。	METAPHONE (‘Yang’)は YNK を返します。
なし	他の場合。	METAPHONE (‘Bobby’)は BB を返します。	
Z	S	すべての場合。	METAPHONE (‘Zack’)は SK を返します。
*整数 0			

- 入力文字列の最初の 2 文字が下記の値のいずれかを含む場合、最初の文字をスキップし、文字列の残りの部分をエンコードします。
  - **KN。**例えば、METAPHONE(‘KNOT’)は‘NT’を返します。
  - **GN。**例えば、METAPHONE(‘GNOB’)は‘NB’を返します。

- **PN。**例えば、METAPHONE(‘PNRX’)は‘NRKS’を返します。
- **AE。**例えば、METAPHONE(‘AERL’)は‘ERL’を返します。

- 入力文字列内に「C」以外の文字が複数ある場合、最初の文字だけをエンコードします。例えば、METAPHONE(‘BBOX’)は‘BKS’を返し、METAPHONE(‘CCOX’)は‘KKKS’を返します。

構文

METAPHONE(*string* [,*length*])

引数	必須/オプション	説明
string	必須	文字列でなければなりません。エンコードしたい値を渡します。最初の文字は、英字（A - Z）でなければなりません。有効な式を必要に応じて入力できます。 *string* 内の英字以外のものはスキップします。
length	オプション	0 より大きい整数でなければなりません。エンコードする *string* 内の文字の数を指定します。有効な式を必要に応じて入力できます。 *length* がゼロ、または *string* の長さより大きな値の場合、入力文字列全体をエンコードします。 デフォルトは 0 です。

戻り値

文字列。

下記の条件のいずれかが真の場合、NULL となります。

- 関数に渡された値がすべて NULL である。
- *string* 内の文字が英字ではない。
- *string* が空である。

例

下記の式は、EMPLOYEE_NAME カラムの最初の 2 文字を文字列にエンコードします。

```
METAPHONE( EMPLOYEE_NAME, 2 )
```

Employee_Name	Return Value
John	JH
*@#$	NULL
P$%%oc&&KMNL	PK

下記の式は、EMPLOYEE_NAME カラムの最初の 4 文字を文字列にエンコードします。

```
METAPHONE( EMPLOYEE_NAME, 4 )
```

Employee_Name	Return Value
John	JHN
1ABC	ABK
*@#$	NULL

Employee_Name	Return Value
P$%%oc&&KMNL	PKKM

MIN (Dates)

フィールドまたはグループ内で最も古い日付を返します。検索において、行を制限するフィルタを適用できます。MIN を使用して、フィールド内またはグループ内における最小の数値もしくは最低の文字列値を返すこともできます。

MIN には他の集計関数は 1 つしかネストできません。また、ネストされた関数は Date データ型を返す必要があります。詳細モードでは集計関数をネストできません。

マッピングタスクでのみ使用します。

構文

MIN(*date* [, *filter_condition*])

引数	必須/オプション	説明
date	必須	Date/Time データ型。最小値を返したい値を渡します。有効なトランスフォーメーション式を必要に応じて入力できます。
filter_condition	オプション	検索される行を制限します。フィルタ条件は数値であるか、TRUE、FALSE、または NULL の値が求められなければなりません。有効なトランスフォーメーション式を必要に応じて入力できます。

戻り値

value 引数が日付の場合は日付を返します。

関数に渡された値がすべて NULL である場合、または行が 1 つも選択されていない場合（たとえば、フィルタ条件の値がすべての行に対して FALSE または NULL であった場合）には、NULL です。

NULL

値の 1 つが NULL であると、MIN はその値を無視します。ただし、フィールドから渡された値がすべて NULL である場合には、NULL を返します。

Group By

MIN は、トランスフォーメーションで定義したグループ化フィールドに基づいて値をグループ分けし、各グループについて 1 つの結果を返します。

グループ化フィールドがない場合には、MIN はすべての行を 1 つのグループとして扱い、1 つの値を返します。

例

次の式は、懐中電灯の最も古い注文日を返します。

```
MIN( ORDER_DATE, ITEM_NAME='Flashlight' )
```

ITEM_NAME	ORDER_DATE
Flashlight	Apr 20 1998
Regulator System	May 15 1998
Flashlight	Sep 21 1998
Diving Hood	Aug 18 1998
Halogen Flashlight	Feb 1 1998
Flashlight	Oct 10 1998
Flashlight	NULL

RETURN VALUE: Feb 1 1998

MIN (numbers)

フィールドまたはグループ内の最小の数値を返します。検索において、行を制限するフィルタを適用できます。MIN を使用して、フィールド内またはグループ内における最新の日付もしくは最低の文字列値を返すこともできます。

MIN には他の集計関数は 1 つしかネストできません。また、ネストされた関数は数値データ型を返す必要があります。詳細モードでは集計関数をネストできません。

マッピングタスクでのみ使用します。

構文

MIN(*numeric_value* [, *filter_condition*])

引数	必須/オプション	説明
numeric_value	必須	数値データ型。最小値を返したい値を渡します。有効なトランスフォーメーション式を必要に応じて入力できます。
filter_condition	オプション	検索される行を制限します。フィルタ条件は数値であるか、TRUE、FALSE、または NULL の値が求められなければなりません。有効なトランスフォーメーション式を必要に応じて入力できます。

戻り値

数値。

関数に渡された値がすべて NULL である場合、または行が 1 つも選択されていない場合（たとえば、フィルタ条件の値がすべての行に対して FALSE または NULL であった場合）には、NULL です。

NULL

値の 1 つが NULL であると、MIN はその値を無視します。ただし、フィールドから渡された値がすべて NULL である場合には、NULL を返します。

Group By

MIN は、トランスフォーメーションで定義したグループ化フィールドに基づいて値をグループ分けし、各グループについて 1 つの結果を返します。

グループ化フィールドがない場合には、MIN はすべての行を 1 つのグループとして扱い、1 つの値を返します。

例

次の式は、懐中電灯の最小価格を返します。

```
MIN ( PRICE, ITEM_NAME='Flashlight' )
```

ITEM_NAME	PRICE
Flashlight	10.00
Regulator System	360.00
Flashlight	55.00
Diving Hood	79.00
Halogen Flashlight	162.00
Flashlight	85.00
Flashlight	NULL

RETURN VALUE: 10.00

MIN (String)

フィールドまたはグループ内の最低の文字列値を返します。検索において、行を制限するフィルタを適用できます。

注: MIN 関数は、Sorter トランスフォーメーションと同じソート順を使用します。ただし、MIN 関数では大文字と小文字が区別されますが、Sorter トランスフォーメーションでは区別されない場合もあります。

MIN を使用して、フィールド内またはグループ内における直近の日付もしくは最小の数値を返すこともできます。

MIN には他の集計関数は 1 つしかネストできません。また、ネストされた関数は文字列データ型を返す必要があります。詳細モードでは集計関数をネストできません。

マッピングタスクでのみ使用します。

構文

MIN(*string* [, *filter_condition*])

引数	必須/オプション	説明
string	必須	String データ型。最小値を返したい値を渡します。有効なトランスフォーメーション式を必要に応じて入力できます。
filter_condition	オプション	検索される行を制限します。フィルタ条件は数値であるか、TRUE、FALSE、または NULL の値が求められなければなりません。有効なトランスフォーメーション式を必要に応じて入力できます。

戻り値

文字列値。

関数に渡された値がすべて NULL である場合、または行が 1 つも選択されていない場合（たとえば、フィルタ条件の値がすべての行に対して FALSE または NULL であった場合）には、NULL です。

NULL

値の 1 つが NULL であると、MIN はその値を無視します。ただし、フィールドから渡された値がすべて NULL である場合には、NULL を返します。

Group By

MIN は、トランスフォーメーションで定義したグループ化フィールドに基づいて値をグループ分けし、各グループについて 1 つの結果を返します。

グループ化フィールドがない場合には、MIN はすべての行を 1 つのグループとして扱い、1 つの値を返します。

例

以下の式は、メーカー ID 104 の最小の項目名を返します。

```
MIN ( ITEM_NAME, MANUFACTURER_ID='104' )
```

MANUFACTURER_ID	ITEM_NAME
101	First Stage Regulator
102	Electronic Console
104	Flashlight
104	Battery (9 volt)
104	Rope (20 ft)
104	60.6 cu ft Tank
107	75.4 cu ft Tank
108	Wristband Thermometer

RETURN VALUE: 60.6 cu ft Tank

MOD

除算の余りを返します。例えば、MOD(8,5)は 3 を返します。

構文

MOD(*numeric_value*, *divisor*)

引数	必須/オプション	説明
numeric_value	必須	数値データ型。割られる値（被除数）です。有効な式を必要に応じて入力できます。
divisor	必須	割る値（除数）です。除数にゼロは指定できません。

戻り値

関数に渡したのと同じデータ型の数値。数値を除数で割った余りです。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、PRICE カラムの値を QTY カラムの値で割った余りを返します。

MOD(PRICE, QTY)

PRICE	QTY	RETURN VALUE
10.00	2	0
12.00	5	2
9.00	2	1
15.00	3	0
NULL	3	NULL
20.00	NULL	NULL
25.00	0	None. データ統合 writes the row into the error rows file.

最後の行（25、0）は、除数をゼロにしたためにエラーとなりました。ゼロでの除算を防ぐために、次のような式を作成することができます。この式では、数量がゼロでないときに限り価格を数量で割った余りを返します。数量がゼロの場合、関数は NULL を返します。

MOD(PRICE, IIF(QTY = 0, NULL, QTY))

PRICE	QTY	RETURN VALUE
10.00	2	0
12.00	5	2
9.00	2	1
15.00	3	0

PRICE	QTY	RETURN VALUE
NULL	3	NULL
20.00	NULL	NULL
25.00	0	NULL

最後の行（25、0）はエラーの代わりに NULL を返します。これは、IIF 関数によって QTY カラムの 0 が NULL に置き換えられたためです。

MOVINGAVG

指定された行のセットについて、行ごとの平均を返します。オプションとして、移動平均を計算する前に、条件に基づいて行をフィルタリングすることができます。

構文

MOVINGAVG(*numeric_value*, *rowset* [, *filter_condition*])

引数	必須/オプション	説明
numeric_value	必須	数値データタイプ。移動平均を計算したい値を渡します。有効なトランスフォーメーション式を必要に応じて入力できます。
rowset	必須	0 より大きい正の整数リテラルでなければなりません。移動平均を計算したい行のセットを定義します。例えば、データの列について一度に 5 行ずつ移動平均を計算したい場合は、MOVINGAVG(SALES, 5)のような式を記述できます。
filter_condition	オプション	検索される行を制限します。フィルタ条件は数値であるか、TRUE、FALSE、または NULL の値が求められなければなりません。有効なトランスフォーメーション式を必要に応じて入力できます。

戻り値

数値。

関数に渡された値がすべて NULL である場合、または行が 1 つも選択されていない場合（たとえば、フィルタ条件の値がすべての行に対して FALSE または NULL であった場合）には、NULL です。

注: 戻り値が 15 を超える精度を持つ 10 進値である場合は、高精度を有効にして、最大 28 桁までの 10 進精度を使用可能にできます。

NULL

MOVINGAVG は、移動平均の計算において NULL 値を無視します。ただし、すべての値が NULL である場合には、NULL を返します。

例

次の式は、SALES 列の最初の 5 行に基づいて固定ベストの平均注文を返し、そのあとは、直前に読み込んだ 5 行についての平均を返します。

```
MOVINGAVG( SALES, 5 )
```

ROW_NO	SALES	RETURN VALUE
1	600	NULL
2	504	NULL
3	36	NULL
4	100	NULL
5	550	358
6	39	245.8
7	490	243

関数は、5 行の組ごとの平均を返します。行番号 1-5 の平均は 358、行番号 2-6 の平均は 245.8、行番号 3-7 の平均は 243 です。

MOVINGSUM

指定された行のセットについて、行ごとの合計を返します。オプションとして、移動合計を計算する前に、条件に基づいて行をフィルタリングすることができます。

構文

```
MOVINGSUM( numeric_value, rowset [, filter_condition] )
```

引数	必須/オプション	説明
numeric_value	必須	数値データタイプ。移動合計を計算したい値を渡します。有効なトランスフォーメーション式を必要に応じて入力できます。
rowset	必須	0 より大きい正の整数リテラルでなければなりません。移動合計を計算したい行のセットを定義します。例えば、データの列について一度に 5 行ずつ移動合計を計算したい場合は、MOVINGSUM(SALES, 5)のような式を記述できます。
filter_condition	オプション	検索される行を制限します。フィルタ条件は数値であるか、TRUE、FALSE、または NULL の値が求められなければなりません。有効なトランスフォーメーション式を必要に応じて入力できます。

戻り値

数値。

関数に渡された値がすべて NULL である場合、または行が 1 つも選択されていない場合（たとえば、フィルタ条件の値がすべての行に対して FALSE または NULL であった場合）には、NULL です。

注: 戻り値が 15 を超える精度を持つ 10 進値である場合は、高精度を有効にして、最大 28 桁までの 10 進精度を使用可能にできます。

NULL

MOVINGSUM は、移動合計の計算において NULL 値を無視します。ただし、すべての値が NULL である場合には、NULL を返します。

例

次の式は、SALES 列の最初の 5 行に基づいて固定ベストの注文合計を返し、そのあとは、直前に読み込んだ 5 行についての合計を返します。

```
MOVINGSUM( SALES, 5 )
```

ROW_NO	SALES	RETURN VALUE
1	600	NULL
2	504	NULL
3	36	NULL
4	100	NULL
5	550	1790
6	39	1229
7	490	1215

関数は、5 行の組ごとの合計を返します。行番号 1-5 の合計は 1790、行番号 2-6 の合計は 1229、行番号 3-7 の合計は 1215 です。

NPER

一定の利率、支払周期、支払額に基づいて、投資の期間数を返します。

構文

NPER(*rate, present value, payment* [, *future value, type*])

引数	必須/オプション	説明
rate	必須	数値。各期間で得た金利。十進数で表示されます。利率を 100 で除算すると、十進数で表示できます。0 以上を指定する必要があります。
present value	必須	数値。今後の支払いの合計に相当する一時金の金額。
payment	必須	数値。期間ごとの支払額。負の数を指定する必要があります。

引数	必須/オプション	説明
future value	オプション	数値。最終支払いの後、獲得する現金残高。この値を省略すると、NPER はゼロを使用します。
type	オプション	ブール。支払時期。期首支払の場合は、1 を入力します。期末支払の場合は、0 を入力します。デフォルトは 0 です。0 または 1 以外の値を入力すると、データ統合ではその値が 1 として処理されます。

戻り値

数値。

例

投資の現在価値は 2,000 ドルです。1 回の支払額は 500 ドルで、この投資の将来価値は 20,000 ドルです。次の式は、支払いが必要な期間数を計算し、9 を返します。

```
NPER( 0.01, -2000, -500, 20000, TRUE )
```

注意事項

期間あたりの利率を計算するには、年利を年間の支払回数で割ります。たとえば、年利 15%で月払いの場合、率引数は 15%を 12 で割った値になります。年払いの場合は、率引数は 15%です。

支払値と現在価値は支払額を指すため、負の値になります。

PERCENTILE

数値のグループ内で、与えられたパーセンタイルに入る値を計算します。

データ統合は、すべてのデータ行を読み取り、パーセンタイル計算を実行します。パフォーマンスは、使用するデータベース結合の種類によっても変わってきます。オプションとして、パーセンタイルを計算するために読み込む行を制限するフィルタを適用できます。

PERCENTILE には他の集計関数は 1 つしかネストできません。また、ネストされた関数は数値データ型を返す必要があります。詳細モードでは集計関数をネストできません。

マッピングタスクでのみ使用します。

構文

PERCENTILE(*numeric_value*, *percentile* [, *filter_condition*])

引数	必須/オプション	説明
numeric_value	必須	数値データタイプ。パーセンタイルを計算したい値を渡します。有効なトランスフォーメーション式を必要に応じて入力できます。
percentile	必須	0 から 100 までの整数。計算したいパーセンタイルを渡します。有効なトランスフォーメーション式を必要に応じて入力できます。0〜100 の範囲外の数値を渡すと、データ統合ではエラーが表示され、行が書き込まれません。
filter_condition	オプション	検索される行を制限します。フィルタ条件は数値であるか、TRUE、FALSE、または NULL の値が求められなければなりません。有効なトランスフォーメーション式を必要に応じて入力できます。

戻り値

数値。

関数に渡された値がすべて NULL である場合、または行が 1 つも選択されていない場合（たとえば、フィルタ条件の値がすべての行に対して FALSE または NULL であった場合）には、NULL です。

NULL

値が NULL であると、PERCENTILE はその行を無視します。ただし、グループ内の値がすべて NULL である場合には、NULL を返します。

Group By

PERCENTILE は、トランスフォーメーションで定義したグループ化フィールドに基づいて値をグループ分けし、各グループについて 1 つの結果を返します。

グループ化フィールドがない場合には、PERCENTILE はすべての行を 1 つのグループとして扱い、1 つの値を返します。

例

データ統合は、次のロジックを使用してパーセンタイルを計算します。

$$i = \frac{(x + 1)\ x\ \text{パーセンタイル}}{100}$$

この方程式では、以下のガイドラインを使用します。

- *x* は、パーセンタイルを計算している値グループにある要素の数です。
- *i* < 1 の場合、PERCENTILE はリスト内の最初の要素の値を返します。
- *i* が整数値である場合、PERCENTILE はリスト内の *i* 番目の要素の値を返します。
- 整数値でない場合、PERCENTILE は *n* の値を返します。

$$n = [\lfloor i \rfloor \text{th?element} \times (\lceil i \rceil - i)] + [\lceil i \rceil \text{th?element} \times (i - \lfloor i \rfloor)]$$

