



Informatica®
10.4.0

トランスフォーメーション 言語リファレンス

Informatica トランスフォーメーション言語リファレンス
10.4.0
2019 年 12 月

© 著作権 Informatica LLC 2009, 2020

本ソフトウェアおよびマニュアルは、使用および開示の制限を定めた個別の使用許諾契約のもとでのみ提供されています。本マニュアルのいかなる部分も、いかなる手段（電子的複製、写真複製、録音など）によっても、Informatica LLC の事前の承諾なしに複製または転載することは禁じられています。

Informatica および Informatica ロゴは、米国およびその他の国における Informatica LLC の商標または登録商標です。Informatica の商標の最新リストは、Web (<https://www.informatica.com/trademarks.html>) にあります。その他の企業名および製品名は、それぞれの企業の商標または登録商標です。

米政府の権利プログラム、ソフトウェア、データベース、および関連文書や技術データは、米国政府の顧客に配信され、「商用コンピュータソフトウェア」または「商業技術データ」は、該当する連邦政府の取得規制と代理店固有の補足規定に基づきます。このように、使用、複製、開示、変更、および適応は、適用される政府の契約に規定されている制限およびライセンス条項に従うものとし、政府契約の条項によって適当な範囲において、FAR 52.227-19、商用コンピュータソフトウェアライセンスの追加権利を規定します。

本ソフトウェアまたはドキュメンテーション（あるいはその両方）の一部は、第三者が保有する著作権の対象となります。必要な第三者の通知は、製品に含まれています。

本マニュアルの情報は、予告なしに変更されることがあります。このドキュメントで問題が見つかった場合は、infa_documentation@informatica.com までご報告ください。

Informatica 製品は、それらが提供される契約の条件に従って保証されます。Informatica は、商品性、特定目的への適合性、非侵害性の保証等を含めて、明示的または黙示的ないかなる種類の保証をせず、本マニュアルの情報を「現状のまま」提供するものとします。

発行日: 2020-02-04

目次

序文	9
Informatica のリソース	9
Informatica Network	9
Informatica ナレッジベース	9
Informatica マニュアル	9
Informatica 製品可用性マトリックス	10
Informatica Velocity	10
Informatica Marketplace	10
Informatica グローバルカスタマサポート	10
第 1 章: トランスフォーメーション言語	11
トランスフォーメーション言語の概要	11
トランスフォーメーション言語の構成要素	11
国際化およびトランスフォーメーション言語	12
式の構文	12
式の構成要素	12
式の構文のルールとガイドライン	13
式へのコメント追加	14
予約語	15
第 2 章: 定数	16
DD_DELETE	16
例	16
DD_INSERT	16
例	17
DD_REJECT	17
例	17
DD_UPDATE	17
例	18
FALSE	18
例	18
NULL	18
ブール式における NULL 値の扱い	19
比較式での NULL 値	19
集計関数における NULL 値	19
フィルタ条件における NULL 値	19
NULL と演算子	19
TRUE	19
例	20

第 3 章 : 演算子	21
演算子の優先順位	21
複合演算子	22
添字演算子	23
ドット演算子	24
ネストされたデータ型の複合演算子	26
算術演算子	30
文字列演算子	31
NULL	31
例	31
比較演算子	32
論理演算子	33
NULL	33
第 4 章 : 変数	34
ビルトイン変数	34
SYSDATE	34
ローカル変数	34
第 5 章 : 日付	35
日付の概要	35
Date/Time データ型	35
ユリウス日、修正ユリウス日、およびグレゴリオ暦	36
西暦 2000 年の日付	36
リレーショナルデータベースの日付	38
フラットファイルの日付	38
デフォルトの日付形式	38
日付形式文字列	39
TO_CHAR 形式文字列	40
例	42
TO_DATE および IS_DATE 形式文字列	43
日付形式文字列のルールとガイドライン	45
例	45
日付の算術演算について	46
第 6 章 : 関数	47
関数の分類	47
集計関数	47
集計関数と NULL	49
文字関数	49
複合関数	50
変換関数	51

データクレンジング関数.	51
日付関数.	52
エンコード関数.	53
財務関数.	53
数値関数.	53
科学関数.	54
特殊関数.	54
文字列関数.	54
テスト関数.	55
ウィンドウ関数.	55
ABORT.	55
ABS.	56
ADD_TO_DATE.	57
AES_DECRYPT.	59
AES_ENCRYPT.	60
ANY.	61
ARRAY.	63
ASCII.	63
AVG.	64
CAST.	66
CEIL.	67
CHOOSE.	67
CHR.	68
CHRCODE.	69
COLLECT_LIST.	70
COLLECT_MAP.	71
COMPRESS.	72
CONCAT.	73
CONCAT_ARRAY.	74
CONVERT_BASE.	75
COS.	76
COSH.	76
COUNT.	77
CRC32.	80
CREATE_TIMESTAMP_TZ.	80
CUME.	81
DATE_COMPARE.	83
DATE_DIFF.	84
DEC_BASE64.	86
DECODE.	87
DECOMPRESS.	89
ENC_BASE64.	90

ERROR.	90
EXP.	91
FIRST.	92
FLOOR.	94
FV.	95
GET_DATE_PART.	96
GET_TIMEZONE.	97
GET_TIMESTAMP.	98
GREATEST.	99
IIF.	100
IN.	103
INDEXOF.	104
INITCAP.	105
INSTR.	106
ISNULL.	109
IS_DATE.	110
IS_NUMBER.	112
IS_SPACES.	114
LAG.	115
LAST.	117
LAST_DAY.	118
LEAD.	119
LEAST.	121
LENGTH.	122
LN.	123
LOG.	124
LOWER.	125
LPAD.	125
LTRIM.	127
MAKE_DATE_TIME.	128
MAP.	129
MAP_FROM_ARRAYS.	130
MAP_KEYS.	131
MAP_VALUES.	132
MAX (Dates).	132
MAX (Numbers).	133
MAX (String).	135
MD5.	136
MEDIAN.	137
METAPHONE.	138
MIN (Dates).	142
MIN (Numbers).	143

MIN (String).	145
MOD.	146
MOVINGAVG.	147
MOVINGSUM.	149
NPER.	150
PARSE_JSON.	151
PARSE_XML.	152
PERCENTILE.	153
PMT.	155
POWER.	156
PV.	157
RAND.	158
RATE.	159
REG_EXTRACT.	159
REG_MATCH.	162
REG_REPLACE.	163
REPLACECHR.	164
REPLACESTR.	167
RESPEC.	170
REVERSE.	171
ROUND (Dates).	172
ROUND (数值)	176
RPAD.	178
RTRIM.	179
SET_DATE_PART.	181
SIGN.	183
SIN.	184
SINH.	185
SIZE.	186
SOUNDEX.	187
SQL_LIKE.	189
SQRT.	190
STDDEV.	191
STRUCT.	192
STRUCT_AS.	193
SUBSTR.	194
SUM.	197
SYSTIMESTAMP.	198
TAN.	199
TANH.	200
TIME_RANGE.	201
TO_BIGINT.	202

TO_CHAR (Dates)	204
TO_CHAR (数值)	209
TO_DATE.	210
TO_DECIMAL.	214
TO_DECIMAL38.	215
TO_FLOAT.	216
TO_INTEGER.	217
TO_TIMESTAMP_TZ.	219
TRUNC (Dates)	220
TRUNC (数值)	223
UPPER.	225
UUID4.	226
UUID_UNPARSE.	226
VARIANCE.	226
索引	228

序文

Developer tool のトランスフォーメーション言語について理解するには、『*Informatica(R) Developer* トランスフォーメーション言語リファレンス』を参照してください。定数、演算子、変数、日付、および関数を使用してソースデータを変換する方法について学びます。

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 Network

Informatica Network は、Informatica ナレッジベースや Informatica グローバルカスタマサポートなど、多くのリソースへの入口です。Informatica Network を利用するには、<https://network.informatica.com> にアクセスしてください。

Informatica Network メンバーは、次のオプションを利用できます。

- ナレッジベースで製品リソースを検索できます。
- 製品の提供情報を表示できます。
- サポートケースを作成して確認できます。
- 最寄りの Informatica ユーザーグループネットワークを検索して、他のユーザーと共同作業を行えます。

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 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 製品可用性マトリックス

製品可用性マトリックス (PAM) には、製品リリースでサポートされるオペレーティングシステム、データベースなどのデータソースおよびターゲットが示されています。Informatica PAM は、<https://network.informatica.com/community/informatica-network/product-availability-matrices> で参照できます。

Informatica Velocity

Informatica Velocity は、Informatica プロフェッショナルサービスが開発したヒントとベストプラクティスのコレクションで、多数のデータ管理プロジェクトから得た実体験に基づいています。Informatica Velocity には、世界中の組織と連携してデータ管理ソリューションを計画、開発、デプロイ、管理する Informatica コンサルタントによる集合知を表しています。

Informatica Velocity リソースには、<http://velocity.informatica.com> からアクセスしてください。Informatica Velocity についての質問、コメント、またはアイデアがある場合は、ips@informatica.com から Informatica プロフェッショナルサービスにお問い合わせください。

Informatica Marketplace

Informatica Marketplace は、お使いの Informatica 製品を拡張したり強化したりするソリューションを検索できるフォーラムです。Marketplace で、Informatica デベロッパーやパートナーからの多数のソリューションを活用すれば、生産性を向上したり、プロジェクトでの実装時間を短縮したりできます。Informatica Marketplace は、<https://marketplace.informatica.com> からアクセスしてください。

Informatica グローバルカスタマサポート

電話または Informatica Network からグローバルサポートセンターに連絡できます。

各地域の Informatica グローバルカスタマサポートの電話番号は、Informatica Web サイト (<https://www.informatica.com/services-and-training/customer-success-services/contact-us.html>) を参照してください。

Informatica Network でオンラインサポートリソースを見つけるには、<https://network.informatica.com> にアクセスし、eSupport オプションを選択します。

第 1 章

トランスフォーメーション言語

この章では、以下の項目について説明します。

- [トランスフォーメーション言語の概要, 11 ページ](#)
- [式の構文, 12 ページ](#)
- [式へのコメント追加, 14 ページ](#)
- [予約語, 15 ページ](#)

トランスフォーメーション言語の概要

Informatica デベロッパでは、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。たとえば、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します。

この場合、ORDERS などのポートや、10 などの数値定数のみを含む単純な式を作成できます。また、複雑な式として、関数の中に別の関数をネストしたり、トランスフォーメーション言語演算子を使って異なる複数のポートを結合したりすることもできます。

トランスフォーメーション言語の構成要素

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、ユーザーはこれらを使用し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。
- **マッピングパラメータ。**各州のサービス税率のような、マッピングまたはマッピングの実行を通して一定の値を保つ値を参照するために、マッピングまたはマップレットで使用するマッピングパラメータを作成します。
- **ビルトイン変数およびローカル変数。**ビルトイン変数を使用して、変化する値（システム日付など）を参照する式を記述できます。トランスフォーメーション内でローカル変数を作成することもできます。
- **戻り値。**Lookup トランスフォーメーションからの戻り値が含まれる式を記述することもできます。

国際化およびトランスフォーメーション言語

トランスフォーメーション言語の関数は、ASCII または Unicode のいずれかのデータ移動モードで文字データを取り扱うことができます。Unicode モードは、マルチバイトデータを扱う場合に使用します。以下の関数およびトランスフォーメーションは、Data Integration Service のコードページおよびデータ移動モードに応じて異なる戻り値を返します。

- INITCAP
- LOWER
- UPPER
- MIN (Date)
- MIN (Number)
- MIN (String)
- MAX (Date)
- MAX (Number)
- MAX (String)
- 条件文を使って文字列を比較する関数 (IIF、DECODE など)

MIN と MAX の戻り値は、Data Integration Service のコードページに関連したソート順によっても異なります。

式のエディタを使って無効な式を検査すると、ダイアログボックスで式の中にエラーを示す“>>>>”が表示されます。このインジケータは、式の中でエラーを含む部分の左側に表示されます。たとえば、式 $a = b + c$ の c の部分にエラーがある場合、エラーメッセージは次のように表示されます。

$a = b + \quad >>>> \quad c$

文字データを評価するトランスフォーメーション言語関数は、バイト基準ではなく、文字基準で処理を行います。たとえば、LENGTH 関数が返すのは文字列中の文字数であり、バイト数ではありません。LOWER 関数は、Data Integration Service のコードページに従って小文字の文字列を返します。

式の構文

トランスフォーメーション言語は標準 SQL に基づいていますが、2 つの言語には異なる点もあります。たとえば、SQL は集計関数に対してキーワード ALL および DISTINCT をサポートしていますが、トランスフォーメーション言語ではサポートしていません。一方、トランスフォーメーション言語は集計関数に対してオプションのフィルタ条件をサポートしていますが、SQL ではサポートしていません。

簡単な式としては、1 つのポート (ORDERS など) または 1 つの数値リテラル (10 など) だけからなる式があります。また、複雑な式として、関数の中に別の関数をネストしたり、トランスフォーメーション言語演算子を使って異なる複数のカラムを結合したりすることもできます。

式の構成要素

式には、以下の要素を組み合わせたことができます。

- ポート (入力、入出力、変数)
- 文字列リテラル、数値リテラル
- 定数

- 関数
- ビルトイン変数とローカル変数
- マッピングパラメータ
- 演算子
- 戻り値

ポートと戻り値

コネクトされていないトランスフォーメーションからのポートまたは戻り値を含む式を記述する場合は、次の表の参照修飾子を使用します。

参照修飾子	説明
:LKP	<p>コネクトされていない Lookup トランスフォーメーションからの戻り値を含む式を作成する場合に必要です。一般的な構文は次のとおりです。</p> <pre>:LKP.lookup_transformation(argument1, argument2, ...)</pre> <p>引数はルックアップ条件で使用されるローカルポートです。引数の順序はトランスフォーメーションにおけるポートの順序と一致しなければなりません。ローカルポートのデータ型は、ルックアップ条件で使用される Lookup ポートのデータ型に必ず一致していなければなりません。</p>

文字列リテラルと数値リテラル

式には数値リテラルまたは文字列リテラルを含むことができます。

文字列リテラルは必ず一重引用符で囲んでください。以下に例を示します。

```
'Alice Davis'
```

文字列リテラルでは大文字と小文字が区別されます。一重引用符を除くすべての文字を使用できます。たとえば、次のような文字列は許されません。

```
'Joan's car'
```

一重引用符を含む文字列を返すには、CHR 関数を使用します。

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

数値リテラルでは一重引用符を使用しないでください。含めたい数値をそのまま入力します。以下に例を示します。

```
.05
```

または

```
$$Sales_Tax
```

式の構文のルールとガイドライン

式を記述する場合、以下のルールおよびガイドラインを適用します。

- Aggregator トランスフォーメーションには、単一レベルの集計関数とネストされた集計関数の両方を含めることはできません。
- 単一レベルの関数とネストされた関数の両方を作成する必要がある場合は、別々の Aggregator トランスフォーメーションを作成してください。
- 数式式では文字列を使用できません。

たとえば、`1 + '1'`という式は無効です。加算は数値データ型でしか実行できないからです。整数と文字列は加算できません。

- 文字列は数値パラメータとして使用できません。

たとえば、式 `SUBSTR(TEXT_VAL, '1', 10)` の場合、`SUBSTR` 関数は開始位置に文字列ではなく整数値が必要なため、無効になります。

- 比較演算子を使用する場合は、データ型を混在させることはできません。

たとえば、`123.4 = '123.4'`という式は無効です。小数と文字列を比較しているからです。

- 式に渡すことができる値は、ポートからの値、文字列リテラルまたは数値リテラル、Lookup トランスフォーメーションからの値、または他の式の結果です。
- 式のエディタの「ポート」タブを使用して、式にポート名を入力します。接続されたトランスフォーメーションのポート名を変更すると、Developer ツールは名称変更をトランスフォーメーションの式に反映させます。
- 関数内の各引数はカンマで区切ります。
- リテラルを除き、トランスフォーメーション言語では大文字と小文字は区別されません。
- リテラルを除き、Developer ツールと Data Integration Service ではスペースが無視されます。
- コロン (:), カンマ (,), ピリオド (.) は特別な意味を持っているため、構文を指定する場合にのみ使用します。
- Data Integration Service では、ダッシュ (-) がマイナス演算子として扱われます。
- 関数にリテラル値を渡す場合、文字列リテラルは一重引用符で囲みます。数値リテラルには引用符を使用しないでください。Data Integration Service では、一重引用符で囲まれた値はすべて文字列として扱われます。
- 式の中で関数にマッピングパラメータを渡す場合、マッピングパラメータを指定する際に引用符を使用しないでください。
- ポートを指定する際に引用符を使用しないでください。
- 式の中に複数の関数をネストすることができます（集計関数は1つしかネストできません）。Data Integration Service では、最も内側の関数から式の評価が開始されます。

式へのコメント追加

トランスフォーメーション言語では、式にコメントを挿入するために2種類の方法を提供しています。

- 2つのダッシュの後にコメントを記述します。
`-- These are comments`
- 2つのスラッシュの後にコメントを記述します。
`// These are comments`

Data Integration Service は、上記の2種類の方法で記述された行のテキストをすべて無視します。たとえば、2つの文字列を連結する場合、連結を記述する式の途中に次のようなコメントの付いた式を挿入することができます。

```
-- This expression concatenates first and last names for customers:
FIRST_NAME -- First names from the CUST table
|| // Concat symbol
LAST_NAME // Last names from the CUST table
// Joe Smith Aug 18 1998
```

Data Integration Service では、コメントを無視して、この式を次のように評価します。

```
FIRST_NAME || LAST_NAME
```

コメントを次の行に続けることはできません。

```
-- This expression concatenates first and last names for customers:
FIRST_NAME -- First names from the CUST table
|| // Concat symbol
LAST_NAME // Last names from the CUST table
Joe Smith Aug 18 1998
```

この場合、最後の行が正しい式でないため、Developer ツールは式を無効と判断します。

コメントを埋め込みたくない場合は、式エディタで [コメント] をクリックしてコメントを追加できます。

予約語

トランスフォーメーション言語で 사용되는いくつかのキーワード（定数、演算子、ビルトイン変数など）は、特定の関数に対する予約語となっています。予約語には以下のものがあります。

- :INFA
- :LKP
- :MCR
- :TYPE
- AND
- DD_DELETE
- DD_INSERT
- DD_REJECT
- DD_UPDATE
- FALSE
- NOT
- NULL
- OR
- PROC_RESULT
- SPOUTPUT
- SYSDATE
- TRUE

注: ポートやローカル変数の名前として予約語を使用することはできません。予約語はトランスフォーメーション式の中でのみ使用できます。予約語は式の中であらかじめ定義された意味を持ちます。

第 2 章

定数

この章では、以下の項目について説明します。

- [DD_DELETE, 16](#) ページ
- [DD_INSERT, 16](#) ページ
- [DD_REJECT, 17](#) ページ
- [DD_UPDATE, 17](#) ページ
- [FALSE, 18](#) ページ
- [NULL, 18](#) ページ
- [TRUE, 19](#) ページ

DD_DELETE

更新方式の式でレコードに削除フラグを設定する場合に使用します。DD_DELETE は整数リテラル 2 と等価です。

注: DD_DELETE 定数は、Update Strategy トランスフォーメーションでのみ使用します。複雑な数値式のトラブルシューティングを円滑に行うには、整数リテラル 2 の代わりに DD_DELETE を使用します。

例

下記に、ID 番号 1001 の項目を削除し、ほかのすべての項目を挿入する式を表わします。

```
IIF( ITEM_ID = 1001, DD_DELETE, DD_INSERT )
```

この更新方式の式では、数値リテラルを使っても同じ結果が得られます。

```
IIF( ITEM_ID = 1001, 2, 0 )
```

注: 数値リテラルを使った式よりも定数を使った式の方が読みやすくなります。

DD_INSERT

更新方式の式でレコードに挿入フラグを設定する場合に使用します。DD_INSERT は整数リテラル 0 と等価です。

注: DD_INSERT 定数は、Update Strategy トランスフォーメーションでのみ使用します。複雑な数値式のトラブルシューティングを円滑に行うには、整数リテラル 0 の代わりに DD_INSERT を使用します。

例

次の例は、販売員ごとの毎月の販売額を計算するマッピングを、1 人の販売員の販売額だけを調べられるように変更する式です。

次の更新方式の式では、従業員の販売額を挿入対象に設定し、その他はすべて拒否対象に設定しています。

```
IIF( EMPLOYEENAME = 'Alex', DD_INSERT, DD_REJECT )
```

この更新方式の式では、数値リテラルを使っても同じ結果が得られます。

```
IIF( EMPLOYEENAME = 'Alex', 0, 3 )
```

ヒント: 数値リテラルを使った式よりも定数を使った式の方が読みやすくなります。

DD_REJECT

更新方式の式でレコードに拒否フラグを設定する場合に使用します。DD_REJECT は整数リテラル 3 と等価です。

注: DD_REJECT 定数は、Update Strategy トランスフォーメーションでのみ使用します。複雑な数値式のトラブルシューティングを円滑に行うには、整数リテラル 3 の代わりに DD_REJECT を使用します。

DD_REJECT は、データのフィルタリングや検査に使用します。レコードに拒否フラグを設定すると、Data Integration Service はそのレコードをスキップし、セッション拒否ファイルに書き込みます。

例

次の例は、現在の月の販売額を計算するマッピングを変更して、正の数だけを含むようにする式です。

この更新方式の式では、0 より小さいレコードを拒否対象に、その他のレコードをすべて挿入対象に設定します。

```
IIF( SALES > 0, DD_INSERT, DD_REJECT )
```

この更新方式の式では、数値リテラルを使っても同じ結果が得られます。

```
IIF( SALES > 0, 0, 3 )
```

数値リテラルを使った式よりも定数を使った式の方が読みやすくなります。

以下のデータドリブンの例では、DD_REJECT および 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が書き込まれるのを回避しています。この式では、スペースのみからなるレコードが拒否対象に設定され、その他はすべて挿入対象に設定されます。

```
IIF( IS_SPACES( CUST_NAMES ), DD_REJECT, DD_INSERT )
```

DD_UPDATE

更新方式の式でレコードに更新フラグを設定する場合に使用します。DD_UPDATE は整数リテラル 1 と等価です。

注: DD_UPDATE 定数は、Update Strategy トランスフォーメーションでのみ使用します。複雑な数値式のトラブルシューティングを円滑に行うには、整数リテラル 1 の代わりに DD_UPDATE を使用します。

例

次の例は、現在の月の販売額を計算するマッピングを変更する式です。マッピングでは 1 人の従業員の販売額がロードされます。

この式では、Alex のレコードを更新対象にフラグを設定し、その他はすべて拒否対象に設定しています。

```
IIF( EMPLOYEENAME = 'Alex', DD_UPDATE, DD_REJECT )
```

