



Informatica® PowerCenter
10.4.0

トランスフォーメーション ガイド

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、Informatica ロゴ、および PowerCenter 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。

本ソフトウェアまたはドキュメントの一部は、次のサードパーティが有する著作権に従います（ただし、これらに限定されません）。Copyright DataDirect Technologies. All rights reserved. Copyright (C) Sun Microsystems. All rights reserved. Copyright (C) RSA Security Inc. All rights reserved. Copyright (C) Ordinal Technology Corp. All rights reserved. Copyright (C) Aandacht c.v. All rights reserved. Copyright Genivia, Inc. All rights reserved. Copyright Isomorphic Software. All rights reserved. Copyright (C) Meta Integration Technology, Inc. All rights reserved. Copyright (C) Intalio. All rights reserved. Copyright (C) Oracle. All rights reserved. Copyright (C) Adobe Systems Incorporated. All rights reserved. Copyright (C) DataArt, Inc. All rights reserved. Copyright (C) ComponentSource. All rights reserved. Copyright (C) Microsoft Corporation. All rights reserved. Copyright (C) Rogue Wave Software, Inc. All rights reserved. Copyright (C) Teradata Corporation. All rights reserved. Copyright (C) Yahoo! Inc. All rights reserved. Copyright (C) Glyph & Cog, LLC. All rights reserved. Copyright (C) Thinkmap, Inc. All rights reserved. Copyright (C) Clearpace Software Limited. All rights reserved. Copyright (C) Information Builders, Inc. All rights reserved. Copyright (C) OSS Nokalva, Inc. All rights reserved. Copyright Edifecs, Inc. All rights reserved. Copyright Cleo Communications, Inc. All rights reserved. Copyright (C) International Organization for Standardization 1986. All rights reserved. Copyright (C) ej-technologies GmbH. All rights reserved. Copyright (C) Jaspersoft Corporation. All rights reserved. Copyright (C) International Business Machines Corporation. All rights reserved. Copyright (C) yWorks GmbH. All rights reserved. Copyright (C) Lucent Technologies. All rights reserved. Copyright (C) University of Toronto. All rights reserved. Copyright (C) Daniel Veillard. All rights reserved. Copyright (C) Unicode, Inc. Copyright IBM Corp. All rights reserved. Copyright (C) MicroQuill Software Publishing, Inc. All rights reserved. Copyright (C) PassMark Software Pty Ltd. All rights reserved. Copyright (C) LogiXML, Inc. All rights reserved. Copyright (C) 2003-2010 Lorenzi Davide, All rights reserved. Copyright (C) Red Hat, Inc. All rights reserved. Copyright (C) The Board of Trustees of the Leland Stanford Junior University. All rights reserved. Copyright (C) EMC Corporation. All rights reserved. Copyright (C) Flexera Software. All rights reserved. Copyright (C) Jinfonet Software. All rights reserved. Copyright (C) Apple Inc. All rights reserved. Copyright (C) Telerik Inc. All rights reserved. Copyright (C) BEA Systems. All rights reserved. Copyright (C) PDFlib GmbH. All rights reserved. Copyright (C) Orientation in Objects GmbH. All rights reserved. Copyright (C) Tanuki Software, Ltd. All rights reserved. Copyright (C) Ricebridge. All rights reserved. Copyright (C) Sencha, Inc. All rights reserved. Copyright (C) Scalable Systems, Inc. All rights reserved. Copyright (C) jQWidgets. All rights reserved. Copyright (C) Tableau Software, Inc. All rights reserved. Copyright (C) MaxMind, Inc. All rights reserved. Copyright (C) TMatte Software s.r.o. All rights reserved. Copyright (C) MapR Technologies Inc. All rights reserved. Copyright (C) Amazon Corporate LLC. All rights reserved. Copyright (C) Highsoft. All rights reserved. Copyright (C) Python Software Foundation. All rights reserved. Copyright (C) BeOpen.com. All rights reserved. Copyright (C) CNRI. All rights reserved.

本製品には、Apache Software Foundation (<http://www.apache.org/>) によって開発されたソフトウェア、およびさまざまなバージョンの Apache License（まとめて「License」と呼んでいます）の下に許諾された他のソフトウェアが含まれます。これらのライセンスのコピーは、<http://www.apache.org/licenses/> で入手できます。適用法にて要求されないか書面に合意されない限り、ライセンスの下に配布されるソフトウェアは「現状のまま」で配布され、明示的あるいは黙示的かを問わず、いかなる種類の保証や条件も付帯することはありません。ライセンス下での許諾および制限を定める具体的文言については、ライセンスを参照してください。

本製品には、Mozilla (<http://www.mozilla.org/>) によって開発されたソフトウェア、ソフトウェア Copyright (c) The JBoss Group, LLC, all rights reserved、ソフトウェア Copyright (c) 1999-2006 by Bruno Lowagie and Paulo Soares および GNU Lesser General Public License Agreement のさまざまなバージョン (<http://www.gnu.org/licenses/lgpl.html> で参照できる場合がある) に基づいて許諾されたその他のソフトウェアが含まれています。資料は、Informatica が無料で提供しており、一切の保証を伴わない「現状渡し」で提供されるものとし、Informatica LLC は市場性および特定の目的の適合性の黙示の保証などを含めて、一切の明示的及び黙示的保証の責任を負いません。

製品には、ワシントン大学、カリフォルニア大学アーバイン校、およびバンダービルト大学の Douglas C. Schmidt および同氏のリサーチグループが著作権を持つ ACE (TM) および TAO (TM) ソフトウェアが含まれています。Copyright (C) 1993-2006, All rights reserved.

本製品には、OpenSSL Toolkit を使用するために OpenSSL Project が開発したソフトウェア (copyright The OpenSSL Project. All Rights Reserved) が含まれています。また、このソフトウェアの再配布は、<http://www.openssl.org> および <http://www.openssl.org/source/license.html> にある使用条件に従います。

本製品には、Curl ソフトウェア Copyright 1996-2013, Daniel Stenberg, <daniel@haxx.se>が含まれます。All rights reserved. 本ソフトウェアに関する許諾および制限は、<http://curl.haxx.se/docs/copyright.html> にある使用条件に従います。すべてのコピーに上記の著作権情報とこの許諾情報が記載されている場合、目的に応じて、本ソフトウェアの使用、コピー、変更、ならびに配布が有償または無償で許可されます。

本製品には、MetaStuff, Ltd. のソフトウェアが含まれます。Copyright 2001-2005 (C) MetaStuff, Ltd. All Rights Reserved. 本ソフトウェアに関する許諾および制限は、<http://www.dom4j.org/license.html> にある使用条件に従います。

本製品には、Per Bothner のソフトウェアが含まれます。Copyright (C) 1996-2006. All rights reserved. お客様がこのようなソフトウェアを使用するための権利は、ライセンスで規定されています。<http://www.gnu.org/software/kawa/Software-License.html> を参照してください。

本製品には、OSSP UUID ソフトウェアが含まれます。Copyright (C) 2002 Ralf S. Engelschall, Copyright (C) 2002 The OSSP Project Copyright (C) 2002 Cable & Wireless Deutschland. 本ソフトウェアに関する許諾および制限は、<http://www.opensource.org/licenses/mit-license.php> にある使用条件に従います。

本製品には、Boost (<http://www.boost.org/>) によって開発されたソフトウェア、または Boost ソフトウェアライセンスの下で開発されたソフトウェアが含まれます。本ソフトウェアに関する許諾および制限は、http://www.boost.org/LICENSE_1_0.txt にある使用条件に従います。

本製品には、University of Cambridge のが含まれます。Copyright (C) 1997-2007. 本ソフトウェアに関する許諾および制限は、<http://www.pcre.org/license.txt> にある使用条件に従います。

本製品には、The Eclipse Foundation のソフトウェアが含まれます。Copyright (C) 2007. All rights reserved. 本ソフトウェアに関する許諾および制限は、<http://www.eclipse.org/org/documents/epl-v10.php> および <http://www.eclipse.org/org/documents/edl-v10.php> にある使用条件に従います。

本製品には、<http://www.tcl.tk/software/tcltk/license.html>、<http://www.bosrup.com/web/overlib/?License>、<http://www.stlport.org/doc/license.html>、<http://www.asm.ow2.org/license.html>、<http://www.cryptix.org/LICENSE.TXT>、<http://hsqldb.org/web/hsqLicense.html>、<http://httpunit.sourceforge.net/doc/license.html>、<http://jung.sourceforge.net/license.txt>、http://www.gzip.org/zlib/zlib_license.html、<http://www.openldap.org/software/release/license.html>、<http://www.libssh2.org>、<http://slf4j.org/license.html>、<http://www.sente.ch/software/OpenSourceLicense.html>、<http://fusesource.com/downloads/license-agreements/fuse-message-broker-v-5-3-license-agreement>、<http://antlr.org/license.html>、<http://aopalliance.sourceforge.net/>、<http://www.bouncycastle.org/license.html>、<http://www.jgraph.com/jgraphdownload.html>、<http://www.jcraft.com/jsch/LICENSE.txt>、http://jotm.objectweb.org/bsd_license.html に基づいて許諾されたソフトウェアが含まれています。<http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231>、<http://www.slf4j.org/license.html>、<http://nanoxml.sourceforge.net/orig/copyright.html>、<http://www.json.org/license.html>、<http://forge.ow2.org/projects/javaservice/>、<http://www.postgresql.org/about/license.html>、<http://www.sqlite.org/copyright.html>、<http://www.tcl.tk/software/tcltk/license.html>、<http://www.jaxen.org/faq.html>、<http://www.jdom.org/docs/faq.html>、<http://www.slf4j.org/license.html>、<http://www.iodbc.org/dataspace/iodbc/wiki/IODBC/License>、<http://www.keplerproject.org/md5/license.html>、<http://www.toedter.com/en/jcalendar/license.html>、<http://www.edankert.com/bounce/index.html>、<http://www.net-snmp.org/about/license.html>、<http://www.openmdx.org/#FAQ>、http://www.php.net/license/3_01.txt、<http://srp.stanford.edu/license.txt>、<http://www.schneider.com/blowfish.html>、<http://www.jmock.org/license.html>、<http://xsom.java.net>、<http://benalman.com/about/license/>、<https://github.com/CreateJS/EaselJS/blob/master/src/easeljs/display/Bitmap.js>、<http://www.h2database.com/html/license.html#summary>、<http://jsoncpp.sourceforge.net/LICENSE>、<http://jdbc.postgresql.org/license.html>、<http://protobuf.googlecode.com/svn/trunk/src/google/protobuf/descriptor.proto>、<https://github.com/rantav/hector/blob/master/LICENSE>、<http://>

web.mit.edu/Kerberos/krb5-current/doc/mitK5license.html、<http://jibx.sourceforge.net/jibx-license.html>、<https://github.com/lyokato/libgeohash/blob/master/LICENSE>、<https://github.com/hjiang/jsonxx/blob/master/LICENSE>、<https://code.google.com/p/lz4/>、<https://github.com/jedisct1/libsodium/blob/master/LICENSE>、<http://one-jar.sourceforge.net/index.php?page=documents&file=license>、<https://github.com/EsotericSoftware/kryo/blob/master/license.txt>、<http://www.scala-lang.org/license.html>、<https://github.com/tinkerpop/blueprints/blob/master/LICENSE.txt>、<http://gee.cs.oswego.edu/dl/classes/EDU/oswego/cs/dl/util/concurrent/intro.html>、<https://aws.amazon.com/asl/>、<https://github.com/twbs/bootstrap/blob/master/LICENSE>、および <https://sourceforge.net/p/xmlunit/code/HEAD/tree/trunk/LICENSE.txt>。

本製品には、Academic Free License (<http://www.opensource.org/licenses/afl-3.0.php>)、Common Development and Distribution License (<http://www.opensource.org/licenses/cddl1.php>)、Common Public License (<http://www.opensource.org/licenses/cpl1.0.php>)、Sun Binary Code License Agreement Supplemental License Terms、BSD License (<http://www.opensource.org/licenses/bsd-license.php>)、BSD License (<http://opensource.org/licenses/BSD-3-Clause>)、MIT License (<http://www.opensource.org/licenses/mit-license.php>)、Artistic License (<http://www.opensource.org/licenses/artistic-license-1.0>)、Initial Developer's Public License Version 1.0 (<http://www.firebirdsql.org/en/initial-developer-s-public-license-version-1-0/>) に基づいて許諾されたソフトウェアが含まれています。

本製品には、ソフトウェア copyright (C) 2003-2006 Joe Walnes, 2006-2007 XStream Committers が含まれています。All rights reserved. 本ソフトウェアに関する許諾および制限は、<http://j.org/license.html> にある使用条件に従います。本製品には、Indiana University Extreme! Lab によって開発されたソフトウェアが含まれています。詳細については、<http://www.extreme.indiana.edu/> を参照してください。

本製品には、ソフトウェア Copyright (C) 2013 Frank Balluffi and Markus Moeller が含まれています。All rights reserved. 本ソフトウェアに関する許諾および制限は、MIT ライセンスの使用条件に従います。

特許については、<https://www.informatica.com/legal/patents.html> を参照してください。

免責: 本文書は、一切の保証を伴わない「現状渡し」で提供されるものとし、Informatica LLC は他社の権利の非侵害、市場性および特定の目的への適合性の黙示の保証などを含めて、一切の明示的および黙示的保証の責任を負いません。Informatica LLC では、本ソフトウェアまたはドキュメントに誤りのないことを保証していません。本ソフトウェアまたはドキュメントに記載されている情報には、技術的に不正確な記述や誤植が含まれる場合があります。本ソフトウェアまたはドキュメントの情報は、予告なしに変更されることがあります。

NOTICES

この Informatica 製品（以下「ソフトウェア」）には、Progress Software Corporation（以下「DataDirect」）の事業子会社である DataDirect Technologies からの特定のドライバ（以下「DataDirect ドライバ」）が含まれています。DataDirect ドライバには、次の用語および条件が適用されます。

1. DataDirect ドライバは、特定物として現存するままの状態を提供され、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。
2. DataDirect または第三者は、予見の有無を問わず発生した ODBC ドライバの使用に関するいかなる直接的、間接的、偶発的、特別、あるいは結果的損害に対して責任を負わないものとします。本制限事項は、すべての訴訟原因に適用されます。訴訟原因には、契約違反、保証違反、過失、厳格責任、詐称、その他の不法行為を含みますが、これらに限るものではありません。

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: 2020-02-24

目次

序文	22
Informatica のリソース.....	22
Informatica Network.....	22
Informatica ナレッジベース.....	22
Informatica マニュアル.....	23
Informatica 製品可用性マトリックス.....	23
Informatica Velocity.....	23
Informatica Marketplace.....	23
Informatica グローバルカスタマサポート.....	23
 第 1 章: トランスフォーメーションに関する作業	24
トランスフォーメーションの概要.....	24
アクティブなトランスフォーメーション.....	25
パッシブトランスフォーメーション.....	25
接続されていないトランスフォーメーション.....	25
ネイティブトランスフォーメーションと非ネイティブトランスフォーメーション.....	25
トランスフォーメーションの説明.....	26
トランスフォーメーションの作成.....	28
トランスフォーメーションの設定.....	29
トランスフォーメーションの名前の変更.....	29
トランスフォーメーションのポート.....	30
ポートの作成.....	30
ポートの設定.....	30
ポートのリンク.....	30
複数グループのトランスフォーメーション.....	31
式に関する作業.....	31
式エディタの使用.....	33
式の評価.....	35
式の評価.....	35
式の評価の制限.....	35
ローカル変数.....	36
データの一時的な格納と複雑な式の簡素化.....	36
複数行の値の格納.....	37
ストアドプロシージャからの値の取得.....	38
変数ポートの設定のガイドライン.....	38
ポートのデフォルト値.....	39
ユーザー定義のデフォルト値.....	40
ユーザー定義のデフォルト入力値.....	42
ユーザー定義のデフォルト出力値.....	43
デフォルト値の一般ルール.....	45

デフォルト値の検証.	46
トランスフォーメーションのトレースレベルの設定.	46
再利用可能なトランスフォーメーション.	47
インスタンスと継承される変更.	48
式のマッピング変数.	48
再利用可能なトランスフォーメーションの作成.	48
再利用不可能なトランスフォーメーションの格上げ.	49
再利用可能なトランスフォーメーションの再利用不可能なインスタンスの作成.	49
マッピングへの再利用可能なトランスフォーメーションの追加.	49
再利用可能なトランスフォーメーションの変更.	50

第2章: アグリゲータトランスフォーメーション. 51

アグリゲータトランスフォーメーションの概要.	51
アグリゲータトランスフォーメーションのコンポーネント.	52
アグリゲータトランスフォーメーションのプロパティの設定.	52
アグリゲータトランスフォーメーションのポートの設定.	53
集計キャッシュの設定.	53
集計式.	54
集計関数.	54
ネストされた集計関数.	54
条件句.	55
非集計関数.	55
集計関数における NULL 値.	55
Group By ポート.	55
非集計式.	57
デフォルト値.	57
ソート済み入力の使用.	57
[ソート済み入力] の条件.	58
データのソート.	58
アグリゲータトランスフォーメーションの作成.	59
アグリゲータトランスフォーメーションに関するヒント.	59
アグリゲータトランスフォーメーションのトラブルシューティング.	60

第3章: カスタムトランスフォーメーション. 61

カスタムトランスフォーメーションの概要.	61
カスタムトランスフォーメーションを使用して構築されたトランスフォーメーションに関する作業.	62
コードページの互換性.	62
カスタムトランスフォーメーションプロシージャの配布.	63
カスタムトランスフォーメーションの作成.	63
カスタムトランスフォーメーションのルールおよびガイドライン.	64
カスタムトランスフォーメーションのコンポーネント.	64
グループおよびポートに関する作業.	64

グループとポートの作成.	65
グループとポートの編集.	65
ポートの関係の定義.	65
ポート属性に関する作業.	66
ポート属性値の編集.	66
カスタムトランスフォーメーションのプロパティ.	67
アップデートストラテジの設定.	69
スレッド特有のプロシージャコードに関する作業.	69
トランザクション制御に関する作業.	70
トランスフォーメーション範囲.	70
トランザクションの生成.	70
トランザクション境界に関する作業.	71
入力データのブロック.	71
データをブロックするプロシージャコードの記述.	72
カスタムトランスフォーメーションをブロックングトランスフォーメーションとして設定する.	72
カスタムトランスフォーメーションでのマッピングの検証.	73
プロシージャのプロパティに関する作業.	73
カスタムトランスフォーメーションプロシージャの作成.	74
手順 1. カスタムトランスフォーメーションの作成.	74
手順 2. C ファイルの生成.	76
手順 3. トランスフォーメーションロジックでコードを記述する.	77
手順 4. モジュールの構築.	82
手順 5. マッピングの作成.	84
手順 6. ワークフローでのセッションの実行.	84
第 4 章 : カスタムトランスフォーメーション関数.	85
カスタムトランスフォーメーション関数の概要.	85
ハンドルに関する作業.	85
関数リファレンス.	86
行に関する作業.	89
行ベースおよび配列ベースのデータアクセスモードに関するルールとガイドライン.	90
生成済み関数.	90
初期化関数.	91
通知関数.	92
初期化解除関数.	94
API 関数.	96
データアクセスモード設定関数.	97
ナビゲーション関数.	97
プロパティ関数.	100
データタイプの再関連付け関数.	108
データ操作関数（行ベースモード）.	109
パススルーポート設定関数.	112
出力通知関数.	112

データ境界出力通知関数.	113
エラー関数.	113
セッションログメッセージ関数.	114
エラーカウントインクリメント関数.	115
終了要求検査関数.	115
ブロック関数.	116
ポインタ関数.	116
文字列モード変更関数.	117
データコードページ設定関数.	118
行更新方式関数（行ベースモード）.	118
デフォルトの行ストラテジ変更関数.	119
配列ベース API 関数.	120
最大行数関数.	121
行数関数.	122
行有効検証関数.	123
データ操作関数（配列ベースモード）.	123
行更新方式関数（配列ベースモード）.	125
入力エラー行設定関数.	127
第 5 章 : データマスキングトランスフォーメーション.	129
データマスキングトランスフォーメーション.	130
マスキングプロパティ.	130
ロケール.	130
マスキングのタイプ.	131
再現可能な出力.	131
シード.	131
マッピングパラメータ.	131
関連出力ポート.	132
キーマスキング.	132
文字列値のマスキング.	133
数値のマスキング.	133
日時の値のマスキング.	134
置換マスキング.	134
ディクショナリ.	134
格納テーブル.	135
置換マスキングプロパティ.	136
リレーショナルディクショナリ.	137
接続要件.	137
置換マスキングのルールおよびガイドライン.	137
依存マスキング.	138
依存マスキングの例.	138
再現可能な依存マスキング.	139
ランダムマスキング.	139

数値のマスキング.....	140
文字列値のマスキング.....	140
日付値のマスキング.....	140
マスキングルールの適用.....	141
マスク形式.....	141
ソース文字列の文字.....	142
結果文字列の置換文字.....	143
範囲.....	143
ブラー.....	144
式マスキング.....	145
再現可能な式マスキング.....	145
式マスキングのルールとガイドライン.....	146
特殊マスク形式.....	147
社会保障番号のマスキング.....	147
社会保障番号（SSN）形式.....	148
地域コードの要件.....	148
再現可能な社会保障番号のマスキング.....	148
クレジットカード番号のマスキング.....	148
電話番号マスキング.....	149
電子メールアドレスマスキング.....	149
詳細電子メールマスキング.....	150
社会保険番号のマスキング.....	151
SIN の開始桁.....	151
再現可能な SIN 番号.....	151
IP アドレスマスキング.....	151
URL アドレスマスキング.....	152
デフォルト値ファイル.....	152
データマスキングトランスフォーメーションのセッションプロパティ.....	153
データマスキングトランスフォーメーションのルールとガイドライン.....	153
第 6 章 : データマスキングの例.....	155
名前と住所のルックアップファイル.....	155
ルックアップトランスフォーメーションを使用したデータの置換.....	155
式トランスフォーメーションを使用したデータのマスキング.....	159
第 7 章 : 式トランスフォーメーション.....	161
式トランスフォーメーションの概要.....	161
式トランスフォーメーションコンポーネント.....	161
ポートの設定.....	162
値の計算.....	162
式トランスフォーメーションの作成.....	163