次の式は、$50,000 を超える給与の 75 番目のパーセンタイルに入る給与を返します。

```
PERCENTILE( SALARY, 75, SALARY > 50000 )
```

SALARY
125000.0
27900.0
100000.0
NULL
55000.0
9000.0
85000.0
86000.0
48000.0
99000.0

RETURN VALUE: 106250.0

PMT

一定の利率で定額を支払う場合の貸付の支払額を返します。

構文

PMT(*rate, terms, present value, future value, type*])

引数	必須/オプション	説明
rate	必須	数値。それぞれの期間における、ローンの金利。十進数で表示されます。利率を 100 で除算すると、十進数で表示できます。0 以上を指定する必要があります。
terms	必須	数値。期間または支払の数値。0 より大きな値を指定する必要があります。
present value	必須	数値。ローンの元金。

引数	必須/オプション	説明
future value	オプション	数値。最終支払いの後、獲得する現金算高。この値を省略すると、PMT はゼロを使用します。
type	オプション	ブール。支払時期。期首支払の場合は、1 を入力します。期末支払の場合は、0 を入力します。デフォルトは 0 です。0 または 1 以外の値を入力すると、データ統合ではその値が 1 として処理されます。

戻り値

数値。

例

以下の式は、ローンの毎月の支払額として-2111.64 を返します。

```
PMT( 0.01, 10, 20000 )
```

注意事項

期間ごとに得た金利を計算するには、年利を 1 年間の支払回数で除算します。たとえば年率 15%の金利で支払いを毎月実行する場合、月利は 15%/12 になります。年払いの場合、金利は 15%になります。

支払値は支払額を指すため、負の値になります。

POWER

関数に渡された指数による値の累乗を返します。

構文

```
POWER( base, exponent )
```

引数	必須/オプション	説明
base	必須	数値。この引数は累乗される基数値です。有効な式を必要に応じて入力できます。底の値が負の場合は、指数は整数である必要があります。
exponent	必須	数値。この引数は累乗の指数値です。有効な式を必要に応じて入力できます。底の値が負の場合は、指数は整数である必要があります。このような場合、小数点以下の値は、結果が返される前に最も近い整数に丸められます。

戻り値

Double 値。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、Exponent カラムの値を指数として、Numbers カラムの値の累乗を返します。

```
POWER( NUMBERS, EXPONENT )
```

NUMBERS	EXPONENT	RETURN VALUE
10.0	2.0	100
3.5	6.0	1838.265625
3.5	5.5	982.594307804838
NULL	2.0	NULL
10.0	NULL	NULL
-3.0	-6.0	0.00137174211248285
3.0	-6.0	0.00137174211248285
-3.0	6.0	729.0
-3.0	5.5	729.0

-3.0 の 6 乗は、-3.0 の 5.5 乗と同じ結果を返します。それ以外の場合、データ統合はその指数を近似値の整数に丸めます。

PV

投資の現在価値を返します。

構文

PV(*rate, terms, payment* [, *future value, type*])

引数	必須/オプション	説明
rate	必須	数値。各期間で得た金利。十進数で表示されます。利率を 100 で除算すると、十進数で表示できます。0 以上を指定する必要があります。
terms	必須	数値。期間または支払の数値。0 より大きな値を指定する必要があります。
payment	必須	数値。期間ごとの支払額。負の数を指定する必要があります。
future value	オプション	数値。最後の支払いの後の現金残高。この値を省略すると、PV はゼロを使用します。
type	オプション	ブール。支払時期。期首支払の場合は、1 を入力します。期末支払の場合は、0 を入力します。デフォルトは 0 です。0 または 1 以外の値を入力すると、データ統合ではその値が 1 として処理されます。

戻り値

数値。

例

以下の式では、今後期首ごとに$500 を預金して 1 年後の将来価値を$20,000 にするために、本日中に口座に入金する必要のある金額として 12,524.43 が返されます。

```
PV( 0.0075, 12, -500, 20000, TRUE )
```

RAND

0〜1 の範囲の乱数を返します。これは、確率を計算する場合に役立ちます。

構文

```
RAND( seed )
```

引数	必須/オプション	説明
seed	オプション	数値。データ統合の開始値を指定して、乱数を生成します。この値は、定数でなければなりません。シードを入力しない場合、データ統合では現在のシステム時間を使用して、1971 年 1 月 1 日からの秒数が算出されます。この値を seed として使用します。

戻り値

数値。

同じシードに対し、データ統合は同じ数字のシーケンスを生成します。

例

以下の式は、0.417022004702574 を戻り値として返します。

```
RAND (1)
```

RATE

証券ごとに、期間別に得た金利を返します。

構文

RATE(*terms, payment, present value*[, *future value, type*])

引数	必須/オプション	説明
terms	必須	数値。期間または支払の数値。0 より大きな値を指定する必要があります。
payment	必須	数値。期間ごとの支払額。負の数を指定する必要があります。
present value	必須	数値。現段階で今後の支払額に相当する一時金の金額。
future value	オプション	数値。最終支払いの後、獲得する現金算高。たとえば、貸付の将来価値は 0 になります。この引数を省略すると、RATE はゼロを使用します。
type	オプション	ブール。支払時期。期首支払の場合は、1 を入力します。期末支払の場合は、0 を入力します。デフォルトは 0 です。0 または 1 以外の値を入力すると、データ統合ではその値が 1 として処理されます。

戻り値

数値。

例

以下の式は、ローンの月利として 0.0077 を返します。

```
RATE( 48, -500, 20000 )
```

ローンの年利を計算するには、0.0077 に 12 を掛けます。年利は、0.0924 つまり 9.24%になります。

REG_EXTRACT

入力値から正規表現のサブパターンを抽出します。たとえばフルネームの正規表現から、姓または名を抽出することもできます。

注: REG_REPLACE 関数を使用して、文字列内の文字を新しい文字パターンに置換します。

構文

REG_EXTRACT (*subject*, *pattern*, *subPatternNum*)

引数	必須/オプション	説明
subject	必須	文字列データ型。正規表現のパターンと比較する値を渡します。
pattern	必須	文字列データ型。一致させる正規表現パターン。Perl 互換の正規表現構文を使用する必要があります。パターンは、一重引用符で囲みます。
subPatternNum	オプション	整数値。一致させる正規表現のサブパターン番号。サブパターン番号は、以下のガイドラインを使用して決定します。 - 値を指定しない、または 1。最初の正規表現サブパターンを抽出します。 - 2. 2 番目の正規表現サブパターンを抽出します。 - n. n 番目の正規表現サブパターンを抽出します。 デフォルトは 1 です。

Perl 互換の正規表現構文の使用

REG_EXTRACT 関数および REG_MATCH 関数では、Perl 互換の正規表現構文を使用する必要があります。

以下の表に、Perl 互換の正規表現構文のガイドラインを示します。

構文	説明
.（ピリオド）	任意の 1 つの文字に一致します。
[a-z]	1 つの文字インスタンスに一致します。たとえば、[a-z]では ab に一致します。大文字のコード
\d	0〜9 の任意の数字のうち、1 つのインスタンスに一致します。
\s	空白 1 文字に一致します。
\w	アンダースコア(_)を含む英数字 1 文字に一致します。
()	式をグループ化します。たとえば(\d\d-\d\d)では、かっこによって正規表現\d\d-\d\d がグループ化されます。この正規表現では 12-34 のように、任意の 2 つの数字の後にハイフンが続き、さらに任意の 2 つの文字が続くものが検索されます。
{}	指定された文字数に一致します。たとえば、\d{3}は、650 または 510 などの任意の 3 桁の番号に一致します。また、[a-z]{2}は、CA または NY などの任意の 2 文字に一致します。
?	前にある文字または文字のグループに、0 回または 1 回一致します。例えば、\d{3}(-{d{4})?は、任意の 3 桁の数字、および任意の 3 桁の数字にハイフンと任意の 4 桁の数字が続くものの両方に一致します。
*（アスタリスク）	アスタリスクの後に続く、0 個以上の値のインスタンスに一致します。たとえば*0 は、任意の値が 0 の前に置かれたものに一致します。
+	プラス記号の後に続く、1 個以上の値のインスタンスに一致します。たとえば、\w+は、英数字 1 文字が続く任意の値です。

たとえば以下の正規表現は、93930 などの 5 桁のアメリカの郵便番号、および 93930-5407 などの 9 桁の郵便番号を検索します。

```
\d{5}(-\d{4})?
```

\d{5}は、93930 などの 5 つの数字を表します。-\d{4}を囲むかっこによって、式のこの部分がグループ化されています。ハイフンは、93930-5407 などの 9 桁の郵便番号の中のハイフンを表します。\d{4}は 4 つの数字、たとえば 5407 などを表します。疑問符は、ハイフンと 4 つの数字がオプションであるか、1 回のみ出現することを表します。

COBOL 構文を Perl 互換の正規表現構文へ変換

COBOL 構文に精通している場合、以下の情報を使用して Perl 互換の正規表現を書き込むことができます。

以下の表に、COBOL 構文の例とそれに対応する Perl 構文を示します。

COBOL **構文**	Perl **構文**	**説明**
9	\d	0〜9 の任意の数字のうち、1 つのインスタンスに一致します。
9999	\d\d\d\d または \d{4}	1234 や 5936 など、0〜9 の任意の 4 桁の数字に一致します。
x	[a-z]	1 つの文字インスタンスに一致します。
9xx9	\d[a-z][a-z]\d	1ab2 など、任意の数字の後に 2 つの文字が続き、その後に 1 つの数字が続くものに一致します。

SQL 構文の Perl 互換の正規表現構文への変換

SQL 構文に精通している場合、以下の情報を使用して Perl 互換の正規表現を書き込むことができます。

以下の表に、SQL 構文の例とそれに対応する Perl 構文を示します。

SQL **構文**	Perl **構文**	**説明**
%	.*	任意の文字列に一致します。
A%	A.*	「A」で始まる任意の文字列（「Area」など）に一致します。
_	.（ピリオド）	任意の 1 つの文字に一致します。
A_	A.	「A」で始まり、その後に任意の 1 文字が続く文字列（「AZ」など）に一致します。

戻り値

入力値の一部である、*n* 番目のサブパターンの値を返します。*nth* のサブパターンは、subPatternNum で指定した値に基づいています。

入力値またはパターンが NULL の場合は NULL です。

例

式で REG_EXTRACT を使用して、姓、ミドルネーム、名が一致する正規表現からミドルネームを抽出することもできます。例えば、以下の式は正規表現のミドルネームを返します。

```
REG_EXTRACT (Employee_Name, '(\w+)\s+(\w+)\s+(\w+)', 2)
```

Employee_Name	Return Value
Stephen Graham Smith	Graham
Juan Carlos Fernando	Carlos

REG_MATCH

値が正規表現のパターンに一致するかどうかを返します。そのため、ID、電話番号、郵便番号、州の名前などのデータパターンを検査できます。

注: 文字列内の文字を新しい文字パターンに置換します。

構文

```
REG_MATCH( subject,pattern )
```

引数	必須/オプション	説明
subject	必須	文字列データ型。正規表現のパターンに一致させる値を渡します。
pattern	必須	文字列データ型。一致させる正規表現パターン。Perl 互換の正規表現構文を使用する必要があります。パターンは、一重引用符で囲みます。詳細については、「REG_EXTRACT」 (ページ 158)を参照してください。

戻り値

データがパターンに一致する場合は TRUE。

データがパターンに一致しない場合は FALSE。

入力値またはパターンが NULL の場合は NULL です。

例

式で REG_MATCH を使用して、電話番号を検査する場合もあります。たとえば、以下の式は 10 桁の電話番号とパターンを一致させ、その結果に基づき論理値を返します。

```
REG_MATCH (Phone_Number, '(\d\d\d-\d\d\d-\d\d\d\d)' )
```

Phone_Number	Return Value
408-555-1212	TRUE
	NULL
510-555-1212	TRUE

Phone_Number	Return Value
92 555 51212	FALSE
650-555-1212	TRUE
415-555-1212	TRUE
831 555 12123	FALSE

ヒント

以下のタスクでも REG_MATCH を使用できます。

- 値とパターンが一致するかを確認する場合。これは、SQL LIKE 関数の使用方法に似ています。
- 値が文字かどうかを確認する場合。これは、SQL IS_CHAR 関数の使用方法に似ています。

式の REG_MATCH 関数でピリオド（.）およびアスタリスク（*）を使用して、値がパターンと一致するかを確認する場合。ピリオドは 1 文字に一致します。アスタリスク（*）は、後続の値の 0 個以上のインスタンスに一致します。

たとえば、以下の式を使用して 1835 で始まる口座番号を検索します。

```
REG_MATCH(ACCOUNT_NUMBER, '1835.*')
```

値が文字であるかを確認するには、[a-zA-Z]+を正規表現とする REG_MATCH 関数を使用します。a-z は、すべての小文字に一致します。A-Z は、すべての大文字に一致します。プラス記号(+)は、少なくとも 1 文字が存在しなければならないことを表しています。

たとえば、以下の式を使用して、姓のリストに文字しか含まれていないことを確認します。

```
REG_MATCH(LAST_NAME, '[a-zA-Z]+')
```

REG_REPLACE

文字列内の文字を新しい文字パターンに置換します。REPLACECHR は、入力文字列から指定文字を検索し、検索されたすべての文字を、指定した新しい文字に置き換えます。グループに取り込みたい EMAIL の出現を Pivot 化します。

構文

REG_REPLACE(*subject*, *pattern*, *replace*, *numReplacements*)

引数	必須/オプション	説明
subject	必須	文字列データ型。検索したい文字列を渡します。
pattern	必須	文字列データ型。変更するフォルダを選択してください。Perl 互換の正規表現構文を使用する必要があります。パターンは、一重引用符で囲みます。詳細については、「REG_EXTRACT」 (ページ 158)を参照してください。

引数	必須/オプション	説明
replace	必須	文字列データ型。文字列内で検索するサブストリングを渡します。
numReplacements	オプション	数値データ型。プレビューしたい行の数を入力します。このオプションを省略すると、ステータスメッセージがウィンドウに出力されます。

戻り値

文字列。

例

次の式は、Employee_name カラムの各行について従業員名データから余分なスペースを削除します。

```
REG_REPLACE( Employee_Name, '\s+', ' ')
```

Employee_Name	RETURN VALUE
Adam Smith	Adam Smith
Greg Sanders	Greg Sanders
Sarah Fe	Sarah Fe
Sam Cooper	Sam Cooper

REPLACECHR

文字列内の文字を 1 文字または文字なしに置換します。REPLACECHR は、入力文字列から指定文字を検索し、検索されたすべての文字を、指定した新しい文字に置き換えます。

構文

REPLACECHR(*CaseFlag*, *InputString*, *OldCharSet*, *NewChar*)

引数	必須/オプション	説明
CaseFlag	必須	整数でなければなりません。この関数の引数の大文字と小文字を区別するかどうかを指定します。有効な式を必要に応じて入力できます。 *CaseFlag* が 0 以外の数値の場合、大文字と小文字を区別します。 *CaseFlag* が NULL 値または 0 の場合、大文字と小文字を区別しません。
InputString	必須	文字列でなければなりません。検索したい文字列を渡します。有効な式を必要に応じて入力できます。数値を渡すと、関数はそれを文字列に変換します。 *>InputString</>が NULL の場合、REPLACECHR は NULL を返します。*

引数	必須/オプション	説明
OldCharSet	必須	文字列でなければなりません。置換したい文字を渡します。1 つまたは複数の文字を入力できます。有効な式を必要に応じて入力できます。'abc'のように文字列リテラルを一重引用符で囲んで入力することもできます。 数値を渡すと、関数はそれを文字列に変換します。 *OldCharSet* が NULL または空の場合、REPLACECHR は *InputString* を返します。
NewChar	必須	文字列でなければなりません。1 文字、空の文字列、または NULL を入力できます。有効な式を必要に応じて入力できます。 *>NewChar</>が NULL または空の場合、REPLACECHR は、<>OldCharSet</>に指定された文字をすべて<>InputString</>から削除します。* *NewChar* に複数の文字が含まれている場合、REPLACECHR は、最初の文字で、*OldCharSet を置換します。*

戻り値

文字列。

REPLACECHR が *InputString* のすべての文字を削除した場合は空の文字列。

InputString が NULL の場合は NULL。

OldCharSet が NULL または空の場合は *InputString*。

例

次の式は、WEBLOG カラムの各行について Web ログデータから二重引用符を削除します。

```
REPLACECHR( 0, WEBLOG, '"', NULL )
```

WEBLOG	RETURN VALUE
"GET /news/index.html HTTP/1.1"	GET /news/index.html HTTP/1.1
"GET /companyinfo/index.html HTTP/1.1"	GET /companyinfo/index.html HTTP/1.1
GET /companyinfo/index.html HTTP/1.1	GET /companyinfo/index.html HTTP/1.1
NULL	NULL