この式では、数値リテラルを使っても同じ結果が得られます。Alex の販売額だけが更新対象 (1) となり、その他の販売額レコードはすべて拒否対象 (3) となります。

```
IIF( EMPLOYEENAME = 'Alex', 1, 3 )
```

数値リテラルを使った式よりも定数を使った式の方が読みやすくなります。

次の更新方式の式では、SYSDATE を使用し、過去 2 日間に出荷された注文だけを検索して挿入フラグを設定しています。DATE_DIFF を使用することにより、この式はシステム日付から DATE_SHIPPED を減算して、2 つの日付の差を返します。DATE_DIFF は Double 値を返すため、式では TRUNC を使って差を切り詰めています。その後で、結果を整数リテラル 2 と比較します。結果が 2 より大きい場合は、レコードに拒否フラグを設定します。結果が 2 以下の場合、レコードに更新フラグを設定します。それ以外の場合、拒否フラグを設定します。

```
IIF( TRUNC( DATE_DIFF( SYSDATE, ORDERS_DATE_SHIPPED, 'DD' ), 0 ) > 2, DD_REJECT, DD_UPDATE )
```

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、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( FALSE,  
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.')
```

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 0 (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、Data Integration Service がいずれかの制約のある列に NULL 値を書き込もうとした場合、データベースはその行を拒否し、Data Integration Service はそれを拒否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、0 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。

関連項目：

- [「関数」 \(ページ 47\)](#)

ブール式における 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。たとえば、Data Integration Service は下記の結果を生成します。

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

比較式での NULL 値

比較演算子を含む式で NULL 値を使用した場合は、Data Integration Service は NULL 値を生成します。コラム内の NULL 値をチェックするには、比較式で ISNULL() を使用する必要があります。

NULL 値が含まれない行を返すには、定数!=の代わりに ISNULL 関数を使用します。例えば、NOT ISNULL(Field_A)を使用します。

次の式の結果は NULL 値になり、フィルタトランスフォーメーションは行を返しません: Field_A!=NULL。

比較処理で NULL 値を高く、または低く扱うように、ルックアップトランスフォーメーションを設定することもできます。ルックアップソースの [NULL の順序付け] プロパティを使用し、Data Integration Service がルックアップトランスフォーメーションの比較式で NULL 値を処理する方法を設定します。

集計関数における NULL 値

Data Integration Service は集計関数において NULL 値を NULL として処理します。ポートまたはグループ全体の NULL 値を渡すと、関数は NULL を返します。

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。選択されたポートのすべてのレコードに対するフィルタ条件の評価結果が NULL となった場合、集計関数は NULL を返します（ただし COUNT 関数は 0 を返します）。集計関数や、CUME、MOVINGAVG、および MOVINGSUM 関数では、フィルタ条件を使用できます。

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, 21 ページ](#)
- [複合演算子, 22 ページ](#)
- [算術演算子, 30 ページ](#)
- [文字列演算子, 31 ページ](#)
- [比較演算子, 32 ページ](#)
- [論理演算子, 33 ページ](#)

演算子の優先順位

トランスフォーメーション言語では、複数の演算子を使用することができ、ネストした式の中でも演算子を使用できます。

複数の演算子を含む式を記述した場合、Data Integration Service は以下の順序で式を評価します。

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

Data Integration Service は、以下の表に示された順序に従い演算子进行评估します。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。

演算子	意味
+, -	加算、減算。
	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）。

トランスフォーメーション言語では、ネストした式の中で演算子を使用することもできます。式にかっこが含まれている場合、Data Integration Service 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

複合データ型の要素にアクセスするには、複合演算子を使用します。配列データ型、Map データ型または構造データ型の要素にアクセスできます。

複合演算子は Spark エンジンで実行されるマッピングで使用できます。

以下の表に、トランスフォーメーション言語の複合演算子を示します。

演算子	意味
[]	添字演算子。 添字演算子を使用して、配列内の 1 つ以上の要素にアクセスします。また、添字演算子を使用して、マップのキーと値のペアにある特定のキーに対応する値にアクセスすることもできます。
.	ドット演算子。 ドット演算子を使用して、構造の要素にアクセスします。また、構造の配列でドット演算子を使用して、各構造の要素にアクセスすることもできます。

構造の配列でドット演算子を使用すると、各構造内で同じ名前を持つ要素を配列として返します。ネストされた配列または構造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できます。

添字演算子

添字演算子を使用して、配列またはマップ内の要素にアクセスします。配列内の特定の要素または要素の範囲にアクセスできます。マップのキーと値のペアにある特定のキーに対応する値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]`

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
array	配列。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。
index	整数。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。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。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。
map	マップ。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として評価される有効なトランスフォーメーション式を入力できます。

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

インデックスを指定すると、式は配列の要素を返します。戻り値の型は、指定した配列の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。式が返すサブ配列の型構成は、指定された配列の型構成と同じです。

キーを指定した場合、式はマップ内のキーに関連付けられた値を返します。戻り値の型は、指定したマップ内の値のデータ型と同じです。

NULL

添字のインデックスが配列のサイズより大きい場合、添字演算子は NULL 値を返します。

インデックスが NULL の場合、添字演算子は NULL 値を返します。[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 である場合、式は NULL を返します。

配列が NULL の場合、添字演算子は NULL 値を返します。

キーがマップに存在しない場合、添字演算子は NULL 値を返します。

例

文字列要素を持つ次のような配列があるとします。

```
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]	[]

文字列型文字列のキーと値の要素を持つ次のマップがあります。

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

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

ドット演算子

ドット演算子を使用して、構造の要素にアクセスします。また、構造の配列でドット演算子を使用して、配列内の各構造の要素にアクセスすることもできます。

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
```

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
struct	構造。アクセスする要素が含まれる構造。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。
array_of_structs	構造要素を持つ配列。アクセスする各構造の要素が含まれる配列。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	アクセスする構造要素の名前。

戻り値

構造にドット演算子を使用すると、式は構造の要素を返します。戻り値の型は、指定した構造の要素のデータ型と同じです。

構造の配列にドット演算子を使用すると、式は各構造で指定された要素を含む配列を返します。

NULL

構造の要素に NULL 値がある場合、式は NULL を返します。

構造が NULL の場合、式は NULL を返します。

例

次のような構造があるとしてします。

```
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

また、ドット演算子を使用して構造の配列の要素にアクセスすることもできます。

例えば、次の配列には構造型の 3 つの要素があり、各構造には 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の文字列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]

ネストされたデータ型の複合演算子

ネストされたデータ型には、複合データ型の要素が含まれます。ネストされたデータ型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。

配列または構造に配列型または構造型の要素が含まれる場合は、複合演算子を組み合わせて要素にアクセスします。多次元配列の要素、構造要素を持つ配列、配列要素を持つ構造、構造要素を持つ構造にアクセスできます。

多次元配列

多次元配列は配列の配列であり、5 レベルまでネストできます。添字演算子を使用すると、任意のレベルの配列にアクセスしたり、最も内側のレベルにある配列内にある特定要素にアクセスしたりできます。

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特定要素。
- 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特定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、アクセスする 1 次元配列を決定します。2 番目の添字演算子は、配列内でアクセスする要素を決定します。

次の 2 次元配列には 3 つの配列が含まれています。各配列には文字列型の要素が含まれています。

```

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

```

次の式では、2 つの添字演算子を使用して、menu_array 内の各 1 次元配列の特定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

次の式では、単一の添字演算子を使用して、menu_array の 1 次元配列を返します。

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

次の式では、2 つの添字演算子を使用して、menu_array 内の配列のサブセットを返します。

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]]

構造要素を持つ配列

構造要素を持つ配列は、構造の配列です。添字とドット演算子の組み合わせを使用して、配列内にある構造内の要素にアクセスします。

配列内の構造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。演算子の順序を逆にすることもできます。戻り値は、演算子の順序に関係なく同じです。複合演算子の順序に基づいて、要素は次のようにアクセスされます。

添字演算子の後にドット演算子を使用する。

添字演算子が先に配列内のインデックス付き要素にアクセスし、構造を返します。次に、ドット演算子が構造内の要素にアクセスします。

ドット演算子の後に添字演算子を使用する。

ドット演算子は、各構造から同じ名前の要素を検索し、配列を返します。次に、添字演算子が配列内の要素にアクセスします。

例えば、次のような配列 `employee_info_array` があるとします。

```
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、配列 `employee_info_array` に対して添字演算子の後にドット演算子を使用します。

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

ドット演算子を先に使用すると、ドット演算子は、各構造から同じ名前の要素による配列を返します。例えば、次の式はドット演算子を使用したときの戻り値を示しています。

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

次に、添字演算子が返された配列内の要素にアクセスします。次の式では、ドット演算子の後に添字演算子を使用します。

Input Value	RETURN VALUE
<code>employee_info_array.name[0]</code>	'Derrick'
<code>employee_info_array.city[1]</code>	'Redwood City'
<code>employee_info_array.state[2]</code>	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

構造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の指定した配列要素にアクセスします。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。

例えば、配列要素 drinks、sandwiches、および salads を持つ次の構造があるとします。

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

式 menu_struct.drinks[0] を使用すると、ドット演算子が先に配列要素 drinks にアクセスします。次に、添字演算子が配列 drinks: ['milk','coffee','tea','chai'] の位置 0 にある要素にアクセスします。戻り値は milk です。

次の式では、ドット演算子の後に添字演算子を使用して、構造 menu_struct の配列要素にアクセスします。

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']

構造要素を持つ構造

1 つ以上のレベルの構造を含む構造は、ネストされた構造です。ドット演算子を使用すると、任意のレベルの構造にアクセスしたり、最も内側のレベルにある構造内にある特定要素にアクセスしたりできます。

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内の指定された要素。
- 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の特定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、構造の要素のデータ型と同じです。例えば、2 つのレベルでネストされた構造では 2 つのドット演算子を使用します。最初のドット演算子が親構造の指定された子構造要素にアクセスします。次に、2 番目のドット演算子が子構造の要素にアクセスします。

次の例では、2 つの子構造 home_address_info および department_info を含む構造 employee_info_struct を使用しています。

```
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 employee_info_struct の要素にアクセスします。

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、トランスフォーメーション言語の算術演算子の優先順位を示します。

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。たとえば、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。

注: トランスフォーメーション言語には組み込み日付関数が用意されており、日付や時刻の値に対して算術演算を実行できます。

関連項目：

- [「日付の算術演算について」 \(ページ 46\)](#)

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。

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。
<>	等しくない。
!=	等しくない。
^=	等しくない。

数値を比較する場合、または特定のポート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。

フィルタ条件の評価結果が NULL の場合、統合サービスは NULL を返します。

複合データ型の比較

配列や構造体などの複合データ型を比較する場合、等しい (=) 演算子および等しくない (!=) 演算子を使用します。

2つの配列を等価にするには、次の条件を適用する必要があります。

- 配列要素は同じデータ型である必要があります。
- 配列は同じサイズである必要があります。
- 各インデックスのエントリは同じである必要があります。

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
```

RETURN VALUE: TRUE (1)

どちらの配列も同じサイズで、各インデックスのエントリは、A[0]=B[0]、A[1]=B[1]、および A[2]=B[2] のように同じです。

2つの構造体を比較すると、構造体は次の条件を満たす場合に等価です。

- 対応する構造体要素は、同じデータ型である必要があります。
- 構造体は同じデータを持つ必要があります。

これらの条件が満たされている場合、構造体の要素の名前が異なっても、2つの構造体は等価です。

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
```

RETURN VALUE: TRUE (1)

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、0以外の値であれば TRUE、0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。たとえば、Data Integration Service は下記の結果を生成します。

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

第 4 章

変数

この章では、以下の項目について説明します。

- [ビルトイン変数, 34 ページ](#)
- [ローカル変数, 34 ページ](#)

ビルトイン変数

トランスフォーメーション言語には、システム日付を返すビルトイン変数 SYSDATE があります。SYSDATE は式の中で使用できます。たとえば、SYSDATE は DATE_DIFF 関数で使用できます。

SYSDATE

SYSDATE は、トランスフォーメーションにより渡される各行について、データを処理するノードの現在の日付と時刻を秒単位で返します。SYSDATE は、トランスフォーメーションの Date/Time データ型の値として格納されます。

例

次の式では、SYSDATE を使用し、過去 2 日間に出荷された注文を検索して挿入フラグを設定しています。DATE_DIFF を使用して、Data Integration Service はシステム日付から DATE_SHIPPED を減算し、2 つの日付の差を返します。DATE_DIFF は Double 値を返すため、式では差が切り詰められます。その後で、結果を整数リテラル 2 と比較します。結果が 2 より大きい場合は、行に拒否フラグを設定します。結果が 2 以下の場合は、挿入フラグを設定します。

```
IIF( TRUNC( DATE_DIFF( SYSDATE, DATE_SHIPPED, 'DD' ),  
0 ) > 2, DD_REJECT, DD_INSERT
```

ローカル変数

マッピングを使用する場合であれば、ローカル変数はマッピング内のどのトランスフォーメーション式でも使用できます。たとえば、ある複雑な税金計算をマッピング全体を通して使用する場合は、その式を一度だけ記述して、変数として指定することができます。これにより、Data Integration Service が実行する計算の回数は 1 回だけになるため、パフォーマンスが改善されます。

ローカル変数は、ストアードプロシージャ式で複数の戻り値を取得する場合に便利です。

第 5 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, 35 ページ](#)
- [日付形式文字列, 39 ページ](#)
- [TO_CHAR 形式文字列, 40 ページ](#)
- [TO_DATE および IS_DATE 形式文字列, 43 ページ](#)
- [日付の算術演算について, 46 ページ](#)

日付の概要

トランスフォーメーション言語には、日付に対するトランスフォーメーションを実行する助けとして、いくつかの日付関数と組み込み日付変数一式が用意されています。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、Data Integration Service のホストノードの現在の日付を取得できます。

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。

- **日付形式文字列。**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 形式文字列。**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 形式文字列。**日付への変換またはテストを行う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

Informatica では、さまざまなソースからデータを変換するために汎用的なデータ型が使用されます。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、ナノ秒単位の日時の値をサポートしています。Informatica は内部ではバイナリ形式で日付を格納します。

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。たとえば、次の式は文字列ポートを 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 )
```

グレゴリオ暦の西暦 1 年から西暦 9999 年の範囲の日付を使用できます。

ユリウス日、修正ユリウス日、およびグレゴリオ暦

グレゴリオ暦の日付のみを使用できます。ユリウス暦の日付はユリウス日（複数）と呼ばれ、Informatica 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。

修正ユリウス日（MJD）の形式は、J フォーマット文字列を使用して操作できます。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。

たとえば、次の TO_DATE 式は、SHIP_DATE_MJD_STRING ポート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 00:00:00.000000000
2415021	Jan 1 1900 00:00:00.000000000

SHIP_DATE_MJD_STR	RETURN_VALUE
2451544	Dec 31 1999 00:00:00.000000000
2415021	Jan 1 1900 00:00:00.000000000

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 00.000000000:00:00 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: Data Integration Service は、TO_CHAR 式の中の日付の時刻部分を無視します。

西暦 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。Informatica デベロッパ

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。

- **0-49 の現在の年。**現在の年が 0-49 の間であり（たとえば 2003 年）、元の文字列の年が 0-49 の場合、Data Integration Service は現在の世紀に元の文字列の 2 桁の年を足した値を返します。元の文字列の年が 50-99 の間であれば、Integration Service は前の世紀に元の文字列の 2 桁の年を足した値を返します。
- **50-99 の現在の年。**現在の年が 50-99 の間であり（たとえば 1998 年）、元の文字列の年が 0-49 の場合、Data Integration Service は次の世紀に元の文字列の 2 桁の年を足した値を返します。元の文字列の年が 50-99 の間であれば、Data Integration Service 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。

以下の表に、RR 形式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	ソースの年	RR 形式文字列の戻り値
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

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 00:00:00.000000000
'11/09/01'	11/09/2001 00:00:00.000000000

形式文字列 YY と RR の違い

Informatica デベロッパ 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っ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、各形式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 00:00:00.000000000	04/12/1998 00:00:00.000000000
11/09/01	1998	11/09/2001 00:00:00.000000000	11/09/1901 00:00:00.000000000
04/12/98	2003	04/12/1998 00:00:00.000000000	04/12/2098 00:00:00.000000000
11/09/01	2003	11/09/2001 00:00:00.000000000	11/09/2001 00:00:00.000000000

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。

リレーショナルデータベースの日付

一般に、リレーショナル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、サブ秒が含まれます。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

TO_DATE 関数を使って文字列を Date/Time 値に変換できます。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。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すには、まず TO_DATE 関数を使用して文字列をトランスフォーメーションの Date/Time データ型に変更する必要があります。

デフォルトの日付形式

Data Integration Service 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。デフォルトの日付フォーマットを指定するには、データビューアの設定で、[Datetime フォーマット文字列] 属性に日付のフォーマットを入力します。デフォルトの日付形式は、MM/DD/YYYY HH24:MI:SS.US です。

Informatica は日付をバイナリ形式で格納するため、Data Integration Service は以下のアクションを実行する場合に、デフォルトの日時形式を使用します。

- **Date/Time ポートを文字列ポートに接続することによって日付を文字列に変換する。** Data Integration Service は、日付を、データビューアの設定で定義されている日付形式の文字列に変換します。
- **文字列ポートを Date/Time ポートに接続することによって文字列を日付に変換する。** Data Integration Service では、文字列値がデータビューアの設定で定義された日付形式に一致していることを前提とします。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、Data Integration Service はその行をスキップします。文字列がこの形式になっている場合、Data Integration Service 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する。** 形式文字列を省略した場合、Data Integration Service はデータビューアの設定で定義された日付形式の文字列を返します。形式文字列を指定した場合、Data Integration Service 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する。** 形式文字列を省略した場合、Data Integration Service は文字列がデータビューアの設定で定義された日付形式に一致していることを前提とします。形式文字列を指定した場合、Data Integration Service は文字列が指定された形式に一致していることを前提とします。

デフォルト日付形式の MM/DD/YYYY HH24:MI:SS.US 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
- マイクロ秒

日付形式文字列

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形式文字列のリストを示します。

形式文字列	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」の部分指定します。
HH, HH12, HH24	時 (0-23)。0 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	日付の年の部分 (0001-9999)。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	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。

機能	説明
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 形式文字列

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 を使用して日付を文字列に変換することによって、レポート用に日付の形式を変更できます。

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形式文字列のリストを示します。

形式文字列	説明
AM、A.M. PM、P.M.	午前または午後。これらのいずれかのフォーマット文字列を使用して、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。
DAY	曜日の英語名（9 文字まで）。例：Wednesday。
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。
DY	曜日の英語名の 3 文字の略称。例：Wed。
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。カレンダー日付を、紀元前 4713 年 1 月 1 日 00:00:00 から計算した、その修正ユリウス日の値に相当する文字列に変換します。日付の時刻部分は無視されます。たとえば、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。

形式文字列	説明
MON	月の英語名の 3 文字の略称。例：Jan。
MS	ミリ秒 (0-999)
NS	ナノ秒 (0-999999999)
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 を使用した場合、Data Integration Service は日付の時刻部分しか評価できません。たとえば、式 TO_CHAR(SHIP_DATE, 'MM/DD/YYYY SSSSS')は「12/31/1999 01:02:03」を「12/31/1999 03723」に変換します。
US	マイクロ秒 (0-999999)
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。たとえば、'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。たとえば、'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。
-/.;:	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、以下のようにピリオドを使用して日付の要素を区切る式を作成します。 TO_CHAR(DATES, 'MM.DD.YYYY')
"テキスト"	出力に表示する文字列。たとえば、TO_CHAR(DATES, 'MM/DD/YYYY "Sales Were Up"')という式を使用して出力ポートを作成し、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""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: Data Integration Service 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	3723
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 形式文字列

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 形式文字列は国際化されていないため、定義済みの形式で入力する必要があります。

注: TO_DATE と IS_DATE では、同じ種類の形式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形式文字列を使用します。ソース文字列の形式と形式文字列が一致している必要があります。日付の検証を行うために日付区切り文字を一致させるようにする必要はありません。一致しない部分がある場合、Data Integration Service は文字列を変換せずに、その行をスキップします。形式文字列を省略する場合は、変換元の文字列がデータビューアの設定で指定された日付形式になっている必要があります。

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、データビューアの設定で指定された日付形式の任意の文字列を意味します。テストする文字列が指定した日付形式ではない場合、「TO_DATE and IS_DATE Format Strings」テーブルに一覧表示された文字列の形式を使用します。文字列の形式が指定された形式に一致しない場合、または文字列が正しい日付を表していない場合、FALSE (0)を返します。文字列の形式が指定した文字列の形式と一致していて、正しい日付である場合、TRUE (1)を返します。IS_DATE 形式文字列は国際化されていないため、下の表に示すいずれかの形式で入力する必要があります。

以下の表に、TO_DATE および IS_DATE 関数で使用する形式文字列を一覧表示します。

表 1. TO_DATE および IS_DATE 形式文字列

形式文字列	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。

形式文字列	説明
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が無視し、すべての日付に 00:00:00.000000000 の時刻を割り当てます。たとえば、TO_DATE('2451544','J')の式は 2451544 を Dec 31 1999 00:00:00.000000000 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 (0-999999999)
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。 - 50-99 の現在の年。現在の年が 50-99 の間であり (たとえば 1998 年)、変換元の文字列に含まれる年の値が 0-49 の間であれば、Data Integration Service 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 の間であれば、Data Integration Service 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。 - 0-49 の現在の年。現在の年が 0-49 の間であり (たとえば 2003 年)、変換元の文字列の年が 0-49 の場合、Data Integration Service は現在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。変換元の文字列の年が 50-99 の間であれば、Data Integration Service は前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。
SS	秒 (0～59)。
SSSSS	深夜 0 時からの通算秒。TO_DATE 式で SSSSS を使用した場合、Data Integration Service は日付の時刻部分しか評価できません。 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	Data Integration Service が実行されているノード上での現在の年の下 1 桁を、文字列値で置き換えます。
YY	Data Integration Service が実行されているノード上での現在の年の下 2 桁を、文字列値で置き換えます。
YYY	Data Integration Service が実行されているノード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。

日付形式文字列のルールとガイドライン

日付形式文字列を使用する場合は、以下のルールおよびガイドラインに従ってください。

- TO_DATE 文字列の形式は、形式文字列と一致する必要があります。一致しない場合、Data Integration Service 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。たとえば、2020 年 5 月 12 日を表す文字列 '20200512' を TO_DATE に渡す場合は、YYYYMMDD 形式文字列を指定する必要があります。形式文字列を指定しない場合、Data Integration Service は文字列の形式としてデータビューアの設定で指定された日付形式を想定します。同様に、形式文字列に一致しない文字列を渡すと、Data Integration Service はエラーを返して行をスキップします。例えば、TO_DATE に 2020120 の文字列を渡して YYYYMMDD 形式文字列を指定すると、文字列が形式文字列と一致しないため、Data Integration Service はエラーを返して行をスキップします。
- 形式文字列は一重引用符で囲んで指定する必要があります。
- Data Integration Service は、セッションで指定されているデフォルトの日時形式を使用します。デフォルトは MM/DD/YYYY HH24:MI:SS.US です。形式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 ポート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 00:00:00.000000000
2415021	Jan 1 1900 00:00:00.000000000

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 00.000000000:00:00 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 00:00:00.000000000
08/17/05	08/17/2005 00:00:00.000000000

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 00:00:00.000000000
08/17/05	08/17/1905 00:00:00.000000000

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 01:02:03.000000000
09/15/1996 86399	09/15/1996 23:59:59.000000000

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み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**。日付の特定部分を加算または減算を行います。
- **DATE_DIFF**。2 つの日付の減算を行います。
- **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) を使用して日付の加算や減算を行うことはできません。

トランスフォーメーション言語はうるう年を認識し、西暦 0001 年 1 月 1 日 00.000000000 時 00 分 00 秒から西暦 9999 年 12 月 31 日 23 時 59.999999999 分 59 秒までの日付に対応しています。

注: トランスフォーメーション言語では、トランスフォーメーションの Date/Time データ型を使っ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 6 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サポートについて説明します。

関数の分類

トランスフォーメーション言語には、以下の種類の関数が用意されています。

- 集計
- 文字
- 複合
- 変換
- データクレンジング
- 日付
- エンコーディング
- 財務
- 数値
- 科学
- 特殊
- 文字列
- テスト
- ウィンドウ

集計関数

集計関数は、選択した複数のポートの非 NULL 値をサマリした値を返します。集計関数は以下のような目的に使用できます。

- 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。
- Aggregator トランスフォーメーションで各グループに対して 1 つの値を返す。
- 選択したポート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。

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。

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

Data Integration Service を Unicode モードで動作するように設定している場合、MIN および MAX の戻り値は、マッピング設定で指定したコードページのソート順に従います。

集計関数は、Aggregator トランスフォーメーションで使用できます。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。Data Integration Service 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。たとえば、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 Aggregator トランスフォーメーション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
2	65
2	75
2	76
3	21

ID	EARNINGS
3	45
3	99

戻り値は 186 です。Data Integration Service は ID でグループ分けし、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。

式トランスフォーメーションでは、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。Spark エンジンでマッピングを実行するときに集計関数をウィンドウ関数として使用するには、ウィンドウ化のトランスフォーメーションを構成する必要があります。集計関数をウィンドウ関数として使用すると、式トランスフォーメーションがアクティブになります。

集計関数と NULL

Data Integration Service を設定する場合、集計関数の NULL 値の扱い方を選択できます。Data Integration Service が、集計関数における NULL 値を NULL として扱うか、または 0 として扱うかを指定できます。

デフォルトでは、Data Integration Service は集計関数において NULL 値を NULL として処理します。ポートまたはグループ全体の NULL 値を渡すと、関数は NULL を返します。必要に応じて、集計関数に NULL 値のポート全体を渡した場合には 0 を返すように Data Integration Service を設定できます。

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、Data Integration Service 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。たとえば、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、数値ポートが含まれています。

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

いずれの場合も、Data Integration Service はフィルタ条件に対して小数値を整数に丸めます（たとえば、1.5 は 2 に、1.2 は 1 に、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、関数はポート内のすべての行を選択します。

文字関数

トランスフォーメーション言語には、以下の文字関数が用意されています。

- ASCII
- CHR
- CHRCODE

- CONCAT
- INITCAP
- INSTR
- LENGTH
- LOWER
- LPAD
- LTRIM
- METAPHONE
- REPLACECHR
- REPLACESTR
- RPAD
- RTRIM
- SOUNDEX
- SUBSTR
- UPPER

文字関数 MAX、MIN、LOWER、UPPER、および INITCAP は、Data Integration Service のコードページを使用して文字データを求めます。

複合関数

複合関数は、入力値または戻り値の型が、配列、マップ、構造などの複合データ型である定義済み関数の一種です。複合関数は Spark エンジンで実行されるマッピングで使用できます。

トランスフォーメーション言語には、以下の複合関数が用意されています。

- ARRAY
- CAST
- COLLECT_LIST
- COLLECT_MAP
- CONCAT_ARRAY
- MAP
- MAP_FROM_ARRAYS
- MAP_KEYS
- MAP_VALUES
- PARSE_JSON
- PARSE_XML
- RESPEC
- SIZE
- STRUCT
- STRUCT_AS

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(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、データーエラーをなくすための関数群があります。データー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。

- 入力値のテスト。
- 入力値のデーター型の変換。
- 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、以下のデータークレンジング関数が用意されています。

- GREATEST
- IN
- INSTR
- IS_DATE
- IS_NUMBER
- IS_SPACES
- ISNULL
- LEAST
- LTRIM
- METAPHONE
- REG_EXTRACT
- REG_MATCH
- REG_REPLACE
- REPLACECHR
- REPLACESTR
- RTRIM
- SQL_LIKE
- 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
- IS_DATE
- LAST_DAY
- MAKE_DATE_TIME
- MAX
- MIN
- ROUND (Date)
- SET_DATE_PART
- Systimestamp
- TO_CHAR(Date)
- TIME_RANGE
- TRUNC (Date)

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形式文字列のいずれかを指定します。日付形式文字列は国際化されていません。

Date/Time データ型は、秒までの精度をサポートします。

関連項目：

- [「日付形式文字列」 \(ページ 39\)](#)

エンコード関数

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

- AES_DECRYPT
- AES_ENCRYPT
- COMPRESS[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

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
- MAX
- MIN
- MOD[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、以下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、以下の特殊関数が用意されています。

- ABORT
- DECODE
- ERROR
- IIF
- LOOKUP
- UUID4
- UUID_UNPARSE

一般に、特殊関数は式、フィルタ、および Update Strategy トランスフォーメーションで使用します。特殊関数の中に他の関数をネストすることもできます。また、集計関数の中に特殊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、以下の文字列関数が用意されています。

- CHOOSE
- INDEXOF
- MAX
- MIN
- REVERSE

テスト関数

トランスフォーメーション言語には、以下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

トランスフォーメーション言語には、現在の行に関連する一連の行に対する計算を実行するウィンドウ関数のグループが含まれます。この関数は、入力行ごとに 1 つの戻り値を計算します。ウィンドウ関数は Spark エンジンで実行されるマッピングで使用できます。

トランスフォーメーション言語には、以下のウィンドウ関数が用意されています。

- LAG
- LEAD

ウィンドウ関数は、式トランスフォーメーションで使用できます。式トランスフォーメーションでウィンドウ関数を使用すると、トランスフォーメーションがアクティブになります。

ABORT

マッピングの実行を停止して、ログに指定のエラーメッセージを出力します。Data Integration Service が ABORT 関数に遭遇した場合、その行のデータのトランスフォーメーションを強制終了します。マッピングの実行が強制終了される前に、読み込まれた行を処理します。Data Integration Service は、強制終了された行までのデータをターゲットに書き込んでから、すべての未コミットデータを最後のコミット時点にロールバックします。

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、セッションを強制終了する規則を設定するために使用されます。

入力ポートおよび出力ポートのデフォルト値に、ABORT 関数を使用します。入力ポートについて ABORT 関数を使用することにより、NULL 値をトランスフォーメーションに渡さないようにすることができます。また、ABORT 関数を用いて、式内での ERROR 関数の呼び出しを含め、あらゆる種類のトランスフォーメーションエラーに対処することができます。デフォルト値により、式内の ERROR 関数がオーバーライドされます。エラーの発生時に必ずセッションが停止するようにしたければ、デフォルト値として ABORT を設定します。

未接続ポートに関する式の中で ABORT を使用すると、Data Integration Service は ABORT 関数を実行しません。

構文

ABORT(*string*)

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
文字列	必須	文字列。マッピングの実行の停止時にログに表示させ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

NULL。

ABS

数値の絶対値を返します。

構文

ABS(*numeric_value*)

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
<i>numeric_value</i>	必須	Numeric データ型。絶対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。

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。

注: 戻り値が 15 桁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 38 桁までの 10 進精度を使用可能にできます。

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

PRICE	COST	RETURN VALUE
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。次の式は、SHIP_DATE ポートのすべての日付に 10 年を加えます。