第 8 章 : エクスターナルプロシージャトランスフォーメーション.....	164
エクスターナルプロシージャトランスフォーメーションの概要.....	164
コードページの互換性.....	165
エクスターナルプロシージャとエクスターナルプロシージャトランスフォーメーション.....	165
エクスターナルプロシージャトランスフォーメーションのプロパティ.....	165
COM プロシージャと Informatica エクスターナルプロシージャ.....	166
BankSoft の例.....	166
エクスターナルプロシージャトランスフォーメーションのプロパティの設定.....	166
COM プロシージャの作成.....	168
COM プロシージャの作成手順.....	168
COM エクスターナルプロシージャのサーバータイプ.....	168
Visual C++による COM プロシージャの作成.....	169
Visual Basic による COM プロシージャの作成.....	174
Informatica エクスターナルプロシージャの作成.....	175
手順 1. エクスターナルプロシージャトランスフォーメーションの作成.....	176
手順 2.C++ファイルの生成.....	178
手順 3. メソッドスタブにインプリメンテーションを記述.....	180
手順 4. モジュールの構築.....	181
手順 5. マッピングの作成.....	182
手順 6. セッションの実行.....	183
エクスターナルプロシージャの配布.....	184
COM プロシージャの配布.....	184
Informatica モジュールの配布.....	185
作成にあたっての注意.....	185
COM データタイプ.....	185
行レベルのプロシージャ.....	186
プロシージャからの戻り値.....	186
プロシージャの呼び出しにおける例外.....	187
プロシージャのためのメモリ管理.....	187
既存の C/C++ライブラリまたは Visual Basic 関数に対するラッパークラス.....	187
エラーメッセージとトレースメッセージの生成.....	187
接続されていないエクスターナルプロシージャトランスフォーメーション.....	189
COM および Informatica モジュールの初期化.....	189
TX に配布されて使用される他のファイル.....	191
初期化プロパティでのサービスプロセス変数.....	191
エクスターナルプロシージャのインタフェース.....	192
ディスパッチ関数.....	192
エクスターナルプロシージャ関数.....	193
プロパティアクセス関数.....	193
パラメータアクセス関数.....	194
コードページアクセス関数.....	196

トランスフォーメーション名アクセス関数.	196
プロシージャアクセス関数.	197
パーティション関連の関数.	197
トレースレベル関数.	197
第 9 章: フィルタトランスフォーメーション.	199
フィルタトランスフォーメーションの概要.	199
フィルタトランスフォーメーションのコンポーネント.	200
フィルタトランスフォーメーションのポートの設定.	200
フィルタ条件.	200
NULL 値を含む行のフィルタ.	201
フィルタトランスフォーメーションの作成手順.	201
フィルタトランスフォーメーションに関するヒント.	202
第 10 章: HTTP トランスフォーメーション.	203
HTTP トランスフォーメーションの概要.	203
認証.	204
HTTP サーバーへの接続.	204
HTTP トランスフォーメーションの作成.	204
[プロパティ] タブの設定.	205
[HTTP] タブの設定.	206
メソッドの選択.	206
グループおよびポートの設定.	207
URL の設定.	211
例.	213
GET の例.	213
POST の例.	214
SIMPLE POST の例.	215
SIMPLE PATCH の例.	216
SIMPLE PUT の例.	217
SIMPLE DELETE の例.	218
第 11 章: ID 解決トランスフォーメーション.	220
ID 解決トランスフォーメーションの概要.	220
トランスフォーメーションの作成と設定.	220
検索サーバーの接続.	221
システムと検索の設定.	221
選択の表示.	222
ID 解決トランスフォーメーションのタブ.	223
グループおよびポート.	223
入力グループおよびポート.	224
出力グループおよびポート.	224

第 12 章 : Java トランスフォーメーション..... 225

Java トランスフォーメーションの概要.....	225
Java トランスフォーメーションを定義する手順.....	226
アクティブ Java トランスフォーメーションとパッシブ Java トランスフォーメーション.....	226
データ型の変換.....	226
[Java コード] タブの使用.....	228
ポートの設定.....	229
グループとポートの作成.....	229
デフォルトポート値の設定.....	229
Java トランスフォーメーションプロパティの設定.....	230
トランザクション制御に関する作業.....	232
アップデートストラテジの設定.....	233
Java コードの開発.....	233
Java コードスニペットの作成.....	234
Java パッケージのインポート.....	235
Helper コードの定義.....	236
[入力行に達したとき] タブ.....	237
[データの終わりに達したとき] タブ.....	237
[トランザクションを受け取ったとき] タブ.....	238
Java コードによるフラットファイルの解析.....	238
Java トランスフォーメーション設定値の設定.....	239
クラスパスの設定.....	239
高精度 10 進演算の有効化.....	240
サブ秒の処理.....	241
Java トランスフォーメーションのコンパイル.....	241
コンパイルエラーの修正.....	241
コンパイルエラーのソースの検出.....	242
コンパイルエラーの原因の特定.....	242

第 13 章 : Java トランスフォーメーション API のリファレンス..... 244

Java トランスフォーメーション API メソッドの概要.....	244
コミット.....	245
defineJExpression.....	246
failSession.....	247
generateRow.....	247
getInRowType.....	248
getMetadata.....	249
incrementErrorCount.....	249
invokeJExpression.....	250
isNull.....	251
logError.....	251
logInfo.....	252

resetNotification.	252
rollBack.	253
setNull.	254
setOutRowType.	254
storeMetadata.	255

第 14 章 : Java 式..... 257

Java 式の概要.	257
式の関数タイプ.	258
[式の定義] [関数の定義] ダイアログボックスを使用した式の定義.	258
手順 1. 関数の設定.	259
手順 2. 式の作成と検証.	259
手順 3. 式の Java コードの生成.	259
[式の定義] [関数の定義] ダイアログボックスを使用した式の作成と Java コードの生成.	259
Java 式のテンプレート.	260
単純なインタフェースに関する作業.	260
invokeJExpression.	260
単純なインタフェースの例.	261
高度なインタフェースに関する作業.	262
高度なインタフェースを使用した式の呼び出し.	262
高度なインタフェースに関する作業のルールとガイドライン.	262
EDatatype クラス.	263
JExprParamMetadata クラス.	263
defineJExpression.	264
JExpression クラス.	265
高度なインタフェースの例.	265
JExpression クラス API リファレンス.	266
getBytes.	266
getDouble.	267
getInt.	267
getLong.	267
getResultDataType.	267
getResultMetadata.	267
getStringBuffer.	268
呼び出し.	268
isResultNull.	268

第 15 章 : Java トランスフォーメーションの例..... 270

Java トランスフォーメーションの例の概要.	270
手順 1. マッピングのインポート.	271
手順 2. トランスフォーメーションの作成とポートの設定.	271
手順 3. Java コードの入力.	272
[パッケージのインポート] タブ.	272

[Helper コード] タブ	273
[入力行に達したとき] タブ	273
手順 4. Java コードのコンパイル	275
手順 5. セッションとワークフローの作成	275
データの例	275

第 16 章: ジョイナトランスフォーメーション..... 277

ジョイナトランスフォーメーションの概要	277
ジョイナトランスフォーメーションに関する作業	278
ジョイナトランスフォーメーションのプロパティ	278
ジョイン条件の定義	280
結合タイプの定義	280
Normal ジョイン	281
Master Outer ジョイン	282
Detail Outer ジョイン	282
Full Outer ジョイン	282
ソート済み入力の使用	283
ソート順の設定	283
マッピングへのトランスフォーメーションの追加	284
ジョイナトランスフォーメーションの設定	284
ジョイン条件の定義	284
1 つのソースからのデータの結合	285
同じパイプラインの 2 つのブランチの結合	286
同じソースの 2 つのインスタンスの結合	286
1 つのソースからのデータの結合に関するガイドライン	287
ソースパイプラインのブロック	287
未ソートのジョイナトランスフォーメーション	287
ソート済みのジョイナトランスフォーメーション	288
トランザクションに関する作業	288
1 つのパイプラインにおけるトランザクション境界の保持	289
明細パイプラインにおけるトランザクション境界の保持	290
2 つのパイプラインのトランザクション境界の削除	290
ジョイナトランスフォーメーションの作成	290
ジョイナトランスフォーメーションに関するヒント	291

第 17 章: ルックアップトランスフォーメーション..... 292

ルックアップトランスフォーメーションの概要	292
ルックアップソースのタイプ	293
リレーショナルルックアップ	293
フラットファイルルックアップ	294
パイプラインルックアップ	295
接続されたルックアップと接続されていないルックアップ	296
接続されたルックアップ	297

接続されていないルックアップ.....	298
ルックアップのコンポーネント.....	299
ルックアップソース.....	299
ルックアップポート.....	299
ルックアップ プロパティ.....	300
ルックアップ条件.....	300
ルックアッププロパティ.....	301
セッションにおけるルックアッププロパティの設定.....	306
ルックアップクエリ.....	308
デフォルトのルックアップクエリ.....	308
ルックアップクエリのオーバーライド.....	308
キャッシュを使用しないルックアップの SQL オーバーライド.....	311
ルックアップソースフィルタ.....	311
ルックアップ条件.....	312
キャッシュを使用しない、または静的キャッシュ.....	313
動的キャッシュ.....	313
複数の一致の処理.....	313
ルックアップキャッシュ.....	314
戻り値としての複数の行.....	315
戻り値としての複数の行に関するルールおよびガイドライン.....	315
接続されていないルックアップトランスフォーメーションの設定.....	316
手順 1. 入力ポートの追加.....	316
手順 2. ルックアップ条件の追加.....	316
手順 3. 戻り値の指定.....	317
手順 4. 式からのルックアップの呼び出し.....	317
データベースデッドロックに対するレジリエンス.....	318
ルックアップトランスフォーメーションの作成.....	318
再利用可能なパイプラインルックアップトランスフォーメーションの作成.....	320
再利用不可能なパイプラインルックアップトランスフォーメーションの作成.....	320
ルックアップトランスフォーメーションのヒント.....	320
第 18 章 : ルックアップキャッシュ.....	322
ルックアップキャッシュの概要.....	322
キャッシュの比較.....	324
接続されたルックアップキャッシュの作成.....	324
連続したキャッシュ.....	325
コンカレントキャッシュ.....	325
パーシステントルックアップキャッシュの使用.....	326
非永続キャッシュの使用.....	326
永続キャッシュの使用.....	326
ルックアップキャッシュの再構築.....	327
キャッシュを使用しないルックアップまたは静的キャッシュに関する作業.....	328
ルックアップキャッシュの共有.....	328

名前なしルックアップキャッシュの共有.....	328
名前付きルックアップキャッシュの共有.....	331
ルックアップキャッシュに関するヒント.....	335

第 19 章 : 動的ルックアップキャッシュ..... 336

動的ルックアップキャッシュの概要.....	336
動的ルックアップのプロパティ.....	337
NewLookupRows.....	338
関連する式.....	338
NULL 値.....	339
比較においてポートを無視.....	340
SQL オーバーライド.....	341
ルックアップトランスフォーメーションの値.....	341
初期キャッシュ値.....	342
入力値.....	342
ルックアップ値.....	343
出力値.....	343
動的ルックアップキャッシュの更新.....	344
挿入でなければ更新.....	344
更新でなければ挿入.....	345
動的ルックアップを使用したマッピング.....	346
アップストリームのアップデートストラテジトランスフォーメーションの設定.....	346
ダウンストリームトランスフォーメーションの設定.....	347
動的ルックアップキャッシュを持つセッションの設定.....	348
条件付きの動的キャッシュの更新.....	349
セッション処理.....	349
条件付きの動的キャッシュルックアップの設定.....	349
式の結果を使用した動的キャッシュの更新.....	350
式の値が NULL.....	350
セッション処理.....	350
動的キャッシュの更新のための式の設定.....	350
キャッシュとルックアップソースの同期.....	351
NewLookupRow.....	351
動的キャッシュ同期の設定.....	352
動的ルックアップキャッシュの例.....	352
動的ルックアップキャッシュのルールとガイドライン.....	353

第 20 章 : ノーマライザトランスフォーメーション..... 355

ノーマライザトランスフォーメーションの概要.....	355
ノーマライザトランスフォーメーションのコンポーネント.....	356
[ポート] タブ.....	356
[プロパティ] タブ.....	357
[ノーマライザ] タブ.....	358

ノーマライザトランスフォーメーションの生成キー.....	359
生成されたキー値の格納.....	359
生成キー値の変更.....	360
VSAM ノーマライザトランスフォーメーション.....	360
VSAM ノーマライザの [ポート] タブ.....	362
VSAM ノーマライザのタブ.....	362
VSAM ノーマライザトランスフォーメーションの作成手順.....	363
パイプラインノーマライザトランスフォーメーション.....	364
パイプラインノーマライザの [ポート] タブ.....	366
パイプラインノーマライザのタブ.....	366
パイプラインノーマライザトランスフォーメーションの作成手順.....	367
マッピングにおけるノーマライザトランスフォーメーションの使用.....	368
キー値の生成.....	370
ノーマライザトランスフォーメーションのトラブルシューティング.....	372
第 21 章 : ランクトランスフォーメーション.....	373
ランクトランスフォーメーションの概要.....	373
文字列値のランク付け.....	374
ランクキャッシュ.....	374
ランクトランスフォーメーションのプロパティ.....	374
ランクトランスフォーメーションのポート.....	374
ランクインデックス.....	375
グループの定義.....	375
ランクトランスフォーメーションの作成.....	376
第 22 章 : ルータトランスフォーメーション.....	378
ルータトランスフォーメーションの概要.....	378
グループに関する作業.....	380
入力グループ.....	380
出力グループ.....	380
グループフィルタ条件の使用.....	380
グループの追加.....	382
ポートに関する作業.....	382
マッピング内のルータトランスフォーメーションの接続.....	383
ルータトランスフォーメーションの作成.....	383
第 23 章 : シーケンスジェネレータトランスフォーメーション.....	384
シーケンスジェネレータトランスフォーメーションの概要.....	384
シーケンスジェネレータのポート.....	385
パススルーポート.....	385
NEXTVAL ポート.....	385
CURRVAL.....	389
シーケンスジェネレータトランスフォーメーションのプロパティ.....	390

開始値.	392
増分.	392
終了値.	393
増分値.	393
値の範囲内でのサイクル.	393
現在の値.	393
キャッシュされる値の数.	394
再利用不可能なシーケンスジェネレータ.	395
再利用可能なシーケンスジェネレータ.	395
リセット.	396
行順序の保持.	396
シーケンスデータオブジェクト.	396
シーケンスデータオブジェクトの作成.	397
シーケンスジェネレータトランスフォーメーションの作成.	398
シーケンスジェネレータトランスフォーメーションの作成.	399
FAQ (よくある質問)	400
シーケンスジェネレータトランスフォーメーション非ネイティブ環境で.	401
Blaze エンジンでのシーケンスジェネレータトランスフォーメーション.	401
Spark エンジンでのシーケンスジェネレータトランスフォーメーション.	401
第 24 章: ソータトランスフォーメーション.	402
ソータトランスフォーメーションの概要.	402
データのソート.	402
ソータトランスフォーメーションのプロパティ.	403
ソータキャッシュサイズ.	404
大文字小文字の区別.	404
作業ディレクトリ.	404
重複しない出力行.	404
トレースレベル.	405
NULL を低位として処理.	405
トランスフォーメーション範囲.	405
ソータトランスフォーメーションの作成.	405
第 25 章: ソース修飾子トランスフォーメーション.	407
ソース修飾子トランスフォーメーションの概要.	407
トランスフォーメーションデータタイプ.	408
ターゲットのロード順.	408
日時の値.	408
パラメータおよび変数.	409
ソース修飾子トランスフォーメーションのプロパティ.	409
デフォルトクエリ.	410
デフォルトのクエリの表示.	411
デフォルトのクエリの上書き.	411

ソースデータの結合.	412
デフォルトジョイン.	412
ユーザー作成のジョイン.	413
異種データのジョイン.	413
キー関係の作成.	414
SQL クエリの追加.	414
ユーザー定義ジョインの入力.	416
アウタージョインのサポート.	416
Informatica ジョイン構文.	417
アウタージョインの作成.	422
一般的なデータベース構文の制約.	423
ソースフィルタの入力.	423
ソート済みポートの使用.	424
個別に選択.	425
セッションでの [個別に選択] の上書き.	426
セッション実行前/実行後の SQL コマンドの追加.	426
ソース修飾子トランスフォーメーションの作成.	427
ソース修飾子トランスフォーメーションの手動での作成.	427
ソース修飾子トランスフォーメーションオプションの設定.	427
ソース修飾子トランスフォーメーションのトラブルシューティング.	428
第 26 章: SQL トランスフォーメーション.	429
SQL トランスフォーメーションの概要.	429
スクリプトモード.	430
例.	430
スクリプトモードのルールとガイドライン.	431
クエリモード.	432
静的 SQL クエリの使用.	432
動的 SQL クエリの使用.	434
パススルーポートの設定.	435
パッシュモードの設定.	436
クエリモードのルールとガイドライン.	436
データベースへの接続.	437
静的データベース接続の使用.	437
論理データベース接続の受け渡し.	437
フル接続情報を渡す.	438
データベース接続のルールおよびガイドライン.	440
セッション処理.	440
トランザクション制御.	441
高可用性.	442
SQL クエリログ.	443
入力行と出力行のカーディナリティ.	444
クエリ文の処理.	444

影響を受けた行数.	444
最大出力行数.	445
エラー行について.	446
SQL エラー時の処理の継続.	447
SQL トランスフォーメーションプロパティ.	448
[プロパティ] タブ.	448
[SQL 設定] タブ.	450
[SQL ポート] タブ.	450
SQL 文.	451
SQL トランスフォーメーションの作成.	452

第 27 章: マッピングにおける SQL トランスフォーメーションの使用. 454

SQL トランスフォーメーションの例の概要.	454
動的更新の例.	454
ソースファイルの定義.	455
ターゲット定義の作成.	456
データベーステーブルの作成.	456
式トランスフォーメーションの設定.	457
SQL トランスフォーメーションの定義.	457
セッション属性の設定.	459
ターゲットデータの結果.	459
動的接続の例.	459
ソースファイルの定義.	460
ターゲット定義の作成.	461
データベーステーブルの作成.	461
データベース接続の作成.	461
式トランスフォーメーションの設定.	461
SQL トランスフォーメーションの定義.	462
セッション属性の設定.	463
ターゲットデータの結果.	464

第 28 章: ストアドプロシージャトランスフォーメーション. 465

ストアドプロシージャトランスフォーメーションの概要.	465
入力データと出力データ.	466
接続されたモードと接続されていないモード.	467
ストアドプロシージャの実行タイミングの指定.	468
マッピングでのストアドプロシージャの使用.	469
ストアドプロシージャの記述.	470
ストアドプロシージャの例.	470
ストアドプロシージャトランスフォーメーションの作成.	472
ストアドプロシージャのインポート.	472
手動によるストアドプロシージャトランスフォーメーションの作成.	473
ストアドプロシージャのオプションの設定.	475

ストアドプロシージャの変更.	476
接続されたトランスフォーメーションの設定.	477
接続されていないトランスフォーメーションの設定.	477
式からストアドプロシージャを呼ぶ場合.	478
セッション実行前または実行後のストアドプロシージャの呼び出し.	480
エラー処理.	481
セッション実行前のエラー.	481
セッション実行後のエラー.	482
セッションのエラー.	482
サポートされるデータベース.	482
SQL の宣言.	482
パラメータの種類.	483
マッピングにおける入出力ポート.	483
サポートされる戻り値の型.	483
式のルール.	483
ストアドプロシージャトランスフォーメーションに関するヒント.	484
ストアドプロシージャトランスフォーメーションのトラブルシューティング.	484
第 29 章 : トランザクション制御トランスフォーメーション.	486
トランザクション制御トランスフォーメーションの概要.	486
トランザクション制御トランスフォーメーションのプロパティ.	487
[プロパティ] タブ.	487
例.	488
マッピングでのトランザクション制御トランスフォーメーションの使用.	489
複数のターゲットを持つトランザクション制御マッピングの例.	490
マッピングのガイドラインと検証.	491
トランザクション制御トランスフォーメーションの作成.	492
第 30 章 : 共有体トランスフォーメーション.	493
共有体トランスフォーメーションの概要.	493
共有体トランスフォーメーションのルールとガイドライン.	493
共有体トランスフォーメーションのコンポーネント.	494
グループおよびポートに関する作業.	494
共有体トランスフォーメーションの作成.	494
マッピングにおける共有体トランスフォーメーションの使用.	495
第 31 章 : 構造化されていないデータのトランスフォーメーション.	496
構造化されていないデータのトランスフォーメーションの概要.	496
構造化されていないデータオプションの設定.	497
Data Transformation リポジトリディレクトリの設定.	498
Data Transformation サービスのタイプ.	498
構造化されていないデータのトランスフォーメーションのコンポーネント.	499
[プロパティ] タブ.	499

[UDT 設定] タブ.....	500
ステータストレースメッセージの表示.....	501
構造化されていないデータのトランスフォーメーションポート.....	502
入力タイプと出力タイプ.....	502
追加の構造化されていないデータのトランスフォーメーションポート.....	503
Data Transformation サービスからのポートの作成.....	504
構造化されていないデータのトランスフォーメーションサービス名.....	504
リレーショナル階層.....	505
階層スキーマのエクスポート.....	505
マッピング.....	506
リレーショナルテーブル用の Word ドキュメントの解析.....	506
XML からの Excel シートの作成.....	506
分割 XML ファイルの出力.....	507
非構造化データマッピングのルールおよびガイドライン.....	508
構造化されていないデータのトランスフォーメーションの作成.....	508
第 32 章 : アップデートストラテジトランスフォーメーション.....	510
アップデートストラテジトランスフォーメーションの概要.....	510
アップデートストラテジの設定.....	511
マッピング内の行のフラグ設定.....	511
拒否された行の転送.....	511
アップデートストラテジ式.....	512
アグリゲータトランスフォーメーションとアップデートストラテジトランスフォーメーション	512
ルックアップトランスフォーメーションとアップデートストラテジトランスフォーメーション	513
セッションのアップデートストラテジの設定.....	513
すべての行に対する操作の指定.....	513
個々のターゲットテーブルに対する操作の指定.....	514
アップデートストラテジチェックリスト.....	515
第 33 章 : XML トランスフォーメーション.....	516
XML ソース修飾子トランスフォーメーション.....	516
XML パーサトランスフォーメーション.....	516
XML ジェネレータトランスフォーメーション.....	517
索引.....	518

序文

Informatica トランスフォーメーションの設定、ガイドライン、使用法、および実行時の動作を確認するには、『PowerCenter(R) トランスフォーメーションガイド』を参照してください。トランスフォーメーションは、Integration Service がデータに対して実行する処理を表すリポジトリオブジェクトです。

このガイドを参照すると、データを生成、変更、または転送するデータ処理を実行できます。マッピングをどこでどのように実行するかに応じて各トランスフォーメーションのサポートを表示します。データは、トランスフォーメーションポートを通過して、マッピングまたはマップレット内のリンクされたターゲットおよびその他のトランスフォーメーションに渡されます。アクティブなトランスフォーメーションでは、式エディタを使用して式を入力できます。トランスフォーメーションには複数の入力グループと出力グループを設定できます。これらは、1 行の入力データまたは出力データを定義する、ポートのセットです。

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に関する作業

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の概要, 24 ページ](#)
- [トランスフォーメーションの作成, 28 ページ](#)
- [トランスフォーメーションの設定, 29 ページ](#)
- [トランスフォーメーションのポート, 30 ページ](#)
- [複数グループのトランスフォーメーション, 31 ページ](#)
- [式に関する作業, 31 ページ](#)
- [ローカル変数, 36 ページ](#)
- [ポートのデフォルト値, 39 ページ](#)
- [トランスフォーメーションのトレースレベルの設定, 46 ページ](#)
- [再利用可能なトランスフォーメーション, 47 ページ](#)