次の式は、WEBLOG カラムの各行について複数の文字を削除します。

```
REPLACECHR ( 1, WEBLOG, '][', NULL )
```

WEBLOG	RETURN VALUE
[29/Oct/2001:14:13:50 -0700]	29/Oct/2001:14:13:50 -0700
[31/Oct/2000:19:45:46 -0700] "GET /news/index.html HTTP/1.1"	31/Oct/2000:19:45:46 -0700 GET /news/index.html HTTP/1.1
[01/Nov/2000:10:51:31 -0700] "GET /news/index.html HTTP/1.1"	01/Nov/2000:10:51:31 -0700 GET /news/index.html HTTP/1.1
NULL	NULL

次の式は、CUSTOMER_CODE カラムの各行について顧客コードの値の一部を変更します。

```
REPLACECHR ( 1, CUSTOMER_CODE, 'A', 'M' )
```

CUSTOMER_CODE	RETURN VALUE
ABA	MBM
abA	abM
BBC	BBC
ACC	MCC
NULL	NULL

次の式は、CUSTOMER_CODE カラムの各行について顧客コードの値の一部を変更します。

```
REPLACECHR ( 0, CUSTOMER_CODE, 'A', 'M' )
```

CUSTOMER_CODE	RETURN VALUE
ABA	MBM
abA	MbM
BBC	BBC
ACC	MCC

次の式は、CUSTOMER_CODE カラムの各行について顧客コードの値の一部を変更します。

```
REPLACECHR ( 1, CUSTOMER_CODE, 'A', NULL )
```

CUSTOMER_CODE	RETURN VALUE
ABA	B
BBC	BBC
ACC	CC
AAA	*[empty string]*
aaa	aaa
NULL	NULL

次の式は、INPUT カラムの各行について複数の数字を削除します。

```
REPLACECHR ( 1, INPUT, '14', NULL )
```

INPUT	RETURN VALUE
12345	235
4141	NULL
111115	5
NULL	NULL

OldCharSet または *NewChar* で一重引用符（'）を使用する場合、CHR 関数を使用する必要があります。一重引用符は、文字列リテラル内で使用できない唯一の文字です。

次の式は、INPUT カラムの各行について、複数の一重引用符の文字を削除します。

```
REPLACECHR (1, INPUT, CHR(39), NULL )
```

INPUT	RETURN VALUE
'Tom Smith' 'Laura Jones'	Tom Smith Laura Jones
Tom's	Toms
NULL	NULL

REPLACESTR

文字列内の文字を 1 文字、複数の文字または文字なしに置換します。REPLACESTR は、入力文字列から、指定されたすべての文字列を検索し、指定された新しい文字列に置換します。

構文

```
REPLACESTR ( CaseFlag, InputString, OldString1, [OldString2, ... OldStringN,] NewString )
```

引数	必須/オプション	説明
CaseFlag	必須	整数でなければなりません。この関数の引数の大文字と小文字を区別するかどうかを指定します。有効な式を必要に応じて入力できます。 *CaseFlag* が 0 以外の数値の場合、大文字と小文字を区別します。 *CaseFlag* が NULL 値または 0 の場合、大文字と小文字を区別しません。
InputString	必須	文字列でなければなりません。検索したい文字列を渡します。有効な式を必要に応じて入力できます。数値を渡すと、関数はそれを文字列に変換します。 *>InputString</>が NULL の場合、REPLACESTR は NULL を返します。*

引数	必須/オプション	説明
OldString	必須	文字列でなければなりません。置換したい文字列を渡します。*最低つの <>OldString</1>引数を入力する必要があります。>OldString</>引数ごとに 1 文字以上を入力できます。*有効な式を必要に応じて入力できます。'abc'のように文字列リテラルを一重引用符で囲んで入力することもできます。 数値を渡すと、関数はそれを文字列に変換します。 *複数の<>OldString</>引数があり、1 つ以上の<>OldString</>引数が NULL または空の場合、REPLACESTR はその<>OldString</>引数を無視します。すべての <>OldString</>引数が NULL または空の場合、REPLACESTR は<>InputString</>を返します。* *>OldString</>引数の文字列は、関数に指定されている順に置換されます。たとえば、複数の<>OldString</>引数を入力した場合、最初の<>OldString</>引数は 2 番目の<>OldString</>引数に優先され、2 番目の<>OldString</>引数は 3 番目の <>OldString</>に優先されます。REPLACESTR が文字列を置き換える場合、次に一致する文字列を検索する前に、<>InputString</>内の置き換えられた文字の後ろにカーソルを置きます。*詳細については、例を参照してください。
NewString	必須	文字列でなければなりません。1 文字、複数の文字、空の文字列、または NULL を入力できます。有効な式を必要に応じて入力できます。 *>NewString</>が NULL または空の場合、REPLACESTR は、<>InputString</>内の <>OldString</>をすべて削除します。*

戻り値

文字列。

REPLACESTR が *InputString* のすべての文字を削除した場合は空の文字列。

InputString が NULL の場合は NULL。

すべての *OldString* 引数が NULL または空の場合は *InputString*。

例

次の式は、WEBLOG カラムの各行について、Web ログデータから二重引用符と 2 つの異なるテキスト文字列を削除します。

```
REPLACESTR( 1, WEBLOG, '"', 'GET ', ' HTTP/1.1', NULL )
```

WEBLOG	RETURN VALUE
"GET /news/index.html HTTP/1.1"	/news/index.html
"GET /companyinfo/index.html HTTP/1.1"	/companyinfo/index.html
GET /companyinfo/index.html	/companyinfo/index.html
GET	*[empty string]*
NULL	NULL

次の式は、TITLE カラムの各行について、ある値のタイトルを変更します。

```
REPLACESTR ( 1, TITLE, 'rs.', 'iss', 's.' )
```

TITLE	RETURN VALUE
Mrs.	Ms.
Miss	Ms.
Mr.	Mr.
MRS.	MRS.

次の式は、TITLE カラムの各行について、ある値のタイトルを変更します。

```
REPLACESTR ( 0, TITLE, 'rs.', 'iss', 's.' )
```

TITLE	RETURN VALUE
Mrs.	Ms.
MRS.	Ms.

次の式は、REPLACESTR 関数が、INPUT カラムの各行について、複数の *OldString* 引数をどのように置換するかを示します。

```
REPLACESTR ( 1, INPUT, 'ab', 'bc', '*' )
```

INPUT	RETURN VALUE
abc	*c
abbc	**
abbbbc	*bb*
bc	*

次の式は、REPLACESTR 関数が、INPUT カラムの各行について、複数の *OldString* 引数をどのように置換するかを示します。

```
REPLACESTR ( 1, INPUT, 'ab', 'bc', 'b' )
```

INPUT	RETURN VALUE
ab	b
bc	b
abc	bc
abbc	bb
abbcc	bbc

OldString または *NewString* に一重引用符（'）を使用する場合、CHR 関数を使用する必要があります。CHR および CONCAT 関数を使って、一重引用符を文字列に連結することができます。一重引用符は、文字列リテラル内で使用できない唯一の文字です。次の例を検討します。

```
CONCAT( 'Joan', CONCAT( CHR(39), 's car' ))
```

戻り値は次のとおりです。

```
Joan's car
```

次の式は、INPUT カラムの各行について、一重引用符を含む文字列を変更します。

```
REPLACESTR ( 1, INPUT, CONCAT('it', CONCAT(CHR(39), 's' )), 'its' )
```

INPUT	RETURN VALUE
it's	its
mit's	mits
mits	mits
mits'	mits'

REVERSE

入力文字列を逆順にします。

構文

```
REVERSE( string )
```

引数	必須/オプション	説明
string	必須	任意の文字値。逆順にする値。

戻り値

文字列。入力値を逆順にします。

例

以下の式は、顧客コードの数字を逆順にします。

```
REVERSE( CUSTOMER_CODE )
```

CUSTOMER_CODE	RETURN VALUE
0001	1000
0002	2000
0003	3000
0004	4000

ROUND (Dates)

日付の一部分を丸めます。また、ROUND を使って数値を丸めることもできます。

この関数は、日付の中の以下の部分を丸めることができます。

- **年。**日付の年の部分を、月に基づいて丸めます。月が 1 月～6 月の場合、関数は入力した年の 1 月 1 日を返し、時刻を 00:00:00 に設定します。月が 7 月～12 月の場合、関数は次の年の 1 月 1 日を返し、時刻を 00:00:00 に設定します。例えば、ROUND(06/30/1998 2:30:55, 'YY')の式は 01/01/1998 00:00:00 を、ROUND(07/1/1998 3:10:15, 'YY')の式は 1/1/1998 00:00:00 を返します。
- **月。**日付の月の部分を、日に基づいて丸めます。日が 1 日～15 日の場合、関数は日付を入力された月の最初の日に丸め、時刻を 00:00:00 に設定します。日が 16 日～月の最後の日の場合、関数は日付を次の月の最初の日に丸め、時刻を 00:00:00 に設定します。例えば、ROUND(4/15/1998 12:15:00, 'MM')の式は 4/1/1998 00:00:00 を、ROUND(4/16/1998 8:24:19, 'MM')の式は 5/1/1998 00:00:00 を返します。
- **日。**日付の日の部分を、時刻に基づいて丸めます。時刻が 00:00:00（AM12 時）～11:59:59AM の場合、関数は現在の日付を返し、時刻を 00:00:00（AM12 時）に設定します。時刻が 12:00:00（PM12 時）以降の場合、関数は日付を次の日に丸めて、時刻を 00:00:00（AM12 時）に設定します。例えば、ROUND(06/13/1998 2:30:45, 'DD')の式は 06/13/1998 00:00:00、ROUND(06/13/1998 22:30:45, 'DD')の式は 06/14/1998 00:00:00 を返します。
- **時間。**日付の時の部分を、分に基づいて丸めます。時刻の分の部分が 0～29 の間の数であれば、関数は現在の時で分と秒をゼロにした値を返します。分の部分が 30 以上の場合、関数は次の時で分と秒をゼロにした値を返します。例えば、ROUND(04/01/1998 11:29:35, 'HH')の式は 04/01/1998 11:00:00 を、ROUND(04/01/1998 13:39:00, 'HH')の式は 04/01/1998 14:00:00 を返します。
- **分。**日付の分の部分を、秒に基づいて丸めます。秒の部分が 0～29 の間の数であれば、関数は現在の分で秒をゼロにした値を返します。秒の部分が 30～59 の間の数であれば、関数は次の分で秒をゼロにした値を返します。例えば、ROUND(05/22/1998 10:15:29, 'MI')の式は 05/22/1998 10:15:00、ROUND(05/22/1998 10:15:30, 'MI')の式は 05/22/1998 10:16:00 を返します。

構文

ROUND(*date* [,*format*])

引数	必須/オプション	説明
date	必須	Date/Time データ型。丸める前に、TO_DATE をネストして文字列を日付に変換できます。
format	オプション	正しいフォーマット文字列を入力します。これは日付の中で丸めたい部分です。日付の中の 1 つの部分だけを丸めることができます。フォーマット文字列を省略すると、関数は日付を最も近い日に丸めます。

Return Value

指定された部分が丸められた日付。ROUND は元の日付と同じ形式で日付を返します。この関数の結果を、Date/Time データ型を持つ任意のカラムにリンクすることができます。

関数に NULL 値を渡した場合は NULL です。

例

以下の式は、DATE_SHIPPED カラムの日付の年の部分を丸めます。

```
ROUND( DATE_SHIPPED, 'Y' )
ROUND( DATE_SHIPPED, 'YY' )
```

```
ROUND( DATE_SHIPPED, 'YYY' )
ROUND( DATE_SHIPPED, 'YYYY' )
```

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:30AM	Jan 1 1998 12:00:00AM
Apr 19 1998 1:31:20PM	Jan 1 1998 12:00:00AM
Dec 20 1998 3:29:55PM	Jan 1 1999 12:00:00AM
NULL	NULL

以下の式は、DATE_SHIPPED カラムの各日付の月の部分を丸めます。

```
ROUND( DATE_SHIPPED, 'MM' )
ROUND( DATE_SHIPPED, 'MON' )
ROUND( DATE_SHIPPED, 'MONTH' )
```

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:30AM	Jan 1 1998 12:00:00AM
Apr 19 1998 1:31:20PM	May 1 1998 12:00:00AM
Dec 20 1998 3:29:55PM	Jan 1 1999 12:00:00AM
NULL	NULL

以下の式は、DATE_SHIPPED カラムの各日付の日の部分を丸めます。

```
ROUND( DATE_SHIPPED, 'D' )
ROUND( DATE_SHIPPED, 'DD' )
ROUND( DATE_SHIPPED, 'DDD' )
ROUND( DATE_SHIPPED, 'DY' )
ROUND( DATE_SHIPPED, 'DAY' )
```

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:30AM	Jan 15 1998 12:00:00AM
Apr 19 1998 1:31:20PM	Apr 20 1998 12:00:00AM
Dec 20 1998 3:29:55PM	Dec 21 1998 12:00:00AM
Dec 31 1998 11:59:59PM	Jan 1 1999 12:00:00AM
NULL	NULL

以下の式は、DATE_SHIPPED カラムの各日付の時の部分を丸めます。

```
ROUND( DATE_SHIPPED, 'HH' )
ROUND( DATE_SHIPPED, 'HH12' )
ROUND( DATE_SHIPPED, 'HH24' )
```

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:31AM	Jan 15 1998 2:00:00AM
Apr 19 1998 1:31:20PM	Apr 19 1998 2:00:00PM
Dec 20 1998 3:29:55PM	Dec 20 1998 3:00:00PM

DATE_SHIPPED	RETURN VALUE
Dec 31 1998 11:59:59PM	Jan 1 1999 12:00:00AM
NULL	NULL

以下の式は、DATE_SHIPPED カラムの各日付の分の部分を丸めます。

```
ROUND( DATE_SHIPPED, 'MI' )
```

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:30AM	Jan 15 1998 2:11:00AM
Apr 19 1998 1:31:20PM	Apr 19 1998 1:31:00PM
Dec 20 1998 3:29:55PM	Dec 20 1998 3:30:00PM
Dec 31 1998 11:59:59PM	Jan 1 1999 12:00:00AM
NULL	NULL

ROUND (Numbers)

数値を指定の桁数または小数点以下の桁数に丸めます。また、ROUND を使って日付を丸めることもできます。

構文

ROUND(*numeric_value* [, *precision*])

引数	必須/オプション	説明
numeric_value	必須	数値データ型。有効な式を必要に応じて入力できます。値を丸める前に、演算子を使用して算術演算を実行できます。
precision	オプション	正または負の整数。正の *precision* を入力すると、関数は数値の小数点以下の桁数をこの値に丸めます。たとえば、ROUND(12.99, 1)は 13.0 を返し、ROUND(15.44, 1)は 15.4 を返します。 負の *precision* を入力すると、関数は小数点の左側をこの桁数だけ丸めて、整数を返します。たとえば、ROUND(12.99, -1)は 10 を返し、ROUND(15.99, -1)は 20 を返します。 小数の *precision* を入力すると、関数はこの値を最も近い整数に丸めてから、式を求めます。たとえば、ROUND(12.99, 0.8)は 13.0 を返します。これは、0.8 を 1 に丸めてから式を求めるからです。 *precision* 引数を省略すると、関数は数値を最も近い整数に丸めて、小数点以下を切り捨てます。たとえば、ROUND(12.99)は 13 を返します。

戻り値

数値。

いずれかの引数が NULL の場合、ROUND は NULL を返します。

例

次の式は、Price カラムの値を小数点以下 3 桁に丸めた値を返します。

```
ROUND( PRICE, 3 )
```

PRICE	RETURN VALUE
12.9936	12.994
15.9949	15.995
-18.8678	-18.868
56.9561	56.956
NULL	NULL

precision 引数に負の整数を渡すことにより、小数点の左側を指定桁数に丸めることもできます。

```
ROUND( PRICE, -2 )
```

PRICE	RETURN VALUE
13242.99	13200.0
1435.99	1400.0
-108.95	-100.0
NULL	NULL

precision 引数に小数値を渡すと、データ統合はその値を近似値の整数に丸めてから式を評価します。

```
ROUND( PRICE, 0.8 )
```

PRICE	RETURN VALUE
12.99	13.0
56.34	56.3
NULL	NULL

precision 引数を省略すると、関数は数値を最も近い整数に丸めます。

```
ROUND( PRICE )
```

PRICE	RETURN VALUE
12.99	13.0
-15.99	-16.0
-18.99	-19.0
56.95	57.0
NULL	NULL

ヒント

ROUND を使用して、計算値の精度を明示的に設定し、期待した結果を得ることもできます。

RPAD

文字列の末尾に空白または文字を追加して、文字列を指定した長さに変換します。

構文

RPAD(*first_string*, *length* [,*second_string*])

引数	必須/オプション	説明
first_string	必須	任意の文字列値。変更したい文字列を渡します。有効な式を必要に応じて入力できます。
length	必須	正の整数リテラルでなければなりません。各文字列の希望の長さを指定します。*length* が負の数である場合、RPAD は NULL を返します。
second_string	オプション	任意の文字列値。*first_string* 値の右側に付加したい文字列を渡します。文字列の末尾に追加したい文字列は、'abc' のように一重引用符で囲みます。この引数は大文字と小文字を区別します。 second_string を省略すると、関数は first_string の末尾に空白を付加します。

戻り値

指定された長さの文字列。

関数に NULL 値を渡した場合、あるいは length が負の数の場合は、NULL です。

例

次の式は、各商品名の末尾に文字列‘.’を追加して、商品名を長さ 16 文字で返します。

```
RPAD( ITEM_NAME, 16, '.')
```

ITEM_NAME	RETURN VALUE
Flashlight	Flashlight.....
Compass	Compass.....
Regulator System	Regulator System
Safety Knife	Safety Knife....