```
ADD_TO_DATE( SHIP_DATE, 'YY', 10 )
```

- **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。次の式は SHIP_DATE ポートのすべての日付から 10 ヶ月を引きます。

```
ADD_TO_DATE( SHIP_DATE, 'MONTH', -10 )
```

- **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。たとえば、式は SHIP_DATE ポートのすべての日付に 10 日を加えます。

```
ADD_TO_DATE( SHIP_DATE, 'DD', 10 )
```

- **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。以下の式は、SHIP_DATE ポートのすべての日付に 14 時間を加えます。

```
ADD_TO_DATE( SHIP_DATE, 'HH', 14 )
```

- **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。以下の式は、SHIP_DATE ポートのすべての日付に 25 分を加えます。

```
ADD_TO_DATE( SHIP_DATE, 'MI', 25 )
```

- **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。以下の式は、SHIP_DATE ポートのすべての日付に 59 秒を加えます。

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

- **ミリ秒。** *amount* 引数に正または負の整数を入力します。秒の設定には MS フォーマット文字列を使用します。以下の式は、DATE_ポートの各日付の年の部分を返します。

```
ADD_TO_DATE( SHIP_DATE, 'MS', 125 )
```

- **マイクロ秒。** *amount* 引数に正または負の整数を入力します。秒の設定には US フォーマット文字列を使用します。以下の式は、DATE_ポートの各日付の年の部分を返します。

```
ADD_TO_DATE( SHIP_DATE, 'US', 2000 )
```

- **ナノ秒。** *amount* 引数に正または負の整数を入力します。秒の設定には NS フォーマット文字列を使用します。以下の式は、DATE_ポートの各日付の年の部分を返します。

```
ADD_TO_DATE( SHIP_DATE, 'NS', 3000000 )
```

構文

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

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
<i>日付</i>	必須	Date/Time データ型。変更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。
<i>形式</i>	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。
<i>amount</i>	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

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 ポートの各日付に 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、Data Integration Service はその月の最後の日を返します。たとえば、Jan 31 1998 に 1 か月を加えると、Data Integration Service は Feb 28 1998 を返します。

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 (<i>Leap Year</i>)
Oct 9 1998 2:30:12PM	Nov 9 1998 2:30:12PM
NULL	NULL

以下の式はすべて、DATE_SHIPPED ポートの各日付から 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

DATE_SHIPPED	RETURN VALUE
Jan 31 1997 6:24:45PM	Jan 21 1997 6:24:45PM
Mar 9 1996 5:32:12AM	Feb 29 1996 5:32:12AM (<i>Leap Year</i>)
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 ポートの各日付から 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 (<i>Leap Year</i>)
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、Data Integration Service はその月の最後の日を返します。たとえば、Jan 31 1998 に 1 か月を加えると、Data Integration Service は Feb 28 1998 を返します。
- 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

復号化されたデータを文字列形式に返します。Data Integration Service は、128 ビットエンコードの AES (Advanced Encryption Standard) アルゴリズムを使用します。AES アルゴリズムは、FIPS (連邦情報処理規格) 認可の暗号アルゴリズムです。

構文

AES_DECRYPT (*value*, *key*)

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
<i>value</i>	必須	バイナリデータ型。解読する値。
キー	必須	文字列データ型。精度 16 文字以下。キーにマッピング変数を使用します。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

復号化したバイナリ値。

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、Data Integration Service は SUBSRT 関数を使用して、社会保険番号の最初の 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
992D6A5D91E7F59D03B940A4B1CBBCBE	832-81-9528

AES_ENCRYPT

暗号化された形式でデータを返します。Data Integration Service は、128 ビットエンコードの AES (Advanced Encryption Standard) アルゴリズムを使用します。AES アルゴリズムは、FIPS (連邦情報処理規格) 認可の暗号アルゴリズムです。

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。たとえば、データウェアハウ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*)

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
<i>value</i>	必須	文字列データ型。暗号化する値。
キー	必須	文字列データ型。精度 16 文字以下。キーにマッピング変数を使用します。

戻り値

暗号化されたバイナリ値。

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、Data Integration Service は SUBSTR 関数を使用して、社会保険番号の最初の 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	992D6A5D91E7F59D03B940A4B1CBBCE
832-81-9528	992D6A5D91E7F59D03B940A4B1CBBCE

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。

ANY

選択したポートの任意の行を返します。オプションとして、データ統合サービスが読み込む行を制限するフィルタを適用できます。ANY の中にネストできる他の集計関数は 1 つだけです。

構文

```
ANY( value [, 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
<i>value</i>	必須	バイナリ以外の任意のデータ型。任意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。
<i>filter_condition</i>	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

ポートの任意の行。毎回別の行を返します。

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。

例

次の式は、ITEM_NAME ポート内で価格が\$10.00 を超える任意の行を返します。

ANY(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:Flashlight

ANY と複合データ型

ANY を使用して、配列型または構造型の複合ポート内の行を返すことができます。

例えば、次のような配列があるとします。

```
emp_phones =  
[205-128-6478, 722-515-2889]  
[107-081-0961, 718-051-8116]  
[344-894-6463, 861-411-8361]  
[107-031-0961, NULL]
```

次の式を使用すると、配列ポート内の任意の行を返すことができます。

ANY(emp_phones)

戻り値: [205-128-6478, 722-515-2889]

ARRAY

指定した引数に基づいて要素を持つ配列を生成します。

構文

ARRAY (array_element1 as any [, array_element2] ...)

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
array_element1	必須	任意のデータ型。配列に追加する要素。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。
array_element2	オプション	array_element1 と同じデータ型。

配列ポートの出力式で ARRAY 関数を使用する場合、関数の引数のデータ型は、配列ポートの型構成で指定された配列要素のデータ型と一致する必要があります。

戻り値

配列。

引数のデータ型によって、配列要素のデータ型が決まります。例えば、文字列引数を渡すと、関数は文字列要素の配列を生成します。

例

次の式は、文字列要素の配列を生成します。

ARRAY (work_phone, home_phone)

work_phone	home_phone	RETURN VALUE
205-128-6478	722-515-2889	[205-128-6478,722-515-2889]
107-081-0961	718-051-8116	[107-081-0961,718-051-8116]
344-894-6463	861-411-8361	[344-894-6463,861-411-8361]
107-031-0961	NULL	[107-031-0961,NULL]

ASCII

Data Integration Service で ASCII モードを使用している場合、ASCII 関数は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。

Data Integration Service で Unicode モードを使用している場合、ASCII 関数は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。Unicode 値の範囲は 0-65,535 です。

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、ASCII または Unicode 値に変換したい特定の文字を取り出します。たとえば、RTRIM などの文字列操作関数を使用できます。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 ASCII または Unicode 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 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*)

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
<i>string</i>	必須	文字列。ASCII 値として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。

戻り値

整数。文字列の最初の文字の ASCII または Unicode 値。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、ITEMS ポートの各値の最初の文字に対応する ASCII または Unicode 値を返します。

ASCII(ITEMS)

ITEMS	RETURN VALUE
Flashlight	70
Compass	67
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さなければなりません。

構文

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

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
<i>numeric_value</i>	必須	数値データ型。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。
<i>filter_condition</i>	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 桁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 38 桁までの 10 進精度を使用可能にできます。

NULL

値が NULL であると、AVG はその行を無視します。ただし、ポート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

AVG は、トランスフォーメーションで定義した Group By ポート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。

Group By ポートがない場合には、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)

CAST

指定した複合データ型定義のデータ型に基づいて、要素の名前を変更し、指定された構造値に対する各要素のデータ型を変更します。

構文

CAST (:Type.type_definition_library.type_definition, struct_value)

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
:Type.type_definition_library.type_definition	必須	構造データのスキーマを表す複合データ型定義。参照修飾子:Type を使用して、複合データ型定義を含む型定義ライブラリを参照します。
struct_value	必須	構造要素のデータ型を変更する構造値。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。

構造値のデータ型と、複合データ型定義のデータ型は互換性がある必要があります。

戻り値

構造。

例

次の式は、複合データ型定義 h1_sales_def のデータ型に基づいて、構造ポート h2_sales の要素のデータ型を変更します。

CAST (:Type.type_definition_library.h1_sales_def, h2_sales)

h1_sales_def	h2_sales	RETURN VALUE
{ q1_total : bigint q2_total : double }	{ q3_total : int q4_total : int }	{ q1_total : bigint q2_total : double }

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*)

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
<i>numeric_value</i>	必須	数値データタイプ。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。

38 を超える宣言された精度を持つ数値を渡した場合は、倍精度浮動小数点数型の値です。

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 関数で次の整数を返す前に、CEIL に渡す値に対して算術演算を実行することができます。例えば、数値に 10 を掛けてからその結果の値より大き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。値と位置が一致する場合、Data Integration Service はその値を返します。

構文

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

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
<i>index</i>	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。
<i>文字列</i>	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

Data Integration Service で ASCII モードを使用している場合、CHR は渡された数値に対応する 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。

Data Integration Service で Unicode モードを使用している場合、CHR は渡された数値に対応する Unicode 文字を返します。Unicode 値の範囲は 0-65,535 です。

構文

CHR(*numeric_value*)

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
<i>numeric_value</i>	必須	数値データ型。ASCII または Unicode 文字として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。

戻り値

ASCII または Unicode 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 ポートの各数値に対応する ASCII または Unicode 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させることができます。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

Data Integration Service で ASCII モードを使用している場合、CHRCODE は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。ASCII 値は 0-255 の範囲になります。

Data Integration Service で Unicode モードを使用している場合、CHRCODE は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。Unicode 値の範囲は 0-65,535 です。

通常は、CHRCODE に文字列値を渡す前に、ASCII または Unicode 値に変換したい特定の文字を取り出します。たとえば、RTRIM などの文字列操作関数を使用できます。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 ASCII または Unicode を返します。

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*)

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
<i>string</i>	必須	文字列。ASCII または Unicode 値として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。

戻り値

整数。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、ITEMS ポートの各値の最初の文字に対応する ASCII または 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

COLLECT_LIST

引数に基づいて要素を持つ配列を返します。引数のデータ型によって、配列のデータ型が決まります。COLLECT_LIST は集計関数です。

構文

COLLECT_LIST(value as any)

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
value	必須	任意のデータ型。配列型の階層データに集計する値。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

配列。

Group By

COLLECT_LIST は、トランスフォーメーションで定義した Group By ポート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。

Group By ポートがない場合には、COLLECT_LIST 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、PRODUCT_NAME 内の要素を持つ配列を返します。

```
COLLECT_LIST(PRODUCT_NAME)
```

PRODUCT_NAME

Flashlight

Compass

Pressure Gauge

Vest

戻り値: [Flashlight,Compass,Pressure Gauge,Vest]

COLLECT_MAP

指定した引数に基づく要素を持つマップを返します。

構文

```
COLLECT_MAP(map_key as ANY, map_value as ANY)
```

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
map_key	必須	任意のプリミティブデータ型。タイプマップの階層データのキーとして集計する要素。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。
map_value	必須	任意のプリミティブまたは複合データ型。タイプマップの階層データの値として集計する要素。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

マップ。

Group By

COLLECT_MAP は、トランスフォーメーションで定義した Group By ポート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。

Group By ポートがない場合には、COLLECT_MAP 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、キーとして PRODUCT_ID に要素があり、値として PRODUCT_NAME に要素があるマップを返します。

```
COLLECT_MAP(PRODUCT_ID, PRODUCT_NAME)
```

PRODUCT_ID	PRODUCT_NAME
34890	Flashlight
12754	Compass
54028	Pressure Gauge
81203	Vest

戻り値:

```
[34890 -> Flashlight, 12754 -> Compass, 54028 -> Pressure Gauge, 81203 -> Vest]
```

COMPRESS

zlib 1.2.1 圧縮アルゴリズムを使用してデータを圧縮します。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 )
```

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
<i>value</i>	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バイナリ値。

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、Data Integration Service がネットワークに書き込むデータの量が縮小されます。そのため、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 ではなく文字列演算子||を使用すると、Data Integration Service のパフォーマンスが向上します。

構文

CONCAT(*first_string*, *second_string*)

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
<i>first_string</i>	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。
<i>second_string</i>	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。

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_NAME ポートと LAST_NAME ポート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 <i>(includes leading blank)</i>
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

CONCAT_ARRAY

指定したセパレータに基づいて配列内の文字列要素を連結し、文字列を返します。

構文

```
CONCAT_ARRAY(' ', array)
```

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
' '	必須	各文字列要素は、指定したセパレータで区切られます。例えば、「,」は値をカンマで区切ります。
array	必須	文字列型の要素を持つ配列。連結する配列。

戻り値

String

NULL

いずれかの文字列要素が NULL の場合、CONCAT_ARRAY 関数はそれを無視してもう一方の文字列を返します。

すべての文字列要素が NULL の場合、CONCAT_ARRAY は空の文字列を返します。

例

次の式は、配列内の文字列要素を連結します。

```
CONCAT_ARRAY( ':', Name )
```

Name	RETURN VALUE
['John','Baer']	'John:Baer'
['Bobbi','Apperley']	'Bobbi:Apperley'
['Jason','']	'Jason:'
['Greg',NULL]	'Greg'
[NULL,NULL]	''

CONVERT_BASE

負以外の数値文字列をある基数値から別の基数値に変換します。

構文

```
CONVERT_BASE( value, source_base, dest_base )
```

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
<i>値</i>	必須	文字列データ型。他のベースに変換する値。最大値は 9,233,372,036,854,775,806 です。
<i>source_base</i>	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。
<i>dest_base</i>	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 )
```

Data Integration Service 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)`

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
<i>numeric_value</i>	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

Double 値。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、Degrees ポート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。たとえば次のように、ポート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)`

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
<i>numeric_value</i>	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

Double 値。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、Angles ポート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。COUNT の中にネストできる他の集計関数は 1 つだけです。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。

構文

`COUNT(value [, filter_condition])`

または

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

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
<i>value</i>	必須	任意のデータ型（Binary を除く）。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。
<i>*</i>	オプション	<i>*</i> を指定すると、トランスフォーメーション内の<>すべての行</>を数えられます。
<i>filter_condition</i>	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 Group By ポート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。Group By ポートがない場合には、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

ITEM_NAME	IN_STOCK
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

COUNT と複合データ型

COUNT を使用して、配列型または構造型の複合ポート内の行数を数えることができます。

例えば、次のような配列があるとして。

```
emp_phones =
[205-128-6478, 722-515-2889]
[107-081-0961, 718-051-8116]
[344-894-6463, 861-411-8361]
[107-031-0961, NULL]
```

次の式を使用すると、配列ポート内の行数を数えることができます。

```
COUNT( emp_phones )
```

戻り値: 4

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。

CRC32 を使用して ASCII モードおよび Unicode モードにおいてデータの冗長性チェックを実行する場合、Data Integration Service は同じ入力値に対して異なる結果を生成することがあります。CRC32 を使用して異なる複数のオペレーティングシステムでデータの冗長性チェックを実行する場合、Data Integration Service は同じ入力値に対して異なる結果を生成することがあります。