トランスフォーメーションの概要

トランスフォーメーションとは、データの生成や変更を行ったり、データ渡したりするリポジトリオブジェクトのことです。Designer には、特定の関数を実行する一連のトランスフォーメーションが用意されています。たとえば、アグリゲータトランスフォーメーションはデータのグループに対して計算を実行します。

マッピング内のトランスフォーメーションは、Integration Service がデータに対して実行する処理を表します。データは、マッピングまたはマップレット内のリンクされたトランスフォーメーションポートを通過します。

トランスフォーメーションには以下のタイプがあります。

- アクティブ/パッシブ
- コネクトされている、またはコネクトされていない
- ネイティブまたは非ネイティブ

アクティブなトランスフォーメーション

アクティブなトランスフォーメーションは、以下のいずれかのアクションを実行できます。

- **トランスフォーメーションを通過する行の数を変更する。**例えば、フィルタトランスフォーメーションはフィルタ条件を満たさない行を削除するため、アクティブなトランスフォーメーションです。複数グループのトランスフォーメーションも、トランスフォーメーションを通過する行の数を変更できるため、アクティブなトランスフォーメーションです。
- **トランザクション境界を変更する。**例えば、トランザクション制御トランスフォーメーションは行ごとに評価される式に基づいてコミットまたはロールバックトランザクションを定義するため、アクティブなトランスフォーメーションです。
- **行タイプを変更する。**例えば、アップデートストラテジトランスフォーメーションは挿入、削除、更新、または拒否のフラグを行に設定するため、アクティブなトランスフォーメーションです。

Integration Service がアクティブなトランスフォーメーションによって渡される行を連結できない場合があるため、Designer では、複数のアクティブなトランスフォーメーションや1つのアクティブおよびパッシブなトランスフォーメーションを同じダウストリームトランスフォーメーションまたはトランスフォーメーション入力グループに接続することはできません。例えば、マッピング内の1つのブランチに、行に削除のフラグを付けるアップデートストラテジトランスフォーメーションが含まれているとします。別のブランチには、行に挿入のフラグを付けるアップデートストラテジトランスフォーメーションが含まれています。これらのトランスフォーメーションを1つのトランスフォーメーション入力グループに接続した場合、Integration Service は行の削除操作と挿入操作を結合できません。

シーケンスジェネレータトランスフォーメーションはこのルールの例外です。Designer では、シーケンスジェネレータトランスフォーメーションおよびアクティブなトランスフォーメーションを同じダウストリームトランスフォーメーションまたはトランスフォーメーション入力グループに接続できます。シーケンスジェネレータトランスフォーメーションはデータを受け取りません。一意の数値を生成します。その結果、Integration Service では、シーケンスジェネレータトランスフォーメーションおよびアクティブなトランスフォーメーションによって渡された行を連結する際に問題が発生しません。

パッシブトランスフォーメーション

パッシブトランスフォーメーションでは、トランスフォーメーションを通過する行数は変更されませんが、トランザクション境界と行タイプが維持されます。

アップストリームブランチのすべてのトランスフォーメーションがパッシブの場合、複数のトランスフォーメーションを同じダウストリームトランスフォーメーションまたは同じトランスフォーメーション入力グループに接続できます。ブランチを発生させるトランスフォーメーションはアクティブである場合とパッシブである場合があります。

接続されていないトランスフォーメーション

トランスフォーメーションはデータフローに接続されているか、接続されていないかのいずれかです。コネクトされていないトランスフォーメーションとは、マッピング内の他のトランスフォーメーションに接続されていないトランスフォーメーションのことです。コネクトされていないトランスフォーメーションは他のトランスフォーメーション内で呼び出され、そのトランスフォーメーションに値を返します。

ネイティブトランスフォーメーションと非ネイティブトランスフォーメーション

ネイティブトランスフォーメーションとは、Designer が提供するトランスフォーメーションのセットのことです。非ネイティブトランスフォーメーションとは、カスタムトランスフォーメーションを使用して作成するトランスフォーメーションのことです。Designer では、Java、SQL、共有体トランスフォーメーションなどの、非ネイティブトランスフォーメーションもいくつか提供しています。カスタムトランスフォーメーションに適

用されるルールは、カスタムトランスフォーメーションを使用して作成される非ネイティブトランスフォーメーションにも適用されます。

トランスフォーメーションの説明

以下の表に、各トランスフォーメーションの簡単な説明を示します。

トランスフォーメーション	タイプ	説明
Aggregator	- アクティブ - 接続済 - ネイティブ	集計の計算を実行します。
アプリケーションソース修飾子	- アクティブ - 接続済 - 非ネイティブ	統合サービスがセッションを実行するときにアプリケーション（ERP ソースなど）から読み込む行を表します。
カスタム	- アクティブ/ パッシブ - 接続済 - 非ネイティブ	共有ライブラリまたは DLL のプロシージャを呼び出します。
データマスキング	- パッシブ - 接続済 - 非ネイティブ	非プロダクション環境で、センシティブなプロダクションデータを現実的なテストデータに置き換えます。
Expression	- パッシブ - 接続済 - ネイティブ	値を計算します。
エクスターナルプロシージャ	- パッシブ - コネクトされている、 またはコネクトされていない - ネイティブ	共有ライブラリまたは Windows の COM 層にあるプロシージャを呼び出します。
Filter	- アクティブ - 接続済 - ネイティブ	データをフィルタリングします。
HTTP	- パッシブ - 接続済 - 非ネイティブ	HTTP サーバにコネクトし、データを読み取るか、または更新します。
入力	- パッシブ - 接続済 - ネイティブ	マプレット入力行を定義します。Mapplet Designer で使用できます。
Java	- アクティブ/ パッシブ - 接続済 - 非ネイティブ	Java で書かれたユーザコードを実行します。ユーザロジックのバイトコードはリポジトリ内に格納されています。

トランスフォーメーション	タイプ	説明
ジョイナ	<ul style="list-style-type: none"> - アクティブ - 接続済 - ネイティブ 	異なるデータベースまたはフラットファイルシステムから得たデータを結合します。
ルックアップ	<ul style="list-style-type: none"> - アクティブ/パッシブ - コネクトされている、またはコネクトされていない - ネイティブ 	フラットファイル、リレーショナルテーブル、ビュー、またはシノニムからのデータを検索し、返します。
ノーマライザ	<ul style="list-style-type: none"> - アクティブ - 接続済 - ネイティブ 	COBOL ソースの Source Qualifier。リレーショナルソースまたはフラットファイルソースから取得したデータを正規化するためにパイプライン内で使用することもできます。
アウトプット	<ul style="list-style-type: none"> - パッシブ - 接続済 - ネイティブ 	マプレット出力行を定義します。Mapplet Designer で使用できます。
ランク	<ul style="list-style-type: none"> - アクティブ - 接続済 - ネイティブ 	レコードのランキング処理を行います。
ルータ	<ul style="list-style-type: none"> - アクティブ - 接続済 - ネイティブ 	グループ条件に基づいて、複数のトランスフォーメーションにデータをルーティングします。
シーケンスジェネレータ	<ul style="list-style-type: none"> - パッシブ - 接続済 - ネイティブ 	プライマリキーを生成します。
ソータ	<ul style="list-style-type: none"> - アクティブ - 接続済 - ネイティブ 	ソートキーに基づいてデータをソートします。
Source Qualifier	<ul style="list-style-type: none"> - アクティブ - 接続済 - ネイティブ 	統合サービスがセッションを実行するときに、リレーショナルソースまたはフラットファイルソースから読み込む行を表します。
SQL	<ul style="list-style-type: none"> - アクティブ/パッシブ - 接続済 - 非ネイティブ 	データベースに対して SQL クエリーを実行します。
ストアドプロシージャ	<ul style="list-style-type: none"> - パッシブ - コネクトされている、またはコネクトされていない - ネイティブ 	ストアドプロシージャを呼び出します。

トランスフォーメーション	タイプ	説明
トランザクションコントロール	- アクティブ - 接続済 - ネイティブ	コミットとロールバックのトランザクションを定義します。
Union	- アクティブ - 接続済 - 非ネイティブ	異なるデータベースまたはフラットファイルシステムから得たデータを結合します。
Unstructured Data	- アクティブ/ パッシブ - 接続済 - 非ネイティブ	データを非構造化形式または半構造化形式に変換します。
アップデートストラテジ	- アクティブ - 接続済 - ネイティブ	行を挿入、削除、更新、または拒否するかどうかを決定します。
XML ジェネレータ	- アクティブ - 接続済 - ネイティブ	1 つ以上の入力ポートからデータを読み込み、出力ポート 1 つに XML を出力します。
XML パーサ	- アクティブ - 接続済 - ネイティブ	入力ポート 1 つから XML を読み込み、1 つ以上の出力ポートに出力します。
XML ソース修飾子	- アクティブ - 接続済 - ネイティブ	統合サービスがセッションを実行するときに XML ソースから読み込む行を表します。

マッピングを作成する場合、ビジネスの目的に応じてデータを処理できるようにトランスフォーメーションを追加します。トランスフォーメーションをマッピングに組み込むには、以下のタスクを実行します。

1. **トランスフォーメーションを作成する。** Mapping Designer でマッピングの一部として作成するか、Mapplet Designer でマップレットの一部として作成するか、または Transformation Developer で再利用可能なトランスフォーメーションとして作成します。
2. **トランスフォーメーションを設定します。** 各タイプのトランスフォーメーションには、設定可能な固有のオプションのセットがあります。
3. **トランスフォーメーションを他のトランスフォーメーションおよびターゲット定義にリンクします。** 1 つのポートを別のポートにドラッグして、マッピングまたはマップレットでリンクします。

トランスフォーメーションの作成

以下の Designer ツールを使用してトランスフォーメーションを作成することができます。

- **Mapping Designer。** ソースをターゲットに接続するトランスフォーメーションを作成します。マッピング内のトランスフォーメーションは、再利用可能に設定しない限り他のマッピングで使用することはできません。

- **Transformation Developer.** 再利用可能なトランスフォーメーションという、複数のマッピングで利用できる独立したトランスフォーメーションを作成します。
- **Mapplet Designer.** 複数のマッピングで使用する一連のトランスフォーメーションを作成および設定します。このトランスフォーメーションは、マップレットと呼ばれます。

同じ手順を用いて、Mapping Designer、Transformation Developer、および Mapplet Designer でトランスフォーメーションを作成できます。

トランスフォーメーションを作成するには：

1. 該当する Designer ツールを開きます。
2. Mapping Designer で、マッピングを開くか作成します。Mapplet Designer で、マップレットを開くか作成します。
3. [トランスフォーメーション] - [作成] をクリックし、作成するトランスフォーメーションのタイプを選択します。
4. マッピング上でトランスフォーメーションを配置したい位置をクリックしてドラッグします。
ワークスペースに、新しいトランスフォーメーションが表示されます。次に、新たにポートを追加し、また他のプロパティを指定することにより、トランスフォーメーションを設定する必要があります。

トランスフォーメーションの設定

トランスフォーメーションを作成したら、設定が必要です。各トランスフォーメーションには、以下の共通タブが含まれています。

- **Transformation.** トランスフォーメーションに名前を付けたり、説明を追加したりします。
- **ポート.** ポートを追加および設定します。
- **Properties.** トランスフォーメーション固有のプロパティを設定します。

トランスフォーメーションによっては、ジョイナトランスフォーメーションまたはノーマライザトランスフォーメーションの条件を入力するための [条件] タブなど、上記以外のタブを含んでいるものもあります。

トランスフォーメーションを設定する場合、以下の作業を実行する場合があります。

- **ポートを追加します。** トランスフォーメーションを通過するデータの列を定義します。
- **グループを追加します。** いくつかのトランスフォーメーションでは、トランスフォーメーションに入力またはトランスフォーメーションから出力されるデータの行を定義する入力および出力グループを定義します。
- **式を入力します。** データの変換を実行するいくつかのトランスフォーメーションに、SQL のような式を入力します。
- **ローカル変数を定義します。** いくつかのトランスフォーメーションに、一時的にデータを格納するローカル変数を定義します。
- **デフォルト値を上書きします。** 入力 NULL および出力トランスフォーメーションエラーを取り扱うための、ポートのデフォルト値を設定します。
- **トレースレベルを入力します。** Integration Service がトランスフォーメーションに関するセッションログに書き込む詳細情報の量を選択します。

トランスフォーメーションの名前の変更

トランスフォーメーションの名前を変更するには、[名前の変更] をクリックし、トランスフォーメーションの分かりやすい名前を入力してから、[OK] をクリックします。

トランスフォーメーションのポート

トランスフォーメーションを作成した後は、ポートを定義します。ポートを作成し、ポートのプロパティを定義します。

トランスフォーメーションを作成する場合に、すべてのポートを手動で作成する必要はありません。例えば、ルックアップトランスフォーメーションを作成し、ルックアップテーブルを参照するとします。トランスフォーメーションポートを表示すると、トランスフォーメーションには、参照するテーブルの各カラムに対して出力ポートがあることがわかります。これらのポートを定義する必要はありません。

ポートの作成

トランスフォーメーションを作成する場合に、すべてのポートを手動で作成する必要はありません。例えば、ルックアップトランスフォーメーションを作成し、ルックアップテーブルを参照するとします。トランスフォーメーションポートを表示すると、トランスフォーメーションには、参照するテーブルの各カラムに対して出力ポートがあることがわかります。これらのポートを定義する必要はありません。

以下の方法でポートを作成します。

- **別のトランスフォーメーションからポートをドラッグする。**別のトランスフォーメーションからポートをドラッグすると、Designer は同じプロパティを持つポートを作成し、2 つのポートをリンクします。[レイアウト] - [カラムのコピー] をクリックして、ポートのコピーを有効化します。
- **[ポート] タブの [追加] ボタンをクリックする。**Designer は、設定可能な空のポートを作成します。

ポートの設定

トランスフォーメーションポートを定義する場合は、ポートのプロパティを定義します。ポートのプロパティには、ポート名、データ型、ポートタイプ、およびデフォルト値が含まれます。

次のポートのプロパティを設定します。

- **ポート名。**ポートの名前。ポートに名前を付けるときは、次の規則に従います。
 - 1 バイトまたは 2 バイトの文字か、1 バイトまたは 2 バイトのアンダスコア (_) で始めます。
 - 1 バイトまたは 2 バイトの文字のうち数値、アンダスコア (_)、\$、#、または @ はいずれも、ポート名に含めることが可能です。
- **データ型、精度、およびスケール。**式または条件を入力する場合は、データ型が式の戻り値に一致することを確認します。
- **ポートタイプ。**トランスフォーメーションには、入力、出力、入出力、および変数のポートの種類の組み合わせが含まれる場合があります。
- **デフォルト値。**NULL 値または出力トランスフォーメーションエラーを含むポートのデフォルト値を割り当てます。いくつかのポートのデフォルト値を上書きすることができます。
- **説明。**ポートの説明。
- **その他のプロパティ。**トランスフォーメーションによっては、式プロパティやグループ化プロパティなどのトランスフォーメーション固有のプロパティを持つものがあります。

ポートのリンク

トランスフォーメーションをマッピング内に追加して設定すると、ターゲットや他のトランスフォーメーションにリンクすることができます。ポートを介してマッピングオブジェクトをリンクすることができます。データは以下のポートを介してマッピングを通過します。

- **入力ポート。**データを受信します。

- **出力ポート。**データを渡します。
- **入出力ポート。**データを受信し、変更せずに渡します。

ポートをリンクするには、異なるマッピングオブジェクトのポート間でドラッグします。Designer はリンクを検証し、リンクが検証条件を満たした場合のみリンクを作成します。

複数グループのトランスフォーメーション

トランスフォーメーションには、複数の入力グループと出力グループを含めることができます。グループは、入力データまたは出力データの行を定義するポートのセットです。

グループは、リレーショナルソースやターゲット定義のテーブルと類似しています。ほとんどのトランスフォーメーションは、1つの入力グループおよび1つの出力グループを持っています。ただし、いくつかのトランスフォーメーションは、複数の入力グループ、複数の出力グループ、またはその両方を持っています。グループは、トランスフォーメーションに入力されるデータ、またはトランスフォーメーションから出力されるデータの行を表すものです。

複数グループのトランスフォーメーションはすべてアクティブなトランスフォーメーションです。複数のアクティブなトランスフォーメーションまたはアクティブなトランスフォーメーションとパッシブなトランスフォーメーションを同じ後続トランスフォーメーションまたはトランスフォーメーション入力グループに接続することはできません。

複数の入力グループを持つトランスフォーメーションの場合、Integration Service がある入力グループからの行を待つ間、Integration Service で別の入力グループの行をブロックする必要がある場合があります。ブロッキングトランスフォーメーションは、複数入力グループを持つトランスフォーメーションであり、入力データをブロックします。以下のトランスフォーメーションは、ブロッキングトランスフォーメーションです。

- **[入力ブロック]** プロパティが有効なカスタムトランスフォーメーション
- **未ソート入力**に対して設定されたジョイナトランスフォーメーション

マッピングを保存または検証する場合、アクティブなトランスフォーメーションまたはブロッキングトランスフォーメーションを含むマッピングが有効にならないことがあります。

式に関する作業

いくつかのトランスフォーメーションでは、式エディタを使用して式を入力できます。以下の関数を使用して、式を作成します。

- **トランスフォーメーション言語関数。** 一般的な式を扱うように設計された、SQL に似た関数
- **ユーザー定義関数。** トランスフォーメーション言語関数に基づいて PowerCenter で作成する関数です。
- **カスタム関数。** カスタム関数 API を使用して作成する関数です。

入力または入出力ポートから得たデータの値を使用するポートに、式を入力します。例えば、全従業員の給与が含まれる入力ポート IN_SALARY を持つトランスフォーメーションがあるとします。あとから、マッピングの IN_SALARY カラムの値と、このトランスフォーメーションで計算する給与総額および給与平均額を使用する必要がある場合もあります。そのため、Designer では各計算値について別の出力ポートを作成する必要があります。

以下の表に、式を入力できるトランスフォーメーションを示します。

トランスフォーメーション	式	戻り値
アグリゲータ	トランスフォーメーションを通過するすべてのデータに基づいて、集計を行います。また、集計するレコードに対してフィルタを指定して、特定のレコードを排除できます。例えば、このトランスフォーメーションを用いて、事務所の全従業員の総数と平均給与を算出できます。	ポートの集計結果。
データマスキング	行の入力ポートまたは出力ポートの値に基づいて計算を実行します。式は、データマスキングトランスフォーメーション内のプロダクションデータをマスキングするメソッドです。	入力ポートまたは出力ポートを使用した行レベルの計算結果。
式	単一の行内の値に基づいて計算を行います。たとえば、特定品目の価格や数量に基づいて、注文におけるその品目の購入価格合計値の算出ができます。	ポートの行レベルの計算結果。
フィルタ	このトランスフォーメーションを通過する行のフィルタリング条件を指定します。たとえば、未払い残高のある顧客についての BAD_DEBT テーブルに顧客データを書き込みたい場合には、フィルタトランスフォーメーションを用いて顧客データのフィルタリングができます。	TRUE または FALSE。これは行が指定条件を満足しているかどうかによります。TRUE を返す行だけがこのトランスフォーメーションから出力されます。このトランスフォーメーションは該当値を各取得行に適用します。
ランク	ランクに含める行の条件を設定します。たとえば、現在会社に所属している販売員の上位 10 名をランク付けできます。	ポートの条件適用結果または計算結果。
ルータ	グループ式に基づいて、複数のトランスフォーメーションにデータをルーティングします。たとえば、このトランスフォーメーションを使用して、3 つの異なる給与レベルに属する従業員の給与を比較します。これは、ルータトランスフォーメーションに 3 つのグループを作成することによって行うことができます。たとえば、各給与範囲に対して 1 つのグループ式を作成します。	TRUE または FALSE。これは行が指定されたグループ式を満足しているかどうかによります。TRUE を返す行のみが、このトランスフォーメーションの各ユーザー定義グループを通過します。FALSE を返す行は、デフォルトグループを通過します。

トランスフォーメーション	式	戻り値
アップデートストラテジ	行に更新、挿入、削除、またはリジェクトのフラグを設定します。一定の条件に基づいてターゲットの更新を管理する場合に、このトランスフォーメーションを使用します。たとえば、アップデートストラテジトランスフォーメーションを用いて、全顧客の行に対して、メールアドレスが変更された際に更新フラグを設定したり、全従業員の行に対して、社員でなくなった者についてはリジェクトフラグを設定したりできます。	更新、挿入、削除、またはリジェクトに対応する数値コード。このトランスフォーメーションは該当値を各取得行に適用します。
トランザクション制御	Integration Service が実行する操作（コミット、ロールバック、またはトランザクション変更なし）を決定する条件を指定します。トランスフォーメーションに渡される行または行のセットに基づくコミットおよびロールバックトランザクションを制御したい場合、このトランスフォーメーションを使用します。たとえば、このトランスフォーメーションを使用して、注文の入力日付に基づいて行のセットをコミットできます。	以下のいずれかのビルトイン変数。行が指定された条件に一致しているかどうかにより戻り値が決まります。 <ul style="list-style-type: none"> - TC_CONTINUE_TRANSACTION - TC_COMMIT_BEFORE - TC_COMMIT_AFTER - TC_ROLLBACK_BEFORE - TC_ROLLBACK_AFTER Integration Service は、戻り値に基づいてアクションを実行します。

式エディタの使用

式エディタは、SQL に似た文を作成するために使用します。式は手動で入力できますが、ポイントアンドクリック機能を使用することをお勧めします。ポイントアンドクリックインタフェースを使用して関数、ポート、変数、および演算子を選択することで、式を作成するときのエラーを減らすことができます。式に含めることができる文字は最大 32,767 文字です。

式トランスフォーメーションの式エディタで構成した式を評価できます。

式をテストする場合、サンプルデータを入力してから式を評価します。Transformation Designer で再利用可能なトランスフォーメーションの式を評価します。Mapping Designer で、再利用不可能なトランスフォーメーションの式を評価します。一部の関数では、ポートのデータ型を Binary または Date/Time に設定している場合、式の評価を実行できません。

式へのポート名の入力

接続されたトランスフォーメーションの式でポート名を使用している場合、トランスフォーメーション内のポート名を変更すると、Designer は式を更新します。たとえば、2 つの日付 Date_Promised と Date_Delivered の間の差を求める有効な式を書いたとします。後で Date_Promised ポート名を Due_Date に変更したとします。このとき、Designer は式内の Date_Promised ポート名を Due_Date に変更します。

注: 名前 Due_Date を、マッピングのこのポートに依存する他の再利用不可能なトランスフォーメーションにプロパゲートできます。

コメントの追加

コメントを式に追加して式に関する説明を記述したり、式に関連するビジネス文書にアクセスするための有効な URL を指定したりできます。

コメントは以下に示す方法を使って追加できます。

- 式にコメントを追加するには、コメントインジケータ「--」または「//」を使用します。
- ダイアログボックスにコメントを追加するには、[コメント] をクリックします。

式の検証

[検証] ボタンを使用して、式を検証してください。式の検証を行わない場合には、式エディタを閉じた時点で、Designer がその検証を行います。式が無効の場合、Designer は警告を表示します。無効な式を保存または変更することができます。無効な式を含むマッピングに対して、セッションを実行することはできません。