RPAD は、長さを左から数えます。したがって、元の文字列が指定した長さよりも長い場合は、文字列が右側から切り詰められます。たとえば、RPAD(‘alphabetical’, 5, ‘x’)は文字列‘alpha’を返します。RPAD は、必要に応じて *second_string* の一部分を使用します。

次の式は、各商品名の末尾に文字列'*..*'を追加して、商品名を長さ 16 文字で返します。

```
RPAD( ITEM_NAME, 16, '*..*' )
```

ITEM_NAME	RETURN VALUE
Flashlight	Flashlight*..**.
Compass	Compass*..**..**
Regulator System	Regulator System
Safety Knife	Safety Knife*..*

次の式は、ITEM_NAME カラムの各行について、*length* 引数が負の数の場合に RPAD がどのように処理するかを示します。

```
RPAD( ITEM_NAME, -5, '.')
```

ITEM_NAME	RETURN VALUE
Flashlight	NULL
Compass	NULL
Regulator System	NULL

RTRIM

文字列の末尾から空白または文字を削除します。

式で *trim_set* パラメータを指定しない場合、RTRIM はシングルバイトのスペースのみを削除します。

RTRIM を使って文字列から文字を削除する場合、RTRIM は *trim_set* を *string* 引数内の各文字と右から 1 文字ずつ比較します。文字列内の文字が *trim_set* 内のいずれかの文字と一致した場合、RTRIM はその文字を削除します。RTRIM は、一致する文字が *trim_set* で見つからなくなるまで、文字を比較して削除します。一致する文字を含まない文字列を返します。

構文

```
RTRIM( string [, trim_set] )
```

引数	必須/オプション	説明
string	必須	任意の文字列値。切り詰めたい値を渡します。有効な式を必要に応じて入力できます。文字列の末尾から空白を削除する前に、演算子を使って文字列の比較や連結を実行します。
trim_set	オプション	任意の文字列値。文字列の末尾から削除したい文字を渡します。文字列リテラルを入力することもできます。ただし、文字列の末尾から削除する文字は、'abc'のように一重引用符で囲む必要があります。2 番目の文字列を省略すると、関数は 1 番目の文字列の末尾から空白を削除します。 RTRIM では、大文字と小文字が区別されます。

戻り値

文字列。*trim_set* 引数で指定された文字を削除した結果の文字列値。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、LAST_NAME カラムの文字列から文字‘re’を削除します。

```
RTRIM( LAST_NAME, 're')
```

LAST_NAME	RETURN VALUE
Nelson	Nelson
Page	Pag
Osborne	Osborn
NULL	NULL
Sawyer	Sawy
H. Bender	H. Bend
Steadman	Steadman

trim_set の最初の文字は‘r’ですが、RTRIM は Page から‘e’を削除します。これは、RTRIM では *trim_set* 引数に指定された文字列を 1 文字ずつ検索していくためです。文字列内の最後の文字が *trim_set* 内の最初の文字と一致した場合、RTRIM はその文字を削除します。ただし、文字列内の最後の文字が一致しない場合、RTRIM は *trim_set* 内の 2 番目の文字と比較します。文字列内の最後から 2 番目の文字が *trim_set* の 2 番目の文字と一致した場合、RTRIM はその文字を削除します。文字列内の文字が *trim_set* と一致しなかった場合、RTRIM はその文字列を返して、次の行の評価を行います。

最後の例では、Nelson の最後の文字が *trim_set* 引数のどの文字とも一致しないため、RTRIM は‘Nelson’を返して次の行を評価します。

ヒント

RTRIM および LTRIM を || または CONCAT とともに使用すると、2 つの文字列を連結したあとで先頭および末尾の空白を削除します。

また、RTRIM をネストして複数の文字列を削除することもできます。たとえば、名前の列内にある各文字列の末尾から末尾の空白と文字‘t’を削除したい場合は、次のような式を作成します。

```
RTRIM( RTRIM( NAMES ), 't' )
```

SET_DATE_PART

Date/Time 値の一部を指定した値に設定します。

SET_DATE_PART を使用して、日付の以下の部分を変更できます。

年

value 引数に正の整数を入力することで、年を変更します。Y、YY、YYY、YYYY のいずれかを年のフォーマット文字列として使用し、年を設定します。例えば、SHIP_DATE カラムのすべての日付について、年を 2001 に変更するには、式 SET_DATE_PART(SHIP_DATE, 'YY', 2001)を使用します。

月

value 引数に 1 から 12（1 月は 1、12 月は 12）までの正の整数を入力することにより、月を変更します。MM、MON、MONTH のいずれかを月のフォーマット文字列として使用し、月を設定します。例えば、SHIP_DATE カラムのすべての日付について、月を 10 月に変更するには、式 SET_DATE_PART(SHIP_DATE, 'MONTH', 10)を使用します。

日

value 引数に 1 から 31 までの正の整数を入力することにより、日を変更します（31 日より少ない 2 月、4 月、6 月、9 月、11 月を除く）。D、DD、DDD、DY、DAY のいずれかを日のフォーマット文字列として使用し、日を設定します。例えば、SHIP_DATE カラムのすべての日付について、日を 10 に変更するには、式 SET_DATE_PART(SHIP_DATE, 'DD', 10)を使用します。

時間

value 引数に 0 から 24 までの正の整数を入力することにより、時を変更します（0=12AM、12=12PM、24=12AM）。HH、HH12、HH24 のいずれかを時のフォーマット文字列として使用し、時を設定します。例えば、SHIP_DATE カラムのすべての日付について、時を 14:00:00（午後 2:00:00）に変更するには、式 SET_DATE_PART(SHIP_DATE, 'HH', 14)を使用します。

分

value 引数に 0 から 59 までの正の整数を入力することにより、分を変更します。分の設定にはフォーマット文字列 MI を使用します。例えば、SHIP_DATE カラムのすべての日付について、分を 25 に変更するには、式 SET_DATE_PART(SHIP_DATE, 'MI', 25)を使用します。

秒

value 引数に 0 - 59 の正の整数を入力することにより、秒を変更します。秒の設定には SS フォーマット文字列を使用します。例えば、SHIP_DATE カラムのすべての日付について、秒を 59 に変更するには、式 SET_DATE_PART(SHIP_DATE, 'SS', 59)を使用します。

構文

SET_DATE_PART(*date*, *format*, *value*)

引数	必須/オプション	説明
date	必須	Date/Time データ型。変更したい日付です。有効な式を必要に応じて入力できます。
format	必須	日付の中で変更したい場所を指定するフォーマット文字列。フォーマット文字列は大文字と小文字を区別しません。
value	必須	日付内の指定した部分に代入される正の整数値。この整数は、変更したい日付の部分に対して有効な値でなければなりません。不適切な値（2 月 30 日など）を入力すると、セッションが失敗します。

戻り値

指定した部分が変更された日付を、元の日付と同じ形式で返します。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、DATE_PROMISED カラムの各日付について、時を午後 4 時に変更します。

```
SET_DATE_PART( DATE_PROMISED, 'HH', 16 )
SET_DATE_PART( DATE_PROMISED, 'HH12', 16 )
SET_DATE_PART( DATE_PROMISED, 'HH24', 16 )
```

DATE_PROMISED	RETURN VALUE
Jan 1 1997 12:15:56AM	Jan 1 1997 4:15:56PM
Feb 13 1997 2:30:01AM	Feb 13 1997 4:30:01PM
Mar 31 1997 5:10:15PM	Mar 31 1997 4:10:15PM
Dec 12 1997 8:07:33AM	Dec 12 1997 4:07:33PM
NULL	NULL

次の式は、DATE_PROMISED カラムの日付について、月を 6 月に変更します。データ統合では、存在しない日付を作成しようとした場合（3 月 31 日を 6 月 31 日に変更しようとした場合など）、エラーが表示されます。

```
SET_DATE_PART( DATE_PROMISED, 'MM', 6 )
SET_DATE_PART( DATE_PROMISED, 'MON', 6 )
SET_DATE_PART( DATE_PROMISED, 'MONTH', 6 )
```

DATE_PROMISED	RETURN VALUE
Jan 1 1997 12:15:56AM	Jun 1 1997 12:15:56AM
Feb 13 1997 2:30:01AM	Jun 13 1997 2:30:01AM
Mar 31 1997 5:10:15PM	None. データ統合 writes the row into the error rows file.
Dec 12 1997 8:07:33AM	Jun 12 1997 8:07:33AM
NULL	NULL

次の式は、DATE_PROMISED カラムの日付について、年を 2000 に変更します。

```
SET_DATE_PART( DATE_PROMISED, 'Y', 2000 )
SET_DATE_PART( DATE_PROMISED, 'YY', 2000 )
SET_DATE_PART( DATE_PROMISED, 'YYY', 2000 )
SET_DATE_PART( DATE_PROMISED, 'YYYY', 2000 )
```

DATE_PROMISED	RETURN VALUE
Jan 1 1997 12:15:56AM	Jan 1 2000 12:15:56AM
Feb 13 1997 2:30:01AM	Feb 13 2000 2:30:01AM
Mar 31 1997 5:10:15PM	Mar 31 2000 5:10:15PM
Dec 12 1997 8:07:33AM	Dec 12 2000 4:07:33PM
NULL	NULL

ヒント

日付の中の複数の部分を一度に変更したい場合は、*date* 引数に複数の SET_DATE_PART 関数をネストすることができます。例えば、DATE_ENTERED カラムのすべての日付を 1998 年 7 月 1 日に変更するには、次の式を作成します。

```
SET_DATE_PART( SET_DATE_PART( SET_DATE_PART( DATE_ENTERED, 'YYYY', 1998),MM', 7), 'DD', 1)
```

SETCOUNTVARIABLE

関数が評価した行を数えて、そのカウントに基づいて入出力パラメータの現在の値を増やします。新しいカレント値を返します。

各タスクセッションの終了時に、ジョブの詳細の最後の現在の値がデータ統合で保存されます。この値がオーバーライドされない限り、次回ユーザーがタスクを使用したときに、保存された値が変数の初期値としてデータ統合で使用されます。

パイプラインの各入出力パラメータで、SETCOUNTVARIABLE 関数を使用するのは、1 回のみです。データ統合は、変数関数をマッピングで検出したときにそれらの関数を処理します。データ統合がマッピングで変数関数を検出する順序は、セッションを実行するたびに異なります。このため、マッピングで同じ変数関数を複数回使用すると、結果が矛盾する場合があります。

SETCOUNTVARIABLE は、式トランスフォーメーションで使用できます。

構文

```
SETCOUNTVARIABLE( $$Variable )
```

引数	必須/ オプション	説明
$$Variable	必須	設定する入出力パラメータの名前。

戻り値

入出力パラメータのカレント値。

SETMAXVARIABLE

入出力パラメータの現在の値を、パラメータの現在の値と指定値のいずれか高い方に設定します。新しいカレント値を返します。

タスクセッションの終了時、データ統合は、最終的な現在の値を保存し、その値をジョブの詳細に書き込みます。この値がオーバーライドされない限り、次回のタスクセッションでは、保存された値がパラメータの初期値として使用されます。

文字列の入出力パラメータとともに使用した場合、SETMAXVARIABLE は、マッピングで選択したソート順で高い方の文字列を返します。

SETMAXVARIABLE 関数は、パイプラインの各入出力パラメータでに対して一度しか使用できません。データ統合は、マッピングで入出力パラメータが検出されたときにそのパラメータを処理します。データ統合がマッ

ピングで変数関数を検出する順序は、タスクセッションごとに異なります。このため、マッピングで同じ変数関数を複数回使用すると、結果が矛盾する場合があります。

SETMAXVARIABLE は、式トランスフォーメーションで使用できます。

構文

```
SETMAXVARIABLE( $$Variable, value )
```

引数	必須/オプション	説明
$$Variable	必須	設定する入出力パラメータの名前。Max 集計型の入出力パラメータを使用します。
value	必須	データ統合が入出力パラメータの現在の値と比較する値。パラメータのデータ型と互換性のあるデータ型に評価する有効なトランスフォーメーション式を入力します。

戻り値

入出力パラメータのカレント値と指定値のいずれか高い方。戻り値は、パラメータの新しいカレント値です。

value が NULL の場合、データ統合は現在の値である *$$Variable* を返します。

SETMINVARIABLE

入出力パラメータの現在の値を、パラメータの現在の値と指定値のいずれか低い方に設定します。新しいカレント値を返します。

データ統合では、ジョブの詳細に最終的な現在の値が保存されます。この値がオーバーライドされない限り、次回のタスクセッションでは、保存された値がパラメータの初期値として使用されます。

文字列の入出力パラメータとともに使用した場合、SETMINVARIABLE は、マッピングで選択したソート順で低い方の文字列を返します。

パイプラインの各入出力パラメータで、SETMINVARIABLE 関数を使用するのは、1 回のみです。データ統合は、変数関数をマッピングで検出したときにそれらの関数を処理します。データ統合がマッピングで変数関数を検出する順序はセッションごとに異なります。このため、マッピングで同じ変数関数を複数回使用すると、結果が矛盾する場合があります。

SETMINVARIABLE は、式トランスフォーメーションで使用できます。

構文

```
SETMINVARIABLE( $$Variable, value )
```

引数	必須/オプション	説明
$$Variable	必須	設定する入出力パラメータの名前。Min 集計型の入出力パラメータとともに使用します。
value	必須	データ統合がパラメータの現在の値と比較する値。パラメータのデータ型と互換性のあるデータ型に評価する有効な式を入力します。

戻り値

パラメータのカレント値と指定値のいずれか低い方。戻り値は、パラメータの新しいカレント値です。

value が NULL の場合、データ統合は現在の値である *$$Variable* を返します。

SETVARIABLE

入力パラメータの現在の値を指定した値に設定します。指定した値を返します。

タスクの実行時、データ統合は、入出力パラメータの最終的な現在の値と開始値を比較します。入出力パラメータの集計タイプに基づいて、ジョブの詳細に最終的な現在の値が保存されます。この値がオーバーライドされない限り、次回タスクを実行するときは、保存された値が入出力パラメータの初期値として使用されます。

パイプラインの各入出力パラメータで、SETVARIABLE 関数を使用するのは、1 回のみです。データ統合は、変数関数をマッピングで検出したときにそれらの関数を処理します。データ統合がマッピングで変数関数を検出する順序は、タスクセッションごとに異なります。このため、マッピングで同じ変数関数を複数回使用すると、結果が矛盾する場合があります。

SETVARIABLE は、式トランスフォーメーションで使用できます。

構文

```
SETVARIABLE( $$Variable, value )
```

引数	必須/オプション	説明
$$Variable	必須	設定する入出力パラメータの名前。Max 集計型または Min 集計型で設定された入出力パラメータとともに使用します。
value	必須	入出力パラメータのカレント値として設定する値。パラメータのデータ型と互換性のあるデータ型に評価する有効な式を使用します。

戻り値

入出力パラメータのカレント値。

value が NULL の場合、データ統合は現在の値である *$$Variable* を返します。

例

次の式は、データ統合が行を評価したときに入出力パラメータ$$Time をシステム日付に設定し、このシステム日付を SET_$$TIME ポートに返します。

```
SETVARIABLE ($$Time, SYSDATE)
```

TRANSACTION	TOTAL	SET_$$TIME
0100002	534.23	10/10/2016 01:34:33
0100003	699.01	10/10/2016 01:34:34
0100004	97.50	10/10/2016 01:34:35

TRANSACTION	TOTAL	SET_$$TIME
0100005	116.43	10/10/2016 01:34:36
0100006	323.95	10/10/2016 01:34:37

セッションの終了時、データ統合は、最後に評価された$$Time の現在の値として 10/10/2016 01:34:37 をジョブの詳細に保存します。次回のタスク実行後、データ統合は、$$Time に対するすべての参照を 10/10/2016 01:34:37 と評価します。

次の式は、行に関連付けられているタイムスタンプに入出力パラメータの$$Timestamp を設定し、タイムスタンプを SET_$$TIMESTAMP ポートに返します。

```
SETVARIABLE ($$Time, TIMESTAMP)
```

TRANSACTION	TIMESTAMP	TOTAL	SET_$$TIME
0100002	10/01/2016 12:01:01	534.23	10/01/2016 12:01:01
0100003	10/01/2016 12:10:22	699.01	10/01/2016 12:10:22
0100004	10/01/2016 12:16:45	97.50	10/01/2016 12:16:45
0100005	10/01/2016 12:23:10	116.43	10/01/2016 12:23:10
0100006	10/01/2016 12:40:31	323.95	10/01/2016 12:40:31