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。マッピングでキーを生成する場合、Sequence Generator トランスフォーメーションを使用します。CRC32 を使用してマッピングで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 )
```

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
<i>値</i>	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 checksum を計算し、ファイルと一緒に保存できます。ソースデータ読み込み時に、Data Integration Service は CRC32 を使用して checksum を計算し、保存されている値と比較できます。2 つの値が一致する場合、データは変更されています。

CREATE_TIMESTAMP_TZ

タイムスタンプとタイムゾーンの値から、Timestamp with Time Zone データ型を作成します。

出力ポートは CREATE_TIMESTAMP_TZ 式の timestampWithTZ である必要があります。

構文

CREATE_TIMESTAMP_TZ (timestamp_value, timezone_value)

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
<i>timestamp_value</i>	必須	Date/Time データ型。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。
<i>timezone_value</i>	必須	String データ型で指定する必要があります。文字列はすべて文字からなっていなければなりません。タイムゾーンとして作成する値を渡します。インストール先に存在するタイムゾーンファイルに定義された任意の有効なトランスフォーメーション式を入力できます。

戻り値

timestamp with time zone データ型を返します。

NULL 値を入力した場合は、NULL です。

例

INPUT VALUE	RETURN VALUE
1947-08-05 10:45:00.221111000 AM, 'America/Los_Angeles'	'1947-08-05 10:45:00.221111000 AM America/Los_Angeles'
1947-08-05 10:45:00.221111000 AM, '-08:00'	'1947-08-05 10:45:00.221111000 AM -08:00'

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*])

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
<i>numeric_value</i>	必須	Numeric データ型。現在の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。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。
<i>filter_condition</i>	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 桁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 38 桁までの 10 進精度を使用可能にできます。

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、選択されたポートの値がすべて NULL である場合には、NULL を返します。

例

CUME 関数を使用すると、次のような行のセットが得られます。

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

CA_SALES	OR_SALES	RETURN VALUE
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*)

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
<i>date1</i>	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。
<i>date2</i>	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

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 ポートと DATE_SHIPPED ポートの各日付を比較して、どちらが早い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

DATE_PROMISED	DATE_SHIPPED	RETURN VALUE
Feb 29 1996	Apr 12 1996	-1 (<i>Leap year</i>)
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。Data Integration Service は、1つ目の日付から2つ目の日付を減算して、差を返します。

Data Integration Service は、日数ではなく月数に基づいて DATE_DIFF 関数を計算します。各月の選択された日で、1カ月に満たない月の日付の差を計算します。1カ月に満たない月の日付の差を計算するために、Data Integration Service は月内の日を加算します。次に、その値を選択された月の全日数で除算します。

Data Integration Service は、うるう年と非うるう年では同じ期間で異なる値を算出します。違いが発生するのは、DATE_DIFF 関数に2月が含まれる場合です。DATE_DIFF は、うるう年の2月は29日間で除算し、非うるう年の場合は28日間で除算します。

たとえば、9月13日から2月19日までの月数を計算するとします。うるう年の場合、DATE_DIFF 関数は2月を19/29ヶ月、つまり0.655ヶ月と計算します。非うるう年の場合、DATE_DIFF 関数は2月を19/28ヶ月、つまり0.678ヶ月と計算します。Data Integration Service は、残りの月の日の差を同様に計算し、DATE_DIFF 関数が指定された期間における合計値を返します。

注: 日付の差を計算するのに異なるアルゴリズムを使用するデータベースもあります。

構文

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

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
<i>date1</i>	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。
<i>date2</i>	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。
<i>形式</i>	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、0 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 ポートと DATE_SHIPPED ポート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 ポートと DATE_SHIPPED ポート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 ポートと DATE_SHIPPED ポート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

DATE_PROMISED	DATE_SHIPPED	RETURN VALUE
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 ポートと DATE_SHIPPED ポート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

次の式は、DATE_PROMISED ポートと DATE_SHIPPED ポート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	LEAP YEAR VALUE (in Months)	NON-LEAP YEAR VALUE (in Months)
Sept 13	Feb 19	-5.237931034	-5.260714286
NULL	Feb 19	NULL	N/A
Sept 13	NULL	NULL	N/A

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。

ENC_BASE64 を使用してエンコードしたデータを DEC_BASE64 を使用してデコードする場合、同じデータ移動モードを使用してマッピングを実行する必要があります。それ以外の場合は、デコードしたデータの出力は元のデータと異なる可能性があります。

構文

DEC_BASE64(*value*)

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
<i>値</i>	必須	文字列データ型。デコードするデータ。

戻り値

デコードしたバイナリ値。

NULL 値を入力した場合は、NULL です。

マッピングの戻り値は、Unicode モードで実行した場合と ASCII モードで実行した場合で異なります。

DECODE

ポートから指定した値を検索します。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 を使って文字列ポート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できますし、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*])

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
<i>value</i>	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。
<i>search</i>	必須	<i>value</i> 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。 たとえば、特定のポートで文字列 '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。

引数	必須/ オプション	説明
<i>result</i>	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。
<i>default</i>	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

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。

複数の条件に一致した場合でも、Data Integration Service 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、Data Integration Service のコードページとデータ移動モード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、次のような式があります。

```
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。

高精度モードでマッピングを実行する場合、結果のうちどれか 1 つでも Double であれば、戻り値のデータ型は Double となります。

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。

たとえば、下記の式は無効となります。

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

上記の式を検査すると、下記のエラーメッセージが表示されます。

Function cannot resolve operands of ambiguously mismatching datatypes.

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
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 1.2.1 圧縮アルゴリズムを使用してデータを解凍します。DECOMPRESS 関数は、COMPRESS 関数または zlib 1.2.1 アルゴリズムを使用する圧縮ツールによって圧縮されたデータで使します。データを解凍するマッピングで、データ圧縮のマッピングと異なるデータ移動モードが使用されている場合、解凍されたデータの出力は、元のデータと異なる場合があります。

構文

```
DECOMPRESS( value, precision )
```

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

戻り値

入力値の解凍されたバイナリ値。

NULL 値を入力した場合は、NULL です。

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。バイナリ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、トランスフォーメーションを利用して文字列フォーマットのバイナリ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*)

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
<i>value</i>	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。

ERROR

Data Integration Service に行をスキップさせて、定義したエラーメッセージを表示させます。エラーメッセージはログに表示されます。Data Integration Service は、スキップした行を拒否ファイルに書き込みません。

Expression トランスフォーメーションで 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。

入力ポートおよび出力ポートのデフォルト値に、ERROR 関数を使用します。入力ポートについて ERROR 関数を使用することにより、NULL 値をトランスフォーメーションに渡さないようにすることができます。

出力ポートについては、式内での ERROR 関数の呼び出しなど、ERROR 関数を用いて、あらゆる種類の変換エラーに対処してください。式の中または出力ポートのデフォルト値で ERROR 関数を使用する場合、Data Integration Service は行をスキップし、式およびデフォルト値からのエラーメッセージの両方を記録します。Data Integration Service がエラーとなる行をスキップするよう設定するには、デフォルト値として ERROR を指定します。

ERROR 以外の出力デフォルト値を使用した場合、デフォルト値により式内の ERROR 関数がオーバーライドされます。たとえば、式内で ERROR 関数を使用し、出力ポートにデフォルト値'1234'を割り当てるとします。Data Integration Service は、式の中で ERROR 関数を見つけるたびに、エラーを'1234'の値で上書きし、次のトランスフォーメーションに'1234'を渡します。行はスキップされず、ログにエラーは記録されません。

構文

ERROR(*string*)

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
文字列	必須	文字列値。Integration Service 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。次に示す式では、Data Integration Service が Salary ポートに負の給与を見つけた場合にその行をスキップしてエラーを表示するよう、IIF 式内に ERROR 関数をネストします。

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)。たとえば、EXP(2)は 7.38905609893065 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*)

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
<i>exponent</i>	必須	数値データ型。e の累乗に対する指数の値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

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 ポート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

ポートまたはグループ内で最初に見つかった値を返します。オプションとして、Data Integration Service が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。

構文

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

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
<i>value</i>	必須	任意のデータ型（Binary を除く）。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。
<i>filter_condition</i>	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、ポート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 Group By ポート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。

Group By ポートがない場合には、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 ポート内で価格が\$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 ポート内で価格が\$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*)

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
<i>numeric_value</i>	必須	数値データ型。数値データ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。

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 ポート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

PRICE	RETURN VALUE
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*])

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
<i>rate</i>	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。
<i>terms</i>	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。
<i>payment</i>	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。
<i>present value</i>	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 0 を使用します。
<i>タイプ</i>	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、Data Integration Service はその値を 1 として処理します。

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。したがって、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*)

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
<i>日付</i>	必須	Date/Time データ型。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。
<i>形式</i>	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 HH: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 ポート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 ポート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 ポート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 ポート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

GET_TIMEZONE

timestamp with time zone 列に指定された値からタイムゾーン値を返します。

例:

```
String TimeZone = GET_TIMEZONE (timestampWithTZ);
```

出力ポートは GET_TIMEZONE 式の String データ型である必要があります。

構文

GET_TIMEZONE (timestamp_with_timezone_value)

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
<i>timestamp_with_timezone_value</i>	必須	timestamp with time zone データ型で指定する必要があります。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。

戻り値

タイムゾーン地域名またはタイムゾーンオフセットを含む String データ型を返します。

NULL 値を入力した場合は、NULL です。

例

INPUT VALUE	RETURN VALUE
'1947-08-05 10:45:00.221111111 AM America/Los_Angeles'	'AMERICA/LOS_ANGELES'
'1947-08-05 10:45:00.221111111 AM -08:00'	'-08:00'

GET_TIMESTAMP

timestampWithTZ 入力タイプの日付/時刻値を返します。要求されたタイムゾーン（第 2 引数として指定可能）の Date/Time 値が返されます。第 2 引数にタイムゾーン値を指定しない場合は、入力 timestampWithTZ 値のタイムスタンプ部分を返します。

例:

GET_TIMESTAMP(Timestamp with Time Zone, "+08:30")

第 1 引数の timestamp with time zone のタイムゾーン値は+05:30 です。この関数は、第 2 引数で指定されたタイムゾーン+08:30 のタイムスタンプを返します。

出力ポートは GET_TIMESTAMP 式の Date/Time 値でなければなりません。

構文

GET_TIMESTAMP (timestamp_with_timezone_value, [timezone_value])

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
<i>timestamp_with_timezone_value</i>	必須	timestamp with time zone データ型で指定する必要があります。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。
<i>timezone_value</i>	オプション	String データ型で指定する必要があります。文字列はすべて文字からなっていなければなりません。タイムゾーン（この関数がタイムスタンプを返すときに使用するタイムゾーン）として表示する値を渡します。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。タイムゾーンを指定しない場合は、第 1 引数のタイムスタンプ部分を返します。

戻り値

タイムゾーンオフセットまたは指定された地域でのタイムスタンプ値を返します。

タイムゾーン値を指定しない場合は、第 1 引数のタイムスタンプ部分を返します。

NULL 値を入力した場合は、NULL です。

例

INPUT VALUE	RETURN VALUE
'1996-01-05 10:45:00.221111111 AM America/Los_Angeles', '+05:30'	Returns the timestamp value in time zone offset of '+05:30': '1996-01-06 12:15:00.221111111 AM'
'1996-01-05 10:45:00.221111111 AM America/Los_Angeles', 'GMT'	Returns the timestamp value in the 'GMT' time zone: '1996-01-05 06:45:00.221111111 PM'
'1996-01-05 10:45:00.221111111 AM America/Los_Angeles'	As the time zone value is not passed as the second input parameter, the timestamp is returned: '1996-01-05 10:45:00.221111111 AM'

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*)

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
値	必須	バイナリ以外の任意のデータ型。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、Integer データ型の値と Double データ型の値が混在している場合、Data Integration Service は各値を 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。入力値引数が文字列値の場合に値を指定します。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。 CaseFlag が 0 以外の数値の場合、関数は大文字と小文字を区別します。 CaseFlag が 0 の場合、関数は大文字と小文字を区別しません。 デフォルトでは大文字と小文字が区別されます。

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。

いずれかの引数が NULL である場合は、NULL。

例

次の式は、注文した項目の最大数を返します。

GREATEST(QUANTITY1, QUANTITY2, QUANTITY3)

QUANTITY1	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*])

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
<i>condition</i>	必須	値を求める条件。TRUE または FALSE にな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。
<i>value1</i>	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。有効なトランスフォーメーション式 (別の IIF 式を含む) を必要に応じて入力できます。
<i>value2</i>	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。有効なトランスフォーメーション式 (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。

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

例えば、下記の式には FALSE 条件が含まれず、*value1* は文字列データ型であるため、Data Integration Service 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
50	Pierre Bleu	' ' (<i>empty string</i>)
120	Sally Green	Sally Green
NULL	Greg Jones	' ' (<i>empty string</i>)

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 が含まれるため、Data Integration Service 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

SALES	EMP_NAME	RETURN VALUE
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、Data Integration Service のコードページとデータ移動モード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。

高精度モードでマッピングを実行し、結果のうちどれか 1 つでも Double であれば、戻り値のデータ型は Double となります。

IIF および複合データ型

IIF を使用すると、配列または構造体、または配列または構造体の要素を返すことができます。

例えば、次のような配列があるとします。

```
names = ['John', 'Kevin', 'Laura']
```

次の式を使用して、配列内のいずれかの値を返すことができます。

```
IIF( SIZE(names) > 2, names[2], names[0] )
```

RETURN VALUE: 'Laura'

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 0 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 87)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)

```

また、IIF の代わりに Filter トランスフォーメーションを使用してパフォーマンスを向上できる場合もよくあります。

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 )
```

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
<i>valueToSearch</i>	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。
<i>値</i>	必須	<i>valueToSearch</i> 引数に指定したタイプに応じて、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。トランスフォーメーションのポートも有効な値です。指定できる値の最大数はありません。
<i>CaseFlag</i>	オプション	整数でなければなりません。 <i>valueToSearch</i> 引数が文字列値の場合に値を指定します。 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。 <i>CaseFlag</i> が 0 以外の数値の場合、関数は大文字と小文字を区別します。 <i>CaseFlag</i> が 0 の場合、関数は大文字と小文字を区別しません。 デフォルトでは大文字と小文字が区別されます。

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] )
```

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
<i>valueToSearch</i>	必須	文字列データ型。文字列リストで検索する値。
<i>string</i>	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。CaseFlag 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す。
<i>CaseFlag</i>	オプション	整数でなければなりません。valueToSearch 引数が文字列値の場合に値を指定します。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。CaseFlag が 0 以外の数値の場合、関数は大文字と小文字を区別します。CaseFlag が 0 の場合、関数は大文字と小文字を区別しません。

戻り値

入力値が<string</1>に一致する場合は 1、入力値が<string</2>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 ポートの値が 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、0 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。たとえば、文字列「...THOMAS」を渡すと、関数は Thomas を返します。

構文

INITCAP(*string*)

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
<i>string</i>	必須	任意のデータ型（Binary を除く）。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。データにマルチバイト文字が含まれている場合、戻り値は Data Integration Service のコードページとデータ移動モードに応じて異なります。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、FIRST_NAME ポート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
?!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* [,*comparison_type*]]])

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
<i>文字列</i>	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。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。
<i>search_value</i>	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 0 を返します。 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。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。

引数	必須/ オプション	説明
<i>start</i>	オプション	<p>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。</p> <p>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。</p> <p>開始位置が 0 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。</p>
<i>occurrence</i>	オプション	<p>0 より大きい正の整数。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。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。</p> <p>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。小数を渡すと、Data Integration Service はそれを最も近い整数値に丸めます。負の整数または 0 を渡すと、セッションは失敗します。</p>
<i>comparison_type</i>	オプション	<p>Data Integration Service を Unicode モードで実行するときの、言語またはバイナリのいずれかの文字列の比較型。Data Integration Service を ASCII モードで実行するときは、比較型は常にバイナリです。</p> <p>言語比較では言語固有の照合規則を考慮し、バイナリ比較ではビット単位の比較を行います。たとえば、ドイツ語のシャープ文字である <i>s</i> は、言語比較では文字列 "ss" に一致しますが、バイナリ比較では一致しません。バイナリ比較は、言語比較より高速に実行されます。</p> <p>0 または 1 のいずれかの整数値である必要があります。</p> <ul style="list-style-type: none"> - 0：INSTR は言語文字列比較を行います。 - 1：INSTR はバイナリ文字列比較を行います。 <p>デフォルトは 0 です。</p>

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、0 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

COMPANY	RETURN VALUE
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

COMPANY	RETURN VALUE
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

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 )
```

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
<i>値</i>	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

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) <i>Empty string is not NULL</i>

ISNULL と複合データ型

ISNULL を使用すると、配列または構造体に NULL 値があるかどうかを確認できます。

次の式は、次の複合データ型の NULL 値をチェックします。

Complex Data Type	Input Value	RETURN VALUE
NULL_array = NULL	ISNULL(NULL_array)	1 (TRUE)
NULL_struct = NULL	ISNULL(NULL_struct)	1 (TRUE)
num_array = [1, 2, 3]	ISNULL(num_array)	0 (FALSE)
num_array = [1, NULL, 3]	ISNULL(num_array)	0 (FALSE)
num_struct{ number: int rank: int }	ISNULL(num_struct)	0 (FALSE)

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、セッションで指定された日時形式で日付を表し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形式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 式の出力ポートは、文字列データ型または数値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、Data Integration Service 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*])

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
<i>value</i>	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。有効なトランスフォーマー式を必要に応じて入力できます。
<i>形式</i>	オプション	正しい TO_DATE 形式文字列を入力します。フォーマット文字列は、 <i>string</i> 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、文字列の値がマッピング設定で指定された日付形式になっている必要があります。

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、Data Integration Service は不正確な値を返すか、レコードをスキップする可能性があります。

例

次の式は、INVOICE_DATE ポート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 00:12:15.7008'	1 (TRUE)
'02/31/1998 12:13:55.9204'	0 (FALSE) (<i>February does not have 31 days</i>)
'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) (<i>February does not have 31 days</i>)
'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 ポートの値を検査して、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、Data Integration Service 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、式の出力ポートは 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 式の出力ポートは、文字列データ型または数値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*)

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
<i>value</i>	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

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 ポート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) <i>Incomplete number</i>
' '	0 (False) <i>Consists entirely of blanks</i>
''	0 (False) <i>Empty string</i>

ITEM_PRICE	RETURN VALUE
'+123abc'	0 (False)
' 123'	1 (True) <i>Leading white blanks</i>
'123 '	1 (True) <i>Trailing white blanks</i>
'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。たとえば、次の式は ITEM_PRICE ポートの値を検査し、有効な数字を倍精度の浮動小数点値に変換します。値が有効な数字でない場合、Data Integration Service 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 <i>Consists entirely of blanks</i>
''	0.00 <i>Empty string</i>
'+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 )
```

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
<i>value</i>	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。有効なトランスフォーマー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 ポートを検査して、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) (<i>Empty string does not contain spaces.</i>)

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。たとえば、ターゲットテーブル内の固定長 CHAR(5)列に顧客名を書き込むトランスフォーマーがある場合、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)

LAG

式トランスフォーマーで現在の行の前にある行のオフセット番号である値を返します。この関数は、Hadoop 環境の Spark エンジンでマッピングを実行するときに、現在の行の値を前の行の値と比較するために使用します。

遅延値は一連のデータの現在の行の前に表示されます。

トランスフォーマーで LAG を使用するときは、ウィンドウ化のトランスフォーマーを構成する必要があります。ウィンドウ化プロパティは、データのパーティション化と順序付けの方法を定義します。

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
<i>column_name</i>	必須	関数で操作する対象の列または式。
<i>offset</i>	必須	整数データ型。値の取得元である現在の行の前の行数。
<i>default</i>	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境界の外側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。

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

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)

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外側にあるため、関数はデフォルト値である NULL を返しました。

次の例では、組織がトリップ 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。

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

最初と 4 番目の行の遅延値は指定されたパーティションの外側にあるため、関数はデフォルト値である NULL を 2 つ返しました。

LAST

選択したポートの最後の行を返します。オプションとして、Data Integration Service が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。

構文

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

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
<i>value</i>	必須	任意のデータ型 (Binary を除く)。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。
<i>filter_condition</i>	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

ポートの最後の行。

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 ポート内で価格が\$10.00 を超える最後の行を返します。

LAST(ITEMS_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

ポート内の各日付に対して、月の最後の日の日付を返します。

構文

LAST_DAY(*date*)

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
<i>日付</i>	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。関数に渡した日付値に対する月の最後の日。

選択したポート内の値が 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、ポート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

LAST_DAY は、トランスフォーメーションで定義した Group By ポート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。Group By ポートがない場合には、LAST_DAY 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

次の例は、ORDER_DATE ポート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
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含まれます。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

式トランスフォーメーションで現在の行の後にある行のオフセット番号である値を返します。この関数は、Hadoop 環境の Spark エンジンでマッピングを実行するときに、現在の行の値を次の行の値と比較するために使用します。

先行値は一連のデータの現在の行の後に表示されます。

注: トランスフォーメーションで LEAD を使用するときは、ウィンドウ化のトランスフォーメーションを構成する必要があります。ウィンドウ化プロパティは、データのパーティション化と順序付けの方法を定義します。

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
<i>column_name</i>	必須	関数で操作する対象の列または式。
<i>offset</i>	必須	整数データ型。値の取得元である現在の行の後の行数。
<i>default</i>	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境界の外側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。

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

次の式は、従業員ごとに、次の従業員が採用された日付を返します。

LEAD (HIRE_DATE, 1, NULL)

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

最後の行の先行値はないため、関数はデフォルト値である NULL を返しました。

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

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。

YEAR	QUARTER	SALES_QUOTA	QUOTA_DIFF
2016	1	300	7700
2016	2	7000	0

YEAR	QUARTER	SALES_QUOTA	QUOTA_DIFF
2016	3	8000	0
2017	1	5000	4000
2017	2	400	0
2017	3	9000	0

第 2 四半期と第 3 四半期の行の先行値は指定されたパーティションの外側にあるため、関数は値「0」を返しました。

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*)

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
<i>値</i>	必須	<p>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。</p> <p>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、Integer データ型の値と Double データ型の値が混在している場合、Data Integration Service は各値を Double データ型に変換します。</p>
<i>CaseFlag</i>	オプション	<p>整数でなければなりません。入力値引数が文字列値の場合に値を指定します。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。</p> <p>CaseFlag が 0 以外の数値の場合、関数は大文字と小文字を区別します。</p> <p>CaseFlag が 0 の場合、関数は大文字と小文字を区別しません。</p> <p>デフォルトでは大文字と小文字が区別されます。</p>

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。

いずれかの引数が NULL である場合は、NULL。

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)

QUANTITY1	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*)

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
文字列	必須	文字列データ型。評価したい文字列。有効なトランスフォーメーション式が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

LENGTH のヒント

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。たとえば、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)

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
numeric_value	必須	数値データ型。これは 0 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

Double 値。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、NUMBERS ポート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	Error. (The Integration Service does not write row.)
0	Error. (The Integration Service does not write row.)

注: 負の数値または 0 を渡すと、Data Integration Service はエラーを表示し、行を書き込みません。
numeric_value は、0 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*)

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
<i>base</i>	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トランスフォーメーション式。
<i>exponent</i>	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トランスフォーメーション式。

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 ポート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	Error. (Data Integration Service does not write the row.)
0	5	Error. (Data Integration Service does not write the row.)
10	-2	Error. (Data Integration Service does not write the row.)

基数の値として負の数値、0、または 1 を渡した場合、あるいは指数として負の値を渡した場合、Data Integration Service はエラーを表示し、行を書き込みません。

LOWER

大文字の文字列を小文字に変換します。

構文

LOWER(*string*)

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
文字列	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

小文字の文字列。データにマルチバイト文字が含まれている場合、戻り値は Integration Service のコードページとデータ移動モードに応じて異なります。

選択したポート内の値が 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*])

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
<i>first_string</i>	必須	文字列。変更したい文字列を渡します。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。
<i>length</i>	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。
<i>second_string</i>	オプション	任意の文字列値。> <i>first_string</i> </>値の左側に付加したい文字列。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。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。> <i>second_string</i> </>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
Regulator System	Regulator System
Safety Knife	*.*.*Safety Knife

LTRIM

文字列の先頭から空白または文字を削除します。LTRIM は、Expression または Update Strategy トランスフォーメーション内で 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。

式に *trim_set* パラメータを指定しない場合は、次のようになります。

- Unicode モードでは、LTRIM は 1 バイトの空白と 2 バイトの空白をともに文字列の先頭から削除します。
- ASCII モードでは、LTRIM は 1 バイトの空白だけを削除します。

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*])

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
文字列	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。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。
<i>trim_set</i>	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。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 ポートの文字列から文字 'S' と '.' を削除します。

LTRIM(LAST_NAME, 'S.')

LAST_NAME	RETURN VALUE
Nelson	Nelson
Osborne	Osborne
NULL	NULL

LAST_NAME	RETURN VALUE
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 ポート内の文字列を返して、次の行に移ります。

LTRIM のヒント

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, nanosecond )
```

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
<i>year</i>	必須	数値データ型。4 桁の正の整数この関数に 2 桁の年を渡す場合、Data Integration Service は "00" を年の最初の 2 桁として返します。
<i>月</i>	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）。
<i>day</i>	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）。
<i>hour</i>	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）。

引数	必須/ オプション	説明
<i>minute</i>	オプション	数値データ型。0 から 59 までの正の整数。
<i>second</i>	オプション	数値データ型。0 から 59 までの正の整数。
ナノ秒	オプション	数値データ型。0 から 999,999,999 までの正の整数。

戻り値

MM/DD/YYYY HH24:MI:SS の日付。関数に年、月、または日を渡さない場合、NULL 値を返します。

例

以下の式は、入力ポート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		06/15/2000 15:17:00
2003	1	3		22	45	01/03/2003 00:22:45
04	3	30	12	5	10	03/30/0004 12:05:10
99	12	12	5		16	12/12/0099 05:00:16

MAP

指定したキーと値のペアに基づく要素を持つマップを生成します。

構文

MAP(map_key1 as any, map_value1 as any [, map_key2, map_value2]...)