式エディタの表示

式エディタでは、可読性を高めるために構文の式が色分けされて表示されます。最新の Rich Edit コントロールである riched20.dll がシステムにインストールされている場合、式エディタは、式の関数を青色、コメントを灰色、および引用符付き文字列を緑色で表示します。

式エディタのサイズは変更が可能です。ダイアログボックスの境界線をドラッグすると、サイズが変更されます。Designer では、変更後のサイズをクライアント設定として保存します。

ポートへの式の追加

データマスキングトランスフォーメーションでは、式を入力ポートに追加できます。その他のすべてのトランスフォーメーションでは、式を出力ポートに追加します。

式をポートに追加するには、以下の手順を実行します。

1. トランスフォーメーションで、ポートを選択し、式エディタを開きます。
2. 式を入力します。
[関数] タブおよび [ポート] タブ、および演算子キーを使用します。
3. 式にコメントを追加します。
コメントインジケータの「--」または「//」を使用します。
4. 式を検証します。
[検証] ボタンを使用して、式を検証してください。

パラメータファイル内での式文字列の定義

Integration Service では式が渡された後、式内のマッピングパラメータおよび変数が展開されます。頻繁に変更される式がある場合は、パラメータファイル内で式文字列を定義しておけば、式が変更されたときに、変更のあった式を用いるマッピングを更新せずに済みます。

パラメータファイルで式文字列を定義するには、マッピングパラメータまたは変数を作成し、式文字列を格納して、パラメータファイル内の式文字列にパラメータまたは変数を設定します。作成されたパラメータまたは変数で、IsExprVar を true にセットする必要があります。IsExprVar が true の場合、パラメータまたは変数が展開された後で、式が解析されます。

たとえば、パラメータファイルで式 IIF(color='red',5) を定義するには、次の手順を実行します。

1. 式を使用するマッピングで、マッピングパラメータ \$\$Exp を作成します。IsExprVar を true に設定し、データタイプを String に設定します。
2. 式エディタで、式をマッピングパラメータの名前に設定します。
\$\$Exp
3. パラメータファイルを使用するようにセッションまたはワークフローを設定します。

4. パラメータファイルで、\$\$Exp の値を次の式文字列にセットします。

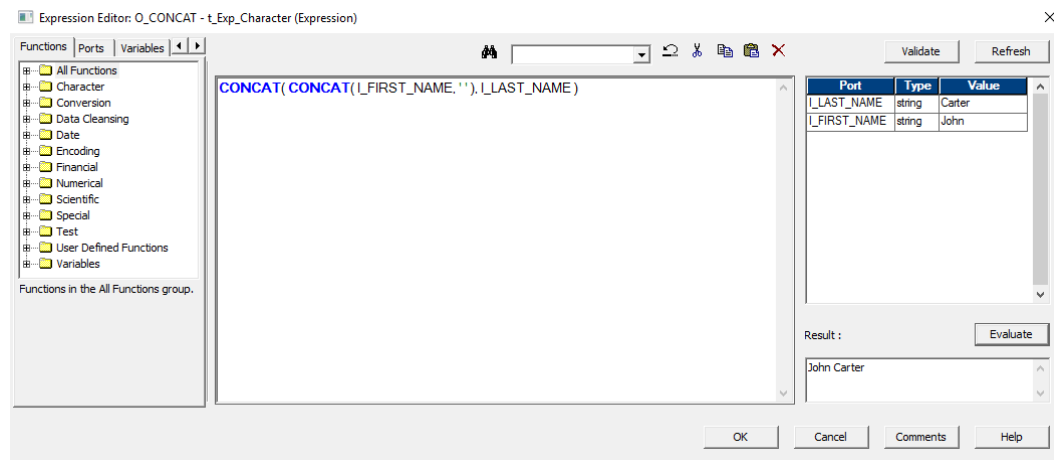
```
$$Exp=IIF(color='red',5)
```

式の評価

式トランスフォーメーションの式エディタで構成した式を評価できます。式をテストする場合、サンプルデータを入力してから式を評価します。

式の条件を編集する場合は、テストデータの評価する前に、[テスト] パネルで[更新] を選択する必要があります。有効な式を入力しないと、[テスト] パネルでポートを設定できません。システム定義のパラメータを指定する場合、入力する値はランタイム値ではありません。

次の図は、式トランスフォーメーションの式エディタ内でのサンプル式の評価を示しています。



注: Date/Time または Binary データ型の式を評価することはできません。

例

すべての結果をターゲットにロードする前に、各顧客の受注の合計数に基づいてプロモーションオファーを計算する必要があります。式トランスフォーメーション内で式を定義することでオファーを計算するためのマッピングを作成します。演算子がマッピングを実行する前に、式を評価して結果を検証します。

式の評価

式トランスフォーメーションの式エディタで式をテストおよび検証できます。

1. 式トランスフォーメーションの式エディタで、式を入力します。
2. 式を検証するには、[検証] をクリックします。
3. 右側のペインの [テスト式] セクションで、式の条件の最新の変更を反映するには、[更新] をクリックします。
4. 右側のペインに、式内で使用される入力ポートのサンプル値を入力します。
5. 式を評価するには、[評価] をクリックします。

式の評価の制限

式を評価するときに、特定の制限が適用されます。

式を評価する場合は、次の制限を考慮してください。

トランスフォーメーション制限

以下のトランスフォーメーションでは式を評価できません。

- アグリゲータ
- データマスキング
- フィルタ
- ランク
- ルータ
- Stored Procedure
- トランザクションコントロール
- アップデートストラテジ

データ型の制限

入力ポートタイプまたは戻り関数に Binary または Date/Time データ型が含まれている場合、式を評価できません。

ポートの制限

ルックアップポートまたはストアードプロシージャポートを使用する式は評価できません。

ローカル変数

パフォーマンスを向上させるには、アグリゲータトランスフォーメーション、式トランスフォーメーション、およびランクトランスフォーメーションでローカル変数を使用します。式中の変数を参照したり、変数を使ってデータの一時的格納ができます。

変数を使用して、以下の作業を実行する場合があります。

- データの一時的格納。
- 複雑な式の簡素化。
- 前の行からの値の格納。
- ストアドプロシージャからの複数の戻り値の取得。
- 値の比較。
- コネクต์されていないルックアップトランスフォーメーションの結果の格納。

データの一時的な格納と複雑な式の簡素化

同じトランスフォーメーションに複数の関連式を入力する際に、変数を使用するとパフォーマンスが向上します。トランスフォーメーションで同じ式のコンポーネントを複数回解析して検証する代わりに、コンポーネントを変数として定義できます。

例えば、アグリゲータトランスフォーメーションが合計と平均を計算する前に同じフィルタ条件を使用する場合、この条件を変数として定義してから両方の集計計算で再利用できます。

複雑な式を単純化できます。アグリゲータで複数の式に同じ計算が含まれる場合は、変数を作成して計算結果を格納することによってパフォーマンスを向上できます。

例えば、次の式を作成し、同じデータを使用して平均給与と給与合計を算出できます。

```
AVG( SALARY, ( ( JOB_STATUS = 'Full-time' ) AND (OFFICE_ID = 1000 ) ) )  
SUM( SALARY, ( ( JOB_STATUS = 'Full-time' ) AND (OFFICE_ID = 1000 ) ) )
```

両方の計算に同じ引数を入力する代わりに、この計算の各条件に変数ポートを作成してから変数を使用するように式を変更できます。

以下の表に、変数を使用して複雑な式を簡素化し、データを一時的に格納する方法を示します。

ポート	値
V_CONDITION1	JOB_STATUS = ヤ Full-time ャ
V_CONDITION2	OFFICE_ID = 1000
AVG_SALARY	AVG (SALARY, (V_CONDITION1 AND V_CONDITION2))
SUM_SALARY	SUM (SALARY, (V_CONDITION1 AND V_CONDITION2))

複数行の値の格納

トランスフォーメーション内で、ソース行からのデータを格納するように変数を設定できます。変数は、トランスフォーメーション式の中に使用できます。

例えば、ソースファイルに次の行が含まれるとします。

```
California  
California  
California  
Hawaii  
Hawaii  
New Mexico  
New Mexico  
New Mexico
```

各行には、州が含まれます。行数をカウントし、州ごとに行カウント数を返す必要があります。

```
California,3  
Hawaii      ,2  
New Mexico,3
```

この場合、アグリゲータトランスフォーメーションを設定して、ソース行が州別にグループ化され、各グループ内の行数がカウントされるようにすることができます。アグリゲータトランスフォーメーション内の変数は、行カウント数が格納されるように設定してください。別の変数を、前の行から州の名前が格納されるように定義します。

アグリゲータトランスフォーメーションは、次のポートを備えています。

ポート	ポートタイプ	式	説明
州	パススルー	n/a	州の名前。ソース行は、州の名前ごとにグループ化されています。Aggregator トランスフォーメーションによって、州ごとに 1 行が返されます。
State_Count	変数	IIF (PREVIOUS_STATE = STATE, STATE_COUNT +1, 1)	現在の州の行カウント。現在の [州] カラムの値が Previous_State カラムと同じ場合、State_Count が増分されます。それ以外の場合、State_Count が 1 にリセットされます。
Previous_State	変数	State	前の行の [州] カラムの値。行の処理時には、州の値が Previous_State に移動されます。
State_Counter	出力	State_Count	Aggregator トランスフォーメーションで、州ごとに処理される行数。Integration Service は州ごとに State_Counter を返します。

ストアドプロシージャからの値の取得

変数は、ストアドプロシージャから複数カラムの戻り値を取得する手段となります。

変数ポートの設定のガイドライン

トランスフォーメーション内に変数ポートを設定するときは、以下の項目について考慮する必要があります。

- **ポートの評価順。** Integration Service は依存関係によってポートを評価します。トランスフォーメーション内のポートの順序は、評価の順序と一致させる必要があります。入力ポート、変数ポート、出力ポートの順になります。
- **データタイプ。** 選択するデータタイプには、入力した式の戻り値が反映されます。
- **変数の初期化。** Integration Service は、カウンタを作成することが可能な変数ポートに初期値を設定します。

ポートの評価順

統合サービスは、最初に入力ポートを評価します。統合サービスは、次に変数ポートを評価し、最後に出力ポートを評価します。

Integration Service は以下の順序でポートを評価します。

1. **入力ポート。** Integration Service では最初に入力ポートがすべて評価されます。これは、入力ポートが他のポートに影響されないという理由からです。したがって、入力ポートは任意の順序で作成できます。入力ポートは他のポートを参照しないので、統合サービスは入力ポートを順序付けしません。
2. **変数ポート。** 変数ポートは入力ポートと変数ポートを参照できますが、出力ポートの参照はできません。変数ポートは入力ポートの参照ができるので、Integration Service は入力ポートの次に変数ポートの評価を行います。変数は他の変数を参照できるので、変数ポートの表示順は、統合サービスがそれぞれの変数进行评估する順序と同じです。

たとえば、建物の取得価格を計算し、ついで原価償却による調整を行う場合には、取得価格計算を変数ポートとして作成できます。この変数は、原価償却による調整ポートの前に配置される必要があります。

3. **出力ポート**。出力ポートは入力ポートと変数ポートを参照できるので、統合サービスは出力ポートを最後に評価します。出力ポートは他の出力ポートを参照できないので、出力ポートの表示順は関係ありません。出力ポートは、必ずポートリストの最下部に表示してください。

データ型

ポートを変数として設定すると、任意の式または条件を入力できます。このポートに対して選択するデータ型は、入力する式の戻り値に反映されます。変数ポートを使用して条件を指定すると、数値データ型は TRUE（非ゼロ）値と FALSE（ゼロ）値を返します。

変数の初期化

統合サービスは、変数の初期値を NULL に設定しません。

統合サービスは、次のガイドラインを使用して変数の初期値を設定します。

- 数値ポートについてはゼロ
- 文字列ポートについては、空の文字列
- Date/Time ポートについては 01/01/0001

そのため、初期値を必要とするカウンタとして変数を使用します。たとえば、以下の式では数値変数が作成できます。

VAR1 + 1

この式では、VAR1 ポートで行数をカウントします。変数の初期値を NULL に設定した場合には、この式は必ず NULL と評価されることになります。そのため、初期値はゼロに設定してあります。

ポートのデフォルト値

すべてのトランスフォーメーションは、統合サービスが入力 NULL 値と出力トランスフォーメーションエラーを処理する方法を決定するデフォルト値を使用します。

入力、出力、および入出力の各ポートには、必要に応じてユーザー定義デフォルト値によってオーバーライドできるシステムデフォルト値が指定されています。デフォルト値は、ポートの種類によって機能が異なります。

- **入力ポート**。NULL 入力ポートのシステムデフォルト値は NULL です。トランスフォーメーションでは、デフォルト値は空白として表示されます。入力値が NULL である場合、Integration Service では NULL のままになります。
- **出力ポート**。出力トランスフォーメーションエラーのシステムデフォルト値は、ERROR です。トランスフォーメーションでは、デフォルト値は ERROR('transformation error')として表示されます。トランスフォーメーションエラーが発生すると、Integration Service ではその行がスキップされます。統合サービスは、ERROR 関数がスキップするすべての入力行をログファイルに記録します。

次に、トランスフォーメーションエラーを示します。

- データ変換エラー（たとえば、日付関数に数値を渡した場合など）。
- ゼロ除算などの式評価エラー。
- ERROR 関数の呼び出し。

- **パススルーポート** NULL 入力 of システムデフォルト値は入力ポート同様 NULL です。トランスフォーメーションでは、システムのデフォルト値は空白として表示されます。出力トランスフォーメーションエラーのデフォルト値は、出力ポートと同じです。出力トランスフォーメーションエラーのデフォルト値はトランスフォーメーションに表示されません。

注: Java トランスフォーメーションは、Java トランスフォーメーションのポートタイプに基づいて、PowerCenter® のデータ型を Java データ型に変換します。NULL 入力のデフォルト値は、Java データタイプによって異なります。

以下の表に、接続されたトランスフォーメーションのポートのシステムデフォルト値を示します。

ポートタイプ	デフォルト値	統合サービスの動作	サポートされているユーザー定義のデフォルト値
入力、パススルー	NULL	統合サービスは、すべての入力 NULL 値を NULL として渡します。	入力、入出力
出力、パススルー	ERROR	統合サービスは、出力ポートでのトランスフォーメーションのエラーに対して ERROR 関数呼び出しします。統合サービスは、エラーが発生した行をスキップし、入力データとエラーメッセージをログファイルに書き込みます。	アウトプット

変数ポートはデフォルト値をサポートしていません。変数ポートはそのデータタイプに応じて Integration Service で初期化されます。

一部のデフォルト値を上書きして、NULL の入力値および出力トランスフォーメーションのエラーに対する Integration Service の動作を変更できます。

ユーザー定義のデフォルト値

接続されているトランスフォーメーションでサポートされている入力、パススルー、および出力の各ポートのシステムデフォルト値は、ユーザー定義のデフォルト値でオーバーライドできます。

ポートのシステムデフォルト値をオーバーライドする場合は、次のルールとガイドラインを使用します。

- **入力ポート。** 統合サービスで NULL 値を NULL として扱わないようにする場合は、入力ポートにユーザー定義のデフォルト値を入力します。NULL が入力ポートに渡されると、統合サービスでは NULL をデフォルト値に置き換えます。
- **出力ポート。** 統合サービスで行をスキップしない場合、またはスキップされた行とともに特定のメッセージをログに書き込む場合は、出力ポートにユーザー定義のデフォルト値を入力します。出力ポートにデフォルト値を定義する場合、統合サービスでは、出力ポートにトランスフォーメーションエラーが発生していれば、その行をデフォルト値に置き換えます。
- **パススルーポート。** 統合サービスで NULL 値を NULL として扱わない場合は、パススルーポートにユーザー定義のデフォルト値を入力します。出力トランスフォーメーションエラーのユーザー定義のデフォルト値をパススルーポートに入力することはできません。

注: 接続されていないトランスフォーメーションのユーザー定義のデフォルト値は、Integration Service によって無視されます。例えば、式を使用してルックアップトランスフォーメーションまたはストアドプロシージャトランスフォーメーションを呼び出す場合、統合サービスはすべてのユーザー定義のデフォルト値を無視してシステムデフォルト値を適用します。

ユーザー定義デフォルト値を入力する場合、以下のオプションを使用します。

- **定数値。** NULL を含めて、任意の定数（数値またはテキスト）が使用できます。

- **定数式**。定数パラメータをもつトランスフォーメーション関数を含めることができます。
- **ERROR**。トランスフォーメーションエラーを生成します。セッションログまたは行エラーログに行とメッセージを書き込みます。
- **ERROR**。トランスフォーメーションエラーを生成します。マッピングログまたは行エラーログに行とメッセージを書き込みます。
- **ABORT**。セッションを強制終了します。
- **ABORT**。マッピングを強制終了します。

定数値

デフォルト値には、任意の定数値を入力できます。定数値はポートのデータ型と一致する必要があります。

たとえば、数値ポートのデフォルト値は数値定数でなければなりません。定数値には、以下のものがあります。

```
0
9999
NULL
'Unknown Value'
'Null input data'
```

定数式

定数式とは、定数式を記述するトランスフォーメーション関数（集計関数以外）を使う式です。定数式には、入力、入出力、または変数ポートからの値は使用できません。

有効な定数式には以下のものがあります。

```
500 * 1.75
TO_DATE('January 1, 1998, 12:05 AM', 'MONTH DD, YYYY, HH:MI AM')
ERROR('Null not allowed')
ABORT('Null not allowed')
SESSSTARTTIME
```

統合サービスはセッションの初期化の際にマッピング全体にデフォルト値を割り当てるので、式内でポートからの値を使用することはできません。

統合サービスはマッピングの初期化の際にマッピング全体にデフォルト値を割り当てるので、式内でポートからの値を使用することはできません。

次の例は、ポートからの値を使用するので無効です。

```
AVG(IN_SALARY)
IN_PRICE * IN_QUANTITY
:LKP(LKP_DATES, DATE_SHIPPED)
```

注: デフォルト値の式からストアードプロシージャまたはルックアップテーブルを呼び出すことはできません。

ERROR 関数と ABORT 関数

ERROR 関数および ABORT 関数は、入力ポートおよび出力ポートのデフォルト値、および入出力ポートの入力値で使います。統合サービスは、ERROR 関数に遭遇するとその行をスキップします。ABORT 関数に遭遇したときは、セッションを強制終了します。

ERROR 関数および ABORT 関数は、入力ポートおよび出力ポートのデフォルト値、および入出力ポートの入力値で使います。統合サービスは、ERROR 関数に遭遇するとその行をスキップします。ABORT 関数に遭遇すると、マッピングを強制終了します。

ユーザー定義のデフォルト入力値

統合サービスが NULL 値を NULL として扱わないようにするには、ユーザー定義のデフォルト入力値を入力します。

NULL 値をオーバーライドするには、次のいずれかのタスクを完了します。

- NULL 値を定数値または定数式によって置き換える。
- ERROR 関数によって、NULL 値をスキップする。
- ABORT 関数によって、マッピングを強制終了する。
- ABORT 関数によって、セッションを強制終了する。

以下の表に、Integration Service が入力および入出力ポートの NULL 入力を処理する方法をまとめます。

デフォルト値	デフォルト値のタイプ	説明
NULL (空白を表示)	System	統合サービスは NULL を渡します。
定数または定数式	ユーザー定義	統合サービスは、NULL 値を定数または定数式の値に置き換えます。
エラー (ERROR)	ユーザー定義	統合サービスは、これをトランスフォーメーションエラーとして処理します。 <ul style="list-style-type: none">- トランスフォーメーションエラーカウントを 1 つ増やします。- その行をスキップし、ログファイルまたは行エラーログにエラーメッセージを書き込みます。 統合サービスはその行を拒否ファイルに書き込みません。
ABORT	ユーザー定義	統合サービスが NULL 入力値を検出すると、セッションが強制終了します。統合サービスはエラーカウントを増やさず、行を拒否ファイルに書き込みません。 統合サービスが NULL 入力値を検出すると、マッピングが強制終了します。統合サービスはエラーカウントを増やさず、行を拒否ファイルに書き込みません。

NULL 値の置換

定数値または定数式を使用して、ポートの NULL 値を指定した値で置き換えます。

例えば、入力文字列ポートの名前が DEPT_NAME で、NULL 値を文字列「UNKNOWN DEPT」で置き換える場合は、デフォルト値を「UNKNOWN DEPT」に設定します。トランスフォーメーションタイプに応じて、Integration Service は「UNKNOWN DEPT」をトランスフォーメーション内の式か変数、またはデータフロー内の次のトランスフォーメーションに渡します。

例えば、Integration Service ではポート内の NULL 値がすべて、文字列「UNKNOWN DEPT」に置き換えられます。

DEPT_NAME	REPLACED VALUE
Housewares	Housewares
NULL	UNKNOWN DEPT
Produce	Produce

NULL レコードのスキップ

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には、ERROR 関数をデフォルト値として使用します。たとえば、DEPT_NAME の入力値が NULL の場合に、行をスキップするとします。以下の式をデフォルト値として使用することができます。

```
ERROR('Error. DEPT is NULL')
```

ERROR 関数をデフォルト値として使用すると、Integration Service では NULL 値を含む行がスキップされます。統合サービスは、ERROR 関数がスキップしたすべての行をログファイルに書き込みます。これらの行は、拒否ファイルには書き込まれません。

DEPT_NAME	RETURN VALUE
Housewares	Housewares
NULL	'Error. DEPT is NULL' (Row is skipped)
Produce	Produce

次のログに、統合サービスが NULL 値を含む行をスキップする場所を示します。

```
TE_11019 Port [DEPT_NAME]: Default value is: ERROR(<<Transformation Error>> [error]: Error. DEPT is NULL
... error('Error. DEPT is NULL')
).
CMN_1053 EXPTRANS: : ERROR: NULL input column DEPT_NAME: Current Input data:
CMN_1053 Input row from SRCTRANS: Rowdata: ( RowType=4 Src Rowid=2 Targ Rowid=2
  DEPT_ID (DEPT_ID:Int:): "2"
  DEPT_NAME (DEPT_NAME:Char.25:): "NULL"
  MANAGER_ID (MANAGER_ID:Int:): "1"
)
```

セッションの強制終了

統合サービスが NULL 入力値に遭遇した場合にセッションを強制終了するには、ABORT 関数を使用します。

ユーザー定義のデフォルト出力値

出力ポートのシステムデフォルト値をオーバーライドする場合は、ユーザー定義デフォルト値を作成できます。

統合サービスがエラーを含む行をスキップしないようにする場合、または統合サービスがログにスキップされた行とともに特定のメッセージを書き込むようにする場合は、出力ポートにユーザー定義デフォルト値を入力できます。統合サービスで出力トランスフォーメーションエラーが発生した場合は、デフォルト値を入力して次の操作を実行します。

- エラーを定数値または定数式によって置き換える。Integration Service は行をスキップしません。
- ABORT 関数によって、セッションを強制終了する。
- ABORT 関数によって、マッピングを強制終了する。
- トランスフォーメーションエラーに対する特定のメッセージをログに書き込む。