セッションの終了時、データ統合は、最後に評価された$$Timestamp の現在の値として、10/01/2016 12:40:31 をジョブの詳細に保存します。

次回のセッション実行時、データ統合は、$$Timestamp の初期値を 10/01/2016 12:40:31 に設定します。

SIGN

指定された数値が正の数、負の数、または 0 のいずれであるかを返します。

構文

```
SIGN( numeric_value )
```

引数	必須/オプション	説明
numeric_value	必須	数値。評価したい値を渡します。有効な式を必要に応じて入力できます。

戻り値

負の値の場合、-1。

0 の場合、0。

正の値の場合、1。

NULL の場合、NULL。

例

次の式は、SALES カラムに負の値が含まれるかどうかを確認します。

```
SIGN( SALES )
```

SALES	RETURN VALUE
100	1
-25.99	-1
0	0
NULL	NULL

SIN

数値（ラジアン単位）の正弦を返します。

構文

```
SIN( numeric_value )
```

引数	必須/オプション	説明
numeric_value	必須	数値データ型。ラジアンで表された数値データ（角度に π を掛け、180 で割ったもの）。正弦を計算したい値を渡します。有効な式を必要に応じて入力できます。演算子を使って、SIN の計算時に数値をラジアンに変換したり算術演算を行ったりすることもできます。

戻り値

Double 値。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、Degrees カラムの値をラジアンに変換してから、各ラジアンに対して正弦を計算します。

```
SIN( DEGREES * 3.14159265359 / 180 )
```

DEGREES	RETURN VALUE
0	0
90	1
70	0.939692620785936
30	0.50000000000003
5	0.0871557427476639

DEGREES	RETURN VALUE
18	0.309016994374967
89	0.999847695156393
NULL	NULL

SIN 関数で正弦を計算する前に、SIN に渡す値に算術演算を実行することができます。以下に例を示します。

```
SIN( ARCS * 3.14159265359 / 180 )
```

SINH

数値の双曲正弦を返します。

構文

```
SINH( numeric_value )
```

引数	必須/オプション	説明
numeric_value	必須	数値データ型。ラジアンで表された数値データ（角度に π を掛け、180 で割ったもの)。双曲正弦を計算したい値を渡します。有効な式を必要に応じて入力できます。

戻り値

Double 値。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、ANGLES カラムの値に対して双曲正弦を返します。

```
SINH( ANGLES )
```

ANGLES	RETURN VALUE
1.0	1.1752011936438
2.897	9.03225804884884
3.66	19.4178051793031
5.45	116.376934801486
0	0.0
0.345	0.35188478309993
NULL	NULL

ヒント

SINH 関数で双曲正弦を計算する前に、SINH に渡す値に算術演算を実行することができます。以下に例を示します。

```
SINH( MEASURES.ARCS / 180 )
```

SOUNDEX

文字列値を 4 字の文字列にエンコードします。

SOUNDEX は英語アルファベット（A～Z）の文字に対して機能します。入力文字列の最初の文字が、戻り値の先頭文字として使用され、それ以外の文字は、子音に基づく 3 桁の数値にエンコーディングします。

SOUNDEX は下記の規則に従って文字をエンコードします。

- *string* の最初の文字を戻り値の最初の文字として使用し、大文字にエンコードします。たとえば、SOUNDEX(‘John’)と SOUNDEX(‘john’)は両方とも‘J500’を返します。
- *string* の第一文字以降の最初の 3 つの異なる子音をエンコードし、残りは無視します。たとえば、SOUNDEX(‘JohnRB’)と SOUNDEX(‘JohnRBCD’)は両方とも‘J561’を返します。
- 音が似ている子音には同じコードを割り当てます。
  以下の表に、SOUNDEX の子音に対するエンコードガイドラインを示します。

コード	子音
1	B、P、F、V
2	C、S、G、J、K、Q、X、Z
3	D、T
4	L
5	M、N
6	R

- A、E、I、O、U、H、W は *string* の最初の文字である場合を除き、これらの文字をスキップします。たとえば、SOUNDEX(‘A123’)は‘A000’を返し、SOUNDEX(‘MAeiouhwC’)は‘M000’を返します。
- *string* が生成した文字数が 4 文字未満の場合、SOUNDEX は生成された文字列の残りの部分にゼロを埋め込みます。たとえば、SOUNDEX(‘J’)は‘J000’を返します。
- 上記の表にリストされたのと同じコードを使用する連続した子音が *string* に含まれている場合、SOUNDEX は最初の子音をエンコードし、残りの同じコードの子音はスキップします。たとえば、SOUNDEX(‘AbbpdMN’)は‘A135’を返します。
- *string* 内の数字をスキップします。たとえば、SOUNDEX(‘Joh12n’)と SOUNDEX(‘1John’)は両方とも‘J500’を返します。
- *string* が NULL の場合、または *string* 内のすべての文字が英字でない場合、NULL を返します。

構文

SOUNDEX(*string*)

引数	必須/オプション	説明
string	必須	文字列。エンコードしたい文字列値を渡します。有効な式を必要に応じて入力できます。

戻り値

文字列。

下記の条件のいずれかが真の場合、NULL となります。

- 関数に渡した値が NULL。
- *string* 内の文字が英字ではない。
- *string* は空です。

例

次の式は、EMPLOYEE_NAME カラムの値をエンコードします。

```
SOUNDEX( EMPLOYEE_NAME )SOUNDEX
```

EMPLOYEE_NAME	RETURN VALUE
John	J500
William	W450
jane	J500
joh12n	J500
1abc	A120
NULL	NULL

SQRT

負以外の数値の平方根を返します。

構文

SQRT(*numeric_value*)

引数	必須/オプション	説明
numeric_value	必須	正の数値。平方根を計算したい値を渡します。有効な式を必要に応じて入力できます。

戻り値

Double 値。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、NUMBERS カラムの値に対する平方根を返します。

```
SQRT( NUMBERS )
```

NUMBERS	RETURN VALUE
100	10
-100	None. データ統合 writes the row into the error rows file.
NULL	NULL
60.54	7.78074546557076

値-100 はセッション時にエラーとなります。これは、関数 SQRT が正の数値だけを評価するからです。負の値または文字値が渡されると、データ統合は、その行をエラー行ファイルに書き込みます。

SQRT 関数で平方根を計算する前に、SQRT に渡す値に算術演算を実行することができます。

STDDEV

関数に渡された数値の標準偏差を返します。STDDEV は統計データの分析に使用されます。

STDDEV には他の集計関数は 1 つしかネストできません。また、ネストされた関数は数値データ型を返す必要があります。詳細モードでは集計関数をネストできません。

マッピングタスクでのみ使用します。

構文

```
STDDEV( numeric_value [,filter_condition] )
```

引数	必須/オプション	説明
numeric_value	必須	数値データ型。この関数は、標準偏差を計算したい値、または他の関数の結果を渡します。有効なトランスフォーメーション式を必要に応じて入力できます。演算子を使用して、さまざまなフィールドの値の平均を取ることができます。
filter_condition	オプション	検索される行を制限します。フィルタ条件は数値であるか、TRUE、FALSE、または NULL の値が求められなければなりません。有効なトランスフォーメーション式を必要に応じて入力できます。

戻り値

数値。

関数に渡された値がすべて NULL である場合、または行が 1 つも選択されていない場合（たとえば、フィルタ条件の値がすべての行に対して FALSE または NULL であった場合）には、NULL です。

アグリゲータトランスフォーメーションで、行が 1 つしかないグループ化フィールドで STDDEV を使用すると、データ統合サーバーは標準偏差 0 を返し、詳細クラスタは NULL を返します。

NULL

値の 1 つが NULL であると、STDDEV はその値を無視します。ただし、すべての値が NULL である場合には、NULL を返します。

Group By

STDDEV は、トランスフォーメーションで定義したグループ化フィールドに基づいて値をグループ分けし、各グループについて 1 つの結果を返します。

グループ化フィールドがない場合、STDDEV はすべての行を 1 つのグループとして扱い、1 つの値を返します。

例

次の式は、TOTAL_SALES フィールド内で 2000.00 ドルを超えるすべての行の標準偏差を計算します。

```
STDDEV( SALES, SALES > 2000.00 )
```

SALES
2198.0
1010.90
2256.0
153.88
3001.0
NULL
8953.0

RETURN VALUE: 3254.60361129688

値 1010.90 と 153.88 は計算に含まれません。これは、*filter_condition* によって$2,000 を超える販売額が指定されているからです。

次の式は、SALES フィールド内のすべての行の標準偏差を計算します。

```
STDDEV(SALES)
```

SALES
2198.0
2198.0
2198.0
2198.0

RETURN VALUE: 0

戻り値は 0 です。これは、各行に同じ数値がある（標準偏差がない）からです。標準偏差がない場合、戻り値は 0 になります。

SUBSTR

文字列の一部を返します。SUBSTR は、空白を含むすべての文字を文字列の先頭から数えます。

構文

SUBSTR(*string*, *start* [,*length*])

引数	必須/オプション	説明
string	必須	文字列でなければなりません。検索したい文字列を渡します。有効な式を必要に応じて入力できます。数値を渡すと、関数はそれを文字列に変換します。
start	必須	整数でなければなりません。文字列内でカウントを開始する位置を示します。有効な式を必要に応じて入力できます。開始位置が正の数である場合、SUBSTR は文字列の先頭からその数だけ数えた位置を開始位置とします。開始位置が負の数である場合、SUBSTR は文字列の最後から数えた位置を開始位置とします。開始位置がゼロの場合、SUBSTR は文字列の最初の文字から検索を開始します。
length	オプション	0 より大きい整数でなければなりません。SUBSTR が返す文字の数を指定します。有効な式を必要に応じて入力できます。length 引数を省略すると、SUBSTR は開始位置から文字列の終わりまで、すべての文字を返します。負の整数またはゼロを渡すと、関数は空の文字列を返します。小数を渡すと、関数はそれを最も近い整数値に丸めます。

戻り値

文字列。

負またはゼロの length 値を渡した場合は、空の文字列です。

関数に NULL 値を渡した場合は NULL です。

例

以下の式は、PHONE カラム内の各行に対して、市外局番を返します。

```
SUBSTR( PHONE, 0, 3 )
```

PHONE	RETURN VALUE
809-555-0269	809
357-687-6708	357
NULL	NULL

```
SUBSTR( PHONE, 1, 3 )
```

PHONE	RETURN VALUE
809-555-3915	809

PHONE	RETURN VALUE
357-687-6708	357
NULL	NULL

以下の式は、PHONE カラム内の各行に対して、市外局番を除いた電話番号を返します。

```
SUBSTR( PHONE, 5, 8 )
```

PHONE	RETURN VALUE
808-555-0269	555-0269
809-555-3915	555-3915
357-687-6708	687-6708
NULL	NULL

負の開始値を渡すことにより、PHONE カラム内の各行に対して、電話番号を返すこともできます。式は、*length* 引数の結果を返す際には、元の文字列を左から順に読みます。

```
SUBSTR( PHONE, -8, 3 )
```

PHONE	RETURN VALUE
808-555-0269	555
809-555-3915	555
357-687-6708	687
NULL	NULL

start 引数、または *length* 引数で INSTR をネストして特定の文字列を検索し、その位置を返すことができます。

次の式は、文字列を最後の文字から順に評価します。文字列内の最後（一番右側）のスペースを見つけて、それより前にある文字をすべて返します。

```
SUBSTR( CUST_NAME,1,INSTR( CUST_NAME,' ' ,-1,1 ) - 1 )
```

CUST_NAME	RETURN VALUE
PATRICIA JONES	PATRICIA
MARY ELLEN SHAH	MARY ELLEN

次の式は、文字列から文字#を削除します。

```
SUBSTR( CUST_ID, 1, INSTR(CUST_ID, '#')-1 ) || SUBSTR( CUST_ID, INSTR(CUST_ID, '#')+1 )
```

length 引数が文字列より長い場合、SUBSTR は開始位置から文字列の終わりまで、すべての文字を返します。次の例を検討します。

```
SUBSTR('abcd', 2, 8)
```

戻り値は‘bcd’です。この結果を、以下の例と比較します。

```
SUBSTR('abcd', -2, 8)
```

戻り値は‘cd’です。

SUM

選択されたフィールドのすべての値の合計を返します。オプションとして、合計を計算するために読み込む行を制限するフィルタを適用できます。

SUM には他の集計関数は 1 つしかネストできません。また、ネストされた関数は数値データ型を返す必要があります。詳細モードでは集計関数をネストできません。

マッピングタスクでのみ使用します。

構文

SUM(*numeric_value* [, *filter_condition*])

引数	必須/オプション	説明
numeric_value	必須	数値データタイプ。追加したい値を渡します。有効なトランスフォーメーション式を必要に応じて入力できます。演算子を使用して、さまざまなフィールドの値を加算できます。
filter_condition	オプション	検索される行を制限します。フィルタ条件は数値であるか、TRUE、FALSE、または NULL の値が求められなければなりません。有効なトランスフォーメーション式を必要に応じて入力できます。

戻り値

数値。

関数に渡された値がすべて NULL である場合、または行が 1 つも選択されていない場合（たとえば、フィルタ条件の値がすべての行に対して FALSE または NULL であった場合）には、NULL です。

NULL

値の 1 つが NULL であると、SUM はその値を無視します。ただし、フィールドから渡された値がすべて NULL である場合、SUM は NULL を返します。

Group By

SUM は、トランスフォーメーションで定義したグループ化フィールドに基づいて値をグループ分けし、各グループについて 1 つの結果を返します。

グループ化フィールドがない場合、SUM はすべての行を 1 つのグループとして扱い、1 つの値を返します。

例

次の式は、Sales フィールド内の 2000 を超えるすべての値の合計を返します。

```
SUM( SALES, SALES > 2000 )
```

SALES
2500.0
1900.0
1200.0
NULL

SALES

3458.0

4519.0

RETURN VALUE: 10477.0

ヒント

SUM 関数で合計を計算する前に、SUM に渡す値に算術演算を実行することができます。以下に例を示します。

```
SUM( QTY * PRICE - DISCOUNT )
```

Systimestamp

タスクを開始する Secure Agent のホストシステムの現在の日時をナノ秒までの精度で返します。日付と時間を取得するときの精度は、Secure Agent をホストするシステムによって異なります。

関数の戻り値は引数の設定方法に応じて異なります。

- SYSTIMESTAMP の引数が変数として設定されている場合、データ統合はトランスフォーメーションの各行で関数を評価します。
- SYSTIMESTAMP の引数が定数として設定されている場合、データ統合はトランスフォーメーションの各行で関数を一度評価し、その値を保持します。

構文

```
SYSTIMESTAMP( [format] )
```

引数	必須/オプション	説明
format	オプション	値を取得する対象となるメトリック。最大個の履歴ログを指定できます。フォーマット文字列は一重引用符で囲んでください。フォーマット文字列は大文字と小文字を区別しません。たとえば、ミリ秒の精度で日時を表示するには、構文 SYSTIMESTAMP('MS')を使用します。デフォルトの精度はマイクロ秒（US）です。

戻り値

タイムスタンプ。入力値に基づく日付と時間を返します。精度は、プラットフォームによって異なります。

例

組織にはオンライン注文サービスがあり、リアルタイムでデータを処理します。SYSTIMESTAMP 関数を使用すると、ターゲットデータベースの各トランザクションに対してプライマリキーを生成できます。

以下のフィールドと値で Expression トランスフォーメーションを作成します。

Field Name	Field Type	Expression
Customer_Name	Input	n/a
Order_Qty	Input	n/a

Field Name	Field Type	Expression
Time_Counter	Variable	'US'
Transaction_ID	Output	SYSTIMESTAMP (Time_Counter)

実行時、SYSTIMESTAMP は各行に対して、マイクロ秒の精度でシステム時間を生成します。

Customer_Name	Order_Qty	Transaction_Id
Vani Deed	14	07/06/2007 18:00:30.701015000
Kalia Crop	3	07/06/2007 18:00:30.701029000
Vani Deed	6	07/06/2007 18:00:30.701039000
Harry Spoon	32	07/06/2007 18:00:30.701048000

TAN

数値（ラジアン単位）の 正接を返します。

構文

TAN(*numeric_value*)

引数	必須/オプション	説明
numeric_value	必須	数値データ型。ラジアンで表された数値データ（角度に π を掛け、180 で割ったもの)。正接を計算したい数値を渡します。有効な式を必要に応じて入力できます。

戻り値

Double 値。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、DEGREES カラムのすべての値の正接を返します。

TAN(DEGREES * 3.14159 / 180)

DEGREES	RETURN VALUE
70	2.74747741945531
50	1.19175359259435
30	0.577350269189672
5	0.0874886635259298

DEGREES	RETURN VALUE
18	0.324919696232929
89	57.2899616310952
NULL	NULL

TANH

この関数に渡された数値の双曲正接を返します。

構文

TANH(*numeric_value*)

引数	必須/オプション	説明
numeric_value	必須	数値データ型。ラジアンで表された数値データ（角度に π を掛け、180 で割ったもの)。双曲正接を計算したい数値を渡します。有効な式を必要に応じて入力できます。

戻り値

Double 値。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、ANGLES カラムの値の双曲正接を返します。

TANH(ANGLES)

ANGLES	RETURN VALUE
1.0	0.761594155955765
2.897	0.993926947790665
3.66	0.998676551914886
5.45	0.999963084213409
0	0.0
0.345	0.331933853503641
NULL	NULL

ヒント

TANH 関数で双曲正接を計算する前に、TANH に渡す値に算術演算を実行することができます。以下に例を示します。

```
TANH( ARCS / 360 )
```