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
map_key1	必須	任意のプリミティブデータ型。マップデータのキーとして追加する要素。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。
map_value1	必須	任意のプリミティブまたは複合データ型。マップデータのキーの値として追加する要素。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。

マップポートの出力式で MAP 関数を使用する場合、関数の引数のデータ型は、マップポートの型構成で指定されたマップ要素のデータ型と一致する必要があります。map_key は NULL にすることはできません。

戻り値

マップ。

引数のデータ型によって、マップ要素のデータ型が決まります。例えば、キーとして整数の引数、値として構造の引数を渡す場合、関数は整数と構造のキーと値のペアでマップデータを生成します。

例

次の式は、整数と文字列の要素のマップを生成します。

```
MAP(emp_id, emp_name)
```

emp_id	emp_name	RETURN VALUE
45781	'Laura'	[45781 -> 'Lauren']
78345	'Derrick'	[78345 -> 'Derrick']
87289	'Kevin'	[87289 -> 'Kevin']
30912		[30912 -> NULL]

MAP_FROM_ARRAYS

指定したキーと値の配列からマップを生成します。

構文

```
MAP_FROM_ARRAYS(map_keys as ARRAY, map_values as ARRAY)
```

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
map_keys	必須	プリミティブデータ型の要素がある配列型。マップデータのキーとして追加する要素の配列。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。
map_values	必須	任意のデータ型の要素がある配列型。マップデータのキーに値として追加する要素の配列。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。

map_keys 配列と map_values 配列の要素の数は一致する必要があります。マップポートの出力式で MAP 関数を使用する場合、関数の引数のデータ型は、マップポートの型構成で指定されたマップ要素のデータ型と一致する必要があります。

戻り値

マップ。

引数のデータ型によって、マップ要素のデータ型が決まります。例えば、キーとして整数の引数、値として構造の引数を渡す場合、関数は整数と構造のキーと値のペアでマップデータを生成します。

例

次の式は、文字列タイプの配列要素からマップを生成します。

MAP_FROM_ARRAYS(cust_name, cust_phone)

cust_type	cust_name	cust_phone	RETURN VALUE
silver	[Adams, Clark]	[205-128-6478, 722-515-2889]	[Adams -> 205-128-6478, Clark -> 722-515-2889]
gold	[Baker, Davis]	[107-081-0961, 718-051-8116]	[Baker -> 107-081-0961, Davis -> 718-051-8116]
platinum	[Evans, Hills]	[344-894-6463, 861-411-8361]	[Evans -> 344-894-6463, Hills -> 861-411-8361]

MAP_KEYS

指定されたマップのキー要素の配列を返します。

構文

MAP_KEYS(map as MAP)

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
map	必須	Map データ型。キーを取得するキーと値のペア要素でマップします。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。

戻り値

配列。

キーのデータ型によって、配列要素のデータ型が決まります。例えば、キーが文字列タイプを持つ場合、関数は文字列要素で配列データを生成します。

マップキーが NULL の場合は-1 を返します。

マップが NULL の場合は NULL を返します。

例

次の式は、文字列タイプの要素がある配列を生成します。

MAP_KEYS(stock_sellprice)

stock_sellprice	RETURN VALUE
[AAPL -> 150.45, GOOGL -> 1150.96]	[AAPL, GOOGL]
[AMZN -> 1400.54, TSLA -> 339.63]	[AMZN, TSLA]

MAP_VALUES

指定したマップの値要素の配列を返します。

構文

MAP_KEYS(map as MAP)

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
map	必須	Map データ型。値を取得するキーと値のペア要素でマップします。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。

戻り値

配列。

マップの値のデータ型によって、配列要素のデータ型が決まります。例えば、値が整数タイプを持つ場合、関数は整数要素で配列データを生成します。

マップの値が NULL の場合は -1 を返します。

マップが NULL の場合は NULL を返します。

例

次の式は、文字列タイプの要素がある配列を生成します。

MAP_VALUES(stock_sellprice)

stock_sellprice	RETURN VALUE
[AAPL -> 150.45, GOOGL -> 1150.96]	[150.45, 1150.96]
[AMZN -> 1400.54, TSLA -> 339.63]	[1400.54, 339.63]

MAX (Dates)

ポート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 の中にネストできる他の集計関数は 1 つだけです。

MAX を使用して、ポート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。

構文

MAX(date [, filter_condition])

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
<i>日付</i>	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。
<i>filter_condition</i>	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

ポート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

MAX (Numbers)

ポート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 の中にネストできる他の集計関数は 1 つだけです。MAX を使用して、ポート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。

構文

```
MAX( numeric_value [, filter_condition] )
```

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
<i>numeric_value</i>	必須	数値データ型。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。
<i>filter_condition</i>	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、ポート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 Group By ポート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。

Group By ポートがない場合には、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)

ポート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 の中にネストできる他の集計関数は 1 つだけです。

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 を使用して、ポート内またはグループ内における直近の日付もしくは最大の数値を返すこともできます。

構文

MAX(*string* [, *filter_condition*])

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
<i>文字列</i>	必須	文字列データ型。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。
<i>filter_condition</i>	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、ポート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 Group By ポート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。

Group By ポートがない場合には、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
104	Battery (9 volt)

MANUFACTURER_ID	ITEM_NAME
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 checksum 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 ビットの一方方向暗号ハッシュ関数です。入力値のチェックサムが異なる場合、入力値が異なると判断できます。MD5 を使用して、データの完全性を確認します。

構文

MD5(*value*)

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
<i>value</i>	必須	データ型は文字列またはバイナリ。checksum 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。MD5 を使用して、ソースから読み込むデータの行のチェックサム値を生成します。マッピングを実行するときに、以前に生成されたチェックサム値と新しいチェックサム値を比較します。次に、更新されたチェックサム値を持つ行をターゲットに書き込みます。チェックサム値の更新は、データが変更されたことを示していると判断できます。

ヒント

戻り値は、ハッシュキーとして使用できます。

MEDIAN

選択したポート内のすべての値のメジアンを返します。

ポート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。ポート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さなければなりません。

Data Integration Service は、データのすべての行を読み込み、メジアン計算を行います。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。

構文

`MEDIAN(numeric_value [, filter_condition])`

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
<i>numeric_value</i>	必須	数値データ型。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。
<i>filter_condition</i>	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 桁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 38 桁までの 10 進精度を使用可能にできます。

NULL

値が NULL であると、MEDIAN はその行を無視します。ただし、ポート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 Group By ポート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。

Group By ポートがない場合には、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、次の式を指定したポートを使って、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	- 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
C	- S	- I、E、または Y の前	- METAPHONE ('Fence')は FNS を返します。
C	- なし	- S の後または I、E、Y の前	- METAPHONE ('Scene')は SN を返します。
C	- 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。
D	- 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。
G	- 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。
G	- 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。
G	- 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。
H	- 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 - K	- C の後 - 他の場合	- METAPHONE ('Ckim')は KM を返します。 - 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	- 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	- 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。
T	- 0 ¹	- H の前	- METAPHONE ('Thor')は 0R を返します。
T	- なし	- CH の前	- METAPHONE ('Glitch')は KLTx を返します。
T	- 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。
W	- 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。

入力	戻り値	条件	例
Y	- 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。

1. 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*])

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
<i>string</i>	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。 > <i>string</i> </>内の英字以外のものはスキップします。
<i>length</i>	オプション	0 より大きい整数でなければなりません。エンコードしたい<> <i>string</i> </>内の文字の数を指定します。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。 > <i>length</i> </>がゼロ、または<> <i>string</i> </>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* は空です。

例

下記の式は、EMPLOYEE_NAME ポートの最初の 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 ポートの最初の 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
P\$%%oc&&KMNL	PKKM

MIN (Dates)

ポート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 には他の集計関数は 1 つしかネストできません。また、ネストされた日付データ型を返さなければなりません。

MIN を使用して、ポート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。

構文

MIN(*date* [, *filter_condition*])

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
<i>日付</i>	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。
<i>filter_condition</i>	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、ポート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 Group By ポート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。

Group By ポートがない場合には、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)

ポート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 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ことが必要です。

MIN を使用して、ポート内またはグループ内における直近の日付もしくは最低の文字列値を返すこともできます。

構文

```
MIN( numeric_value [, filter_condition] )
```

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
<i>numeric_value</i>	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。
<i>filter_condition</i>	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 桁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 38 桁までの 10 進精度を使用可能にできます。

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、ポート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 Group By ポート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。

Group By ポートがない場合には、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)

ポート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。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。

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、ポート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。

構文

MIN(*string* [, *filter_condition*])

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
文字列	必須	文字列データ型。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。
<i>filter_condition</i>	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、ポート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 Group By ポート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。

Group By ポートがない場合には、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

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: 60.6 cu ft Tank

MOD

除算の余りを返します。たとえば、MOD(8,5)は 3 を返します。

構文

MOD(*numeric_value*, *divisor*)

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
<i>numeric_value</i>	必須	数値データ型。割られる値（被除数）です。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。
<i>divisor</i>	必須	割る値（除数）です。除数に 0 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 ポートの値を QTY ポート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

PRICE	QTY	RETURN VALUE
9.00	2	1
15.00	3	0
NULL	3	NULL
20.00	NULL	NULL
25.00	0	<i>Error. Integration Service does not write row.</i>

最後の行（25、0）は、除数を 0 にしたためにエラーとなりました。0 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 0 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 0 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
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 ポートの 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])`

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
<i>numeric_value</i>	必須	数値データ型。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。
<i>rowset</i>	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。たとえば、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。
<i>filter_condition</i>	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 桁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 38 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 ポートの最初の 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])`

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
<i>numeric_value</i>	必須	数値データ型。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。
<i>rowset</i>	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。たとえば、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、 <code>MOVINGSUM(SALES, 5)</code> のような式を記述できます。
<i>filter_condition</i>	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 桁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 38 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 ポートの最初の 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

ROW_NO	SALES	RETURN VALUE
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*])

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
<i>rate</i>	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。
<i>present value</i>	必須	数値。今後の支払いの合計に相当する一時金の金額。
<i>payment</i>	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。
<i>future value</i>	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 0 を使用します。
<i>タイプ</i>	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、Data Integration Service はその値を 1 として処理します。

戻り値

数値。

例

投資の現在価値は 500 ドルです。1 回の支払額は 2000 ドルで、この投資の将来価値は 20,000 ドルです。この場合、以下の式は支払い期間として 9 を返します。

NPER (0.015, -500, -2000, 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。

PARSE_JSON

文字列データ型の JSON 階層データを解析し、構造を生成します。構造スキーマは、引数で渡す指定された複合データ型定義に基づきます。この関数は、マッピングのミッドストリーム階層データを受け取り、ダウンストリーム処理のためにデータを解析する必要がある場合に適しています。

構文

```
PARSE_JSON(upstream_string, :Type.type_definition_library.type_definition)
```

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
upstream_string	必須	JSON 形式の階層データを含むソース文字列フィールド、マッピングのミッドストリーム。
:Type.type_definition_library.type_definition	必須	構造データのスキーマを表す複合データ型定義。 参照修飾子:Type を使用して、複合データ型定義を含む型定義ライブラリを参照します。

戻り値

構造。

例

次の式は、指定された複合データ型定義 **customer_def** に基づいて構造を生成します。

```
PARSE_JSON(upstream_string, :Type.type_definition_library.customer_def)
```

次の例は、文字列データ型として割り当てられた、アップストリームフィールドの JSON データを示しています。

```
{"customer" : "ABC Co", "contract" : "1010", "city" : "Phoenix", "saleamt" : "150000.00"}  
{"customer" : "Data Inc", "contract" : "1111", "city" : "Portland", "saleamt" : "20000.00"}
```

複合データ型定義 **customer_def** は、次のように型定義ライブラリで定義されます。

```
customer_def{  
  customer : string  
  contract : int  
  city : string  
  saleamt : int  
}
```

次の表に、マッピング式の PARSE_JSON 関数が **customer_def** を使用してアップストリーム文字列を解析し、構造を返す方法を示します。

customer	contract	city	saleamt	RETURN VALUE
ABC Co	1010	Phoenix	150000.00	{ customer:ABC Co contract:1010 city:Phoenix saleamt:150000.00 }
Data Inc	1111	Portland	20000.00	{ customer:Data Inc contract:1111 city:Portland saleamt:20000.00 }

PARSE_XML

文字列データ型の XML 階層データを解析し、構造を生成します。構造スキーマは、引数で渡す指定された複合データ型定義に基づきます。この関数は、マッピングのミッドストリーム階層データを受け取り、ダウンストリーム処理のためにデータを解析する必要がある場合に適しています。

構文

PARSE_XML(upstream_string,:Type.type_definition_library.type_definition)

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
upstream_string	必須	XML 形式の階層データを含むソース文字列フィールド、マッピングのミッドストリーム。
:Type.type_definition_library.type_definition	必須	構造データのスキーマを表す複合データ型定義。 参照修飾子:Type を使用して、複合データ型定義を含む型定義ライブラリを参照します。

戻り値

構造。

例

次の式は、指定された複合データ型定義 **customer_def** に基づいて構造を生成します。

PARSE_XML(upstream_string,:Type.type_definition_library.customer_def)

次の例は、文字列データ型として割り当てられた、アップストリームフィールドの XML データを示しています。

```
<Customers>
```



```

<Customer>
  <CustName>ABC Co</CustName>
  <Contract>1010</Contract>
  <City>Phoenix</City>
  <SaleAmt>150000.00</SaleAmt>
</Customer>
<Customer>
  <CustName>Data Inc</CustName>
  <Contract>1111</Contract>
  <City>Portland</City>
  <SaleAmt>20000.00</SaleAmt>
</Customer>
</Customers>

```

複合データ型定義 **customer_def** は、次のように型定義ライブラリで定義されます。

```

customer_def{
  CustName : string
  Contract : int
  City : string
  SaleAmt : int
}

```

次の表に、マッピング式の PARSE_XML 関数が **customer_def** を使用してアップストリーム文字列を解析し、構造を返す方法を示します。

CustName	Contract	City	SaleAmt	RETURN VALUE
ABC Co	1010	Phoenix	150000.00	{ CustName:ABC Co Contract:1010 City:Phoenix SaleAmt:150000.00 }
Data Inc	1111	Portland	20000.00	{ CustName:Data Inc Contract:1111 City:Portland SaleAmt:20000.00 }

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。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さなければなりません。

Data Integration Service は、データのすべての行を読み込み、パーセンタイル計算を行います。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。

構文

```
PERCENTILE( numeric_value, percentile [, filter_condition ] )
```

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
<i>numeric_value</i>	必須	数値データ型。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。
<i>percentile</i>	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、Data Integration Service はエラーを表示し、行を書き込みません。
<i>filter_condition</i>	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 桁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 38 桁までの 10 進精度を使用可能にできます。

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 Group By ポート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。

Group By ポートがない場合には、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

Data Integration Service 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。

$$i = \frac{(x + 1) \times \text{パーセンタイル}}{100}$$

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。

$$n = \lceil i \rceil \text{th?element} \times (\lceil i \rceil - i) + \lceil i \rceil \text{th?element} \times (i - \lfloor i \rfloor)$$

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*])

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
<i>rate</i>	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。
<i>terms</i>	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。
<i>present value</i>	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 0 を使用します。
タイプ	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、Data Integration Service はその値を 1 として処理します。

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*)

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
<i>base</i>	必須	数値。この引数は累乗される基数値です。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。底の値が負の場合は、指数は整数である必要があります。
<i>exponent</i>	必須	数値。この引数は累乗の指数値です。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。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 ポートの値を指数として、Numbers ポート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。基数が負の場合、指数は整数でなければなりません。そうでない場合、Data Integration Service は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*])

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
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。期間数または支払回数。0 より大きな値を指定する必要があります。
支払	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。最後の支払いの後の現金残高。この値を省略すると、PV は 0 を使用します。
types	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、Data Integration Service はその値を 1 として処理します。

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、確率シナリオに活用できます。

構文

RAND(seed)

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
シード	オプション	数値。Integration Service が乱数を生成するための開始値。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、Data Integration Service は現在のシステム時間を使用して、1971 年 1 月 1 日からの秒数を算出します。この値をシードとして使用します。

戻り値

数値。

同じシードに対し、Data Integration Service 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

証券の期間あたりの利率を返します。

構文

`RATE(terms, payment, present value[, future value, type])`

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
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。
支払	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 0 を使用します。
types	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。期末に支払う場合は 0 を指定します。デフォルトは 0 です。0 または 1 以外の値を入力すると、Data Integration Service はその値を 1 として処理します。

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ができます。

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)`

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
<i>subject</i>	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。
<i>pattern</i>	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。各サブパターンをカッコで囲みます。
<i>subPatternNum</i>	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。 <ul style="list-style-type: none"> - 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、REG_MATCH 関数および REG_REPLACE 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、カッコによって正規表現\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。たとえば、\d{3}(-{d{4}})?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。

構文	説明
* (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	<code>\d</code>	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	<code>\d\d\d\d</code> または <code>\d{4}</code>	1234 や 5936 など、0~9 の任意の 4 桁の数字に一致します。
x	<code>[a-z]</code>	1 つの文字インスタンスに一致します。
9xx9	<code>\d[a-z][a-z]\d</code>	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
%	<code>.*</code>	任意の文字列に一致します。
A%	<code>A.*</code>	「A」で始まる任意の文字列（「Area」など）に一致します。
_	<code>.</code> (ピリオド)	任意の 1 つの文字に一致します。
A_	<code>A.</code>	「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。たとえば、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 )
```

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
<i>subject</i>	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。
<i>pattern</i>	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 159)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
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

文字列内の文字を別の文字パターンで置換します。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*)

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
<i>subject</i>	必須	文字列データ型。検索したい文字列を渡します。
<i>pattern</i>	必須	文字列データ型。変更するフォルダを選択して下さい。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 159)を参照してください。
<i>replace</i>	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。
<i>numReplacements</i>	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

String

例

次の式は、WEBLOG ポートの各行について Web ログデータから二重引用符を削除します。

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)

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
<i>CaseFlag</i>	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。 > <i>CaseFlag</i> が</>0 以外の数値の場合、大文字と小文字を区別します。 > <i>CaseFlag</i> </>が NULL 値または 0 の場合、大文字と小文字を区別しません。
<i>InputString</i>	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。数値を渡すと、関数はそれを文字列に変換します。 > <i>InputString</i> </>が NULL の場合、REPLACECHR は NULL を返します。
<i>OldCharSet</i>	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。'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。 <i>OldCharSet</i> が NULL または空の場合、REPLACECHR は <i>InputString</i> を返します。
<i>NewChar</i>	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。 <i>NewChar</i> が NULL または空の場合、REPLACECHR は、 <i>OldCharSet</i> に指定された文字をすべて <i>InputString</i> から削除します。 > <i>NewChar</i> </>に複数の文字が含まれている場合、REPLACECHR は、最初の文字で、<> <i>OldChar</i> </>Set 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 ポートの各行について 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 ポート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 ポート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 ポート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 ポート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

CUSTOMER_CODE	RETURN VALUE
AAA	<i>[empty string]</i>
aaa	aaa
NULL	NULL

次の式は、INPUT ポート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 ポートの各行について、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*)

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
<i>CaseFlag</i>	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。 <i>CaseFlag</i> が 0 以外の数値の場合、大文字と小文字を区別します。 <i>CaseFlag</i> が NULL 値または 0 の場合、大文字と小文字を区別しません。
<i>InputString</i>	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。数値を渡すと、関数はそれを文字列に変換します。 <i>InputString</i> が NULL の場合、REPLACESTR は NULL を返します。
<i>OldString</i>	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 <i>OldString</i> 引数を入力する必要があります。 <i>OldString</i> 引数ごとに 1 文字以上を入力できます。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。'abc' のようにテキスト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 <i>OldString</i> 引数があり、1 つ以上の <i>OldString</i> 引数が NULL または空の場合、REPLACESTR はその <i>OldString</i> 引数を無視します。すべての <i>OldString</i> 引数が NULL または空の場合、REPLACESTR は <i>InputString</i> を返します。 <i>OldString</i> 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 <i>OldString</i> 引数を入力した場合、最初の <i>OldString</i> 引数は 2 番目の <i>OldString</i> 引数に優先され、2 番目の <i>OldString</i> 引数は 3 番目の <i>OldString</i> 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、 <i>InputString</i> 内の置き換えられた文字の後ろにカーソルを置きます。
<i>NewString</i>	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。 <i>NewString</i> が NULL または空の場合、REPLACESTR は、 <i>InputString</i> 内の <i>OldString</i> 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 ポートの各行について、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 ポートの各行について、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 ポートの各行について、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 ポートの各行について、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 ポートの各行について、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 ポートの各行について、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'

RESPEC

指定した複合データ型定義の要素の名前に基づいて、指定した構造値の各要素の名前を変更します。

構文

RESPEC(:Type.type_definition_library.type_definition, struct_value)

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
:Type.type_definition_library.type_definition	必須	構造データのスキーマを表す複合データ型定義。参照修飾子:Type を使用して、複合データ型定義を含む型定義ライブラリを参照します。
struct_value	必須	要素名を変更する構造値。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。

複合データ型定義の各要素のデータ型は、構造の対応する要素のデータ型と一致する必要があります。

戻り値

構造。

例

次の式は、複合データ型定義 h1_sales_def の名前に基づいて、構造ポート h2_sales の要素の名前を変更します。