入出力ポートには、ユーザー定義デフォルト出力値を入力できません。

以下の表に、Integration Service による出力ポートトランスフォーメーションエラーの処理方法と、トランスフォーメーションのデフォルト値を示します。

デフォルト値	デフォルト値のタイプ	説明
トランスフォーメーションエラー	System	デフォルト値を上書きしていない場合にトランスフォーメーションエラーが発生すると、統合サービスは次のタスクを実行します。 <ul style="list-style-type: none"> - トランスフォーメーションエラーカウントを 1 つ増やします。 - その行をスキップし、セッション設定に従って、セッションログファイルまたは行エラーログにエラーおよび入力行を書き込みます。 統合サービスはその行を拒否ファイルに書き込みません。
定数または定数式	ユーザー定義	統合サービスは、エラーをデフォルト値で置き換えます。 統合サービスは、エラーカウントを増やさず、ログにメッセージを書き込みません。
ABORT	ユーザー定義	セッションは強制終了し、統合サービスはメッセージをセッションログに書き込みます。 マッピングは強制終了し、統合サービスはメッセージをログに書き込みます。 統合サービスはエラーカウントを増やさず、行を拒否ファイルに書き込みません。

エラーの置換

トランスフォーメーションエラーが発生した場合に統合サービスが行をスキップしないように設定するには、出力ポートのデフォルト値として定数または定数式を使用します。

例えば、NET_SALARY という数値出力ポートがあり、トランスフォーメーションエラーが発生した場合に定数値「9999」を使用する場合、NET_SALARY ポートにデフォルト値 9999 を割り当てます。NET_SALARY の値の計算中にゼロ除算などのトランスフォーメーションエラーが発生した場合、統合サービスはデフォルト値 9999 を使用します。

セッションの強制終了

トランスフォーメーションエラーを容認したくない場合は、出力ポートのデフォルト値として ABORT 関数を使用します。

セッションログまたは行エラーログへのメッセージの書き込み

統合サービスがスキップした行と共に特定のメッセージをセッションログに書き込むようにする場合は、出力ポートにユーザー定義デフォルト値を入力することができます。システムのデフォルト値は ERROR ('transformation error') なので、統合サービスは、スキップされた行と共に「transformation error」というメッセージをセッションログに書き込みます。別のメッセージを書き込む場合は、'transformation error'を置き換えることができます。

行エラーのログを有効にすると、統合サービスはセッションログではなくエラーログにエラーメッセージを書き込みます。統合サービスでのトランザクション制御トランスフォーメーションのロールバックエラーまたはコミットエラーは記録されません。行エラーログのほかに、行自体もセッションログに記録する場合は、冗長データのトレースを有効にします。

出力ポート式の ERROR 関数

ERROR 関数を使用する式を入力すると、出力ポートのユーザー定義デフォルト値によって式の ERROR 関数が上書きされる場合があります。

例えば、エラーが発生したときに Integration Service が 'Negative Sale' という値を使用するように指示する以下の式を入力するとします。

```
IIF( TOTAL_SALES>0, TOTAL_SALES, ERROR ('Negative Sale'))
```

以下の例は、ユーザー定義デフォルト値が式の ERROR 関数をどのように上書きするのかを示しています。

- **定数値または定数式。**定数値または定数式は、出力ポート式の ERROR 関数を上書きします。

例えば、「0」をデフォルト値として入力すると、Integration Service は出力ポート式の ERROR 関数を上書きします。エラーが発生すると、値 0 が渡されます。行をスキップしたり、ログに「Negative Sale」と書き込むことは行いません。

- **ABORT.**ABORT 関数は、出力ポート式の ERROR 関数を上書きします。

ABORT 関数をデフォルト値として使用すると、統合サービスはトランスフォーメーションエラーの発生時にセッションを強制終了します。ABORT 関数は、出力ポート式の ERROR 関数を上書きします。

- **ERROR.**ERROR 関数をデフォルト値として使用すると、統合サービスは次の情報をログに記録します。

- デフォルト値からのエラーメッセージ
- 出力ポート式の ERROR 関数に示されるエラーメッセージ
- スキップされた行

たとえば、以下の ERROR 関数でデフォルト値を上書きすることができます。

```
ERROR('No default value')
```

Integration Service は行をスキップし、両方のエラーメッセージをログに書き込みます。

```
TE_7007 Transformation Evaluation Error; current row skipped...
TE_7007 [<<Transformation Error>> [error]: Negative Sale
... error('Negative Sale')
]
```

```
Sun Sep 20 13:57:28 1998
```

```
TE_11019 Port [OUT_SALES]: Default value is: ERROR(<<Transformation Error>> [error]: No default value
... error('No default value')
```

デフォルト値の一般ルール

デフォルト値を作成する場合には、次の規則およびガイドラインに従ってください。

- デフォルト値は、NULL、定数値、定数式、ERROR 関数、または ABORT 関数のいずれかでなければなりません。
- 入出力ポートの場合、Integration Service はデフォルト値を使用して NULL 入力値を処理します。入出力ポートの出力デフォルト値は常に ERROR (Transformation Error) です。
- 変数ポートはデフォルト値を使用しません。
- アグリゲータトランスフォーメーションおよびリンクトランスフォーメーションでは、Group By ポートにデフォルト値を指定できます。
- トランスフォーメーションのすべてのポートの種類について、ユーザー定義のデフォルト値を指定できるとは限りません。ポートがユーザー定義デフォルト値を使用できない場合は、デフォルト値フィールドは無効になります。
- すべてのトランスフォーメーションでユーザー定義デフォルト値を使用できるわけではありません。
- トランスフォーメーションがマッピングデータフローに接続されていない場合、Integration Service はユーザー定義値を無視します。

- 入力ポートのいずれかが未接続の場合、Integration Service は値が NULL であると仮定し、この入力ポートについてはデフォルト値を使用します。
- 入力ポートのデフォルト値に ABORT 関数が含まれ、また入力値が NULL の場合には、Integration Service はセッションを直ちに停止します。ABORT 関数をデフォルト値として使用することにより、NULL 入力値の制限ができます。入力ポートで最初に NULL 値が見つかった時点で、セッションは終了します。
- 出力ポートのデフォルト値に ABORT 関数が含まれていて、このポートでトランスフォーメーションエラーが発生すると、セッションは直ちに停止します。トランスフォーメーションエラーに対して厳密なルールを適用する場合、ABORT 関数をデフォルト値として使用します。このポートにトランスフォーメーションエラーが初めて発生した時点で、セッションは終了します。
- 入力ポートのデフォルト値に ABORT 関数が含まれ、入力値が NULL の場合、統合サービスはマッピングを直ちに停止します。ABORT 関数をデフォルト値として使用することにより、NULL 入力値の制限ができます。入力ポートで最初に NULL 値が見つかった時点でマッピングは終了します。
- 出力ポートのデフォルト値に ABORT 関数が含まれ、そのポートでトランスフォーメーションエラーが発生すると、マッピングは直ちに停止します。トランスフォーメーションエラーに対して厳密なルールを適用する場合、ABORT 関数をデフォルト値として使用します。このポートで最初のトランスフォーメーションエラーが発生した時点でマッピングは終了します。
- ABORT 関数、定数値、および定数式は、出力ポート式に設定されている ERROR 関数をオーバーライドします。

デフォルト値の検証

入力したデフォルト値が有効かどうかを検証することができます。Designer には、デフォルト値が有効であるかを確認するための「検証」ボタンがあります。メッセージが表示されて、デフォルト値が有効であるかどうかを示します。

Designer では、マッピングの保存時に、デフォルト値の検証も行われます。無効なデフォルト値を入力すると、Designer はマッピングを無効としてマークします。

Developer ツールは、入力時にデフォルト値を検証します。

Developer ツールは、マッピングの保存時にデフォルト値を検証します。無効なデフォルト値を入力すると、Developer ツールはマッピングを無効としてマークします。

トランスフォーメーションのトレースレベルの設定

トランスフォーメーションを設定する場合、Integration Service がセッションログに書き込む詳細レベルを指定できます。

以下の表に、セッションログのトレースレベルを示します。

トレースレベル	説明
ノーマル	Integration Service は、初期化情報、ステータス情報、発生したエラー、およびトランスフォーメーション行エラーが原因でスキップされた行をログに記録します。セッションの結果のまとめを行います。個別行のレベルでのまとめは行いません。
Terse	Integration Service は、初期化情報、エラーメッセージ、および拒否されたデータの通知をログに記録します。

トレースレベル	説明
Verbose Initialization	Normal トレースの内容に加え、Integration Service は初期化の詳細、インデックスおよび使用されたデータファイルの名前、および詳細なトランスフォーメーション統計もログに記録します。
Verbose Data	Integration Service は、Verbose Initialization トレースの内容に加え、マッピングに渡された各行をログに記録します。また、Integration Service はカラムの精度に合わせて文字列データを切り詰めた場所を示し、詳細なトランスフォーメーション統計も記録します。 行エラーのログを有効にした場合、Integration Service はセッションログおよびエラーログの両方にエラーを書き込みます。 トレースレベルを Verbose Data に設定した場合、Integration Service はトランスフォーメーションを処理するときにブロック中のすべての行の行データを書き込みます。

デフォルトでは、すべてのトランスフォーメーションのトレースレベルは Normal です。正常に動作しないトランスフォーメーションをデバッグする場合にだけ、トレースレベルを Verbose 設定に変更します。パフォーマンスを若干高めたい場合は、トレースレベルを Terse に設定し、トランスフォーメーションを含むワークフローの実行時にセッションログに書き込む詳細項目を最小限にできます。

セッションの設定時、すべてのトランスフォーメーションに適用される 1 つのトレースレベルを設定して、個々のトランスフォーメーションのトレースレベルを上書きすることができます。

再利用可能なトランスフォーメーション

マッピングには、再利用可能なトランスフォーメーションと再利用不可能なトランスフォーメーションを含めることができます。再利用不可能なトランスフォーメーションは、単一のマッピングで使用されます。再利用可能なトランスフォーメーションは複数のマッピングで使用できます。

たとえば、カナダでの販売付加価値税を計算する式トランスフォーメーションを作成すれば、カナダにおける事業コストの分析に役立ちます。いつも同じ作業を行うのであれば、再利用可能なトランスフォーメーションを作成できます。当該のトランスフォーメーションをマッピングに組み込む必要がある場合には、そのインスタンスをマッピングに追加してください。あとからトランスフォーメーションの定義を変更した場合には、そのインスタンスはすべてこの変更を受け継ぎます。

Designer は再利用可能なトランスフォーメーションを、それを使用するマッピングとは別のメタデータとして格納します。ナビゲータでフォルダの内容を参照すると、このフォルダ内のすべての再利用可能なトランスフォーメーションのリストが表示されます。

それぞれの再利用可能なトランスフォーメーションは、Designer で使用可能なトランスフォーメーションのカテゴリに対応しています。たとえば、再利用可能なアグリゲータトランスフォーメーションを作成して複数のマッピングで同一の集計を行ったり、また再利用可能なストアードプロシージャトランスフォーメーションを作成して複数のマッピングで同一のストアードプロシージャを呼び出すことができます。

ほとんどのトランスフォーメーションを再利用不可能または再利用可能として作成できます。ただし、エクスターナルプロシージャトランスフォーメーションは再利用可能なトランスフォーメーションとしてのみ作成できます。

再利用可能なトランスフォーメーションのインスタンスをマッピングに追加する場合には、トランスフォーメーションを変更してマッピングを無効にしたり、予期しないデータが生成されたりしないように注意してください。

インスタンスと継承される変更

再利用可能なトランスフォーメーションをマッピングに追加する場合には、このトランスフォーメーションのインスタンスを追加してください。トランスフォーメーションの定義はまだマッピング外にあるのに対して、トランスフォーメーションのインスタンスはマッピング内に表示されます。

再利用可能なトランスフォーメーションのインスタンスはそのトランスフォーメーションへのポインタであるため、Transformation Developer でトランスフォーメーションを変更すると、そのインスタンスには変更が反映されます。同じトランスフォーメーションを使用するマッピングでそれぞれトランスフォーメーションを更新する代わりに、再利用可能なトランスフォーメーションを一度更新すると、そのトランスフォーメーションのインスタンスすべてに変更が反映されます。インスタンスはプロパティ設定項目の変更は継承せず、ポート、式、およびトランスフォーメーション名の変更だけを継承することに注意してください。

式のマッピング変数

再利用可能なトランスフォーメーション式で、マッピングパラメータおよび変数を使用します。Designer はパラメータまたは変数を検証する際に、Integer データタイプとして扱います。マッピングまたはマプレットでトランスフォーメーションを使用すると、Designer は式を再度検証します。マッピングパラメータまたは変数がマプレットまたはマッピングに存在しない場合、Designer はエラーを記録します。

再利用可能なトランスフォーメーションの作成

以下の方法によって、再利用可能なトランスフォーメーションを作成することができます。

- **Transformation Developer で設計する。**Transformation Developer で、新しい再利用可能なトランスフォーメーションを構築できます。
- **再利用不可能なトランスフォーメーションを Mapping Designer から格上げする。**マッピングにトランスフォーメーションを追加した後、再利用可能なトランスフォーメーションのステータスに格上げすることができます。マッピング内に設計されたトランスフォーメーションは、リポジトリ内のどこかで管理される再利用可能なトランスフォーメーションのインスタンスとなります。

トランスフォーメーションを再利用可能なトランスフォーメーションに格上げした場合、再利用可能なトランスフォーメーションを格下げすることはできません。ただし、そのトランスフォーメーションの再利用不可能なインスタンスを作成することはできます。

注: マプレット内のシーケンスジェネレータトランスフォーメーションは再利用可能でなければなりません。マプレット内のシーケンスジェネレータトランスフォーメーションを再利用不可能なトランスフォーメーションに格下げすることはできません。

再利用可能なトランスフォーメーションを作成するには：

1. Designer で、Transformation Developer に切り替えます。
2. トランスフォーメーションツールバー上で、作成したいトランスフォーメーションに対応するボタンをクリックします。
3. ワークブック内でドラッグし、トランスフォーメーションを作成します。
4. トランスフォーメーションのタイトルバーをダブルクリックし、そのプロパティを表示するダイアログを開きます。
5. [名前の変更] をクリックし、トランスフォーメーションの分かりやすい名前を入力してから、[OK] をクリックします。
6. [ポート] タブをクリックしてから、このトランスフォーメーションに必要な入力ポートと出力ポートを追加します。

7. トランスフォーメーションの他のプロパティの設定を行い、[OK] をクリックします。

このプロパティは作成するトランスフォーメーションに応じて異なります。たとえば、式トランスフォーメーションを作成する場合には、このトランスフォーメーションの1つ以上の出力ポートについて式を入力する必要があります。ストアードプロシージャトランスフォーメーションを作成する場合には、呼び出すストアードプロシージャを指定する必要があります。

再利用不可能なトランスフォーメーションの格上げ

再利用可能なトランスフォーメーションの作成は、既存のトランスフォーメーションをマッピング内で格上げすることによっても行えます。[トランスフォーメーションの編集] ダイアログボックスの[再利用可能にする] オプションにチェックマークを付けると、Designer は指示に従ってトランスフォーメーションの格上げを行い、マッピング内にそのインスタンスを作成します。

再利用不可能なトランスフォーメーションを格上げするには：

1. Designer でマッピングを開き、格上げしたいトランスフォーメーションのタイトルバーをダブルクリックします。
2. [再利用可能にする] オプションを選択します。
3. トランスフォーメーションの格上げを行うかどうか聞かれたら、[はい] をクリックします。
4. [OK] をクリックして、マッピングに戻ります。

作業中のフォルダの再利用可能なトランスフォーメーションリストを見ると、新たに格上げたトランスフォーメーションがこのリストに表示されているのが分かります。

再利用可能なトランスフォーメーションの再利用不可能なインスタンスの作成

マッピング内の再利用可能なトランスフォーメーションの再利用不可能なインスタンスを作成することができません。再利用可能なトランスフォーメーションを再利用不可能にするにあたっては、同じフォルダ内で行う必要があります。別のフォルダに再利用可能なトランスフォーメーションの再利用不可能なインスタンスを作成したい場合は、まず、ソースフォルダ内でトランスフォーメーションの再利用不可能なインスタンスを作成してから、それをターゲットフォルダにコピーします。

再利用可能なトランスフォーメーションの再利用不可能なインスタンスを作成するには：

1. Designer でマッピングを開きます。
2. ナビゲータで既存のトランスフォーメーションを選択し、トランスフォーメーションをマッピングワークスペースにドラッグします。トランスフォーメーションを離すまで、Ctrl キーを押し続けてください。
ステータスバーには以下のメッセージが表示されます。
Make a non-reusable copy of this transformation and add it to this mapping.
3. トランスフォーメーションを離します。
Designer は、既存の再利用可能なトランスフォーメーションの再利用不可能なインスタンスを作成します。

マッピングへの再利用可能なトランスフォーメーションの追加

作成した再利用可能なトランスフォーメーションをマッピングに追加できます。

再利用可能なトランスフォーメーションを追加するには：

1. Designer を Mapping Designer モードに切り替えます。
2. マッピングを開くか作成します。

3. リポジトリオブジェクトのリストで、フォルダの Transformations 部に対象の再利用可能なトランスフォーメーションが見つかるまでドリルダウンします。
4. ナビゲータからマッピングにトランスフォーメーションをドラッグします。
再利用可能なトランスフォーメーションのコピー（インスタンス）が表示されます。
5. 新しいトランスフォーメーションを他のトランスフォーメーションまたはターゲット定義にリンクします。

再利用可能なトランスフォーメーションの変更

Transformation Developer で入力した再利用可能なトランスフォーメーションの変更を行うと、そのすべてのインスタンスに変更が直ちに反映されます。この機能は、作業を保存し、標準的な形式を維持するには効果的な方法ですが、再利用可能なトランスフォーメーションを変更すると、マッピングが無効になる可能性があります。

トランスフォーメーションに加えた変更によって影響を受けるマッピング、マプレット、またはショートカットを確認するには、ワークスペースまたはナビゲータ内でトランスフォーメーションを選択して右クリックし、[依存関係を表示] を選択します。

再利用可能なトランスフォーメーションに以下のいずれかの変更を行うと、そのインスタンスを用いるマッピングが無効となる場合があります。

- トランスフォーメーションの 1 つ以上のポートを削除すると、マッピングの一部またはすべてのデータフローから当該インスタンスが切り離されます。
- ポートのデータタイプを変更すると、一致しないデータタイプを用いてそのポートから別のポートへマッピングすることが不可能となります。
- ポート名を変更すると、そのポートを参照する式は無効となります。
- 再利用可能なトランスフォーメーションに無効な式を入力すると、このトランスフォーメーションを使用するマッピングは無効となります。Integration Service では無効なマッピングに基づいたセッションは実行できません。

再利用可能なトランスフォーメーションの元の状態への復元

マッピング内の再利用可能なトランスフォーメーションのプロパティを変更した場合、[元に戻す] ボタンをクリックすれば再利用可能なトランスフォーメーションのプロパティを元の状態に戻すことができます。

第 2 章

アグリゲータトランスフォーメーション

この章では、以下の項目について説明します。

- [アグリゲータトランスフォーメーションの概要, 51 ページ](#)
- [アグリゲータトランスフォーメーションのコンポーネント, 52 ページ](#)
- [集計キャッシュの設定, 53 ページ](#)
- [集計式, 54 ページ](#)
- [Group By ポート, 55 ページ](#)
- [ソート済み入力の使用, 57 ページ](#)
- [アグリゲータトランスフォーメーションの作成, 59 ページ](#)
- [アグリゲータトランスフォーメーションに関するヒント, 59 ページ](#)
- [アグリゲータトランスフォーメーションのトラブルシューティング, 60 ページ](#)

アグリゲータトランスフォーメーションの概要

アグリゲータトランスフォーメーションは、平均値や合計値などの集計計算を実行します。統合サービスは、データグループおよび行データを読み取って集計キャッシュに保存する際に、集計計算を実行します。アグリゲータトランスフォーメーションはアクティブなトランスフォーメーションです。

アグリゲータトランスフォーメーションは、式トランスフォーメーションとは異なり、グループに対して計算を実行するために使用します。式トランスフォーメーションでは、行単位の計算が可能です。

トランスフォーメーション言語を使用して集計式を作成する場合、条件句を使用して行をフィルタリングすることで、SQL 言語より優れた柔軟性を実現できます。

アグリゲータトランスフォーメーションを含むセッションを作成すれば、セッションの差分集計オプションが使用できます。統合サービスでの差分集計実行時には、マッピングを通してソースデータが渡され、履歴キャッシュデータを使用して差分を反映した形で集計計算が実行されます。

アグリゲータトランスフォーメーションのコンポーネント

アグリゲータはアクティブなトランスフォーメーションであり、パイプライン内の行数を変更します。アグリゲータトランスフォーメーションには、以下のコンポーネントとオプションがあります。

- **集計キャッシュ。**Integration Service は、集計計算を完了するまでデータを集計キャッシュに格納します。Integration Service によってインデックスキャッシュにグループ値が格納され、データキャッシュに行データが格納されます。
- **集計式。**出力ポートに式を入力します。式には非集計式や条件句を含めることができます。
- **Group By ポート。**グループの作成方法を示します。グループには、入力ポート、入出力ポート、出力ポート、または変数ポートを設定できます。データのグループ分けを行う場合、アグリゲータトランスフォーメーションは、原則として、各グループの最終行を出力します。
- **ソート済み入力。**セッションのパフォーマンスを高めるには、このオプションを選択します。ソート済み入力を使用するには、Group By ポートに基づいて昇順または降順でソートされたデータをアグリゲータトランスフォーメーションに渡す必要があります。

アグリゲータトランスフォーメーションのコンポーネントおよびオプションは、[プロパティ] タブおよび [ポート] タブで設定することができます。

アグリゲータトランスフォーメーションのプロパティの設定

アグリゲータトランスフォーメーションのプロパティは、[プロパティ] タブで修正します。

以下のオプションを設定します。

アグリゲータの設定	説明
キャッシュディレクトリ	統合サービスによるインデックスキャッシュファイルとデータキャッシュファイルの作成先となるローカルディレクトリ。統合サービスではデフォルトで、Workflow Manager でプロセス変数\$PMCacheDir に対して入力されたディレクトリが使用されます。新しいディレクトリを指定する場合には、そのディレクトリが存在し、また集計キャッシュ用のディスクスペースが十分にあることを確認してください。 差分集計を有効化している場合、統合サービスはセッションを実行するたびにファイルのバックアップを作成します。そのため、2つのファイルを格納するだけの十分なディスクスペースをキャッシュディレクトリで確保する必要があります。
トレースレベル	トランスフォーメーションのセッションログに表示される情報の詳細度。
ソート済み入力	入力データがグループ単位に前もってソートされていることを示します。このオプションは、マッピングで Aggregator トランスフォーメーションにソート済みデータが渡される場合にのみ選択してください。
Aggregator のデータキャッシュサイズ	トランスフォーメーションのデータキャッシュサイズ。デフォルトのキャッシュサイズは 2,000,000 バイトです。設定したセッションキャッシュの合計サイズが 2GB (2,147,483,648 バイト) 以上の場合は、そのセッションを 64 ビットの統合サービスで実行します。キャッシュの数値を使用できます。パラメータファイルのキャッシュ値を使用するか、統合サービスを設定し、自動設定を使用してキャッシュサイズを設定します。キャッシュサイズが決定されるように統合サービスを設定した場合、キャッシュに割り当てられる最大メモリ量も設定できます。