TO_BIGINT

文字列または数値を Bigint 値に変換します。TO_BIGINT 構文にはオプションの引数があり、数字を近似値の整数に丸めるか、小数点以下を切り詰めるかを選択できます。TO_BIGINT は、先頭の空白を無視します。

構文

```
TO_BIGINT( value [, flag] )
```

引数	必須/オプション	説明
value	必須	文字列データ型または数値データ型。Bigint 値に変換したい値を渡します。有効な式を必要に応じて入力できます。
flag	オプション	小数点以下を切り捨てるか丸めるかを指定します。フラグは整数リテラルか、TRUE または FALSE の定数でなければなりません。 - TO_INTEGER は、フラグが TRUE または非ゼロの数字の場合に小数点以下を切り捨てます。 - フラグが FALSE またはゼロの場合、あるいはこの引数が省略された場合、TO_INTEGER は値を近似値の整数に丸めます。 既定ではフラグは設定されません。

戻り値

Bigint。

関数に NULL 値を渡した場合は NULL です。

関数に渡された値に英数字が含まれている場合はゼロです。

例

次の式は、IN_TAX ソースカラムからの値を使用します。

```
TO_BIGINT( IN_TAX, TRUE )
```

IN_TAX	RETURN VALUE
'7,245,176,201,123,435.6789'	7,245,176,201,123,435
'7,245,176,201,123,435.2'	7,245,176,201,123,435
'7,245,176,201,123,435.2.48'	7,245,176,201,123,435
NULL	NULL
'A12.3Grove'	0
' 176,201,123,435.87'	176,201,123,435

IN_TAX	RETURN VALUE
'-7,245,176,201,123,435.2’	-7,245,176,201,123,435
'-7,245,176,201,123,435.23'	-7,245,176,201,123,435
-9,223,372,036,854,775,806.9	-9,223,372,036,854,775,806
9,223,372,036,854,775,806.9	9,223,372,036,854,775,806

TO_BIGINT(IN_TAX)

IN_TAX	RETURN VALUE
'7,245,176,201,123,435.6789'	7,245,176,201,123,436
'7,245,176,201,123,435.2'	7,245,176,201,123,435
'7,245,176,201,123,435.348'	7,245,176,201,123,435
NULL	NULL
'A12.3Grove'	0
' 176,201,123,435.87'	176,201,123,436
'-7,245,176,201,123,435.6789'	-7,245,176,201,123,436
'-7,245,176,201,123,435.23'	-7,245,176,201,123,435
-9,223,372,036,854,775,806.9	-9,223,372,036,854,775,807
9,223,372,036,854,775,806.9	9,223,372,036,854,775,807

TO_CHAR (Dates)

日付を文字列に変換します。TO_CHAR は、数値を文字列に変換することもできます。TO_CHAR フォーマット文字列を使用して、日付を任意の形式に変換できます。

構文

TO_CHAR(*date* [,*format*])

引数	必須/オプション	説明
date	必須	Date/Time データ型。文字列に変換したい日付値を渡します。有効な式を必要に応じて入力できます。
format	オプション	正しい TO_CHAR フォーマット文字列を入力します。このフォーマット文字列は戻り値の形式を定義するものであり、date 引数の値の形式を定義するものではありません。フォーマット文字列を省略すると、関数はデフォルト日付形式（MM/DD/YYYY HH24:MI:SS）の文字列を返します。

戻り値

文字列。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、DATE_PROMISED カラム内の日付を MON DD YYYY 形式のテキストに変換します。

```
TO_CHAR( DATE_PROMISED, 'MON DD YYYY' )
```

DATE_PROMISED	RETURN VALUE
Apr 1 1998 12:00:10AM	'Apr 01 1998'
Feb 22 1998 01:31:10PM	'Feb 22 1998'
Oct 24 1998 02:12:30PM	'Oct 24 1998'
NULL	NULL

format_string 引数を省略すると、TO_CHAR はデフォルト日付形式の文字列を返します。

```
TO_CHAR( DATE_PROMISED )
```

DATE_PROMISED	RETURN VALUE
Apr 1 1998 12:00:10AM	'04/01/1997 00:00:01'
Feb 22 1998 01:31:10PM	'02/22/1997 13:31:10'
Oct 24 1998 02:12:30PM	'10/24/1997 14:12:30'
NULL	NULL

次の式は、カラム内の各日付の曜日を返します。

```
TO_CHAR( DATE_PROMISED, 'D' )
```

DATE_PROMISED	RETURN VALUE
04-01-1997 12:00:10AM	'3'
02-22-1997 01:31:10PM	'7'
10-24-1997 02:12:30PM	'6'
NULL	NULL

```
TO_CHAR( DATE_PROMISED, 'DAY' )
```

DATE_PROMISED	RETURN VALUE
04-01-1997 12:00:10AM	'Tuesday'
02-22-1997 01:31:10PM	'Saturday'
10-24-1997 02:12:30PM	'Friday'

DATE_PROMISED	RETURN VALUE
NULL	NULL

次の式は、カラム内の各日付の日の部分を返します。

```
TO_CHAR( DATE_PROMISED, 'DD' )
```

DATE_PROMISED	RETURN VALUE
04-01-1997 12:00:10AM	'01'
02-22-1997 01:31:10PM	'22'
10-24-1997 02:12:30PM	'24'
NULL	NULL

次の式は、カラム内の各日付に対して、その年の通算何日目になるかを返します。

```
TO_CHAR( DATE_PROMISED, 'DDD' )
```

DATE_PROMISED	RETURN VALUE
04-01-1997 12:00:10AM	'091'
02-22-1997 01:31:10PM	'053'
10-24-1997 02:12:30PM	'297'
NULL	NULL

次の式は、カラム内の各日付の時の部分を返します。

```
TO_CHAR( DATE_PROMISED, 'HH' )
TO_CHAR( DATE_PROMISED, 'HH12' )
```

DATE_PROMISED	RETURN VALUE
04-01-1997 12:00:10AM	'12'
02-22-1997 01:31:10PM	'01'
10-24-1997 02:12:30PM	'02'
NULL	NULL

```
TO_CHAR( DATE_PROMISED, 'HH24' )
```

DATE_PROMISED	RETURN VALUE
04-01-1997 12:00:10AM	'00'
02-22-1997 01:31:10PM	'13'
10-24-1997 11:12:30PM	'23'
NULL	NULL

次の式は、日付値を文字列で表された MJD 値に変換します。

```
TO_CHAR( SHIP_DATE, 'J')
```

SHIP_DATE	RETURN_VALUE
Dec 31 1999 03:59:59PM	2451544
Jan 1 1900 01:02:03AM	2415021

次の式は、日付を MM/DD/YY 形式の文字列に変換します。

```
TO_CHAR( SHIP_DATE, 'MM/DD/RR')
```

SHIP_DATE	RETURN_VALUE
12/31/1999 01:02:03AM	12/31/99
09/15/1996 03:59:59PM	09/15/96
05/17/2003 12:13:14AM	05/17/03

SSSSS フォーマット文字列を TO_CHAR 式で使用することもできます。例えば、次の式は SHIP_DATE カラムの日付を深夜 0 時からの通算秒を示す文字列に変換します。

```
TO_CHAR( SHIP_DATE, 'SSSSS')
```

SHIP_DATE	RETURN_VALUE
12/31/1999 01:02:03AM	3783
09/15/1996 03:59:59PM	86399

TO_CHAR 式では、YY フォーマット文字列は RR フォーマット文字列と同じ結果を返します。

次の式は、日付を MM/DD/YY 形式の文字列に変換します。

```
TO_CHAR( SHIP_DATE, 'MM/DD/YY')
```

SHIP_DATE	RETURN_VALUE
12/31/1999 01:02:03AM	12/31/99
09/15/1996 03:59:59PM	09/15/96
05/17/2003 12:13:14AM	05/17/03

次の式は、カラム内の各日付について、その月の何週目であるかを返します。

```
TO_CHAR( DATE_PROMISED, 'W' )
```

DATE_PROMISED	RETURN VALUE
04-01-1997 12:00:10AM	'01'
02-22-1997 01:31:10AM	'04'
10-24-1997 02:12:30PM	'04'
NULL	NULL

次の式は、カラム内の各日付について、その年の何週目であるかを返します。

```
TO_CHAR( DATE_PROMISED, 'WW' )
```

DATE_PROMISED	RETURN VALUE
04-01-1997 12:00:10PM	'18'
02-22-1997 01:31:10AM	'08'
10-24-1997 02:12:30AM	'43'
NULL	NULL

ヒント

TO_CHAR と TO_DATE を組み合わせて次のような関数を記述すれば、月の数値を対応する月の文字列に変換することができます。

```
TO_CHAR( TO_DATE( numeric_month, 'MM' ), 'MONTH' )
```

TO_CHAR（数値）

数値をテキスト文字列に変換します。TO_CHAR は、日付を文字列にも変換します。

TO_CHAR は、次のように数値をテキスト文字列に変換します。

- Double 値を 16 桁までの文字列に変換し、15 桁までの精度を表示します。15 桁を超える数値を渡すと、TO_CHAR は数値を 16 桁に丸めます。
- (-1e16,-1e-16]および[1e-16, 1e16)の範囲の数値は、10 進表記で返します。この範囲以外の数値は科学的表記で返します。

注: データ統合は、1e-16 と-1e16 の値は科学的表記に変換しますが、1e-16 と-1e-16 の値は 10 進数表記で返します。

構文

```
TO_CHAR( numeric_value )
```

引数	必須/オプション	説明
numeric_value	必須	数値データ型。文字列に変換したい数値です。有効な式を必要に応じて入力できます。

戻り値

文字列。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、SALES カラム内の値をテキストに変換します。

```
TO_CHAR( SALES )
```

SALES	RETURN VALUE
1010.99	'1010.99'
-15.62567	'-15.62567'
10842764968208837340	'1.084276496820884e+019' *(rounded to 16th digit)*
1.234567890123456789e-10	'0.00000000012345678901234 57' *(greater than 1e-16 but less than 1e16)*
1.23456789012345e17	'1.23456789012345e17' *(greater than 1e16)*
0	'0'
33.15	'33.15'
NULL	NULL

TO_DATE

文字列をそれと同じ形式で日付データ型に変換します。元の文字列の形式を指定するには、TO_DATE フォーマット文字列を使用します。

TO_DATE 式の場合、ターゲットカラムは date/time でなければなりません。

TO_DATE で 2 桁の年を変換する場合には、RR または YY のフォーマット文字列を使ってください。YYYY フォーマット文字列を使用してはなりません。

構文

```
TO_DATE( string [, format] )
```

引数	必須/オプション	説明
string	必須	文字列データ型であること。日付に変換したい値を渡します。有効な式を必要に応じて入力できます。
format	オプション	正しい TO_DATE フォーマット文字列を入力します。フォーマット文字列は、*string* 引数の各部分と一致しなければなりません。例えば、'Mar 15 1998 12:43:10AM'の文字列を渡す場合、フォーマット文字列'MON DD YYYY HH12:MI:SSAM'を使用する必要があります。フォーマット文字列を省略する場合は、文字列値がデフォルトの日付形式（MM/DD/YYYY HH24:MI:SS）になっていなければなりません。 TO_DATE フォーマット文字列の詳細については、「TO_DATE および IS_DATE フォーマット文字列」 (ページ 35)を参照してください。

戻り値

日付。

TO_DATE は常に日付と時刻を返します。時刻値を含まない文字列を渡すと、返される日付の時刻は常に 00:00:00 になります。この関数の結果は、Date データ型を持つ任意のターゲット列にマッピングできます。

関数に NULL 値を渡した場合は NULL です。

警告: フォーマット文字列は、日付区切り文字を含め、TO_DATE 文字列の形式と一致しなければなりません。一致しない場合、データ統合は不正確な値を返すか行をスキップする可能性があります。

例

次の式は、DATE_PROMISED カラムの文字列の日付値を返します。TO_DATE は常に日付と時刻を返します。時刻値を含まない文字列を渡すと、返される日付の時刻は常に 00:00:00 になります。20 世紀にセッションを実行した場合、年の最初の 2 桁は 19 になります。データ統合を実行しているコンピュータの現在の年は 1998 です。

```
TO_DATE( DATE_PROMISED, 'MM/DD/YY' )
```

DATE_PROMISED	RETURN VALUE
'01/22/98'	Jan 22 1998 00:00:00
'05/03/98'	May 3 1998 00:00:00
'11/10/98'	Nov 10 1998 00:00:00
'10/19/98'	Oct 19 1998 00:00:00
NULL	NULL

次の式は、DATE_PROMISED カラムの文字列の日付と時間の値を返します。時間の値を持たない文字列が渡された場合、データ統合は、その行をエラー行ファイルに書き込みます。20 世紀にセッションを実行した場合、年の最初の 2 桁は 19 になります。データ統合を実行しているコンピュータの現在の年は 1998 です。

```
TO_DATE( DATE_PROMISED, 'MON DD YYYY HH12:MI:SSAM' )
```

DATE_PROMISED	RETURN VALUE
'Jan 22 1998 02:14:56PM'	Jan 22 1998 02:14:56PM
'Mar 15 1998 11:11:11AM'	Mar 15 1998 11:11:11AM
'Jun 18 1998 10:10:10PM'	Jun 18 1998 10:10:10PM
'October 19 1998'	None. データ統合 writes the row into the error rows file.
NULL	NULL

次の式は、SHIP_DATE_MJD_STRING カラムの文字列をデフォルト日付形式の日付値に変換します。

```
TO_DATE (SHIP_DATE_MJD_STR, 'J')
```

SHIP_DATE_MJD_STR	RETURN_VALUE
'2451544'	Dec 31 1999 00:00:00
'2415021'	Jan 1 1900 00:00:00

J フォーマット文字列には日付の時間部分が含まれないため、戻り値では時間が 00:00:00 に設定されています。

次の式は、文字列を 4 桁形式の年に変換します。現在の年は 1998 です。

```
TO_DATE( DATE_STR, 'MM/DD/RR')
```

DATE_STR	RETURN VALUE
'04/01/98'	04/01/1998 00:00:00
'08/17/05'	08/17/2005 00:00:00

次の式は、文字列を 4 桁形式の年に変換します。現在の年は 1998 です。

```
TO_DATE( DATE_STR, 'MM/DD/YY')
```

DATE_STR	RETURN VALUE
'04/01/98'	04/01/1998 00:00:00
'08/17/05'	08/17/1905 00:00:00

注: 2 番目の行については、RR は 2005 年を返しますが、YY は 1905 年を返します。

次の式は、文字列を 4 桁形式の年に変換します。現在の年は 1998 です。

```
TO_DATE( DATE_STR, 'MM/DD/Y')
```

DATE_STR	RETURN VALUE
'04/01/8'	04/01/1998 00:00:00
'08/17/5'	08/17/1995 00:00:00

次の式は、文字列を 4 桁形式の年に変換します。現在の年は 1998 です。

```
TO_DATE( DATE_STR, 'MM/DD/YYY')
```

DATE_STR	RETURN VALUE
'04/01/998'	04/01/1998 00:00:00
'08/17/995'	08/17/1995 00:00:00

次の式は、深夜からの通算秒を含む文字を日付値に変換します。

```
TO_DATE( DATE_STR, 'MM/DD/YYYY SSSSS')
```

DATE_STR	RETURN_VALUE
'12/31/1999 3783'	12/31/1999 01:02:03
'09/15/1996 86399'	09/15/1996 23:59:59

ターゲットで複数の異なる日付形式を使用できる場合は、DECODE 関数で TO_DATE と IS_DATE を使用して、使用可能な形式をテストできます。以下に例を示します。

```
DECODE( TRUE,
 --test first format
   IS_DATE( CLOSE_DATE,'MM/DD/YYYY HH24:MI:SS' ),
   --if true, convert to date
      TO_DATE( CLOSE_DATE,'MM/DD/YYYY HH24:MI:SS' ),
 --test second format; if true, convert to date
   IS_DATE( CLOSE_DATE,'MM/DD/YYYY'), TO_DATE( CLOSE_DATE,'MM/DD/YYYY' ),
```

```
 --test third format; if true, convert to date
   IS_DATE( CLOSE_DATE,'MON DD YYYY'), TO_DATE( CLOSE_DATE,'MON DD YYYY'),
 --if none of the above
   ERROR( 'NOT A VALID DATE') )
```

TO_CHAR と TO_DATE を組み合わせて次のような関数を記述すれば、月の数値を対応する月の文字列に変換することができます。

```
TO_CHAR( TO_DATE( numeric_month, 'MM' ), 'MONTH' )
```

TO_DECIMAL

文字列または数値を 10 進値に変換します。TO_DECIMAL は、先頭のスペースを無視します。

構文

```
TO_DECIMAL( value [, scale] )
```

引数	必須/オプション	説明
値	必須	文字列データ型または数値データ型であること。10 進に変換したい値を渡します。有効な式を必要に応じて入力できます。
スケール	必須またはオプション	0 から 28 までの整数リテラル。小数点以下の桁数を指定します。この引数を省略すると、関数は入力値と同じ位取りの値を返します。 詳細モードでは、この引数を省略できません。