```
RESPEC(:Type.type_definition_library.h2_sales_def, h2_sales)
```

h2_sales_def	h2_sales	RETURN VALUE
{ q1_sales : int q2_sales : bigint }	{ q3_total : int q4_total : bigint }	{ q1_sales : int q2_sales : bigint }

REVERSE

入力文字列を逆順にします。

構文

```
REVERSE( string )
```

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
文字列	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

日付の月の部分を、日に基づいて丸めます。

日

日付の日の部分を、時刻に基づいて丸めます。

時間

日付の時の部分を、分に基づいて丸めます。

分

日付の分の部分を、秒に基づいて丸めます。

秒

日付の秒の部分を、ミリ秒に基づいて丸めます。

ミリ秒

日付のミリ秒の部分を、マイクロ秒に基づいて丸めます。

マイクロ秒

日付のマイクロ秒の部分を、ナノ秒に基づいて丸めます。

以下の表に、ROUND 式で使用する条件と戻り値を示します。

条件	式	戻り値
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 00:00:00.000000000 に設定します。	ROUND(TO_DATE('04/16/1998 8:24:19', 'MM/DD/YYYY HH24:MI:SS'), 'YY')	01/01/1998 00:00:00.000000000
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 00:00:00.000000000 に設定します。	ROUND(TO_DATE('07/30/1998 2:30:55', 'MM/DD/YYYY HH24:MI:SS'), 'YY')	01/01/1999 00:00:00.000000000
日が 1 日-15 日の場合、関数は入力した月の最初の日を返し、時刻を 00:00:00.000000000 に設定します。	ROUND(TO_DATE('04/15/1998 8:24:19', 'MM/DD/YYYY HH24:MI:SS'), 'MM')	04/01/1998 00:00:00.000000000
日が 16 日-月の最後の日の場合、関数は次の月の最初の日を返し、時刻を 00:00:00.000000000 に設定します。	ROUND(TO_DATE('05/22/1998 10:15:29', 'MM/DD/YYYY HH24:MI:SS'), 'MM')	06/01/1998 00:00:00.000000000
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 00:00:00.000000000 (AM12 時) に設定します。	ROUND(TO_DATE('06/13/1998 2:30:45', 'MM/DD/YYYY HH24:MI:SS'), 'DD')	06/13/1998 00:00:00.000000000
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 00:00:00.000000000 (AM12 時) に設定します。	ROUND(TO_DATE('06/13/1998 22:30:45', 'MM/DD/YYYY HH24:MI:SS'), 'DD')	06/14/1998 00:00:00.000000000
時刻の分の部分が 0-29 の間の場合、関数は現在の時間を返し、分、秒、ミリ秒、ナノ秒を 0 に設定します。	ROUND(TO_DATE('04/01/1998 11:29:35', 'MM/DD/YYYY HH24:MI:SS'), 'HH')	04/01/1998 11:00:00.000000000
分の部分が 30 以上の場合、関数は次の時間を返し、分、秒、ミリ秒、ナノ秒を 0 に設定します。	ROUND(TO_DATE('04/01/1998 13:39:00', 'MM/DD/YYYY HH24:MI:SS'), 'HH')	04/01/1998 14:00:00.000000000
秒の部分が 0-29 の場合、関数は現在の分を返し、秒、ミリ秒、ナノ秒を 0 に設定します。	ROUND(TO_DATE('05/22/1998 10:15:29', 'MM/DD/YYYY HH24:MI:SS'), 'MI')	05/22/1998 10:15:00.000000000
秒の部分が 30-59 の場合、関数は次の分を返し、秒、ミリ秒、ナノ秒を 0 に設定します。	ROUND(TO_DATE('05/22/1998 10:15:30', 'MM/DD/YYYY HH24:MI:SS'), 'MI')	05/22/1998 10:16:00.000000000
ミリ秒の部分が 0-499 の場合、関数は現在の秒を返し、ミリ秒を 0 に設定します。	ROUND(TO_DATE('05/22/1998 10:15:29.499', 'MM/DD/YYYY HH24:MI:SS.MS'), 'SS')	05/22/1998 10:15:29.000000000
ミリ秒の部分が 500-999 の場合、関数は次の秒を返し、ミリ秒を 0 に設定します。	ROUND(TO_DATE('05/22/1998 10:15:29.500', 'MM/DD/YYYY HH24:MI:SS.MS'), 'SS')	05/22/1998 10:15:30.000000000
マイクロ秒の部分が 0-499 の場合、関数は現在のミリ秒を返し、マイクロ秒を 0 に設定します。	ROUND(TO_DATE('05/22/1998 10:15:29.498125', 'MM/DD/YYYY HH24:MI:SS.US'), 'MS')	05/22/1998 10:15:29.498000000

条件	式	戻り値
マイクロ秒の部分が 500-999 の場合、関数は次のミリ秒を返し、マイクロ秒を 0 に設定します。	ROUND(TO_DATE('05/22/1998 10:15:29.498785', 'MM/DD/YYYY HH24:MI:SS.US'), 'MS')	05/22/1998 10:15:29.499000000
ナノ秒の部分が 0-499 の場合、関数は現在のマイクロ秒を返し、ナノ秒を 0 に設定します。	ROUND(TO_DATE('05/22/1998 10:15:29.498125345', 'MM/DD/YYYY HH24:MI:SS.NS'), 'US')	05/22/1998 10:15:29.498125000
ナノ秒の部分が 500-999 の場合、関数は次のマイクロ秒を返し、ナノ秒を 0 に設定します。	ROUND(TO_DATE('05/22/1998 10:15:29.498125876', 'MM/DD/YYYY HH24:MI:SS.NS'), 'US')	05/22/1998 10:15:29.498126000

構文

ROUND(*date* [, *format*])

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
<i>日付</i>	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。
<i>形式</i>	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。

戻り値

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ポート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 ポート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 12:00:00.000000000AM
Apr 19 1998 1:31:20PM	Jan 1 1998 12:00:00.000000000AM
Dec 20 1998 3:29:55PM	Jan 1 1999 12:00:00.000000000AM
NULL	NULL

以下の式は、DATE_SHIPPED ポート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 12:00:00.000000000AM
Apr 19 1998 1:31:20PM	May 1 1998 12:00:00.000000000AM
Dec 20 1998 3:29:55PM	Jan 1 1999 12:00:00.000000000AM
NULL	NULL

以下の式は、DATE_SHIPPED ポート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 12:00:00.000000000AM
Apr 19 1998 1:31:20PM	Apr 20 1998 12:00:00.000000000AM
Dec 20 1998 3:29:55PM	Dec 21 1998 12:00:00.000000000AM
Dec 31 1998 11:59:59PM	Jan 1 1999 12:00:00.000000000AM
NULL	NULL

以下の式は、DATE_SHIPPED ポート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 2:00:00.000000000AM
Apr 19 1998 1:31:20PM	Apr 19 1998 2:00:00.000000000PM
Dec 20 1998 3:29:55PM	Dec 20 1998 3:00:00.000000000PM
Dec 31 1998 11:59:59PM	Jan 1 1999 12:00:00.000000000AM
NULL	NULL

以下の式は、DATE_SHIPPED ポート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 2:11:00.000000000AM
Apr 19 1998 1:31:20PM	Apr 19 1998 1:31:00.000000000PM
Dec 20 1998 3:29:55PM	Dec 20 1998 3:30:00.000000000PM
Dec 31 1998 11:59:59PM	Jan 1 1999 12:00:00.000000000AM
NULL	NULL

ROUND (数値)

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 を使用して日付を丸めることもできます。

構文

ROUND(*numeric_value* [, *precision*])

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
<i>numeric_value</i>	必須	数値データ型。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。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。
<i>precision</i>	オプション	<p>正または負の整数。正の<code><precision></code>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。</p> <p>負の<code><precision></code>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。</p> <p>小数の<code><precision></code>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。</p> <p><code>>precision</code> 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。</p>

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。

注: 戻り値が 15 桁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 38 桁までの 10 進精度を使用可能にできます。

例

次の式は、Price ポートの値を小数点以下 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</i>>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、Data Integration Service はそれ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

PRICE	RETURN VALUE
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。Data Integration Service が低精度モードで実行されている場合、値の精度が 15 桁を超えると計算結果を切り捨てます。たとえば、低精度モードで以下の式を処理するとします。

$7/3 * 3 = 7$

この場合、Data Integration Service は最初の除算の結果を切り捨てるため、式の左辺から 6.999999999999999 を求めます。Data Integration Service は、式全体を FALSE と評価します。これは期待した結果ではないかもしれません。

期待した結果を得るには、ROUND を使って、式の左辺の切り捨て結果を、期待する結果に丸めます。Data Integration Service は、以下の式を TRUE と評価します。

$\text{ROUND}(7/3 * 3) = 7$

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])`

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
<i>first_string</i>	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。
<i>length</i>	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。
<i>second_string</i>	オプション	任意の文字列値。 <i>first_string</i> 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。 <i>second_string</i> を省略すると、関数は <i>first_string</i> 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*. *

RTRIM

文字列の末尾から空白または文字を削除します。

式に *trim_set* パラメータを指定しない場合は、次のようになります。

- Unicode モードでは、RTRIM は 1 バイトの空白と 2 バイトの空白をともに文字列の末尾から削除します。
- ASCII モードでは、RTRIM は 1 バイトの空白だけを削除します。

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。一致した文字を含まない文字列を返します。

構文

```
RTRIM( string [, trim_set] )
```

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
文字列	必須	任意の文字列値。切り詰めたい値を渡します。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。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。
<i>trim_set</i>	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。テキスト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 ポートの文字列から文字'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。

RTRIM のヒント

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。たとえば、次の式は、SHIP_DATE ポート内の全ての日付における年を 2001 に変更します。

```
SET_DATE_PART( SHIP_DATE, 'YY', 2001 )
```

- **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。たとえば、次の式は SHIP_DATE ポートのすべての日付に対して、月を 10 月に変更します。

```
SET_DATE_PART( SHIP_DATE, 'MONTH', 10 )
```

- **日。** 1 から 31 までの正の整数（日数が 31 日未満の月である 2 月、4 月、6 月、9 月、11 月を除く）を *value* 引数に入力することにより、日を変更します。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。たとえば、次の式は SHIP_DATE ポートのすべての日付に対して、日を 10 日に変更します。

```
SET_DATE_PART( SHIP_DATE, 'DD', 10 )
```

- **時間。** 0 から 24 までの正の整数（0=12AM、12=12PM、24=12AM）を *value* 引数に入力することにより、時を変更します。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。たとえば、次の式は SHIP_DATE ポートのすべての日付に対して、時を 14:00:00（2:00:00PM）に変更します。

```
SET_DATE_PART( SHIP_DATE, 'HH', 14 )
```

- **分。** 0 から 59 までの正の整数を *value* 引数に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。たとえば、次ののは SHIP_DATE ポートのすべての日付に対して、分を 25 分に変更します。

```
SET_DATE_PART( SHIP_DATE, 'MI', 25 )
```

- **秒。** 0 から 59 までの正の整数を *value* 引数に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。たとえば、次の式は、SHIP_DATE ポート内のすべての日付における秒を 59 に変更します。

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

- **ミリ秒。** *value* 引数に 0 から 999 までの正の整数を入力することにより、分を変更します。秒の設定には MS フォーマット文字列を使用します。たとえば、次の式は SHIP_DATE ポートのすべての日付に対して、分を 125 分に変更します。

```
SET_DATE_PART( SHIP_DATE, 'MS', 125 )
```

- **マイクロ秒。** 1000 から 999999 までの正の整数を *value* 引数に入力することにより、マイクロ秒を変更します。秒の設定には US フォーマット文字列を使用します。たとえば、次ののは SHIP_DATE ポートのすべての日付に対して、分を 12555 分に変更します。

```
SET_DATE_PART( SHIP_DATE, 'US', 12555 )
```

- **ナノ秒。** 1000000 から 999999999 までの正の整数を *value* 引数に入力することにより、ナノ秒を変更します。秒の設定には NS フォーマット文字列を使用します。たとえば、式<>は SHIP_DATE ポートのすべての日付に対して、分を 12555555 分に変更します。

```
SET_DATE_PART( SHIP_DATE, 'NS', 12555555 )
```

構文

`SET_DATE_PART(date, format, value)`

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
<i>日付</i>	必須	Date/Time データ型。変更したい日付です。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。
<i>形式</i>	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。
<i>value</i>	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

以下の式は、DATE_PROMISED ポートの各日付に対して、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

以下の式は、DATE_PROMISED ポートの日付に対して、月を 6 月に変更します。Data Integration Service は、存在しない日付を作成しようとした場合（5月31日を6月31日に変更しようとした場合など）、エラーを表示します。

```
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

DATE_PROMISED	RETURN VALUE
Feb 13 1997 2:30:01AM	Jun 13 1997 2:30:01AM
Mar 31 1997 5:10:15PM	<i>Error. Integration Service doesn't write row.</i>
Dec 12 1997 8:07:33AM	Jun 12 1997 8:07:33AM
NULL	NULL

以下の式は、DATE_PROMISED ポートの日付に対して、年を 2000 年に変更します。

```
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。たとえば、DATE_ENTERED ポート内のすべての日付を 1998 年 7 月 1 日に変更したい場合、次のような式を記述できます。

```
SET_DATE_PART( SET_DATE_PART( SET_DATE_PART( DATE_ENTERED, 'YYYY', 1998), MM', 7), 'DD', 1)
```

SIGN

数値が正数、負数、0 のいずれであるかを返します。

構文

SIGN(*numeric_value*)

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
<i>numeric_value</i>	必須	数値。評価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。

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 ポート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*)

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
<i>numeric_value</i>	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 ポート内の値をラジアンに変換してから、各ラジアン値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	0.500000000000003
5	0.0871557427476639
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)`

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
<i>numeric_value</i>	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

Double 値。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、Angles ポート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)

SIZE

指定した配列またはマップのサイズを返します。

構文

SIZE(value)

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
value	必須	配列またはマップのデータ型。サイズを決定する配列またはマップ。 配列を指定する場合、関数は配列の要素の数を返します。 マップを指定する場合、関数はマップのキーと値のペアの数を返します。

戻り値

Int。

配列またはマップが NULL の場合は-1 を返します。

例

次の式は、配列 ITEM_NAME のサイズを返します。

SIZE(ITEM_NAME)

ITEM_NAME	RETURN VALUE
['apples','bananas','oranges']	3
['milk','coffee','tea','chai']	4
['cookie','cake']	2
['croissant',NULL]	2
NULL	-1

次の式は、マップ SHIPMENT_DETAILS のサイズを返します。

SIZE(SHIPMENT_DETAILS)

SHIPMENT_DETAILS	RETURN VALUE
[CA -> CNTR345, TX -> T234]	2
[AZ -> CNTR123, AL -> CNTR730, CA -> CNTR345]	3
[FL -> CNTR208, NULL]	2
NULL	-1

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。例えば、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。例えば、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。

表 2. SOUNDEX 子音用エンコードガイドライン

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。例えば、SOUNDEX('A123')は'A000'を返し、SOUNDEX('MAeiouhWC')は'M200'を返します。
- *string* が 4 文字未満の文字数で文字を生成した場合、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。例えば、SOUNDEX('J')は'J000'を返します。
- *string* で「[SOUNDEX](#)」(ページ 187)にリストされた同じコードを使用する子音が続けて登場する場合は、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。例えば、SOUNDEX('AbbpdMN')は'A135'を返します。
- *string* 内の数字をスキップします。例えば、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*)

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
<i>string</i>	必須	文字列。エンコードしたい文字列値を渡します。有効な変換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

下記の式は EMPLOYEE_NAME ポート内の値をエンコードします。

SOUNDEX(EMPLOYEE_NAME)

EMPLOYEE_NAME	RETURN VALUE
John	J500
William	W450
jane	J500
joh12n	J500
1abc	A120
NULL	NULL

SQL_LIKE

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。これにより、ID、電話番号、郵便番号、州の名前などのデータパターンを確認できます。

構文

SQL_LIKE(subject, pattern, escape character)

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
subject	必須	String データ型。正規表現と照合する値を渡します。この値は一重引用符で囲みます。
pattern	必須	String データ型。照合に使用する正規表現。パターンは一重引用符で囲みます。
エスケープ文字	オプション	String データ型。SQL_LIKE 関数では、パーセント記号 (%) とアンダースコア (_) をエスケープ文字として使用できます。エスケープ文字は一重引用符で囲みます。

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 SQL_LIKE を使用すると、パターンに一致する名前を検索できます。例えば、次の式は、エスケープ文字 '#' を使用して、名前をパターン "A_#%" と照合しています。

```
SQL_LIKE(ENAME, 'A_#%', '#')
```

ENAME	値
SMITH	FALSE
AX%	TRUE
MILLER	FALSE
A%	FALSE
JONES	FALSE
BLAKE	FALSE
A%l	FALSE

SQRT

負以外の数値の平方根を返します。

構文

```
SQRT( numeric_value )
```

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
<i>numeric_value</i>	必須	正の数値。平方根を計算したい値を渡します。有効な変換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 ポートの値に対して平方根を返します。

```
SQRT( NUMBERS )
```

NUMBERS	RETURN VALUE
100	10

NUMBERS	RETURN VALUE
-100	<i>Error. Data Integration Service does not write row.</i>
NULL	NULL
60.54	7.78074546557076

値-100 はエラーとなります。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を渡すと、Data Integration Service はトランスフォーメーション評価エラーを表示し、行を書き込みません。

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さなければなりません。

構文

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

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
<i>numeric_value</i>	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。演算子を使って、複数のポートの値の平均を取ることもできます。
<i>filter_condition</i>	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 桁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 38 桁までの 10 進精度を使用可能にできます。

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 Group By ポート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。

Group By ポートがない場合には、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 ポート内で 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 ポート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。

STRUCT

指定した引数に基づいて、要素名とデータ型を持つ構造を生成します。

構文

STRUCT(element_name1, value1 as any [, element_name2, value2 as any] ...)

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
element_name1	必須	構造要素の名前。
value1	必須	任意のデータ型。構造要素の値。

構造ポートの出力式で STRUCT 関数を使用する場合、関数の引数は、複合データ型定義の要素のデータ型と一致する必要があります。

戻り値

構造。

例

次の式は、構造を生成します。

```
STRUCT(city , 'New York', state, 'NY')
```

RETURN VALUE

```
{
  city:New York
  state:NY
}
```

次の式は、複合データ型定義 **adrs_typedef** を持つ構造出力ポートの構造を生成します。

```
STRUCT(city, cust_city, state, cust_state)
```

複合データ型定義 **adrs_typedef** は、次のように型定義ライブラリで定義されます。

```
adrs_typedef{
  city : string
  state : string
}
```

cust_city	cust_state	RETURN VALUE
NEWYORK	NY	{ city:NEWYORK state:NY }
REDWOOD CITY	CA	{ city:REDWOOD CITY state:CA }

STRUCT_AS

指定した複合データ型定義と、引数として渡す値に基づいて、スキーマを持つ構造を生成します。

構文

```
STRUCT_AS (:Type.type_definition_library.type_definition, struct_value)
```

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
:Type.type_definition_library.type_definition	必須	構造データのスキーマを表す複合データ型定義。 参照修飾子:Type を使用して、複合データ型定義を含む型定義ライブラリを参照します。
struct_value	必須	カンマで区切られた複合データ型定義の各要素の値。

戻り値

構造。

例

次の式は、構造要素の引数として渡す値を使用して、指定した複合データ型定義 h1_address_def に基づく構造を生成します。

STRUCT_AS (:Type.type_definition_library.h1_address_def, City, State, ZIP)

複合データ型定義 **h1_address_def** は、次のように型定義ライブラリで定義されます。