アグリゲータの設定	説明
Aggregator のインデックスキャッシュサイズ	トランスフォーメーションのインデックスキャッシュサイズ。デフォルトのキャッシュサイズは 1,000,000 バイトです。設定したセッションキャッシュの合計サイズが 2GB (2,147,483,648 バイト) 以上の場合は、そのセッションを 64 ビットの統合サービスで実行します。キャッシュの数値を使用できます。パラメータファイルのキャッシュ値を使用するか、統合サービスを設定し、自動設定を使用してキャッシュサイズを設定します。キャッシュサイズが決定されるように統合サービスを設定した場合、キャッシュに割り当てられる最大メモリ量も設定できます。
トランスフォーメーション範囲	統合サービスが入力データにトランスフォーメーションロジックを適用する方法を指定します。 <ul style="list-style-type: none"> - Transaction。トランスフォーメーションロジックをトランザクションのすべての行に適用します。データの行が同一トランザクション内のすべての行に依存し、他のトランザクションの行には依存していない場合には、[Transaction] を選択します。 - すべての入力。トランスフォーメーションロジックをすべての入力データに適用します。[すべての入力] を選択すると、PowerCenter は入力トランザクションの境界を削除します。データの行がソース内のすべての行に依存している場合は、[すべての入力] を選択します。

アグリゲータトランスフォーメーションのポートの設定

アグリゲータトランスフォーメーションのポートを設定するには、次の作業を行います。

- 任意の出力ポートに式を入力して、条件句や非集計関数を使用する。
- 複数の集計出力ポートを作成する。
- 任意の入力ポート、入出力ポート、出力ポート、または変数ポートを Group By ポートとして設定する。
- 必要な入出力ポートのみを次のトランスフォーメーションに接続することによってパフォーマンスを向上させ、データキャッシュのサイズを小さくする。
- ローカル変数に対して変数ポートを使用する。
- 式の入力時に、他のトランスフォーメーションへの接続を作成する。

集計キャッシュの設定

アグリゲータトランスフォーメーションを使用するセッションを実行すると、統合サービスはメモリにインデックスおよびデータキャッシュを作成してトランスフォーメーションを処理します。統合サービスですらに多くのスペースが必要になると、オーバーフローした値がキャッシュファイルに格納されます。

キャッシュの数値を使用できます。パラメータファイルのキャッシュ値を使用するか、統合サービスを設定し、自動設定を使用してキャッシュサイズを設定します。

注: 統合サービスではメモリを使用して、ソート済みのポートを使用したアグリゲータトランスフォーメーションが処理されます。統合サービスはキャッシュメモリを使用しません。ソート済みポートを使用するアグリゲータトランスフォーメーションでは、キャッシュメモリを設定する必要はありません。

集計式

Designer では、アグリゲータトランスフォーメーションでのみ集計式が使えます。集計式には、条件句や非集計関数を含めることができます。式では、以下のように一方の集計関数を他方の集計関数に含めることもできます。

MAX(COUNT(ITEM))

集計式の結果は、トランスフォーメーション内の Group By ポートに基づいて変わります。例えば Integration Service によって、Group By ポートが定義されていない状態で以下の集計式を使用すると、全品目の合計販売数量が算出されます。

SUM(QUANTITY)

ただし、同じ式で ITEM ポートごとのグループ分けを指定すると、Integration Service は品目ごとの販売数量を返します。

任意の出力ポートで集計式の作成ができ、また 1 つのトランスフォーメーションで複数の集計ポートを使用できます。

関連項目：

- [「式に関する作業」 \(ページ 31\)](#)

集計関数

以下の集計関数は、アグリゲータトランスフォーメーション内で使用できます。また、1 つの集計関数を別の集計関数内にネストすることができます。

トランスフォーメーション言語には、以下の集計関数が用意されています。

- AVG
- COUNT
- FIRST
- LAST
- MAX
- MEDIAN
- MIN
- PERCENTILE
- STDDEV
- SUM
- VARIANCE

これらの関数は、アグリゲータトランスフォーメーション内の式で使用する必要があります。

ネストされた集計関数

アグリゲータトランスフォーメーションの異なる出力ポートに、単一レベルの関数を複数含めたり、複数レベルにネストされた関数を複数含めたりできます。ただし、アグリゲータトランスフォーメーションに単一レベルの関数とネストされた関数の両方を含めることはできません。そのため、アグリゲータトランスフォーメーションの出力ポートに単一レベルの関数が含まれる場合には、そのトランスフォーメーションの他のポートでネストされた関数を使うことはできません。単一レベルの関数とネストされた関数を同じアグリゲータトランスフォーメーションに含めると、そのマッピングまたはマップレットは Designer によって無効とされます。

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、別々のアグリゲータトランスフォーマーションを作成します。

条件句

集計で使用する行の数を削減するには、集計式で条件句を使用します。条件句には、TRUE または FALSE に評価される任意の句を使用できます。

たとえば以下の式を使用して、四半期単位のノルマを超過して達成した従業員の歩合総額を算出します。

```
SUM( COMMISSION, COMMISSION > QUOTA )
```

非集計関数

集計式では非集計関数を使用することもできます。

以下の式は品目単位の最高販売数を返します。まったく売れなかった場合には、0 が戻り値となります。

```
IIF( MAX( QUANTITY ) > 0, MAX( QUANTITY ), 0)
```

集計関数における NULL 値

Integration Service を設定する際に、Integration Service による集計関数での NULL 値の処理方法を選択できます。集計関数において NULL 値をゼロとして処理するか、または NULL として処理するかを選択できます。デフォルトでは、Integration Service は集計関数において NULL 値を NULL として処理します。

Group By ポート

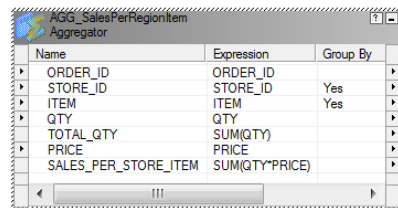
アグリゲータトランスフォーマーションを使用すると、すべての入力データで集計を実行する代わりに、集計対象となるグループを定義できます。たとえば、会社の総売上高を求めるのではなく、地域ごとにグループ分けした総売上高の確認ができます。

集計式に対してグループを定義するには、アグリゲータトランスフォーマーションで該当する入力、入出力、出力、および変数ポートを選択します。複数の Group By ポートを選択して、一意なグループの組み合わせそれぞれについて新しいグループを作成することができます。その後、Integration Service ではグループごとに定義済みの集計が実行されます。

値のグループ分けを行う場合、Integration Service はグループごとに 1 つずつ行を生成します。値のグループ分けを行わない場合、Integration Service はすべての入力行に対して 1 つの行を返します。一般には、各グループの最終行（最後に受信した行）と集計結果を返します。ただし、（例えば FIRST 関数を使用して）特定の行を返すように指定した場合、Integration Service はその指定された行を返します。

アグリゲータトランスフォーマーションで複数の Group By ポートを選択すると、Integration Service はポートの順序に基づいてグループ分けの順序を決定します。グループの順序は結果に影響を及ぼす場合もあるため、グループ分けが妥当なものとなるように、Group By ポートの順序付けを行ってください。たとえば、ITEM_ID の後で QUANTITY のグループ分けを行った場合の結果と、QUANTITY の後で ITEM_ID のグループ分けを行った場合の結果が異なることもありますが、その理由は、数量の数値が必ずしも一意ではないからです。

下記のアグリゲータトランスフォーメーションは、最初に STORE_ID で、次に ITEM でグループ分けを行います。



このアグリゲータトランスフォーメーションに以下のデータを渡すとしてします。

STORE_ID	ITEM	QTY	PRICE
101	'battery'	3	2.99
101	'battery'	1	3.19
101	'battery'	2	2.59
101	'AAA'	2	2.45
201	'battery'	1	1.99
201	'battery'	4	1.59
301	'battery'	1	2.45

Integration Service は、以下の一意なグループで集計計算を実行します。

STORE_ID	ITEM
101	'battery'
101	'AAA'
201	'battery'
301	'battery'

その後、Integration Service は、最後に受信した行に集計の結果を格納して、以下のとおりに渡します。

STORE_ID	ITEM	TOTAL_QTY	SALES_PER_STORE_ITEM
101	'AAA'	2	4.90
101	'battery'	6	17.34
201	'battery'	5	8.35
301	'battery'	1	2.45

非集計式

グループの変更または置き換えを実行するには、Group By ポートで非集計式を使用します。たとえば、グループ分けの前に「AAA battery」を置換したい場合には、以下の式を用いて、CORRECTED_ITEM という名の Group By 出力ポートを新たに作成できます。

```
IIF( ITEM = 'AAA battery', battery, ITEM )
```

デフォルト値

NULL 入力値を置き換えるには、グループ内のポートごとにデフォルト値を定義します。これにより、Integration Service は NULL 品目グループを集計に含めることができます。

関連項目：

- [「ポートのデフォルト値」 \(ページ 39\)](#)

ソート済み入力の使用

ソート済み入力オプションを使用して、アグリゲータトランスフォーメーションのパフォーマンスを向上させることができます。ソート済み入力を使用すると、Integration Service ではすべてのデータがグループ別にソートされているものと想定され、グループごとに行を読み込みながら集計計算が実行されます。必要に応じて、メモリにグループ情報が格納されます。ソート済み入力オプションを使用するには、ソート済みデータをアグリゲータトランスフォーメーションに渡す必要があります。セッションを複数のパーティションに分けて設定すると、ソート済みポート使用時のパフォーマンスが向上します。

ソート済み入力を使用しない場合でも、Integration Service は読み込みながら集計計算を実行します。この場合、データはソートされていないため、すべての集計計算が正確に実行されるように、Integration Service はソース全体の読み込みが完了するまで各グループのデータを格納しておきます。

例えば、あるアグリゲータトランスフォーメーションでポート別の STORE_ID と ITEM グループがあり、ソート済み入力オプションが選択されているとします。アグリゲータを通して以下のデータを渡すと、Integration Service は新しいグループである 201/battery を検出するとすぐに、101/battery グループの 3 つの行について集計を実行します。

STORE_ID	ITEM	QTY	PRICE
101	'battery'	3	2.99
101	'battery'	1	3.19
101	'battery'	2	2.59
201	'battery'	4	1.59
201	'battery'	1	1.99

ソート済み入力オプションを使用する場合、前もってデータを正しくソートしておかないと、予期しない結果が生じます。

[ソート済み入力] の条件

以下のいずれかの条件が当てはまる場合には、[ソート済み入力] オプションを使用しないでください。

- 集計式がネストされた集計関数を使用している。
- セッションで差分集計を使用している。

[ソート済み入力] オプションを使用した場合、データを正しくソートしないと、セッションは失敗します。

データのソート

[ソート済み入力] オプションを使用する場合は、ソート済みデータをアグリゲータへ渡します。

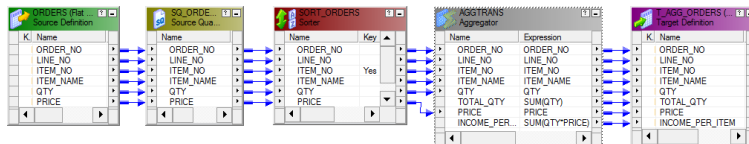
データは次の方法でソートする必要があります。

- アグリゲータの Group By ポートによって、アグリゲータトランスフォーメーションに表示される順でソートする。
- セッションに設定したのと同じソート順を使用する。データが厳密にセッションソート順に基づいた昇順または降順ではない場合、Integration Service はセッションに失敗します。たとえば、フランス語のソート順を使用するようにセッションを設定した場合、アグリゲータトランスフォーメーションに渡すデータは、フランス語のソート順を使用してソートされていなければなりません。

リレーショナルソースおよびファイルソースの場合、アグリゲータトランスフォーメーションにデータを渡す前に、ソータトランスフォーメーションを使用してマッピング内でデータをソートします。トランスフォーメーションによってソート済みデータの順序が変わらない場合は、マッピングの中でアグリゲータより前の任意の場所にソータトランスフォーメーションを配置できます。アグリゲータトランスフォーメーションの中の Group By カラムの順序はソータトランスフォーメーションと同じでなければなりません。

セッションでリレーショナルソースを使用する場合には、ソース修飾子の [ソートするポート数] オプションを用いて、ソースデータベースの Group By カラムをソートすることもできます。Group By カラムは、アグリゲータトランスフォーメーションおよびソース修飾子の両方で同じ順でなければなりません。

次のマッピングは、ITEM_NO の昇順にソースデータをソートするよう構成されたソータトランスフォーメーションを示しています。



ソータトランスフォーメーションは下記のようにデータをソートします。

ITEM_NO	ITEM_NAME	QTY	PRICE
345	Soup	4	2.95
345	Soup	1	2.95
345	Soup	2	3.25
546	Cereal	1	4.49
546	Cereal	2	5.25

[ソート済み入力] オプションを使用した場合、アグリゲータトランスフォーメーションは下記の結果を返します。

ITEM_NO	ITEM_NAME	TOTAL_QTY	INCOME_PER_ITEM
345	Soup	7	21.25
546	Cereal	3	14.99

アグリゲータトランスフォーメーションの作成

マッピングでアグリゲータトランスフォーメーションを使用するには、マッピングにアグリゲータトランスフォーメーションを追加します。次に、集計式および Group By ポートを使用してトランスフォーメーションを設定します。

アグリゲータトランスフォーメーションを作成するには：

1. Mapping Designer で、[トランスフォーメーション] - [作成] をクリックします。アグリゲータトランスフォーメーションを選択します。
2. アグリゲータの名前を入力して、[作成] をクリックします。次に [完了] をクリックします。
Designer がアグリゲータトランスフォーメーションを作成します。
3. アグリゲータトランスフォーメーションにポートをドラッグします。
Designer は、対象ポートのそれぞれについて入出力ポートを作成します。
4. トランスフォーメーションのタイトルバーをダブルクリックして [トランスフォーメーションの編集] ダイアログボックスを開きます。
5. [ポート] タブを選択します。
6. アグリゲータでグループを作成する際に使用したい各カラムの [Group By] オプションをクリックします。
必要に応じて、NULL グループを置き換えるデフォルト値を入力します。
7. [追加] をクリックして、式ポートを追加します。
式ポートは、出力ポートでなければなりません。[入力ポート (I)] のチェックマークを外して、ポートを出力ポートとします。
8. 必要に応じて、特定のポートのデフォルト値を追加します。
NULL 値をターゲットに渡したくない場合、トランスフォーメーションを使用して NULL 値をデフォルト値に変換します。
9. [プロパティ] タブで、プロパティを設定します。

アグリゲータトランスフォーメーションに関するヒント

[ソート済み入力] オプションを使用し、集計キャッシュの使用量を減らしてください。

[ソート済み入力] オプションはセッション時にキャッシュに格納されるデータ量を減らし、セッションのパフォーマンスを向上させます。このオプションをソータトランスフォーメーションと共に使用して、ソート済みデータをアグリゲータトランスフォーメーションに渡します。

アグリゲータトランスフォーメーションで、ソートされた出力が提供されないこともあります。

アグリゲータトランスフォーメーションからの出力をソートするには、ソータトランスフォーメーションを使用します。

接続される入出力ポートまたは出力ポートを制限してください。

接続される入出力ポートまたは出力ポートの数を制限することにより、アグリゲータトランスフォーメーションがデータキャッシュに格納するデータ量を減らしてください。

集計に先立ってデータをフィルタリングします。

マッピングでフィルタトランスフォーメーションを使用する場合は、そのあとにアグリゲータトランスフォーメーションを配置して、不要な集計を減らしてください。

アグリゲータトランスフォーメーションのトラブルシューティング

[ソート済み入力] オプションを選択しましたが、ワークフローの処理にかかる時間が前と変わりません。

以下のいずれかの条件では、[ソート済み入力] オプションを使用できません。

- 集計式が、ネストされた集計関数を含んでいる。
- セッションで差分集計を使用している。
- ソースデータがデータドリブンである。

以上の条件のいずれかが満たされている場合、Integration Service はソート済み入力オプションが使用されていない場合と同様にトランスフォーメーションを処理します。

アグリゲータトランスフォーメーションを使用するとセッションのパフォーマンスが低下します。

Integration Service は、ワークフロー実行中にディスクへのページングを行う場合があります。トランスフォーメーションプロパティでインデックスキャッシュとデータキャッシュのサイズを増やせば、セッションのパフォーマンスを高めることができます。

アグリゲータトランスフォーメーションにオーバーライドキャッシュディレクトリを入力しましたが、Integration Service はセッションの差分集計ファイルをどこか別の場所に保存してしまいます。

トランスフォーメーションキャッシュディレクトリの上書きはセッションレベルで行うことができます。Integration Service はキャッシュディレクトリをセッションログに記録します。オーバーライドキャッシュディレクトリもセッションプロパティで確認できます。

第 3 章

カスタムトランスフォーメーション

この章では、以下の項目について説明します。

- [カスタムトランスフォーメーションの概要, 61 ページ](#)
- [カスタムトランスフォーメーションの作成, 63 ページ](#)
- [グループおよびポートに関する作業, 64 ページ](#)
- [ポート属性に関する作業, 66 ページ](#)
- [カスタムトランスフォーメーションのプロパティ, 67 ページ](#)
- [トランザクション制御に関する作業, 70 ページ](#)
- [入力データのブロック, 71 ページ](#)
- [プロシージャのプロパティに関する作業, 73 ページ](#)
- [カスタムトランスフォーメーションプロシージャの作成, 74 ページ](#)

カスタムトランスフォーメーションの概要

カスタムトランスフォーメーションは、Designer の外部で作成したプロシージャと連絡して動作し、PowerCenter の機能を拡張します。カスタムトランスフォーメーションを作成し、カスタムトランスフォーメーションの関数を使用して開発するプロシージャにバインドすることができます。カスタムトランスフォーメーションは、アクティブまたはパッシブなトランスフォーメーションにすることができます。

カスタムトランスフォーメーションを使用すると、ソートや集計など、出力行を出力する前にすべての入力行を処理する必要があるトランスフォーメーションアプリケーションを作成できます。この処理のために、カスタムトランスフォーメーションではエクスターナルプロシージャトランスフォーメーションと異なり、入力関数と出力関数が別々に存在します。

Integration Service は、入力関数を使って入力データをプロシージャに渡します。出力関数はこれとは別の関数で、プロシージャコードに入力して出力データを Integration Service に渡す必要があります。これとは対照的に、エクスターナルプロシージャトランスフォーメーションでは、エクスターナルプロシージャ関数は入力と出力の両方を行い、パラメータとしてトランスフォーメーションのすべてのポートを取ります。

カスタムトランスフォーメーションを使って、複数の入力グループ、複数の出力グループ、またはその両方を必要とするトランスフォーメーションを作成することもできます。グループは、トランスフォーメーションに入力されるデータ、またはトランスフォーメーションから出力されるデータの行を表すものです。例えば、XML データを解析する、1 つの入力グループと複数の出力グループを持つカスタムトランスフォーメーションを作成することができます。また、2 つの入力データのストリームを 1 つの出力データのストリームに結合す

る、2つの入力グループと1つの出力グループを持つカスタムトランスフォーメーションを作成することもできます。

カスタムトランスフォーメーションを使用して構築されたトランスフォーメーションに関する作業

カスタムトランスフォーメーションを使用して、トランスフォーメーションを構築できます。一部の PowerCenter トランスフォーメーションは、カスタムトランスフォーメーションを使用して構築されています。Informatica 製品に付属の以下のトランスフォーメーションはネイティブのトランスフォーメーションであり、カスタムトランスフォーメーションを使用して構築されたものではありません。

- アグリゲータトランスフォーメーション
- 式トランスフォーメーション
- エクスターナルプロシージャトランスフォーメーション
- フィルタトランスフォーメーション
- ジョイナトランスフォーメーション
- ルックアップトランスフォーメーション
- ノーマライザトランスフォーメーション
- ランクトランスフォーメーション
- ルータトランスフォーメーション
- シーケンスジェネレータトランスフォーメーション
- ソータトランスフォーメーション
- ソース修飾子トランスフォーメーション
- ストアドプロシージャトランスフォーメーション
- トランザクション制御トランスフォーメーション
- アップデートストラテジトランスフォーメーション

その他すべてのトランスフォーメーションは、カスタムトランスフォーメーションを使用して作成されます。ブロックに関するルールなど、カスタムトランスフォーメーションに適用されるルールは、カスタムトランスフォーメーションを使用して作成されたトランスフォーメーションにも適用されます。例えば、マッピング内のカスタムトランスフォーメーションを接続する場合、Integration Service ですべてのソースをブロックすることなく、ターゲットロード順グループ内のすべてのソースからターゲットにデータが流れることを確認する必要があります。同様に、カスタムトランスフォーメーションを使用して作成されたトランスフォーメーションでも、このデータフローを確認する必要があります。

コードページの互換性

Integration Service が ASCII モードで動作している場合は、データをカスタムトランスフォーメーションプロシージャに ASCII 形式で渡します。Integration Service が Unicode モードで動作している場合は、データをプロシージャに UCS-2 形式で渡します。

データを別の形式または別のコードページで要求するには、カスタムトランスフォーメーションプロシージャコードで `INFA_CTChangeStringMode()` 関数および `INFA_CTSetDataCodePageID()` 関数を使用します。

使用できる関数は、Integration Service のデータ移動モードによって異なります。

- **ASCII モード**。データを UCS-2 形式で要求するには、INFA_CTChangeStringMode()関数を使用します。この関数を使う場合、プロシージャで UCS-2 形式の ASCII 文字だけを Integration Service に渡す必要があります。Integration Service が ASCII モードで動作している場合は、INFA_CTSetDataCodePageID()関数を使用してコードページを変更することはできません。
- **Unicode モード**。データを MBCS (マルチバイト文字セット) 形式で要求するには、INFA_CTChangeStringMode()関数を使用します。プロシージャによって MBCS 形式のデータが要求された場合、Integration Service はデータを Integration Service のコードページで渡します。Integration Service のコードページとは別のコードページでデータを要求するには、INFA_CTSetDataCodePageID()関数を使用します。INFA_CTSetDataCodePageID()関数で指定するコードページは、Integration Service のコードページと双方向の互換性がある必要があります。

注: また、INFA_CTRebindInputDataType()関数を使用して、カスタムトランスフォーメーションの特定のポートの形式を変更することもできます。

カスタムトランスフォーメーションプロシージャの配布

カスタムトランスフォーメーションはリポジトリ間でコピーすることができます。カスタムトランスフォーメーションをリポジトリ間でコピーする場合、コピー先リポジトリが使用する Integration Service マシンにカスタムトランスフォーメーションプロシージャが組み込まれていることを確認する必要があります。

カスタムトランスフォーメーションの作成

Transformation Developer で再利用可能なカスタムトランスフォーメーションを作成し、そのトランスフォーメーションのインスタンスをマッピングに追加することができます。Mapping Designer または Mapplet Designer で再利用不可能なカスタムトランスフォーメーションを作成することもできます。