戻り値

0 から 28 までの精度と位取りを持つ 10 進値。

選択したカラムの値が空の文字列または数値以外の文字であった場合は、ゼロ。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、カラム IN_TAX からの値を使用します。データ型は Decimal で、精度は 10、スケールは 3 です。

```
TO_DECIMAL( IN_TAX, 3 )
```

IN_TAX	RETURN VALUE
'15.6789'	15.679
'60.2'	60.200
'118.348'	118.348
NULL	NULL
'A12.3Grove'	0

TO_FLOAT

文字列または数値を倍精度浮動小数点数（Double データ型）に変換します。 TO_FLOAT は、先頭のスペースを無視します。

構文

TO_FLOAT(*value*)

引数	必須/オプション	説明
value	必須	文字列データ型または数値データ型であること。Double 値に変換したい値を渡します。有効な式を必要に応じて入力できます。

戻り値

Double 値。

カラムの値が空白、または数値以外の文字であった場合は 0。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、カラム IN_TAX からの値を使用します。

TO_FLOAT(IN_TAX)

IN_TAX	RETURN VALUE
'15.6789'	15.6789
'60.2'	60.2
'118.348'	118.348
NULL	NULL
'A12.3Grove'	0

TO_INTEGER

文字列または数値を整数に変換します。TO_INTEGER 構文にはオプションの引数があり、数字を近似値の整数に丸めるか、小数点以下を切り詰めるかを選択できます。TO_INTEGER は、先頭のスペースを無視します。

構文

TO_INTEGER(*value* [, *flag*])

引数	必須/オプション	説明
value	必須	文字列データ型または数値データ型。整数に変換したい値を渡します。有効な式を必要に応じて入力できます。
flag	オプション	小数点以下を切り捨てるか丸めるかを指定します。フラグは整数リテラルか、TRUE または FALSE の定数でなければなりません。 - TO_INTEGER は、フラグが TRUE または非ゼロの数字の場合に小数点以下を切り捨てます。 - フラグが FALSE またはゼロの場合、あるいはこの引数が省略された場合、TO_INTEGER は値を近似値の整数に丸めます。

戻り値

整数。

関数に NULL 値を渡した場合は NULL です。

関数に渡された値に英数字が含まれている場合はゼロです。

例

次の式は、カラム IN_TAX からの値を使用します。

TO_INTEGER(IN_TAX, TRUE)

IN_TAX	RETURN VALUE
'15.6789'	15
'60.2'	60
'118.348'	118
NULL	NULL
'A12.3Grove'	0
' 123.87'	123
'-15.6789'	-15
'-15.23'	-15

TO_INTEGER(IN_TAX, FALSE)

IN_TAX	RETURN VALUE
'15.6789'	16
'60.2'	60
'118.348'	118
NULL	NULL

IN_TAX	RETURN VALUE
'A12.3Grove'	0
' 123.87'	124
'-15.6789'	-16
'-15.23'	-15

TRUNC (Dates)

日付を特定の年、月、日、時、または分に切り詰めます。また、TRUNC を使って数値を切り詰めることもできます。

日付の中の以下の部分を切り詰めることができます。

- **年。**日付の年の部分を切り詰めると、関数は入力された年の 1 月 1 日を返し、時刻を 00:00:00 に設定します。例えば、式 TRUNC(6/30/1997 2:30:55, 'YY')は 1/1/1997 00:00:00 を返し、TRUNC(12/1/1997 3:10:15, 'YY')は 1/1/1997 00:00:00 を返します。
- **月。**日付の月の部分を切り詰めると、関数は月の最初の日を返し、時刻を 00:00:00 に設定します。例えば、式 TRUNC(4/15/1997 12:15:00, 'MM')は 4/1/1997 00:00:00 を返し、TRUNC(4/30/1997 3:15:46, 'MM')は 4/1/1997 00:00:00 を返します。
- **日。**日付の日の部分を切り詰めると、その日付を返し、時刻を 00:00:00 に設定します。例えば、式 TRUNC(6/13/1997 2:30:45, 'DD')は 6/13/1997 00:00:00 を返し、TRUNC(12/13/1997 22:30:45, 'DD')は 12/13/1997 00:00:00 を返します。
- **時間。**日付の時の部分を切り詰めると、この関数は日付の分と秒をゼロに設定して返します。例えば、式 TRUNC(4/1/1997 11:29:35, 'HH')は 4/1/1997 11:00:00 を返し、TRUNC(4/1/1997 13:39:00, 'HH')は 4/1/1997 13:00:00 を返します。
- **分。**日付の分の部分を切り詰めると、この関数は日付の秒をゼロに設定して返します。例えば、式 TRUNC(5/22/1997 10:15:29, 'MI')は 5/22/1997 10:15:00 を返し、TRUNC(5/22/1997 10:18:30, 'MI')は 5/22/1997 10:18:00 を返します。

構文

TRUNC(*date* [,*format*])

引数	必須/オプション	説明
date	必須	Date/Time データ型。切り詰めを行う日付です。日付を求める有効な式を必要に応じて入力できます。
format	オプション	正しいフォーマット文字列を入力します。フォーマット文字列は大文字と小文字を区別しません。フォーマット文字列を省略すると、関数は日付の時刻部分を切り詰めて、00:00:00 に設定します。

戻り値

日付。

関数に NULL 値を渡した場合は NULL です。

例

以下の式は、DATE_SHIPPED カラムの日付の年の部分を切り詰めます。

```
TRUNC( DATE_SHIPPED, 'Y' )
TRUNC( DATE_SHIPPED, 'YY' )
TRUNC( DATE_SHIPPED, 'YYY' )
TRUNC( DATE_SHIPPED, 'YYYY' )
```

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:30AM	Jan 1 1998 12:00:00AM
Apr 19 1998 1:31:20PM	Jan 1 1998 12:00:00AM
Jun 20 1998 3:50:04AM	Jan 1 1998 12:00:00AM
Dec 20 1998 3:29:55PM	Jan 1 1998 12:00:00AM
NULL	NULL

以下の式は、DATE_SHIPPED カラムの各日付の月の部分を切り詰めます。

```
TRUNC( DATE_SHIPPED, 'MM' )
TRUNC( DATE_SHIPPED, 'MON' )
TRUNC( DATE_SHIPPED, 'MONTH' )
```

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:30AM	Jan 1 1998 12:00:00AM
Apr 19 1998 1:31:20PM	Apr 1 1998 12:00:00AM
Jun 20 1998 3:50:04AM	Jun 1 1998 12:00:00AM
Dec 20 1998 3:29:55PM	Dec 1 1998 12:00:00AM
NULL	NULL

以下の式は、DATE_SHIPPED カラムの各日付の日の部分を切り詰めます。

```
TRUNC( DATE_SHIPPED, 'D' )
TRUNC( DATE_SHIPPED, 'DD' )
TRUNC( DATE_SHIPPED, 'DDD' )
TRUNC( DATE_SHIPPED, 'DY' )
TRUNC( DATE_SHIPPED, 'DAY' )
```

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:30AM	Jan 15 1998 12:00:00AM
Apr 19 1998 1:31:20PM	Apr 19 1998 12:00:00AM
Jun 20 1998 3:50:04AM	Jun 20 1998 12:00:00AM
Dec 20 1998 3:29:55PM	Dec 20 1998 12:00:00AM
Dec 31 1998 11:59:59PM	Dec 31 1998 12:00:00AM
NULL	NULL

以下の式は、DATE_SHIPPED カラムの各日付の時の部分を切り詰めます。

```
TRUNC( DATE_SHIPPED, 'HH' )
TRUNC( DATE_SHIPPED, 'HH12' )
TRUNC( DATE_SHIPPED, 'HH24' )
```

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:31AM	Jan 15 1998 2:00:00AM
Apr 19 1998 1:31:20PM	Apr 19 1998 1:00:00PM
Jun 20 1998 3:50:04AM	Jun 20 1998 3:00:00AM
Dec 20 1998 3:29:55PM	Dec 20 1998 3:00:00PM
Dec 31 1998 11:59:59PM	Dec 31 1998 11:00:00AM
NULL	NULL

以下の式は、DATE_SHIPPED カラムの各日付の分の部分を切り詰めます。

```
TRUNC( DATE_SHIPPED, 'MI' )
```

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:30AM	Jan 15 1998 2:10:00AM
Apr 19 1998 1:31:20PM	Apr 19 1998 1:31:00PM
Jun 20 1998 3:50:04AM	Jun 20 1998 3:50:00AM
Dec 20 1998 3:29:55PM	Dec 20 1998 3:29:00PM
Dec 31 1998 11:59:59PM	Dec 31 1998 11:59:00PM
NULL	NULL

TRUNC（Numbers）

数値が特定の桁数になるように切り詰めます。また、TRUNC を使って日付を切り詰めることもできます。

構文

TRUNC(*numeric_value* [, *precision*])

引数	必須/オプション	説明
numeric_value	必須	数値データ型。切り詰めを行いたい値を渡します。数値データ型に評価する有効な式を入力します。
precision	オプション	正または負の整数です。整数を求める有効な式を必要に応じて入力できます。この整数は、切り詰め後の桁数を指定します。

precision が正の整数である場合、TRUNC は *precision* で指定された値を小数点以下の桁数として *numeric_value* を返します。*precision* が負の整数である場合、TRUNC は小数点の左側の指定された桁数をゼロに変更します。*precision* 引数を省略すると、TRUNC は *numeric_value* の小数部分を切り捨てて、整数を返します。

10 進数の *precision* 値が渡されると、データ統合は、*numeric_value* を近似値の整数に丸めてから式を評価します。

戻り値

数値。

いずれかの引数が NULL である場合は、NULL。

例

以下の式は、PRICE カラム内の値の切り詰めを行います。

```
TRUNC( PRICE, 3 )
```

PRICE	RETURN VALUE
12.9995	12.999
-18.8652	-18.865
56.9563	56.956
15.9928	15.992
NULL	NULL

```
TRUNC( PRICE, -1 )
```

PRICE	RETURN VALUE
12.99	10.0
-187.86	-180.0
56.95	50.0
1235.99	1230.0
NULL	NULL

```
TRUNC( PRICE )
```

PRICE	RETURN VALUE
12.99	12.0
-18.99	-18.0
56.95	56.0
15.99	15.0
NULL	NULL

UPPER

小文字の文字列を大文字に変換します。

構文

UPPER(*string*)

引数	必須/オプション	説明
string	必須	文字列データ型。大文字の文字列に変換したい値を渡します。有効な式を必要に応じて入力できます。

戻り値

大文字の文字列。データにマルチバイト文字が含まれる場合、タスクを実行する Secure Agent のコードページに応じた戻り値が返されます。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、FIRST_NAME カラム内のすべての名前を大文字に変換します。

UPPER(FIRST_NAME)

FIRST_NAME	RETURN VALUE
Ramona	RAMONA
NULL	NULL
THOMAS	THOMAS
PierRe	PIERRE
Bernice	BERNICE

VARIANCE

渡された値の分散を返します。VARIANCE は統計データの分析に使用されます。

VARIANCE には他の集計関数は 1 つしかネストできません。また、ネストされた関数は数値データ型を返す必要があります。詳細モードでは集計関数をネストできません。

マッピングタスクでのみ使用します。

構文

VARIANCE(*numeric_value* [, *filter_condition*])

引数	必須/オプション	説明
numeric_value	必須	数値データタイプ。分散を計算したい値を渡します。有効なトランスフォーメーション式を必要に応じて入力できます。
filter_condition	オプション	検索される行を制限します。フィルタ条件は数値であるか、TRUE、FALSE、または NULL の値が求められなければなりません。有効なトランスフォーメーション式を必要に応じて入力できます。

計算式

この関数は、次の式を使って分散を計算します。

$$\frac{\sum_{i=1}^{n} x_i^2 - \frac{1}{n}\left[\sum_{i=1}^{n} x_i\right]^2}{n-1}$$

この式では、以下のガイドラインを使用します。

- *xi* は数値の 1 つです。
- *n* は、数値セットの要素の数です。n が 1 の場合、分散は 0 になります。

戻り値

Double 値。

関数に渡された値がすべて NULL である場合、または行が 1 つも選択されていない場合（たとえば、フィルタ条件の値がすべての行に対して FALSE または NULL であった場合）には、NULL です。

NULL

値の 1 つが NULL である場合、VARIANCE はその値を無視します。ただし、関数に渡された値がすべて NULL である場合、または行が 1 つも選択されていない場合、VARIANCE は NULL を返します。

Group By

VARIANCE は、トランスフォーメーションで定義したグループ化フィールドに基づいて値をグループ分けし、各グループについて 1 つの結果を返します。

グループ化フィールドがない場合、VARIANCE はすべての行を 1 つのグループとして扱い、1 つの値を返します。

例

次の式は、TOTAL_SALES フィールド内のすべての行の差異を計算します。

```
VARIANCE( TOTAL_SALES )
```

TOTAL_SALES
2198.0
2256.0

TOTAL_SALES
3001.0
NULL
8953.0

RETURN VALUE: 10592444.6666667

第 6 章

システム変数

現在のタスクや実行 ID など、変化するシステム値を返します。

注: 詳細モードでは使用できません。

$CurrentMappingName

タスクのベースとなっているマッピングの名前を文字列値として返します。文字列データ型を使用できる関数であれば、どの関数でも CurrentMappingName を使用できます。

データ統合は、タスクの実行時に CurrentMappingName を解決します。マッピングをテスト実行する場合は、CurrentMappingName は解決されません。マッピングを実行して現在のマッピング名を返すには、CurrentTaskName を使用します。

CurrentRunId

各タスク実行には、一意の ID が付けられます。CurrentRunId は、マッピングタスクに対してシステムで生成された実行 ID を文字列値として返します。文字列データ型を使用できる関数であれば、どの関数でも CurrentRunId を使用できます。

$CurrentTaskName

文字列値としてタスク名を返します。文字列データ型を使用できる関数であれば、どの関数でも CurrentTaskName を使用できます。マッピングを実行した場合はマッピング名も返します。

SESSSTARTTIME

ジョブを実行するエージェントの現在の日付と時刻を返します。

SESSSTARTTIME は、トランスフォーメーションの日付/時刻データ型の値として格納されます。トランスフォーメーションの日付/時刻データ型を使用できる関数であれば、どの関数でも SESSSTARTTIME を使用できます。SESSSTARTTIME は式の言語内だけで使用できます。

SYSDATE

トランスフォーメーションを通過する各行のエージェントシステムの日付と時刻を返します。

SYSDATE は、トランスフォーメーションの Date/Time データ型の変数として格納されます。

静的な日付と時刻を返すには、SESSSTARTTIME 変数を使用します。

第 7 章

データ型リファレンス

この章では、以下の項目について説明します。

データ型リファレンスの概要

マッピングまたはタスクを作成するときに、ソースからデータを読み取り、ターゲットに書き込むために、データ統合の一連の命令を作成します。データ統合は、データフローと各フィールドに割り当てられているデータ型に基づいて変換を行います。

データ統合は、次のタイプのデータ型を使用します。

ネイティブデータ型

ネイティブデータ型は、マッピングまたはタスクのソース、ターゲット、およびルックアップオブジェクトに固有です。マッピングでは、ソース、ターゲット、およびルックアップオブジェクトにネイティブデータ型が表示されます。

トランスフォーメーションデータ型

トランスフォーメーションデータ型は、マッピング内のすべてのトランスフォーメーションに表示される内部データ型です。これらは ANSI SQL-92 汎用データ型に基づいています。データ統合は、トランスフォーメーションのデータ型を使用して、プラットフォーム間のデータ移動を行います。

レプリケーションタスクなど、変換を行わずにデータを移動するタスクでは、データ統合は、ソースのネイティブデータ型をターゲット内の同等のネイティブデータ型に変換します。マッピングおよびマッピングタスクでは、データ統合は、データの変換を開始する前に、ソースのネイティブデータ型を同等のトランスフォーメーションデータ型に変換します。データ統合は、ターゲットへの書き込み時に、トランスフォーメーションデータ型を同等のネイティブデータ型に変換します。

マルチバイト文字セットを指定した場合、データ型は 3 バイトまで文字を格納する追加スペースをデータベース内に割り当てます。

データ型に関するルールとガイドライン

データ型と変換には以下のルールおよびガイドラインを使用します。

- フラットファイルのすべてのフィールドのデフォルトのデータ型は Nstring(255)です。
  データ型を変更する必要がある場合は、**［メタデータの編集］**オプションを使用して、フィールドのネイティブタイプまたは精度を編集します。
- ソースフィールドとターゲットフィールド間または式とターゲットフィールドの出力間で互換性のないデータ型をマッピングした場合、タスクは予期しない結果になる可能性があります。
  例えば、同期タスクで MySQL データベースソースの日時カラムを Salesforce ターゲットの整数カラムにマッピングした場合、同期タスクは失敗します。
- 精度が 28 を超える数値データは、ターゲットに書き込まれるときに精度 15 に切り詰められます。
  例えば、4567823451237864532451.12345678 はターゲットに 4567823451237864000000.00000000 として書き込まれる可能性があります。
- 以下のデータ型とデータベースタイプのソースフィールドを Salesforce ターゲットにマッピングした場合、同期タスクは失敗します。

データ型	データベースタイプ
tinyint	MySQL
tinyint	SQL Server
interval year to month	Oracle
interval day to second	Oracle
rowid	Oracle
urowid	Oracle