```
h1_address_def{
  city : string
  state : string
  zip : int
}
```

city	state	zip	RETURN VALUE
NEWYORK	NY	12345	{ city:NEWYORK state:NY zip:12345 }
REDWOOD CITY	CA	23452	{ city:REDWOOD CITY state:CA zip:23452 }

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*])

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
文字列	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。数値を渡すと、関数はそれを文字列に変換します。
<i>start</i>	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。開始位置が正の数である場合、SUBSTR は文字列の先頭から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 0 の場合、SUBSTR は文字列の最初の文字から検索を開始します。
<i>length</i>	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。 <i>length</i> 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 0 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 0 の長さ値を渡した場合は、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 ポート内の各行に対して、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
357-687-6708	357
NULL	NULL

以下の式は、Phone ポート内の各行に対して、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 ポート内の各行に対して電話番号を返こともできます。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

選択したポート内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さなければなりません。

構文

`SUM(numeric_value [, filter_condition])`

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
<i>numeric_value</i>	必須	数値データ型。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。演算子を使って、複数のポートの値を加算することもできます。
<i>filter_condition</i>	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 桁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 38 桁までの 10 進精度を使用可能にできます。

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、ポート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

SUM は、トランスフォーメーションで定義した Group By ポート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。

Group By ポートがない場合には、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 ポート内で 2000 より大きいすべての値の合計を返します。

`SUM(SALES, SALES > 2000)`

SALES

2500.0

1900.0

1200.0

SALES

NULL

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

Data Integration Service のホストノードの現在の日付と時刻をナノ秒の精度で返します。日時を表示する制度はプラットフォームにより異なります。

関数の戻り値は引数の設定方法に応じて異なります。

- SYSTIMESTAMP の引数を変数として設定する場合、Data Integration Service で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 の引数を定数として設定する場合、Data Integration Service で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]*)

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
形式	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。

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。

以下のポートと値で Expression トランスフォーメーションを作成します。

Port Name	Port Type	Expression
Customer_Name	Input	n/a
Order_Qty	Input	n/a
Time_Counter	Variable	'US'
Transaction_Id	Output	SYSTIMESTAMP (Time_Counter)

実行時、Data Integration Service 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*)

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
<i>numeric_value</i>	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

Double 値。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、Degrees ポートのすべての値に対して正接を返します。

TAN(DEGREES * 3.14159 / 180)

DEGREES	RETURN VALUE
70	2.74747741945531
50	1.19175359259435
30	0.577350269189672
5	0.0874886635259298
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*)

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
<i>numeric_value</i>	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

Double 値。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、Angles ポートの値に対して双曲正接を返します。

TANH(ANGLES)

ANGLES	RETURN VALUE
1.0	0.761594155955765

ANGLES	RETURN VALUE
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)

TIME_RANGE

結合するストリーミングイベントの時間範囲を決定します。

TIME_RANGE 関数は、ストリーミングマッピングのジョイナトランスフォーメーションに対してのみ適用できます。

構文

TIME_RANGE(EventTime1,EventTime2,Format,Interval)

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
EventTime1	必須	Date データ型。ジョイナトランスフォーメーションのマスタポートでストリーミングイベントが生成された時刻。
EventTime2	必須	Date データ型。ジョイナトランスフォーメーションの詳細ポートでストリーミングイベントが生成された時刻。

引数	必須/オプション	説明
形式	必須	<p>イベント時刻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一重引用符で囲んでください。例えば、'Seconds'のようにします。フォーマット文字列では大文字と小文字は区別されません。</p> <p>フォーマット引数は次の値を受け付けます。</p> <ul style="list-style-type: none"> - 年 - 月 - Weeks - 日 - 時間 - 分 - 秒 - Milliseconds - Microseconds
間隔	必須	フォーマットに基づいて変更したいイベント時刻値の整数値。

戻り値

関数に NULL 値を渡した場合は NULL です。

例

次の例は、ジョイナトランスフォーメーションの時間範囲式を返します。

```
TIME_RANGE(EventTime1,EventTime2,'Second',4)
```

戻り値:

```
(EventTime1.<=(EventTime2).&&(EventTime2.<=(EventTime1.(expr("INTERVAL 4 SECONDS")))))
```

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] )
```

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
<i>value</i>	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。
<i>flag</i>	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 0 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 0 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。

関数に渡された値が bigint の値として無効なデータを含んでいる場合、データ統合サービスはエラー行である
ことを示すマークをその行に付けるか、あるいはマッピングを中止します。

例

次の式は、ポート IN_TAX からの値を使用します。

TO_BIGINT(IN_TAX, TRUE)

IN_TAX	RETURN VALUE
'7245176201123435.6789'	7245176201123435
'7245176201123435.2'	7245176201123435
'7245176201123435.2.48'	7245176201123435
NULL	NULL
'A12.3Grove'	<i>Error. Integration Service skips this row.</i>
'176201123435.87'	176201123435
'-7245176201123435.2'	-7245176201123435
'-7245176201123435.23'	-7245176201123435

IN_TAX	RETURN VALUE
-9223372036854775806.9	-9223372036854775806
9223372036854775806.9	9223372036854775806
TO_BIGINT(IN_TAX)	
IN_TAX	RETURN VALUE
'7245176201123435.6789'	7245176201123436
'7245176201123435.2'	7245176201123435
'7245176201123435.348'	7245176201123435
NULL	NULL
'A12.3Grove'	<i>Error. Integration Service skips this row.</i>
' 176201123435.87'	176201123436
'-7245176201123435.6789'	-7245176201123436
'-7245176201123435.23'	-7245176201123435
-9223372036854775806.9	-9223372036854775807
9223372036854775806.9	9223372036854775807

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。

TO_CHAR (date [,format]) は、date、Timestamp、Timestamp with Time Zone、Timestamp with Local Time Zone の各データ型または内部値を、フォーマット文字列で指定された String データ型の値に変換します。

構文

TO_CHAR(date [, format])

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
日付	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。
形式	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、マッピング設定に指定されている日付形式で文字列が返されます。

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 ポート内の日付を MON DD YYYY:形式の文字列に変換します。

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 引数を省略すると、TO_CHAR はデフォルトではマッピング設定に指定されている日付形式である MM/DD/YYYY HH24:MI:SS.US で文字列を返します。

TO_CHAR(DATE_PROMISED)

DATE_PROMISED	RETURN VALUE
Apr 1 1998 12:00:10AM	'04/01/1998 00:00:10.000000'
Feb 22 1998 01:31:10PM	'02/22/1998 13:31:10.000000'
Oct 24 1998 02:12:30PM	'10/24/1998 14:12:30.000000'
NULL	NULL

次の式は、ポート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'
NULL	NULL

次の式は、ポート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、ポート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、ポート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。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 01:02:03AM	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、ポート内の各日付に対して、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、ポート内の各日付に対して、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、次のように倍精度浮動小数点数をテキスト文字列に変換します。

- 16 桁までの倍精度浮動小数点数値を文字列に変換し、15 桁までの精度を提供します。15 桁を超える数値を渡すと、TO_CHAR は数値を 16 桁に基づいて丸め、数値の文字列表現を科学的表記で返します。例えば、1234567890123456 倍精度浮動小数点数値は、'1.23456789012346e+015'文字列値に変換されます。
- (-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。TO_CHAR は、この範囲以外の数値を科学的表記で返します。例えば、10842764968208837340 倍精度浮動小数点数値は、'1.08427649682088e+019'文字列値に変換されます。

TO_CHAR は、次のように 10 進数値をテキスト文字列に変換します。

- 高精度モードの場合、TO_CHAR は、38 桁までの 10 進数値を文字列に変換します。38 桁を超える 10 進数値を渡すと、TO_CHAR は 38 桁を超える数値の科学的表記を返します。
- 低精度モードの場合、TO_CHAR は、10 進数値を倍精度浮動小数点数値として扱います。

構文

TO_CHAR(*numeric_value*)

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
<i>numeric_value</i>	必須	数値データタイプ。文字列に変換したい数値です。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。

戻り値

文字列。

関数に NULL 値を渡した場合は NULL です。

倍精度浮動小数点数の変換例

次の式は、SALES ポート内の倍精度浮動小数点数値を文字列に変換します。

TO_CHAR(SALES)

SALES	RETURN VALUE
1010.99	'1010.99'
-15.62567	'-15.62567'
10842764968208837340	'1.08427649682088e+019' (rounded based on the 16th digit and returns the value in scientific notation)
236789034569723	'236789034569723'
0	'0'

SALES	RETURN VALUE
33.15	'33.15'
NULL	NULL

10 進数値の変換例

次の式は、SALES ポート内の 10 進数値を、高精度モードで文字列に変換します。

TO_CHAR(SALES)

SALES	RETURN VALUE
2378964536789761	'2378964536789761'
1234567890123456789012345679	'1234567890123456789012345679'
1.234578945469649345876123456	'1.234578945469649345876123456'
0.99999999999999999999999999999999	'0.99999999999999999999999999999999'
12345678901234567890123456799	'12345678901234567890123456799'
23456788992233456678458934567123465239	'23456788992233456678458934567123465239'
423456789012345678901234567991234567899 (38 より大きい)	'4.23456789012346e+038'

TO_DATE

Date/Time データ型に文字列を変換します。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 式の出力ポートは 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*])

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
文字列	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。
形式	オプション	正しい TO_DATE 形式文字列を入力します。フォーマット文字列は、 <i>string</i> 引数の各部分と一致しなければなりません。たとえば、文字列 'Mar 15 1998 12:43:10AM' を渡す場合は、フォーマット文字列 'MON DD YYYY HH12:MI:SSAM' を使用します。フォーマット文字列を省略する場合は、文字列の値がセッションで指定された日付形式になっている必要があります。

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 00:00:00.000000000 になります。この関数の結果は、Date/Time 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。ターゲット列の精度がナノ秒未満である場合、Data Integration Service はターゲットに datetime 値を書き込むときにターゲット列の精度に合わせて datetime 値を切り詰めます。

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、Data Integration Service は不正確な値を返すか、レコードをスキップする可能性があります。

例

次の式は、DATE_PROMISED ポートの文字列に対して日付値を返します。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 00:00:00.000000000 になります。20 世紀にマッピングを実行した場合、年の最初の 2 桁は 19 になります。この例では、Data Integration Service が実行されているノードでの現在の年は 1998 です。ターゲット列の日付形式が MON DD YY HH24:MI SS であるため、Data Integration Service はターゲットに書き込むときに datetime 値を秒単位に切り詰めます。

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 ポートの文字列に対して日付および時刻値を返します。時刻値を含まない文字列を渡すと、Data Integration Service はエラーを返します。20 世紀にマッピングを実行した場合、年の最初の 2 桁は 19 になります。Data Integration Service が実行されているノードでの現在の年は、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'	<i>Error. Integration Service skips this row.</i>
NULL	NULL

以下の式は、SHIP_DATE_MJD_STRING ポートの文字列を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 00:00:00.000000000
'2415021'	Jan 1 1900 00:00:00.000000000

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 00.000000000:00:00 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 00:00:00.000000000
'08/17/05'	08/17/2005 00:00:00.000000000

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 00:00:00.000000000
'08/17/05'	08/17/1905 00:00:00.000000000

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 00:00:00.000000000
'08/17/5'	08/17/1995 00:00:00.000000000

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/YYYY')

DATE_STR	RETURN VALUE
'04/01/998'	04/01/1998 00:00:00.000000000
'08/17/995'	08/17/1995 00:00:00.000000000

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')

関連項目：

- [「日付形式文字列のルールとガイドライン」 \(ページ 45\)](#)

TO_DECIMAL

文字列または数値を 10 進値に変換します。TO_DECIMAL は、先頭の空白を無視します。

構文

TO_DECIMAL(*value* [, *scale*])

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
<i>値</i>	必須	文字列データ型または数値データ型であること。10 進数値に変換する値を渡します。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。
<i>スケール</i>	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。

戻り値

0 から 28 までの精度と位取りを持つ 10 進値。

関数に NULL 値を渡した場合は NULL です。

関数に渡された値が 10 進数値として無効なデータを含んでいる場合、データ統合サービスはエラー行であることを示すマークをその行に付けます。

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。

例

次の式は、ポート IN_TAX からの値を使用します。IN_TAX は、精度 44 桁の String データ型です。戻り値は、精度 28 桁、スケール 3 桁の Decimal データ型です。

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'	<i>Error. Integration Service skips this row.</i>

IN_TAX	RETURN VALUE
'711A1'	Error. Integration Service skips this row.
'1234567890.123'	1234567890.123
'123456789012345678901234567890.123'	Error. Integration Service skips this row.
'1234567890123456789012345678901234567890.123'	Error. Integration Service skips this row.

TO_DECIMAL38

文字列または数値を 10 進値に変換します。TO_DECIMAL38 は、先頭の空白を無視します。

構文

TO_DECIMAL38(*value* [, *scale*])

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
値	必須	String または Numeric データ型で指定する必要があります。10 進数値に変換する値を渡します。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。
スケール	オプション	0 から 38 までの整数リテラルで指定する必要があります。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。

戻り値

0 から 38 桁までの精度とスケールを持つ 10 進値。

関数に NULL 値を渡した場合は NULL です。

関数に渡された値が 10 進数値として無効なデータを含んでいる場合、データ統合サービスはエラー行であることを示すマークをその行に付けます。例えば、TO_DECIMAL38("123456789012345678901234567890.12")を渡すと、データ統合サービスはこの行を拒否します。

注: 戻り値が 15 桁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 38 桁までの 10 進精度を使用可能にできます。

例

次の式は、ポート IN_TAX からの値を使用します。IN_TAX は、精度 44 桁の String データ型です。戻り値は、精度 38 桁、スケール 3 桁の Decimal データ型です。

TO_DECIMAL38(IN_TAX, 3)

IN_TAX	RETURN VALUE
'15.6789'	15.679
'60.2'	60.200
'118.348'	118.348
NULL	NULL
'A12.3Grove'	Error. Integration Service skips this row.
'1234567890.123'	1234567890.123
'123456789012345678901234567890.123'	123456789012345678901234567890.123
'1234567890123456789012345678901234567890.123'	Error. Integration Service skips this row.
'711A1'	Error. Integration Service skips this row.

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、先頭の空白を無視します。

構文

TO_FLOAT(*value*)

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
<i>value</i>	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

Double 値。

関数に NULL 値を渡した場合は NULL です。

関数に渡された値が float の値として無効なデータを含んでいる場合、データ統合サービスはエラー行であることを示すマークをその行に付けるか、あるいはマッピングがを中止します。

例

次の式は、ポート 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'	Error. Integration Service skips this row.

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、先頭の空白を無視します。

構文

TO_INTEGER(*value* [, *flag*])

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
<i>value</i>	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。
<i>flag</i>	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 0 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 0 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。

関数に渡された値が integer の値として無効なデータを含んでいる場合、データ統合サービスはエラー行であることを示すマークをその行に付けるか、あるいはマッピングを中止します。

例

次の式は、ポート IN_TAX からの値を使用します。変換によって数値のオーバーフローが発生すると、Data Integration Service はエラーを表示します。

TO_INTEGER(IN_TAX, TRUE)

IN_TAX	RETURN VALUE
'15.6789'	15
'60.2'	60
'118.348'	118
'5,000,000,000'	Error. Integration Service skips this row.
NULL	NULL
'A12.3Grove'	Error. Integration Service skips this row.
' 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
'5,000,000,000'	Error. Integration Service skips this row.
NULL	NULL
'A12.3Grove'	Error. Integration Service skips this row.
' 123.87'	124
'-15.6789'	-16
'-15.23'	-15

TO_TIMESTAMP_TZ

文字列を Timestamp with Time Zone 値に変換します。関数は、Timestamp with Time Zone データ型を返します。元の文字列のフォーマットを指定するには、TO_TIMESTAMP_TZ フォーマット文字列を使用します。

構文

TO_TIMESTAMP_TZ (*String* , [*format*])

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
文字列	必須	String データ型で指定する必要があります。Timestamp with Time Zone 値に変換する値を渡します。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。 文字列はすべて文字からなっていなければなりません。
形式	オプション	有効な TO_TIMESTAMP_TZ 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 1997 12:43:10AM ASIA/CALCUTTA'の文字列を渡す場合、フォーマット文字列'MON DD YYYY HH12:MI:SSAM TZR'を使用する必要があります。フォーマット文字列を指定しない場合は、[実行設定] ダイアログで指定したデフォルトの日時フォーマットが使用されます。

戻り値

timestamp with time zone データ型を返します。

NULL 値を入力した場合は、NULL です。

関数に渡された値が timestamp with time zone の値として無効なデータを含んでいる場合、データ統合サービスはエラー行であることを示すマークをその行に付けるか、あるいはマッピングを中止します。

例

INPUT VALUE	RETURN VALUE
'1947-08-05 10:45:00.221111000 AM America/Los_Angeles', 'YYYY-MM-DD HH:MI:SS.NS AM TZR'	次のデータについて、timestamp with time zone データ型を返します。 '1947-08-05 10:45:00.221111000 AM AMERICA/LOS_ANGELES'
'1947-08-05 10:45:00.221111000 AM America/Los_Angeles', 'YYYY-MM-DD HH:MI:SS.NS AM'	タイムゾーン地域フォーマットでタイムゾーン地域を指定しない場合でも、timestamp with time zone データ型を返します。 '1947-08-05 10:45:00.221111000 AM AMERICA/LOS_ANGELES'

INPUT VALUE	RETURN VALUE
'1947-08-05 10:45:00.221111000 AM America/Los_Angeles'	タイムゾーンフォーマットでタイムスタンプを指定しない場合でも、timestamp with time zone データ型を返します。 '1947-08-05 10:45:00.221111000 AM AMERICA/LOS_ANGELES'
'1947-08-05 10:45:00.221111000 AM America/Los_Angeles', 'MM-DD-YYYY HH:MI:SS.NS AM'	関数レベルでフォーマットを指定しない場合は、[実行設定] ダイアログのデフォルトの日付/時刻フォーマットが使用されます。 デフォルトの日付/時刻フォーマット: 'YYYY-MM-DD HH:MI:SS.NS AM TZR' timestamp with time zone データが指定されたフォーマットと一致しない場合は、次のエラーが表示されます。 Process row failed for function [TO_TIMESTAMP_TZ]: Failed to convert the string to timestamp with time zone value. Verify that the specified date format string is valid. Verify that the timestamp with time zone string used in the first argument is compatible with the specified date format.

TRUNC (Dates)

日付を特定の年、月、日、時、分、秒、ミリ秒、またはマイクロ秒に切り詰めます。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 00:00:00.000000000 に設定します。たとえば、次の式では 1/1/1997 00:00:00.000000000 を返します。
 TRUNC(12/1/1997 3:10:15, 'YY')
- 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 00:00:00.000000000 に設定します。たとえば、次の式では 4/1/1997 00:00:00.000000000 を返します。
 TRUNC(4/15/1997 12:15:00, 'MM')
- 日。** 日付の日の部分を切り詰めると、関数は日付を返し、時刻を 00:00:00.000000000 に設定します。たとえば、次の式では 6/13/1997 00:00:00.000000000 を返します。
 TRUNC(6/13/1997 2:30:45, 'DD')
- 時間。** 日付の時の部分を切り詰めると、関数は日付の分、秒、およびサブ秒を 0 に設定して返します。たとえば、次の式では 4/1/1997 11:00:00.000000000 を返します。
 TRUNC(4/1/1997 11:29:35, 'HH')
- 分。** 日付の分の部分を切り詰めると、関数は日付の秒およびサブ秒を 0 に設定して返します。たとえば、次の式では 5/22/1997 10:15:00.000000000 を返します。
 TRUNC(5/22/1997 10:15:29, 'MI')
- 秒。** 日付の秒の部分を切り詰めると、関数は日付のミリ秒を 0 に設定して返します。たとえば、次の式では 5/22/1997 10:15:29.000000000 を返します。
 TRUNC(5/22/1997 10:15:29.135, 'SS')
- ミリ秒。** 日付のミリ秒の部分を切り詰めると、関数は日付のマイクロ秒を 0 に設定して返します。たとえば、次の式では 5/22/1997 10:15:30.135000000 を返します。
 TRUNC(5/22/1997 10:15:30.135235, 'MS')
- マイクロ秒。** 日付のマイクロ秒の部分を切り詰めると、関数は日付のナノ秒を 0 に設定して返します。たとえば、次の式では 5/22/1997 10:15:30.135235000 を返します。
 TRUNC(5/22/1997 10:15:29.135235478, 'US')

構文

TRUNC(*date* [, *format*])

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
<i>日付</i>	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。
<i>形式</i>	オプション	正しい形式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、00.000000000:00:00 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 ポート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 12:00:00.000000000
Apr 19 1998 1:31:20PM	Jan 1 1998 12:00:00.000000000
Jun 20 1998 3:50:04AM	Jan 1 1998 12:00:00.000000000
Dec 20 1998 3:29:55PM	Jan 1 1998 12:00:00.000000000
NULL	NULL

以下の式は、DATE_SHIPPED ポート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 12:00:00.000000000AM
Apr 19 1998 1:31:20PM	Apr 1 1998 12:00:00.000000000AM
Jun 20 1998 3:50:04AM	Jun 1 1998 12:00:00.000000000AM

DATE_SHIPPED	RETURN VALUE
Dec 20 1998 3:29:55PM	Dec 1 1998 12:00:00.000000000AM
NULL	NULL

以下の式は、DATE_SHIPPED ポート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 12:00:00.000000000AM
Apr 19 1998 1:31:20PM	Apr 19 1998 12:00:00.000000000AM
Jun 20 1998 3:50:04AM	Jun 20 1998 12:00:00.000000000AM
Dec 20 1998 3:29:55PM	Dec 20 1998 12:00:00.000000000AM
Dec 31 1998 11:59:59PM	Dec 31 1998 12:00:00.000000000AM
NULL	NULL

以下の式は、DATE_SHIPPED ポート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 2:00:00.000000000AM
Apr 19 1998 1:31:20PM	Apr 19 1998 1:00:00.000000000PM
Jun 20 1998 3:50:04AM	Jun 20 1998 3:00:00.000000000AM
Dec 20 1998 3:29:55PM	Dec 20 1998 3:00:00.000000000PM
Dec 31 1998 11:59:59PM	Dec 31 1998 11:00:00.000000000AM
NULL	NULL

以下の式は、DATE_SHIPPED ポート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 2:10:00.000000000AM
Apr 19 1998 1:31:20PM	Apr 19 1998 1:31:00.000000000PM

DATE_SHIPPED	RETURN VALUE
Jun 20 1998 3:50:04AM	Jun 20 1998 3:50:00.000000000AM
Dec 20 1998 3:29:55PM	Dec 20 1998 3:29:00.000000000PM
Dec 31 1998 11:59:59PM	Dec 31 1998 11:59:00.000000000PM
NULL	NULL

TRUNC (数値)

数値が特定の桁数になるように切り詰めます。また、TRUNC を使用して日付を切り詰めることもできます。

構文

TRUNC(*numeric_value* [, *precision*])

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
<i>numeric_value</i>	必須	数値データ型。切り詰める値を渡します。数値データ型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。
<i>precision</i>	オプション	正または負の整数です。整数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。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* 値を渡すと、Data Integration Service は *numeric_value* を近似値の整数に丸めてから式を評価します。

高精度モードでマッピングを実行する場合は、切り詰めを行う前に ROUND 関数を使用します。

例えば、以下の式を使用して QTY ポートの値の切り詰めを行うとします。

TRUNC (QTY / 15)

QTY の値が 15000000 である場合は、セッションから値 999999 が返されます。しかし、正しい値は 1000000 です。

実行時に Data Integration Service は、式の定数部分を評価し、その後に変数部分を評価します。

上の式で、QTY は変数値であり、(1/15)は定数値です。

QTY = 15000000 の場合、この式は以下のように評価されます。

TRUNC (15000000 * (1/15)
TRUNC (15000000 * (1/15)

PRICE	RETURN VALUE
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*)

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
文字列	必須	文字列データ型。大文字のテキストに変換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。

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ている場合、戻り値は Data Integration Service のコードページとデータ移動モードに応じて異なります。

関数に NULL 値を渡した場合は NULL です。

例

次の式は、FIRST_NAME ポート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

UUID4

RFC 4122 の UUID 仕様のバリエーション 4 に基づき、無作為に生成された 16 バイトのバイナリ値を返します。UUID4 は引数を取りません。

構文

UUID4()