各カスタムトランスフォーメーションでは、モジュールとプロシージャの名前を指定します。カスタムトランスフォーメーションは、プロシージャが含まれた既存の共有ライブラリまたは DLL に基づいて作成できます。また、カスタムトランスフォーメーションをプロシージャを作成するためのベースとして作成することもできます。既存の共有ライブラリまたは DLL を使用してカスタムトランスフォーメーションを作成する場合には、正しいモジュールとプロシージャの名前を定義します。

プロシージャを作成するためのベースとしてカスタムトランスフォーメーションを作成する場合には、トランスフォーメーションを選択してコードを生成します。Designer は、手続きのコードを生成するときにトランスフォーメーションのプロパティを使用します。Designer は共通のモジュール名を共有するすべてのトランスフォーメーションのコードを、単一のディレクトリに生成します。

Designer は以下のファイルを生成します。

- **m_<モジュール名>.c**。モジュールを定義します。このファイルには、Integration Service によってモジュールのロード時に実行されるコードを記述できる初期化関数 m_<モジュール名>_moduleInit()が含まれます。同様にこのファイルには、Integration Service によってモジュールをアンロードする前に実行されるコードを記述できる初期化解除関数 m_<モジュール名>_moduleDeinit()が含まれます。
- **p_<プロシージャ名>.c**。モジュール内のプロシージャを定義します。このファイルには、データクレンジングやマーキングといったプロシージャのロジックを実装するコードが含まれます。

- **makefile.aix、makefile.aix64、makefile.hpparisc64、makefile.linux、makefile.sol、および makefile.sol64。** UNIX プラットフォーム（zLinux を除く）用のファイルを作成します。AIX プラットフォーム（64 ビット）には makefile.aix64 を、Solaris プラットフォーム（64 ビット）には makefile.sol64 を使用します。

注: zLinux の場合、makefile.linux を更新する必要があります。FLAGS セクションには「-m64」を追加します。例えば、FLAGS=-Wall -fPIC -DUNIX -m64 のように指定します。

カスタムトランスフォーメーションのルールおよびガイドライン

カスタムトランスフォーメーションを作成する場合は、以下のルールおよびガイドラインに従います。

- カスタムトランスフォーメーションは、接続されたトランスフォーメーションです。式中でカスタムトランスフォーメーションを参照することはできません。
- 1つのモジュールに複数の手続きを含めることができます。たとえば、同じモジュールにXML 書き出し手続きと XML 解析手続きを含めることができます。
- 複数のカスタムトランスフォーメーションインスタンスを扱うプロシージャコードを記述する場合、1つの共有ライブラリまたは DLL を複数のカスタムトランスフォーメーションインスタンスにバインドすることができます。
- プロシージャコードを記述するときは、そのコードがマッピングの基本ルールに違反しないようにする必要があります。
- カスタムトランスフォーメーションは、高精度 10 進値を高精度 10 進値として受け渡します。
- カスタムトランスフォーメーションプロシージャではマルチスレッドコードを使用します。

カスタムトランスフォーメーションのコンポーネント

カスタムトランスフォーメーションの設定時には、以下のコンポーネントを定義します。

- **【トランスフォーメーション】タブ。** このタブでは、トランスフォーメーションの名前を変更したり、説明を追加することができます。
- **【ポート】タブ。** カスタムトランスフォーメーションに対してポートやグループを追加したり編集することができます。出力ポートが依存する入力ポートの定義も行えます。
- **【ポート属性定義】タブ。** カスタムトランスフォーメーションのポートに関するユーザー定義のポート属性を作成できます。
- **【プロパティ】タブ。** モジュールと関数の識別子、トランザクションのプロパティ、実行時位置などのトランスフォーメーションのプロパティを定義できます。
- **【初期化プロパティ】タブ。** エクスターナルプロシージャが初期化時などの実行時に使用するプロパティを定義できます。
- **【メタデータエクステンション】タブ。** メタデータエクステンションを作成して、プロシージャが初期化時などの実行時に使用するプロパティを定義できます。

グループおよびポートに関する作業

カスタムトランスフォーメーションには、入力と出力の両方のグループがあります。また、入力ポート、出力ポート、および入出力ポートを持つこともできます。カスタムトランスフォーメーションの【ポート】タブで、グループとポートの作成および編集を行います。また【ポート】タブでは、入力ポートと出力ポートのリレーションも定義できます。

グループとポートの作成

カスタムトランスフォーメーションに複数の入力グループと複数の出力グループを作成することができます。最低でも入力グループ1つと出力グループ1つを作成する必要があります。入力グループを作成するには、[入力グループの作成] アイコンをクリックします。出力グループを作成するには、[出力グループの作成] アイコンをクリックします。既存のグループ名を変更するには、グループヘッダを入力します。パッシブなカスタムトランスフォーメーションを作成した場合は、入力グループと出力グループをそれぞれ1つだけ作成できます。

ポートを作成すると、Designer はそのポートを現在選択されている行またはグループの下に追加します。ポートは、1つ上に表示される入力グループおよび出力グループに所属させることができます。入力グループの下に表示される入力/出力ポートは、出力グループの一部でもあります。出力グループの下に表示される入力/出力ポートは、入力グループの一部でもあります。

ポートを共有するグループを結合グループと言います。反対タイプの隣接グループがポートを共有できます。1つのグループを複数の結合グループに含めることができます。例えば、[「手順 1。カスタムトランスフォーメーションの作成」 \(ページ 74\)](#)の図では、InputGroup1 および OutputGroup1 は ORDER_ID1 を共有する結合グループです。

トランスフォーメーションに [ポート属性定義] タブがある場合、ポートごとに属性を編集できます。

グループとポートの編集

カスタムトランスフォーメーション内のポートおよびグループを編集する場合は、以下のルールおよびガイドラインに従います。

- グループヘッダに入力することにより、グループ名を変更できます。
- ポートとグループの名前には ASCII 文字しか入力できません。
- グループを作成した後では、グループタイプを変更することはできません。グループタイプを変更する必要がある場合は、そのグループを削除してから、新しいグループを追加してください。
- グループを削除すると、Designer はそのグループ内の同じタイプのポートをすべて削除します。ただし、入出力ポートはすべてトランスフォーメーション内に残ってその上のグループに属し、削除されるグループのタイプに応じて入力ポートまたは出力ポートに変更されます。たとえば、出力グループに出力ポートと入出力ポートが含まれているとします。ここで出力グループを削除します。Designer は出力ポートを削除します。入出力ポートは入力ポートに変更されます。これらの入力ポートは、すぐ上のヘッダの入力グループに属することになります。
- グループを上または下に移動するには、グループヘッダを選択して、[上方に移動] または [下に移動] をクリックします。グループヘッダの上と下のポートはそのまま残りますが、ポートが属するグループは変更される場合があります。
- 入力/出力ポートを作成するには、トランスフォーメーションに入力グループおよび出力グループが設定されている必要があります。

ポートの関係の定義

デフォルトでは、カスタムトランスフォーメーション内の出力ポートはすべての入力ポートに依存します。ただし、カスタムトランスフォーメーション内の入力ポートと出力ポートの関係を定義できます。この場合、カスタムトランスフォーメーションを含むマッピング内のリンクパスを表示して、出力ポートがどの入力ポートに依存するかを確認することができます。また、カスタムトランスフォーメーションを含むマッピング内のターゲットポートについてのソースカラムの依存関係を表示することもできます。

カスタムトランスフォーメーション内のポートの関係を定義するには、ポートの依存性を作成します。ポートの依存性は、1つの出力または入出力ポートと、1つ以上の入力または入出力ポートとの間の関係です。ポートの依存性を作成するときは、コード内の手続きロジックに基づいてください。

ポートの依存性を作成するには、[ポート] タブの [編集] をクリックして、[ポート依存関係] を選択します。

例えば、XML データを解析するエクスターナルプロシージャを作成するとします。1つの入力ポートを持つ1つの入力グループと、複数の出力ポートを持つ複数の出力グループを使用して、カスタムトランスフォーメーションを作成します。エクスターナルプロシージャのロジックに従って、すべての出力ポートは入力ポートに依存します。各出力ポートについてポートの依存性を作成することにより、カスタムトランスフォーメーション内でこの関係を定義できます。各ポートの依存性を定義して、各出力ポートが1つの入力ポートに依存するようにします。

ポートの依存性を作成するには：

1. [ポート] タブで、[編集] をクリックし、[ポート依存関係] を選択します。
2. [ポート依存関係の出力] ダイアログボックスの [出力ポート] フィールドで、出力ポートまたは入出力ポートを選択します。
3. [入力ポート] ペインで、その出力ポートまたは入出力ポートが依存する入力ポートまたは入出力ポートを選択します。
4. [追加] をクリックします。
5. ポートの依存性に入力ポートまたは入出力ポートをさらに追加するには、[3](#) から [4](#) の手順を繰り返します。
6. 別のポートの依存性を作成するには、[2](#)～[5](#) の手順を繰り返します。
7. [OK] をクリックします。

ポート属性に関する作業

ポートには、データタイプや精度などの属性があります。カスタムトランスフォーメーションを作成する場合、ユーザー定義のポート属性を作成することができます。ユーザー定義のポート属性は、カスタムトランスフォーメーション内のすべてのポートに適用されます。

例えば、XML データを解析するエクスターナルプロシージャを作成するとします。その際に、XML 階層内の要素の位置を定義できる「XML パス」というポート属性を作成することができます。

カスタムトランスフォーメーションの [ポート属性定義] タブで、ポート属性を作成し、デフォルト値を割り当てることができます。[ポート] タブで、各ポートごとに固有のポート属性を定義できます。

ポート属性を作成する場合は、以下のプロパティを定義します。

- **名前。**ポート属性の名前。
- **データタイプ。**ポート属性値のデータタイプ。Boolean、Numeric、String のいずれかを選択できます。
- **値。**ポート属性のデフォルト値。このプロパティはオプションです。ここに値を入力すると、その値がカスタムトランスフォーメーション内のすべてのポートに適用されます。[ポート] タブで、各ポートのポート属性値を上書きすることができます。

ポート属性は各カスタムトランスフォーメーションに対して定義します。ポート属性をカスタムトランスフォーメーション間でコピーすることはできません。

ポート属性値の編集

ポート属性を作成した後で、トランスフォーメーション内の各ポートのポート属性値を編集することができます。ポート属性値を編集するには、[ポート] タブで [カスタムトランスフォーメーション] をクリックし、[ポート属性を編集] を選択します。

特定のポートのポート属性値は、[開く] ボタンをクリックして変更することができます。[ポート属性デフォルト値を編集する] ダイアログボックスが開きます。あるいは、[値] カラムに新しい値を直接入力することもできます。

[ポートレベル属性を編集する] ダイアログボックスに表示されるポートは、 [グループの選択] フィールドからグループを選択することにより、フィルタリングすることができます。

カスタムトランスフォーメーションのプロパティ

カスタムトランスフォーメーションのプロパティは、プロシージャとトランスフォーメーションの両方に適用されます。 カスタムトランスフォーメーションのプロパティの設定は、カスタムトランスフォーメーションの [プロパティ] タブで行います。

以下の表で、カスタムトランスフォーメーションのプロパティについて説明します。

オプション	説明
言語	プロシージャコードで使用される言語。カスタムトランスフォーメーションを作成する場合は、言語を定義します。言語を変更する必要がある場合、新規のカスタムトランスフォーメーションを作成します。
モジュール識別子	モジュールの名前。C 言語または C++言語を使用して開発されたカスタムトランスフォーメーションプロシージャに適用されます。 このフィールドに入力できるのは、ASCII 文字のみです。マルチバイト文字は入力できません。 このプロパティは、手続きを含む DLL または共有ライブラリの基本となる名前です。エクスターナルプロシージャコードを生成するときは、Designer でこの名前を使って C ファイルを作成します。
関数識別子	モジュール内の手続きの名前。C 言語を使用して開発されたカスタムトランスフォーメーションプロシージャに適用されます。 このフィールドに入力できるのは、ASCII 文字のみです。マルチバイト文字は入力できません。 Designer では、この名前を使って、手続きコードを入力する C ファイルを作成します。
クラス名	カスタムトランスフォーメーションプロシージャのクラス名。C++言語または Java 言語を使用して開発されたカスタムトランスフォーメーションプロシージャに適用されます。 このフィールドに入力できるのは、ASCII 文字のみです。マルチバイト文字は入力できません。
実行時位置	DLL または共有ライブラリの格納場所。デフォルトは\$PMExtProcDir です。カスタムトランスフォーメーションのセッションを実行する Integration Service ノードへの相対パスを入力します。 このプロパティが空白の場合、Integration Service は、Integration Service ノードで定義されている環境変数を使用して DLL または共有ライブラリの位置を探します。 Integration Service ノードで定義されている実行時位置または環境変数に、すべての DLL または共有ライブラリをコピーする必要があります。DLL、共有ライブラリ、または参照されるファイルが見つからない場合、Integration Service は手続きのロードに失敗します。

オプション	説明
トレースレベル	トランスフォーメーションのセッションログに表示される情報の詳細度。デフォルトは [Normal] です。
パーティション化可能	<p>このトランスフォーメーションを使用するパイプラインで、複数のパーティションを作成できるかどうかを指定します。</p> <ul style="list-style-type: none"> - いいえ。トランスフォーメーションはパーティション化できません。同一パイプライン内のこのトランスフォーメーションおよびその他のトランスフォーメーションは、1つのパーティションに含まれる必要があります。 - ローカルで。トランスフォーメーションをパーティション化することはできませんが、同じノード上のパイプラインですべてのパーティションが実行される必要があります。Csutom トランスフォーメーションの別のパーティションがメモリ内のオブジェクトを共有する必要がある場合、[ローカルで] を選択します。 - グリッドをまたがる。トランスフォーメーションをパーティション化することができ、各パーティションは異なるノードに配分されます。 <p>デフォルトは [No]。</p>
入力ブロック	トランスフォーメーションに関連付けられる手続きが入力データをブロックできるようにする必要があるかどうかを指定します。デフォルトでは有効になっています。
アクティブ	<p>このトランスフォーメーションがアクティブかパッシブかを指定します。</p> <p>カスタムトランスフォーメーションを作成した後は、このプロパティを変更できません。このプロパティを変更する必要がある場合は、新しいカスタムトランスフォーメーションを作成してから、正しいプロパティ値を選択します。</p>
アップデートストラテジトランスフォーメーション	トランスフォーメーションが出力行のアップデートストラテジを定義するかどうかを指定します。デフォルトでは無効になっています。これは、アクティブなカスタムトランスフォーメーションの場合に有効にすることができます。
トランスフォーメーション範囲	<p>Integration Service が入力データにトランスフォーメーションロジックを適用する方法を示します。</p> <ul style="list-style-type: none"> - 行 - トランザクション - すべての入力 <p>トランスフォーメーションがパッシブな場合、このプロパティは必ず [Row] にしてください。トランスフォーメーションがアクティブな場合、このプロパティはデフォルトでは [すべての入力] となります。</p>
トランザクションの生成	<p>このトランスフォーメーションがトランザクションを生成できるかどうかを指定します。カスタムトランスフォーメーションがトランザクション生成するときに、すべての出力グループについてトランザクションを生成します。</p> <p>デフォルトでは無効になっています。このプロパティは、アクティブなカスタムトランスフォーメーションの場合にのみ有効にすることができます。</p>
出力が再現可能	<p>出力データの順序をセッションの実行ごとに一致させるかどうかを指定します。</p> <ul style="list-style-type: none"> - Never。出力データの順序はセッションの実行ごとに異なります。アクティブなトランスフォーメーションの場合、これがデフォルトです。 - 入力順による。入力データの順序がセッションの実行ごとに一致している場合、出力順序をセッションの実行ごとに一致させます。パッシブなトランスフォーメーションの場合、これがデフォルトです。 - Always。入力データの順序がセッションの実行ごとに異なる場合でも、出力データの順序は常に同じです。

オプション	説明
パーティションごとに1つのスレッドを要求します	Integration Service によってプロシージャの各パーティションが1つのスレッドで処理される場合に指定します。このオプションを有効化した場合、プロシージャコードはスレッド特有の操作を実行できます。デフォルトでは有効になっています。
出力が確定的かどうか	トランスフォーメーションが、セッションの実行ごとに一貫した出力データを生成するかどうかを指定します。このトランスフォーメーションを使用するセッションでリカバリを実行するには、このプロパティを有効にします。

警告: トランスフォーメーションを繰り返し可能で一意に定まるものとして設定する場合は、データが繰り返し可能で一意に定まることを保証する必要があります。セッションとリカバリで同じデータが生成されないトランスフォーメーションを使用してセッションをリカバリしようとする、リカバリプロセスを実行した結果、データが破損する可能性があります。

アップデートストラテジの設定

アクティブなカスタムトランスフォーメーションを使用して、以下のレベルでマッピングのアップデートストラテジを設定します。

- **プロシージャ内。** 出力行のアップデートストラテジを設定するエクスターナルプロシージャコードを記述できます。エクスターナルプロシージャでは、挿入、更新、削除、または拒否のフラグを行に設定できます。
- **マッピング内。** マッピング内でカスタムトランスフォーメーションを使用して、挿入、更新、削除、または拒否のフラグを行に設定します。カスタムトランスフォーメーション用のアップデートストラテジトランスフォーメーションプロパティを選択します。
- **セッション内。** ソース行をデータドリブンとして扱うようにセッションを設定します。

アップデートストラテジを定義するようにカスタムトランスフォーメーションを設定しない場合、またはセッションをデータドリブンとして設定しない場合、Integration Service は出力行にフラグを設定するためにエクスターナルプロシージャコードを使用することはありません。その代わりに、カスタムトランスフォーメーションがアクティブな場合、Integration Service は出力行に挿入のフラグを設定します。カスタムトランスフォーメーションがパッシブである場合、Integration Service は行タイプを保持します。例えば、更新のフラグが設定されている行がパッシブなカスタムトランスフォーメーションに入力されると、Integration Service はその行のタイプを保持し、更新として出力します。

スレッド特有のプロシージャコードに関する作業

カスタムトランスフォーメーションプロシージャには、スレッド特有の操作が含まれる場合があります。スレッド特有の操作とは、プロシージャを処理するスレッドに基づいてアクションを実行するコードです。

[パーティションごとに1つのスレッドを要求します] プロパティを使用した各パーティションでカスタムトランスフォーメーションを処理する場合に、Integration Service が1つのスレッドを使用するよう、カスタムトランスフォーメーションを設定できます。

各パーティションを1つのスレッドで処理するようカスタムトランスフォーメーションを設定した場合、Integration Service は各パーティションで同じスレッドを持つ以下の関数を呼び出します。

- p_<手続き名>_partitionInit()
- p_<手続き名>_partitionDeinit()
- p_<手続き名>_inputRowNotification()
- p_<手続き名>_dataBdryRowNotification()
- p_<手続き名>_eofNotification()

Integration Service は同じスレッドを使用して各パーティションで上記の関数进行处理するため、これらの関数にはスレッド特有の操作を含めることができます。例えば、Java バージョナルマシン（JVM）にスレッドをアタッチまたはデタッチする場合があります。

注: 各パーティションを 1 つのスレッドで処理するようにカスタムトランスフォーメーションを設定した場合、マッピングの設定に基づき Workflow Manager がパーティションポイントを追加します。

トランザクション制御に関する作業

カスタムトランスフォーメーションのトランザクション制御は、以下のトランスフォーメーションプロパティを使って定義することができます。

- **トランスフォーメーション範囲。** Integration Service が入力データにトランスフォーメーションロジックを適用する方法を指定します。
- **トランザクションの生成。** プロシージャが、トランザクション行を生成して出力グループに出力するように指定します。

トランスフォーメーション範囲

Integration Service が入力データにトランスフォーメーションロジックを適用する方法を設定できます。以下の値のいずれかを選択できます。

- **行。** トランスフォーメーションロジックを一度に 1 行のデータに適用します。手続きの結果がデータの単一の行に依存する場合は [Row] を選択してください。たとえば、手続きが XML ファイルを含む行を解析する場合には [Row] を選択します。
- **トランザクション。** トランスフォーメーションロジックをトランザクションのすべての行に適用します。手続きの結果が同一トランザクションのすべての行に依存し、他のトランザクションの行には依存していない場合には、[Transaction] を選択します。[Transaction] を選択した場合、すべての入力グループを同じトランザクション制御ポイントに接続する必要があります。例えば、エクスターナルプロシージャが単一のトランザクションのデータに対して集計計算を行う場合に、[トランザクション] を選択します。
- **すべての入力。** トランスフォーメーションロジックをすべての入力データに適用します。[すべての入力] を選択すると、Integration Service はトランザクション境界を削除します。[すべての入力] は、プロシージャの結果がソース内のすべてのデータ行に依存する場合に選択します。例えば、エクスターナルプロシージャがすべての入力データに対して集計計算やソートを行う場合に、[すべての入力] を選択します。

トランザクションの生成

行のコミットやロールバックなどの、トランザクションを出力するエクスターナルプロシージャコードを作成することができます。エクスターナルプロシージャで行のコミットおよびロールバックを出力する場合は、トランザクションを生成するようにカスタムトランスフォーメーションを設定します。[トランザクションの生成] トランスフォーメーションプロパティを選択します。アクティブなカスタムトランスフォーメーションでは、このプロパティを有効にすることができます。

エクスターナルプロシージャが行のコミットまたはロールバックを出力する場合、すべての出力グループに対して出力またはロールバックを実行します。

トランザクションを生成するようにトランスフォーメーションを設定すると、Integration Service はカスタムトランスフォーメーションをトランザクション制御トランスフォーメーションと同様に扱います。マッピング内でトランザクション制御トランスフォーメーションに適用されるルールのほとんどは、カスタムトランスフォーメーションにも適用されます。例えば、トランザクションを生成するようカスタムトランスフォーメーション

ョンを設定すると、トランスフォーメーションを含むパイプラインまたはパイプラインブランチを連結することはできません。

トランザクションを生成するように設定されたカスタムトランスフォーメーションを使用するセッションを編集または作成する場合は、ユーザー定義コミット用に設定します。

トランザクション境界に関する作業

Integration Service は、マッピングの設定およびカスタムトランスフォーメーションのプロパティに基づいてトランザクション境界のカスタムトランスフォーメーションへの入出力を処理します。