- ほとんどの場合、レプリケーションタスクはソースと同じ精度とスケールをターゲットに作成します。それ以外の場合、レプリケーションタスクは異なる精度またはスケールを使用します。
  - 場合によっては、レプリケーションタスクが精度またはスケールを指定せず、データベースがデフォルトを使用することがあります。例えば、ソースが MySQL でソースのデータ型が Double(12)とします。レプリケーションタスクは Oracle ターゲットに Number データ型を作成し、精度を指定しません。
  - ソースが Oracle ではなくターゲットが Oracle、ソースのデータ型が nchar、nvarchar、または nclob の場合、アプリケーションはソースフィールドの精度を 2 倍（最大 64000）にし、ターゲットファイルの精度を取得します。
  - ソースが MySQL でターゲットが Microsoft SQL Server 2000 または 2005、ソースのデータ型が日付または時刻の場合、ターゲットフィールドのデータ型は Timestamp(23, 3)になります。
- レプリケーションタスクが MySQL のデータをフラットファイルに書き込み、ソースに時刻データが含まれる場合、データ統合は時刻データを date/time データ型に変換します。このとき、日付は現在日付、時刻はソースで指定された時刻になります。
  式で文字列関数を使用して、フラットファイルをロードする前に日付を削除できます。
- ソースまたはターゲットが Oracle の場合、Number フィールドの精度はスケール以上にする必要があります。そうでない場合、タスクは失敗します。
- 同期タスクが Salesforce からスケールのない 17 桁または 18 桁の数値データを Number データ型の Oracle カラムに書き込むと、タスクはターゲットに予期しない出力を生成する可能性があります。

例えば、同期タスクは Salesforce の値 67890123456789045 を Oracle ターゲットに 67890123456789048 として書き込みます。

- Oracle データベースターゲットで ODBC 接続を使用する場合、Oracle テーブルカラムの最大精度が char(1999)、varchar(3999)、nvarchar(3998)、nchar(3998)を超えていないことを確認します。
- データが Microsoft SQL Server の Real データ型のソースフィールドから取得される場合、タスクはターゲットに破損データをロードする可能性があります。

トランスフォーメーションデータ型

以下の表で、トランスフォーメーションのデータ型について説明します。

データタイプ	サイズ（バイト数）	説明
Bigint	8 バイト	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 精度 19、位取り 0。 整数値。
Binary	精度	1〜104,857,600 バイト フラットファイルソースでバイナリデータは使用できません。
Date/Time	16 バイト	西暦 0001 年 1 月 1 日〜西暦 9999 年 12 月 31 日 精度 29、位取り 9 （精度はナノ秒まで） 日付値と時刻値が結合されたデータ型。
Decimal	8 バイト（高精度がオフまたは 28 より大きい場合） 16 バイト（精度が 18 以下で高精度がオンの場合） 20 バイト（精度が 18 より大きく、28 以下の場合）	宣言された精度と位取りを持つ 10 進型の値。位取りは、精度以下にする必要があります。 精度 1〜28 桁、位取り 0〜28
Double	8 バイト	倍精度の浮動小数点数値。 精度と位取りは編集できます。スケールは精度以下にする必要があります。
Integer	4 バイト	-2,147,483,648〜2,147,483,647 精度 10、位取り 0 整数値。
Nstring	Unicode モード: （精度+ 1）* 2 ASCII モード: 精度+ 1	1〜104,857,600 文字 固定長または可変長文字列。
Ntext	Unicode モード: （精度+ 1）* 2 ASCII モード: 精度+ 1	1〜104,857,600 文字 固定長または可変長文字列。

データタイプ	サイズ（バイト数）	説明
Real	8 バイト	精度 7、位取り 0 倍精度の浮動小数点数値。
Small Integer	4 バイト	-32,768 および 32,767 精度 5、位取り 0 整数値。
String	Unicode モード: （精度+ 1）* 2 ASCII モード: 精度+ 1	1〜104,857,600 文字 固定長または可変長文字列。
Text	Unicode モード: （精度+ 1）* 2 ASCII モード: 精度+ 1	1〜104,857,600 文字 固定長または可変長文字列。

Integer データ型

ソースからターゲットに整数データを渡して、整数データでトランスフォーメーションを実行できます。データ統合は、Bigint、Integer、および Small Integer データ型をサポートします。

トランスフォーメーションの Integer データ型は、厳密値を表します。

計算における整数値

計算に整数値を使用した場合、データ統合では、計算実行の前に、整数値が浮動小数点数に変換されることがあります。

例えば、MOD(12.00, 5)を評価するために、データ統合は、除算を実行する前に整数値「5」を浮動小数点数に変換します。データ統合は、マッピングタスクの**［高精度を有効にする］**詳細セッションプロパティの設定内容に応じて、整数値を倍精度浮動小数点数または 10 進数の値に変換します。

データ統合は、次の算術演算で整数値を変換します。

操作	高精度が無効	高精度が有効
小数点を使用できない関数と計算。 例えば、整数の加算、減算、乗算、および CUME、SUM などの関数。	変換なし。 ただし、計算結果が範囲外となった場合、データ統合は行エラーを書き込みます。	Decimal
小数点を使用できる非科学関数と計算。 たとえば、整数の除算、および AVG、MEDIAN、PERCENTILE などの関数。	Double	Decimal
すべての科学関数、EXP 関数、LN 関数、LOG 関数、POWER 関数、および SQRT 関数。	Double	Double

トランスフォーメーションの Double データ型は最大 15 桁の精度をサポートし、Bigint データ型は最大 19 桁の精度をサポートします。このため、計算結果が、精度が 15 桁より大きい Bigint 値となる計算では、精度の損失が発生することがあります。

例えば、Expression トランスフォーメーションには以下の計算が含まれています。

```
POWER( BIGINTVAL, EXPVAL )
```

データ統合は、計算を開始する前に、POWER 関数への入力を倍精度浮動小数点数値に変換します。BIGINTVAL フィールドに Bigint 値 9223372036854775807 が含まれている場合、データ統合はこの値を 9.22337203685478e+18 に変換し、最後の 4 桁の精度が失われます。EXPVAL フィールドに値 1.0 が含まれており、結果フィールドが Bigint の場合、計算結果の 9223372036854780000 が最大 Bigint 値を超えているため、行エラーが作成されます。

結果が 10 進数値となる計算で Integer データ型を使用しており、高精度を有効にした場合、データ統合では、整数値が 10 進数値に変換されます。

最大 28 桁精度の Decimal データ型をサポートするトランスフォーメーションの場合、計算結果が高精度モードで 28 桁より大きい値にならない限り、精度の損失は発生しません。この場合、データ統合は、結果を倍精度浮動小数点数として格納します。フィールドの精度が 28 桁以下で、結果が高精度モードの 28 桁より大きい場合、データ統合は行を却下します。

式の整数定数

データ統合では、計算結果が整数の場合でも、式の定数が浮動小数点数として解釈されます。

例えば、式 `INTVALUE + 1000` では、データ統合は高精度が無効な場合に整数値「1000」を倍精度浮動小数点数値に変換します。高精度が有効な場合は、値「1000」を 10 進数値に変換します。値 1000 を整数値として処理するには、Integer データ型を使用して変数フィールドを作成して定数を保持し、式を変更して 2 つのフィールドを追加します。

NaN 値

NaN（非数）とは、通常、特に浮動小数点数の計算で、無効な入力オペランドでの演算の結果として返される値のことです。たとえば、ゼロをゼロで除算する演算の場合、結果として NaN が返されます。

NaN の表示は、オペレーティングシステムおよびプログラミング言語によって異なります。例えば、以下のリストのような NaN の表示があります。

- nan
- NaN
- NaN%
- NAN
- NaNQ
- NaNS
- qNaN
- sNaN
- 1.#QNAN
- 1.#SNAN

データ統合は QNAN 値を 1 に変換します。1.#QNAN に変換します。1.#QNAN は、NaN の有効な表示です。

詳細クラスタで実行した式の結果が NaN となる場合、出力内の式の戻り値は空白になります。

文字列値の整数値への変換

データ統合は、文字列値の整数値への暗黙の変換を実行する場合、最初の数値以外の文字でデータを切り捨てます。

例えば、「9,000,000,000,000,000,000.777」という値を含む文字列フィールド Bigint フィールドにリンクするとします。データ統合は、文字列を Bigint 値 9,000,000,000,000,000,000 に変換します。

フラットファイルへの整数値の書き込み

整数値を固定長のフラットファイルに書き込む場合、ファイルライタではデータが範囲内であることが確認されません。

例えば、ターゲットカラムのフィールド長が少なくとも 13 の場合、ファイルライタによりターゲットの Integer カラムに結果 3,000,000,000 が書き込まれます。ファイルライタは、結果が整数値の有効範囲外であるため、行を拒否しません。

Binary データ型

マッピングにバイナリデータが含まれている場合は、データ統合がデータをソースからターゲットに移動するために必要なメモリを割り当てることができるように、トランスフォーメーションの Binary データ型の精度を設定します。

フラットファイルソースで Binary データ型は使用できません。

Date/Time データ型

Date/Time データ型は、グレゴリオ暦の西暦 1 年から西暦 9999 年までの年を処理します。西暦 9999 年より後の年を使用するとエラーが発生します。マッピングは、Secure Agent マシンのローカルタイムゾーンを使用して、Parquet、Avro、ORC などの復号ファイル形式の Date/Time データ型を処理します。

Date/Time データ型は、ナノ秒単位の精度で日付をサポートします。データ型の精度は 29 で、位取りは 9 です。ネイティブデータ型の中には、精度が低いものがあります。日時の値を含むソースをインポートするときに、インポートプロセスによってソースカラムから正しい精度がインポートされます。例えば、Microsoft SQL Server の Datetime データ型の精度は 23 で、位取りは 3 です。日時の値を含む Microsoft SQL Server ソースをインポートすると、マッピングソースの日時カラムは精度が 23 になり、スケールが 3 になります。

データ統合は、マッピングソースに指定されている精度で、ソースから日時の値を読み取ります。データ統合は、日時の値を変換する際に、29 桁までの精度をサポートします。例えば、ミリ秒の精度で日付の値をインポートする場合、Expression トランスフォーメーションで ADD_TO_DATE 関数を使用して日付にナノ秒を追加することができます。

より短い精度をサポートするターゲットカラムに Date/Time 値を書き込むと、データ統合では、ターゲットカラムの精度まで値が切り詰められます。より長い精度をサポートするターゲットカラムに Date/Time 値を書き込むと、データ統合では、日時の値でサポートされていない部分にゼロが挿入されます。

詳細クラスタは、マイクロ秒までの Date/Time の精度を処理します。Date/Time の値にナノ秒が含まれている場合、詳細クラスタでは末尾の数字が切り詰められます。

Decimal データ型および Double データ型

Decimal データおよび Double データをソースからターゲットに渡して、Decimal データおよび Double データに対するトランスフォーメーションを実行することができます。

データ統合は、次のデータ型をサポートしています。

Decimal

精度 1〜28 桁、位取り 0〜28。位取りが精度より大きい 10 進型の値、および負の精度を持つ 10 進型の値は使用できません。トランスフォーメーションには Decimal データ型に割り当てられている範囲がすべて表示されますが、データ統合でサポートされる精度は、最大 28 桁までです。

高精度を有効にしており、フィールド精度が 28 桁より大きい場合、データ統合では、結果が Double 型として保存されます。

Double

倍精度の浮動小数点数値。

精度と位取りは編集できます。スケールは精度以下にする必要があります。

計算における Decimal 値と Double 値

計算結果が最大精度を超える値になる場合、Decimal と Double のデータ型を計算に使用すると、精度の損失が発生する可能性があります。

高精度を無効にすると、データ統合では、10 進数値が倍精度浮動小数点数に変換されます。Decimal 値が 15 桁を超える精度を持つ場合には、精度の損失が発生します。例えば、Decimal（20,0）を使って数値 40012030304957666903 を渡すマッピングがあるとします。高精度を無効にすると、データ統合は、10 進数値を倍精度浮動小数点数に変換し、$4.00120303049577 \times 10^{19}$ を渡します。

最大 28 桁の精度の Decimal データ型をサポートするトランスフォーメーションでは、Decimal データ型を使用し、高精度を有効にして、最大 28 桁の精度を確保します。

計算結果が、最大許容桁より大きい精度を持つ値にならない限り、精度の損失は発生しません。この場合、データ統合は、結果を倍精度浮動小数点数として格納します。

ルックアップ条件や結合条件など、等価条件で使用するデータに Double データ型を使用しないでください。

次の表に、データ統合が、**[高精度を有効にする]** 詳細セッションプロパティの設定内容に基づいて 10 進値を処理する方法を示します。

フィールドのデータ型	精度	高精度が無効	高精度が有効
Decimal	0〜15	Decimal	Decimal
Decimal	15〜28	Double	Decimal
Decimal	28 より大きい	Double	Double

高精度を有効にすると、データ統合では式関数の数値定数が 10 進数に変換されます。高精度を有効にしない場合、データ統合では数値定数が倍精度浮動小数点数に変換されます。

数値の最大精度を、トランスフォーメーションに応じて、28 または 38 桁よりも大きくすることができます。トランスフォーメーション関数を使用して計算またはトランスフォーメーションを実行する前に、大きい数字を切り詰めるか、丸めます。

Double 値の丸め処理方式

システムの実行時ライブラリと、データベースが double データ型計算を行うコンピュータシステムの違いにより、予想外の結果になる場合があります。プラットフォームの違いによる計算結果への影響を少なくするため、データ統合は、Double データ型の値の 15 桁の有効桁数を格納します。

double データ型は IEEE 794 標準に準拠します。データベースクライアントライブラリへの変更、データベースの異なるバージョン、システム実行時ライブラリへの変更は、算術的に同等な値のバイナリ表現に影響しま

す。また、多くのシステム実行時ライブラリは偶数丸め方法や対称算術法を実装しています。偶数丸め方法は、数字が次の上下の数の中間にある場合、偶数の最小位ビットの最も近い値に丸められます。たとえば、偶数丸め方法では、0.125 は 0.12 に丸められます。対称算術法では末尾桁が 5 以上の場合は次の桁に丸められます。たとえば、対称算術法では 0.125 は 0.13 に丸められ、0.124 は 0.12 に丸められます。

データ統合は、Double データ型の値の 15 桁の有効桁数を格納します。例えば、Windows で計算を実行すると、数値 1234567890.1234567890 が返され、同じ計算を UNIX で実行すると、1234567890.1234569999 が返される場合、データ統合はこの数値を 1234567890.1234600000 に変換します。

詳細モードでの Decimal 値と Double 値

詳細モードでは、マッピングの階層フィールドのタイプに基づいて、マッピングで高精度または低精度が使用されます。マッピング内の階層フィールドに 10 進数データ型の子フィールドが含まれている場合、マッピングは低精度を使用して実行されます。それ以外の場合、マッピングは高精度で実行されます。

詳細モードのマッピングで高精度が使用されている場合、詳細クラスタは、Decimal データ型を 38 の精度までの 10 進数として処理します。

詳細クラスタとデータ統合サーバーは、10 進数値を異なる方法でスケーリングします。データ統合サーバーでは 10 進数データの行間でスケールを変えることができますが、詳細クラスタでは行ごとに固定スケールが使用されます。例えば、詳細クラスタで 10 進数の 1.1234567 をスケール 9 で処理すると、出力は 1.123456700 になります。しかし、データ統合サーバーでは、この出力は 1.1234567 になります。

詳細モードのマッピングでデータがフラットファイルに書き込まれた場合、指数表記の 6 を超えるスケールを持つ 10 進数のソース値は、指数表記を使用せずにターゲットに書き込まれます。例えば、ソースの 10 進数値 0E-20 は、0.00000000000000000000 としてターゲットに書き込まれます。

文字列データ型

トランスフォーメーション文字列のデータ型には、Nstring、Ntext、String、および Text が含まれます。Nstring、Ntext、String、および Text データ型は 104,857,600 文字までの同精度をサポートしますが、データ統合は、ソースからターゲットに文字列データを移動する場合は String データ型を使用し、テキストデータを移動する場合は Text データ型を使用します。

一部のデータベースではテキストデータと文字列データの格納方法が異なるため、データ統合ではこの 2 つの文字データ型を区別する必要があります。ソースで String を表示する場合は、ターゲットカラムを String に設定します。同様に、Text を表示する場合は、ターゲットカラムを Text、Long、または Long Varchar（ソースに応じたデータ型）に設定します。

一般には、サイズの小さい文字列データ型（Char や Varchar など）は、ソース、ルックアップトランスフォーメーション、および SQL トランスフォーメーションでは String として表示され、サイズの大きいテキストデータ型（Text、Long、Long Varchar など）はソースでは Text として表示されます。

トランスフォーメーション内では、Nstring、Ntext、String、および Text を相互に入れ替えて使用できます。ただし、ソース、ルックアップトランスフォーメーション、および SQL トランスフォーメーションでは、ターゲットデータ型が一致している必要があります。データベースドライバは、データが正確に渡されるように文字列データ型とトランスフォーメーションデータ型を一致させる必要があります。たとえば、ルックアップテーブルの Nchar は、Lookup トランスフォーメーションの Nstring と一致しなければなりません。

索引

記号

数字

A

B

C

D

E

F

G

I

J

L

M

N

P

R

S

T

V

W

Y

ね

は

ふ

ま

み

め

も

ゆ

よ

り

れ