戻り値

バイナリ。

UUID4 が NULL 値やエラーを返すことはありません。

UUID_UNPARSE

RFC 4122 に基づき、16 バイトのバイナリ値を 36 文字の文字列表現に変換します。

構文

UUID_UNPARSE(*binary*)

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
<i>binary</i>	必須	バイナリデータ型。36 文字の文字列に変換する 16 バイトのバイナリ値。

戻り値

36 文字の文字列。

引数が NULL の場合は NULL を、引数が 16 バイトのバイナリ値でない場合はエラーを返します。

例

場合によっては、次の式は値「6948DF80-14BD-4E04-8842-7668D9C001F5」を返します。

UUID_UNPARSE(UUID4())

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さなければなりません。

構文

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

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
<i>numeric_value</i>	必須	数値データ型。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。
<i>filter_condition</i>	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

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 Group By ポート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。

Group By ポートがない場合には、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 ポート内のすべての行の分散を計算します。

VARIANCE(TOTAL_SALES)

TOTAL_SALES

2198.0

2256.0

3001.0

NULL

8953.0

RETURN VALUE: 10592444.6666667

索引

数字

10 進数値

変換 [80](#), [97](#), [98](#), [214](#), [215](#), [219](#)

A

ABORT 関数

説明 [55](#)

ABS 関数

説明 [56](#)

ADD_TO_DATE 関数

説明 [57](#)

Advanced Encryption Standard アルゴリズム

説明 [59](#), [60](#)

AES_DECRYPT 関数

説明 [59](#)

AES_ENCRYPT 関数

説明 [60](#)

AND

予約語 [15](#)

ANY 関数

説明 [61](#)

ARRAY 関数

説明 [63](#)

ASCII

ASCII 値の変換 [68](#)

CHR 関数 [68](#)

Unicode 値への変換 [69](#)

文字を ASCII 値に変換 [63](#)

ASCII 関数

説明 [63](#)

AVG 関数

説明 [64](#)

B

bigint

値を変換 [202](#)

C

CAST 関数

説明 [66](#)

CEIL 関数

説明 [67](#)

CHOOSE 関数

説明 [67](#)

CHRCODE 関数

説明 [69](#)

CHR 関数

一重引用符の挿入 [13](#), [68](#)

説明 [68](#)

COBOL 構文

Perl 構文への変換 [159](#)

COLLECT_LIST 関数

説明 [70](#)

COLLECT_MAP 関数

説明 [71](#)

COMPRESS 関数

説明 [72](#)

CONCAT_ARRAY 関数

説明 [74](#)

CONCAT 関数

使用した一重引用符の挿入 [73](#)

説明 [73](#)

CONVERT_BASE 関数

説明 [75](#)

COSH 関数

説明 [76](#)

COS 関数

説明 [76](#)

COUNT 関数

説明 [77](#)

CRC32 関数

説明 [80](#)

CREATE_TIMESTAMP_TZ 関数

説明 [80](#)

CUME 関数

説明 [81](#)

D

DATE_COMPARE 関数

説明 [83](#)

DATE_DIFF 関数

説明 [84](#)

DD_DELETE 定数

更新方式の例 [16](#)

説明 [16](#)

予約語 [15](#)

DD_INSERT 定数

更新方式の例 [16](#)

説明 [16](#)

予約語 [15](#)

DD_REJECT 定数

更新方式の例 [17](#)

説明 [17](#)

予約語 [15](#)

DD_UPDATE 定数

更新方式の例 [17](#)

説明 [17](#)

予約語 [15](#)

DEC_BASE64 関数

説明 [86](#)

DECODE 関数

国際化 [12](#)

説明 [87](#)

DECOMPRESS 関数
説明 [89](#)

E

ENC_BASE64 関数
説明 [90](#)
ERROR 関数
デフォルト値 [90](#)
説明 [90](#)
EXP 関数
説明 [91](#)

F

FALSE 定数
説明 [18](#)
予約語 [15](#)
FIRST 関数
説明 [92](#)
FLOOR 関数
説明 [94](#)
FLOOR 関数 (式)
説明 [94](#)
FV 関数
説明 [95](#)

G

GET_DATE_PART 関数
説明 [96](#)
GET_TIMESTAMP 関数
説明 [98](#)
GET_TIMEZONE 関数
説明 [97](#)
GREATEST 関数
説明 [99](#)

I

IIF 関数
国際化 [12](#)
説明 [100](#)
INDEXOF 関数
説明 [104](#)
:INFA 参照修飾子
予約語 [15](#)
INITCAP 関数
国際化 [12](#)
説明 [105](#)
INSTR 関数
説明 [106](#)
IN 関数
説明 [103](#)
IS_NUMBER 関数
説明 [112](#)
IS_SPACES 関数
説明 [114](#)
IS_DATE 関数
形式文字列 [43](#)
説明 [110](#)
ISNULL 関数
説明 [109](#)

J

J 形式文字列
IS_DATE での使用 [45](#)
TO_CHAR での使用 [42](#)
TO_DATE での使用 [45](#)

L

LAG 関数
説明 [115](#)
LAST_DAY 関数
説明 [118](#)
LAST 関数
説明 [117](#)
LEAD 関数
説明 [119](#)
LEAST 関数
説明 [121](#)
LENGTH 関数
空の文字列テスト [122](#)
説明 [122](#)
:LKP 参照修飾子
説明 [13](#)
予約語 [15](#)
LN 関数
説明 [123](#)
LOG 関数
説明 [124](#)
LOWER 関数
国際化 [12](#)
説明 [125](#)
LPAD 関数
説明 [125](#)
LTRIM 関数
説明 [127](#)

M

MAKE_DATE_TIME 関数
説明 [128](#)
MAP_FROM_ARRAYS 関数
説明 [130](#)
MAP_KEYS 関数
説明 [131](#)
MAP_VALUES 関数
説明 [132](#)
MAP 関数
説明 [129](#)
MAX (Dates)関数
国際化 [12](#)
説明 [132](#)
MAX (numbers) 関数
説明 [133](#)
MAX (numbers)関数
国際化 [12](#)
MAX (String)関数
説明 [135](#)
:MCR 参照修飾子
予約語 [15](#)
MD5 関数
説明 [136](#)
MEDIAN 関数
説明 [137](#)
METAPHONE
説明 [138](#)

MIN (numbers)関数

国際化 [12](#)

説明 [143](#), [145](#)

MIN (Dates)関数

国際化 [12](#)

説明 [142](#)

MOD 関数

説明 [146](#)

MOVINGAVG 関数

説明 [147](#)

MOVINGSUM 関数

説明 [149](#)

N

NOT

予約語 [15](#)

NPER 関数

説明 [150](#)

NULL 値

ISNULL [109](#)

演算子 [19](#)

集計関数 [19](#), [49](#)

チェックイン [109](#)

比較式 [19](#)

フィルタ条件 [19](#)

文字列演算子 [31](#)

論理演算子 [33](#)

NULL 定数

説明 [18](#)

予約語 [15](#)

O

OR

予約語 [15](#)

P

PARSE_JSON 関数

説明 [151](#)

PARSE_XML 関数

説明 [152](#)

PERCENTILE 関数

説明 [153](#)

Perl 互換の正規表現構文

REG_EXTRACT 関数での使用 [159](#)

REG_MATCH 関数での使用 [159](#)

PMT 関数

説明 [155](#)

POWER 関数

説明 [156](#)

PPER 関数

国際化 [12](#)

説明 [225](#)

PROC_RESULT 変数

予約語 [15](#)

PV 関数

説明 [157](#)

R

RAND 関数

説明 [158](#)

RATE 関数

説明 [159](#)

REG_REPLACE 関数

説明 [163](#)

REG_EXTRACT 関数

Perl 構文の使用 [159](#)

説明 [159](#)

REG_MATCH 関数

Perl 構文の使用 [159](#)

説明 [162](#)

REPLACECHR 関数

説明 [164](#)

REPLACESTR 関数

説明 [167](#)

RESPEC 関数

説明 [170](#)

REVERSE 関数

説明 [171](#)

ROUND (数値) 関数

説明 [176](#)

ROUND (Dates)関数

サブ秒の処理 [172](#)

説明 [172](#)

RPAD 関数

説明 [178](#)

RR 形式文字列

IS_DATE での使用 [45](#)

TO_CHAR での使用 [42](#)

TO_DATE での使用 [45](#)

YY と RR の違い [37](#)

説明 [36](#)

RTRIM 関数

説明 [179](#)

S

SESSSTARTTIME 変数

日付関数での使用 [46](#)

SET_DATE_PART 関数

説明 [181](#)

SIGN 関数

説明 [183](#)

SINH 関数

説明 [185](#)

SIN 関数

説明 [184](#)

SIZE 関数

説明 [186](#)

SOUNDEX 関数

説明 [187](#)

SPOUTPUT

予約語 [15](#)

SQL IS_CHAR 関数

REG_MATCH の使用 [162](#)

SQL LIKE 関数

REG_MATCH の使用 [162](#)

SQL LIKE 関数

説明 [189](#)

SQL 構文

Perl 構文への変換 [159](#)

SQRT 関数

説明 [190](#)

SSSSS 形式文字列

IS_DATE での使用 [46](#)

TO_CHAR での使用 [42](#)

TO_DATE での使用 [46](#)

STDDEV 関数
説明 [191](#)
STRUCT_AS 関数
説明 [193](#)
STRUCT 関数
説明 [192](#)
SUBSTR 関数
説明 [194](#)
SUM 関数
説明 [197](#)
SYSDATE 変数
式の使用 [34](#)
説明 [34](#)
予約語 [15](#)
システム変数 [34](#)
SYSTIMESTAMP 関数
説明 [198](#)

T

TANH 関数
説明 [200](#)
TAN 関数
説明 [199](#)
TO__TIMESTAMP_TZ 関数
説明 [219](#)
TO_CHAR (dates)関数
形式文字列 [40](#)
TO_CHAR (Dates)関数
説明 [204](#)
例 [42](#)
TO_DECIMAL38 関数
説明 [215](#)
TO_DECIMAL 関数
説明 [214](#)
TO_CHAR (数値) 関数
説明 [209](#)
TO_DATE 関数
形式文字列 [43](#)
例 [45](#)
説明 [210](#)
TO_FLOAT 関数
説明 [216](#)
TO_INTEGER 関数
説明 [217](#)
TRUE 定数
説明 [19](#)
予約語 [15](#)
TRUNC (Dates)関数
サブ秒の処理 [220](#)
説明 [220](#)
TRUNC (数値) 関数
説明 [223](#)
:TYPE 参照修飾子
予約語 [15](#)

U

Unicode
ASCII 値への変換 [69](#)
Unicode 値の変換 [68](#)
文字を Unicode 値に変換 [63](#)
UUID_UNPARSE 関数
説明 [226](#)
UUID4 関数
説明 [226](#)

V

VARIANCE 関数
説明 [226](#)

Y

YY 形式文字列
IS_DATE での使用 [46](#)
RR と YY の違い [37](#)
TO_CHAR での使用 [43](#)
TO_DATE での使用 [46](#)

い

一重引用符
CHR 関数を使用した一重引用符の挿入 [13](#)

う

ウィンドウ関数
LAG [115](#)
LEAD [119](#)
説明 [55](#)

え

エンコード
ENC_BASE64 関数 [90](#)
文字 [138](#), [187](#)
エンコード関数
AES_DECRYPT [59](#)
AES_ENCRYPT [60](#)
COMPRESS [72](#)
CRC32 [80](#)
DEC_BASE64 [86](#)
DECOMPRESS [89](#)
ENC_BASE64 [90](#)
MD5 [136](#)
説明 [53](#)
演算子
NULL 値 [19](#)
算術演算 [30](#)
算術演算での文字列の使用 [30](#)
説明 [11](#)
比較演算子 [32](#)
比較演算での文字列の使用 [32](#)
文字列演算子 [31](#)
論理演算子 [33](#)
複合 [22](#)
演算子の優先順位
式 [21](#)

お

大文字化
文字列 [105](#), [125](#), [225](#)

か

科学関数
COS [76](#)

科学関数 (続く)

COSH [76](#)
SIN [184](#)
SINH [185](#)
TAN [199](#)
TANH [200](#)
説明 [54](#)

カレンダー

サポートされている日付タイプ [36](#)

関数

文字列 [54](#)
ウィンドウ [55](#)
エンコード [53](#)
科学的 [54](#)
カテゴリ [47](#)
国際化 [12](#)
財務 [53](#)
数値 [53](#)
説明 [11](#)
テスト [55](#)
データクレンジング [51](#)
特殊 [54](#)
変換 [51](#)
文字 [49](#)
集計 [47](#)
日付 [52](#)
複合 [50](#)

き

行

数える [77](#)
スキップ [90](#)
スペースの回避 [114](#)
任意の行を返す [61](#)
合計を返す [149](#)
最後の行を返す [117](#)
最初の行を返す [92](#)
修正移動合計 [81](#)
平均を返す [147](#)

く

グレゴリオ暦

日付関数 [36](#)

け

形式

日付から文字文字列へ [204](#)
文字文字列から日付 [210](#)

形式文字列

IS_DATE 関数 [43](#)
TO_CHAR 関数 [40](#)
TO_DATE 関数 [43](#)
一致 [45](#)
修正ユリウス日 [40, 43](#)
定義 [35](#)
日付 [39](#)
ユリウス日 [40, 43](#)

ケース

大文字に変換 [225](#)

こ

更新方式

DD_DELETE の例 [16](#)
DD_INSERT の例 [16](#)
DD_REJECT の例 [17](#)
DD_UPDATE の例 [17](#)

高精度

ABS [56](#)
ABS 関数 [56](#)
AVG [64](#)
AVG 関数 [64](#)
CEIL [67](#)
CREATE_TIMESTAMP_TZ 関数 [80](#)
CUME [81](#)
CUME 関数 [81](#)
EXP [91](#)
GET_TIMESTAMP 関数 [98](#)
GET_TIMEZONE 関数 [97](#)
LOG [124](#)
MAX (numbers) [133](#)
MAX 関数 [133](#)
MEDIAN [137](#)
MEDIAN 関数 [137](#)
MIN (numbers) [143](#)
MIN 関数 [143](#)
MOD [146](#)
MOVINGAVG [147](#)
MOVINGAVG 関数 [147](#)
MOVINGSUM [149](#)
MOVINGSUM 関数 [149](#)
PERCENTILE [153](#)
PERCENTILE 関数 [153](#)
POWER [156](#)
ROUND 関数 [176](#)
ROUND (数値) [176](#)
SIGN [183](#)
SIN [184](#)
STDDEV 関数 [191](#)
SUM [197](#)
SUM 関数 [197](#)
TO_DECIMAL38 関数 [215](#)
TO_DECIMAL 関数 [214](#)
TO_TIMESTAMP_TZ 関数 [219](#)
TRUNC 関数 [223](#)
算術演算子 [30](#)

struct

生成 [192, 193](#)

構文

一般的なルール [13](#)
式 [12](#)
ポート [13](#)
戻り値 [13](#)

国際化

影響のある関数 [12](#)
ソート順 [12](#)
無効な式 [12](#)

コメント

式への追加 [14](#)

さ

サイズ

マップ [186](#)
配列 [186](#)

財務関数

FV 関数 [95](#)

財務関数 (続く)

NPV 関数 [150](#)

PMT 関数 [155](#)

PV 関数 [157](#)

RATE 関数 [159](#)

説明 [53](#)

サブ秒

ROUND (Dates)関数での処理 [172](#)

TRUNC (Dates)関数での処理 [220](#)

算術演算

日付/時刻値 [46](#)

算術演算子

式での文字列の使用 [30](#)

説明 [30](#)

データ変換に使用 [30](#)

参照修飾子

説明 [13](#)

し

式

演算子の使用 [21](#)

概要 [11](#)

構文 [12](#)

コメントの追加 [14](#)

条件 [18](#)

集計関数

ANY [61](#)

AVG [64](#)

COUNT [77](#)

FIRST [92](#)

LAST [117](#)

MAX (Dates) [132](#)

MAX (numbers) [133](#)

MAX (String) [135](#)

MEDIAN [137](#)

MIN (dates) [142](#)

MIN (numbers) [143](#), [145](#)

NULL 値 [19](#), [49](#)

PERCENTILE [153](#)

STDDEV [191](#)

SUM [197](#)

VARIANCE [226](#)

説明 [47](#)

修正ユリウス日

形式文字列 [40](#), [43](#)

す

数値関数

ABS [56](#)

CEIL [67](#)

CONVERT_BASE [75](#)

CUME [81](#)

EXP [91](#)

FLOOR [94](#)

LN [123](#)

LOG [124](#)

MOD [146](#)

MOVINGAVG [147](#)

MOVINGSUM [149](#)

POWER [156](#)

RAND [158](#)

ROUND (数値) [176](#)

SIGN [183](#)

SQRT [190](#)

数値関数 (続く)

TRUNC (数値) [223](#)

説明 [53](#)

スキップ

行 [90](#)

スペース

DD_REJECT での削除 [17](#)

行での回避 [114](#)

せ

整数

値を変換 [217](#)

西暦 2000 年

日付 [36](#)

そ

ソート順

国際化 [12](#)

て

定数

DD_INSERT [16](#)

DD_REJECT [17](#)

DD_UPDATE [17](#)

FALSE [18](#)

NULL [18](#)

TRUE [19](#)

説明 [11](#)

データ統合サービス

比較式での NULL の取り扱い [19](#)

テキスト文字列

数値の変換 [209](#)

デコード

DEC_BASE64 関数 [86](#)

テスト関数

IS_DATE [110](#)

IS_NUMBER [112](#)

IS_SPACES [114](#)

ISNULL [109](#)

説明 [55](#)

データ型

日付/時刻 [35](#)

データクレンジング関数

GREATEST [99](#)

IN [103](#)

LEAST [121](#)

説明 [51](#)

デフォルトの日時形式

設定 [38](#)

デフォルト値

ERROR 関数 [90](#)

と

特殊関数

ABORT [55](#)

DECODE [87](#)

ERROR [90](#)

IIF [100](#)

説明 [54](#)

ドット演算子
構造要素を持つ構造 [29](#)
説明 [24](#)
データへのアクセスに使用する [24](#)
ネストされたデータ型 [26](#)
複合データ型 [22](#)
トランスフォーメーション言語
SQL との比較 [12](#)
演算子 [21](#)
予約語 [15](#)
トランスフォーメーション言語のアップデート
比較式 [19](#)
ブール式 [19](#)
トランスフォーメーション言語の構成要素
概要 [11](#)
トランスフォーメーション式
NULL 制約 [18](#)
概要 [11](#)

ね

ネストした式
演算子 [21](#)

は

倍精度値
浮動小数点数 [216](#)
配列
生成 [63](#), [70](#), [131](#), [132](#)

ひ

比較演算子
式での文字列の使用 [32](#)
説明 [32](#)
日付
概要 [35](#)
形式文字列 [39](#)
算術演算の実行 [46](#)
修正ユリウス [36](#)
西暦 2000 年 [36](#)
デフォルトの日時形式 [38](#)
フラットファイル [38](#)
丸め処理 [172](#)
文字文字列への変換 [204](#)
ユリウス [36](#)
リレーショナルデータベース [38](#)
関数 [52](#)
切り詰め [220](#)
ビルトイン変数
説明 [34](#)

ふ

フィルタトランスフォーメーション
ISNULL 関数の使用 [109](#)
フィルタ条件
NULL 値 [19](#)
集計関数 [49](#)
複数検索
TRUE 定数の例 [20](#)
プライマリキー制約
NULL 値 [18](#)

フラットファイル
日付 [38](#)

へ

変換
日付文字列 [36](#)
変換関数
CREATE_TIMESTAMP_TZ [80](#)
GET_TIMESTAMP [98](#)
GET_TIMEZONE [97](#)
TO_CHAR (dates) [204](#)
TO_DATE [210](#)
TO_DECIMAL [214](#)
TO_DECIMAL38 [215](#)
TO_FLOAT [216](#)
TO_INTEGER [217](#)
TO_TIMESTAMP_TZ [219](#)
TO_CHAR (数値) [209](#)
説明 [51](#)
変数
SYSDATE [34](#)
ビルトイン変数 [34](#)

ほ

ポート
構文 [13](#)

ま

マッピングパラメータ
定義 [11](#)
マッピング変数
ビルトイン変数 [34](#)
マップ
生成 [71](#), [129](#), [130](#)
丸め処理
数値 [176](#)
日付 [172](#)

も

文字
1 つの置換 [164](#)
ASCII 文字 [63](#), [68](#)
Unicode 文字 [63](#), [68](#), [69](#)
エンコード [138](#), [187](#)
大文字化 [105](#), [125](#), [225](#)
文字列への追加 [125](#), [178](#)
数える [194](#)
数値を返す [122](#)
複数の置換 [167](#)
文字列からの削除 [127](#), [179](#)
文字関数
ASCII [63](#)
CHR [68](#)
CHRCODE [69](#)
CONCAT 関数 [73](#)
INITCAP [105](#)
INSTR [106](#)
LENGTH [122](#)
LOWER [125](#)
LPAD [125](#)

文字関数 (続く)

LTRIM [127](#)
METAPHONE [138](#)
REG_EXTRACT [159](#)
REG_MATCH [162](#)
REG_REPLACE [163](#)
REPLACECHR [164](#)
REPLACESTR [167](#)
RPAD [178](#)
RTRIM [179](#)
SOUNDEX [187](#)
SUBSTR [194](#)
UPPER [225](#)
リスト [49](#)

文字列

1 文字の置換 [164](#)
大文字化 [105](#), [125](#), [225](#)
空白の追加 [125](#)
数値からテキスト文字列への変換 [209](#)
日付から文字への変換 [204](#)
文字セット [106](#)
文字の追加 [125](#)
連結 [31](#), [73](#)
一部を返す [194](#)
空白と文字の削除 [179](#)
空白の削除 [127](#)
長さの変換 [178](#)
複数文字の置換 [167](#)
文字の削除 [127](#)
文字数 [122](#)
文字文字列への変換 [210](#)

文字文字列

日付からの変換 [204](#)
日付への変換 [210](#)

文字列演算子

説明 [31](#)

文字列関数

CHOOSE [67](#)
INDEXOF [104](#)
REVERSE [171](#)
説明 [54](#)

文字列変換

日付 [36](#)

文字列リテラル

一重引用符 [68](#), [73](#)

文字列リテラル (続く)

一重引用符の要件 [13](#)

文字列リテラルの一重引用符

CHR 関数 [68](#)
CHR 関数と CONCAT 関数の使用 [73](#)

戻り値

構文 [13](#)

説明 [11](#)

ゆ

ユリウス日

形式文字列 [40](#), [43](#)

日付関数 [36](#)

よ

予約語

リスト [15](#)

り

リテラル

一重引用符 [68](#), [73](#)

一重引用符の要件 [13](#)

リレーショナルデータベース

日付 [38](#)

れ

連結

文字列 [31](#), [73](#)

ろ

ローカル変数

説明 [11](#)

論理演算子

説明 [33](#)