以下の表に、Integration Service がカスタムトランスフォーメーションでのトランザクション境界を処理する方法を示します。

トランスフォーメーション範囲	【トランザクションの生成】が有効	【トランザクションの生成】が無効
行	Integration Service は、入力されたトランザクション境界を削除し、データ境界通知関数を呼び出しません。 Integration Service は、すべての出力グループに対してプロシージャロジックに従ってトランザクション行を出力します。	すべての入力グループの入力データが同じトランザクション制御ポイントからのものである場合、Integration Service は入力されたトランザクション境界を保持し、すべての出力グループに出力します。ただし、データ境界通知関数は呼び出しません。 入力グループの入力データが異なるトランザクション制御ポイントからのものである場合、Integration Service は入力されたトランザクション境界を削除します。データ境界通知関数は呼び出しません。Integration Service は、1つのオープントランザクションですべての行を出力します。
トランザクション	Integration Service は、入力されたトランザクション境界を保持し、データ境界通知関数を呼び出します。 ただし、Integration Service は、すべての出力グループに対してプロシージャロジックに従ってトランザクション行を出力します。	Integration Service は、入力されたトランザクション境界を保持し、データ境界通知関数を呼び出します。 PowerCenter Server は、すべての出力グループに対してトランザクション行を出力します。
すべての入力	Integration Service は、入力されたトランザクション境界を削除し、データ境界通知関数を呼び出しません。Integration Service は、すべての出力グループに対してプロシージャロジックに従ってトランザクション行を出力します。	Integration Service は、入力されたトランザクション境界を削除し、データ境界通知関数を呼び出しません。1つのオープントランザクションですべての行を出力します。

入力データのブロック

デフォルトでは、Integration Service はターゲットロード順グループ内のソースを同時に読み込みます。ただし、一部の入力グループに対して入力データをブロックするエクスターナルプロシージャコードを記述することができます。ブロックとは、複数の入力グループトランスフォーメーションの入力グループへのデータフローを一時停止することです。

カスタムトランスフォーメーションを使って入力データをブロックするには、データのブロックとブロック解除を行うプロシージャコードを記述する必要があります。また、カスタムトランスフォーメーションの [プロパティ] タブでブロックを有効にしておく必要があります。

データをブロックするプロシージャコードの記述

入力データのブロックとブロック解除を行う手続きを書くことができます。入力データをブロックするには `INFA_CTBlockInputFlow()` 関数を使用します。入力データのブロックを解除するには `INFA_CTUnblockInputFlow()` 関数を使用します。

エクスターナルプロシージャで入力グループからの読み込みを切り換える必要がある場合は、入力データをブロックする必要があります。ブロック機能を使わない場合、入力データをバッファリングする手続きコードを書く必要があります。通常はセッションのパフォーマンスを低下させるバッファリングの代わりに、入力データをブロックすることができます。

たとえば、2つの入力グループでエクスターナルプロシージャを作成するとします。このエクスターナルプロシージャは、最初の入力グループのデータを1行読み込んでから、2つ目の入力グループのデータを1行読み込みます。ブロックを使用すれば、一方の入力グループからのデータを処理している間は他の入力グループからのデータフローをブロックするエクスターナルプロシージャコードを書くことができます。データをブロックするエクスターナルプロシージャコードを書くと、そのプロシージャではソースデータをバッファにコピーする必要がないため、パフォーマンスが向上します。しかし、データを処理する準備ができるまで、バッファを割り当てて入力グループからバッファにデータをコピーするエクスターナルプロシージャを書くこともできます。ソースデータのバッファへのコピーはパフォーマンスを低下させます。

関連項目：

- [「ブロック関数」 \(ページ 116\)](#)

カスタムトランスフォーメーションをブロッキングトランスフォーメーションとして設定する

カスタムトランスフォーメーションを作成するときは、Designer によって [入力ブロック] トランスフォーメーションプロパティがデフォルトで有効になります。このプロパティは、マッピングの保存または検証時のデータフロー検証に影響を与えます。このプロパティを有効にすると、カスタムトランスフォーメーションはブロックはブロッキングトランスフォーメーションになります。このプロパティをクリアすると、カスタムトランスフォーメーションはブロッキングトランスフォーメーションではなくなります。

エクスターナルプロシージャコードによって入力データをブロックできるようにする必要がある場合は、カスタムトランスフォーメーションをブロッキングトランスフォーメーションとして設定します。

以下のいずれかの条件が満たされる場合には、カスタムトランスフォーメーションを非ブロッキングトランスフォーメーションとして設定できます。

- 手続きコードにブロック関数が含まれていない。
- 手続きコードに2つのアルゴリズムが含まれ、一方はブロックを使用し、もう一方はデータをブロックする代わりに手続きが割り当てたバッファにソースデータをコピーする。このコードによって、Integration Service がカスタムトランスフォーメーションによるデータのブロックを許可するかどうかをチェックします。プロシージャは、ブロックが可能な場合はブロック関数を伴うアルゴリズムを使用し、ブロックが不可の場合は他のアルゴリズムを使用します。これにより、複数のマッピング設定で使用できるカスタムトランスフォーメーションを作成できます。

注: プロシージャでデータをブロックする場合にカスタムトランスフォーメーションを非ブロッキングトランスフォーメーションとして設定すると、Integration Service はセッションに失敗します。

カスタムトランスフォーメーションでのマッピングの検証

マッピングにカスタムトランスフォーメーションを含めると、Designer と Integration Service の両方でマッピングが検証されます。Designer ではマッピングの保存または検証時に、Integration Service ではセッションの実行時に検証されます。

設計時の検証

マッピングを保存または検証すると、Designer はデータフローの検証を行います。Designer はこのとき、ブロックトランスフォーメーションがすべてのソースをブロックすることなく、すべてのソースからターゲットロード順グループヘッダーが流れるかどうかを確認します。ブロックトランスフォーメーションを含むマッピングは無効となる場合があります。

実行時の検証

セッションを実行すると、Integration Service は実行時のプロシージャコードに対してマッピングを検証します。このとき Integration Service は、カスタムトランスフォーメーションによるデータのブロックを許可するかどうかを追跡します。

- **カスタムトランスフォーメーションをブロックトランスフォーメーションとして設定する。**カスタムトランスフォーメーションによるデータのブロックを常に許可します。
- **カスタムトランスフォーメーションを非ブロックトランスフォーメーションとして設定する。**カスタムトランスフォーメーションによるデータのブロックをマッピングの設定に応じて許可します。Integration Service がターゲットロード順グループ内のすべてのソースを同時にブロックせずに、カスタムトランスフォーメーションでデータをブロックできる場合、カスタムトランスフォーメーションによるデータのブロックは許可されます。

プロシージャコードを作成して、Integration Service がカスタムトランスフォーメーションにデータのブロックを許可するかどうかをチェックすることができます。INFA_CT_getInternalProperty()関数を使って INFA_CT_TRANS_MAY_BLOCK_DATA プロパティ ID にアクセスします。Integration Service は、カスタムトランスフォーメーションがデータをブロックできる場合は TRUE を返し、カスタムトランスフォーメーションがデータをブロックできない場合は FALSE を返します。

プロシージャのプロパティに関する作業

Integration Service でプロシージャを実行する際（初期化時など）にプロシージャが使用するプロパティの名前と値のペアは、カスタムトランスフォーメーションで定義することができます。カスタムトランスフォーメーションの以下のタブでユーザー定義プロパティを作成できます。

- **メタデータエクステンション。**プロパティの名前、データタイプ、精度、および値を指定できます。メタデータエクステンションを使用して、プロシージャに情報を渡します。
- **初期化プロパティ。**プロパティの名前と値を指定できます。

カスタムトランスフォーメーションのどちらのタブでもプロパティを定義できますが、[メタデータエクステンション] タブではプロパティを詳細に指定できます。メタデータエクステンションを使用して、プロシージャにプロパティを渡します。

例えば、トランスフォーメーション後にデータをソートするカスタムトランスフォーメーションのエクスターナルプロシージャを作成します。ここでは、Sort_Ascending という名前の Boolean 型メタデータエクステンションを作成します。マッピング内でカスタムトランスフォーメーションを使用する場合、プロシージャでのデータのソート方法に応じて、メタデータエクステンションに True または False を選択することができます。

カスタムトランスフォーメーションでプロパティを定義する場合、INFA_CTGetAllPropertyNamesM()など、すべてのプロパティ名を取得する関数を使用して、[初期化プロパティ] タブおよび [メタデータエクステンション] タブで定義されているすべてのプロパティの名前にアクセスできます。INFA_CT_getExternalPropertyM()などの外部プロパティ取得関数を使用して、指定したプロパティ ID に対応するプロパティの名前と値にアクセスできます。

注: メタデータエクステンションと初期化プロパティを同じ名前で定義した場合、メタデータエクステンションの情報を返すのはプロパティ関数だけになります。

カスタムトランスフォーメーションプロセスの作成

32 ビットまたは 64 ビットの Integration Service マシンで実行されるカスタムトランスフォーメーションプロセスを作成できます。カスタムトランスフォーメーションプロセスを作成する場合は、次の手順に従います。

1. Transformation Developer で、再利用可能なカスタムトランスフォーメーションを作成します。または Mapping Designer か Mapplet Designer で、再利用不可能なカスタムトランスフォーメーションを作成します。
2. プロセスのテンプレートコードを作成します。
プロセスコードを生成するときは、Designer によってカスタムトランスフォーメーションからの情報を使用して C 言語のソースコードファイルとメイクファイルが作成されます。
3. C 言語のファイルを修正して、プロセスのロジックを追加します。
4. C/C++ コンパイラを使用してソースコードファイルをコンパイルして DLL または共有ライブラリにリンクし、Integration Service マシンにコピーします。
5. カスタムトランスフォーメーションでマッピングを作成します。
6. ワークフローでセッションを実行します。

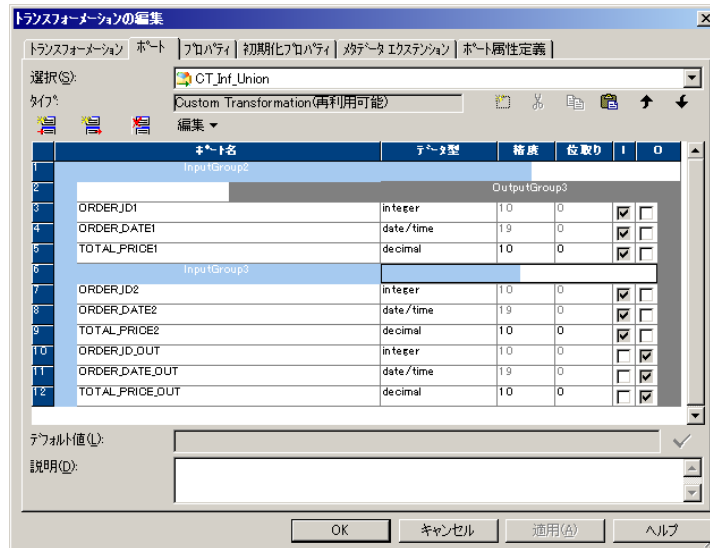
この節には、このプロセスを実演する例が示されています。この節での手順では、2 つの入力グループと 1 つの出力グループが含まれるカスタムトランスフォーメーションを作成します。カスタムトランスフォーメーションプロセスは、カスタムトランスフォーメーションが 2 つの入力グループと 1 つの出力グループを使用することを検査します。また、すべてのグループのポートの数が等しいこと、およびすべてのグループのポートのデータタイプが同じであることも検査します。この手続きは、各入力グループのデータ行を受け取り、すべての行を出力グループに出力します。

手順 1. カスタムトランスフォーメーションの作成

カスタムトランスフォーメーションを作成する手順

1. Transformation Developer で、[トランスフォーメーション] - [作成] を選択します。
2. [トランスフォーメーションの作成] ダイアログボックスで [カスタムトランスフォーメーション] を選択し、トランスフォーメーションの名前を入力して [作成] をクリックします。
共有体の例では、トランスフォーメーションの名前として CT_Inf_Union と入力します。
3. [アクティブ/パッシブ] ダイアログボックスで、トランスフォーメーションをパッシブまたはアクティブトランスフォーメーションのどちらとして作成するか選択し、[OK] をクリックします。
共有体の例では、[アクティブ] を選択します。
4. [完了] をクリックして [トランスフォーメーションの作成] ダイアログボックスを閉じます。

5. トランスフォーメーションを開き、[ポート] タブをクリックします。グループとポートを作成します。
必要に応じて、後でグループおよびポートを編集できます。
以下の図に、グループが2つある共有体トランスフォーメーションの例を示します。



この共有体の例では、グループ InputGroup1 と InputGroup2 を作成します。InputGroup1 の以下のポートを作成します。

ポート名	データ型	精度	スケール
ORDER_ID1	integer	10	0
ORDER_DATE1	date/time	19	0
TOTAL_PRICE1	decimal	10	2

InputGroup2 に対して以下のポートを作成します。

ポート名	データ型	精度	スケール	入出力
ORDER_ID2	integer	10	0	入力
ORDER_DATE2	date/time	19	0	入力
TOTAL_PRICE2	decimal	10	2	入力
ORDER_ID_OUT	integer	10	0	出力
ORDER_DATE_OUT	date/time	19	0	出力
TOTAL_PRICE_OUT	decimal	10	2	出力

6. [プロパティ] タブを選択して、モジュールと関数の識別子、および実行場所を入力します。[トレースレベル]、[パーティション化可能]、[入力ブロック]、[アクティブ]、[アップデートストラテジトランス

フォーメーション]、[トランスフォーメーション範囲]、[トランザクションの生成] の値/チェックボックスなど、ほかのトランスフォーメーションプロパティの属性を編集します。

この共有体の例では、次のプロパティを設定します。

プロパティ名	値
モジュール識別子	UnionDemo
関数識別子	共有体
実行時位置	\$PMExtProcDir
トレースレベル	標準
パーティション化可能	いいえ
入力はブロック	いいえ
アクティブ	はい
アップデートストラテジトランスフォーメーション	いいえ
トランスフォーメーション範囲	すべての入力
トランザクションの生成	いいえ

7. [メタデータエクステンション] タブをクリックして、エクスターナルプロシージャが初期化で必要とするプロパティなどのメタデータエクステンションを入力します。

共有体の例では、メタデータエクステンションは作成しないでください。

8. [ポート属性定義] タブをクリックし、必要に応じてポート属性を作成します。

共有体の例では、ポート属性は作成しないでください。

これで、プロシージャを呼び出すカスタムトランスフォーメーションが作成できました。次の手順では C ファイルを生成します。

手順 2. C ファイルの生成

カスタムトランスフォーメーションの作成後、ソースコードファイルを生成します。Designer は、ファイル名を小文字で生成します。

カスタムトランスフォーメーションプロシージャのコードを生成する手順

1. Transformation Developer でトランスフォーメーションを選択し、[トランスフォーメーション] - [外部プロシージャコードの生成] を選択します。
2. 作成した手続きを選択します。Designer は、<モジュール名>.<手続き名>のフォーマットで手続きを一覧表示します。

共有体の例では、UnionDemo.Union を選択します。

3. ファイルの生成先ディレクトリを指定し、[生成] をクリックします。

共有体の例では、<クライアントインストールディレクトリ>/TX を選択します。

指定したディレクトリの下に、<モジュール名>サブディレクトリが作成されます。共有体の例では、<クライアントインストールディレクトリ>/TX/UnionDemo が作成されます。また、以下のファイルを作成します。

- m_UnionDemo.c
- m_UnionDemo.h
- p_Union.c
- p_Union.h
- makefile.aix (32 ビット)、makefile.aix64 (64 ビット)、makefile.hp (32 ビット)、makefile.hp64 (64 ビット)、makefile.hpparisc64、makefile.linux (32 ビット)、makefile.sol (32 ビット)

手順 3. トランスフォーメーションロジックでコードを記述する

手続きの C ファイルを記述する必要があります。オプションとして、モジュールの C ファイルを記述することもできます。共有体の例では、手続きの C ファイルだけを記述します。モジュールの C ファイルを記述する必要はありません。

手続きの C ファイルを記述するには：

1. 手続きの p_<手続き名>.c を開きます。
共有体の例では、p_Union.c を開きます。
2. 手続きの C コードを入力します。
3. 変更したファイルを保存します。

共有体の例では、以下のコードを使用します。

```
/*
*****
* Custom Transformation p_union Procedure File
*
* This file contains code that functions that will be called by the main
* server executable.
*
* for more information on these files,
* see $(INFA_HOME)/ExtProc/include/Readme.txt
*****
*/

/*
* INFORMATICA 'UNION DEMO' developed using the API for custom
* transformations.

* File Name: p_Union.c
*
* An example of a custom transformation ('Union') using PowerCenter
*
* The purpose of the 'Union' transformation is to combine pipelines with the
* same row definition into one pipeline (i.e. union of multiple pipelines).
* [ Note that it does not correspond to the mathematical definition of union
* since it does not eliminate duplicate rows.]
*
* This example union transformation allows N input pipelines ( each
* corresponding to an input group) to be combined into one pipeline.
*
* To use this transformation in a mapping, the following attributes must be
* true:
* a. The transformation must have >= 2 input groups and only one output group.
* b. In the Properties tab set the following properties:
*   i. Module Identifier: UnionDemo
*   ii. Function Identifier: Union
*   iii. Inputs May Block: Unchecked
*   iv. Is Active: Checked
*   v. Update Strategy Transformation: Unchecked
*/
```

```

*      vi. Transformation Scope: All
*      vii. Generate Transaction: Unchecked *
*
*      * This version of the union transformation does not provide code for
*      * changing the update strategy or for generating transactions.
* c. The input groups and the output group must have the same number of ports
*    and the same datatypes. This is verified in the initialization of the
*    module and the session is failed if this is not true.
* d. The transformation can be used in multiple number of times in a Target
*    Load Order Group and can also be contained within multiple partitions.
*/
/*****
*****
***** Includes
*****
*****
*****
*****
*****
***** Forward Declarations
*****
*****
INFA_STATUS validateProperties(const INFA_CT_PARTITION_HANDLE* partition);
*****
***** Functions
*****
*****
*****
*****
***** Function: p_union_procInit
*****
Description: Initialization for the procedure. Returns INFA_SUCCESS if
procedure initialization succeeds, else return INFA_FAILURE.

Input: procedure - the handle for the procedure
Output: None
Remarks: This function will get called once for the session at
initialization time. It will be called after the moduleInit function.
*****
*****
INFA_STATUS p_union_procInit( INFA_CT_PROCEDURE_HANDLE procedure)
{
    const INFA_CT_TRANSFORMATION_HANDLE* transformation = NULL;
    const INFA_CT_PARTITION_HANDLE* partition = NULL;
    size_t nTransformations = 0, nPartitions = 0, i = 0;

    /* Log a message indicating beginning of the procedure initialization */
    INFA_CTLogMessageM( eESL_LOG,
        "union_demo: Procedure initialization started ..." );

    INFA_CTChangeStringMode( procedure, eASM_MBCS );

    /* Get the transformation handles */
    transformation = INFA_CTGetChildrenHandles( procedure,
        &nTransformations,
        TRANSFORMATIONTYPE);

    /* For each transformation verify that the 0th partition has the correct
    * properties. This does not need to be done for all partitions since rest
    * of the partitions have the same information */
    for (i = 0; i < nTransformations; i++)
    {
        /* Get the partition handle */
        partition = INFA_CTGetChildrenHandles(transformation[i],
            &nPartitions, PARTITIONTYPE );

        if (validateProperties(partition) != INFA_SUCCESS)
        {
            INFA_CTLogMessageM( eESL_ERROR,

```

```

                                "union_demo: Failed to validate attributes of "
                                "the transformation");
        return INFA_FAILURE;
    }
}

INFA_CTLogMessageM( eESL_LOG,
    "union_demo: Procedure initialization completed." );

return INFA_SUCCESS;
}

/*****
Function: p_union_procDeinit

Description: Deinitialization for the procedure. Returns INFA_SUCCESS if
procedure deinitialization succeeds, else return INFA_FAILURE.

Input: procedure - the handle for the procedure
Output: None
Remarks: This function will get called once for the session at
deinitialization time. It will be called before the moduleDeinit
function.
*****/

INFA_STATUS p_union_procDeinit( INFA_CT_PROCEDURE_HANDLE procedure,
                                INFA_STATUS sessionStatus )
{
    /* Do nothing ... */
    return INFA_SUCCESS;
}

/*****
Function: p_union_partitionInit

Description: Initialization for the partition. Returns INFA_SUCCESS if
partition deinitialization succeeds, else return INFA_FAILURE.

Input: partition - the handle for the partition
Output: None
Remarks: This function will get called once for each partition for each
transformation in the session.
*****/

INFA_STATUS p_union_partitionInit( INFA_CT_PARTITION_HANDLE partition )
{
    /* Do nothing ... */
    return INFA_SUCCESS;
}

/*****
Function: p_union_partitionDeinit

Description: Deinitialization for the partition. Returns INFA_SUCCESS if
partition deinitialization succeeds, else return INFA_FAILURE.

Input: partition - the handle for the partition
Output: None
Remarks: This function will get called once for each partition for each
transformation in the session.
*****/

INFA_STATUS p_union_partitionDeinit( INFA_CT_PARTITION_HANDLE partition )
{
    /* Do nothing ... */
    return INFA_SUCCESS;
}

/*****
Function: p_union_inputRowNotification

```

Description: Notification that a row needs to be processed for an input group in a transformation for the given partition. Returns INFA_ROWSUCCESS if the input row was processed successfully, INFA_ROWFAILURE if the input row was not processed successfully and INFA_FATALERROR if the input row causes the session to fail.

Input: partition - the handle for the partition for the given row
group - the handle for the input group for the given row

Output: None

Remarks: This function is probably where the meat of your code will go, as it is called for every row that gets sent into your transformation.

*****/

```
INFA_ROWSTATUS p_union_inputRowNotification( INFA_CT_PARTITION_HANDLE partition,
                                             INFA_CT_INPUTGROUP_HANDLE inputGroup )
```

```
{
    const INFA_CT_OUTPUTGROUP_HANDLE* outputGroups = NULL;
    const INFA_CT_INPUTPORT_HANDLE* inputGroupPorts = NULL;
    const INFA_CT_OUTPUTPORT_HANDLE* outputGroupPorts = NULL;
    size_t nNumInputPorts = 0, nNumOutputGroups = 0,
          nNumPortsInOutputGroup = 0, i = 0;

    /* Get the output group port handles */
    outputGroups = INFA_CTGetChildrenHandles(partition,
                                             &nNumOutputGroups,
                                             OUTPUTGROUPTYPE);

    outputGroupPorts = INFA_CTGetChildrenHandles(outputGroups[0],
                                                  &nNumPortsInOutputGroup,
                                                  OUTPUTPORTTYPE);

    /* Get the input groups port handles */
    inputGroupPorts = INFA_CTGetChildrenHandles(inputGroup,
                                                  &nNumInputPorts,
                                                  INPUTPORTTYPE);

    /* For the union transformation, on receiving a row of input, we need to
     * output that row on the output group. */
    for (i = 0; i < nNumInputPorts; i++)
    {
        INFA_CTSetData(outputGroupPorts[i],
                      INFA_CTGetDataVoid(inputGroupPorts[i]));

        INFA_CTSetIndicator(outputGroupPorts[i],
                          INFA_CTGetIndicator(inputGroupPorts[i]) );

        INFA_CTSetLength(outputGroupPorts[i],
                        INFA_CTGetLength(inputGroupPorts[i]) );
    }

    /* We know there is only one output group for each partition */
    return INFA_CTOutputNotification(outputGroups[0]);
}
```

```
/* *****
Function: p_union_eofNotification
```

Description: Notification that the last row for an input group has already been seen. Return INFA_FAILURE if the session should fail as a result of seeing this notification, INFA_SUCCESS otherwise.

Input: partition - the handle for the partition for the notification
group - the handle for the input group for the notification

Output: None

*****/

```
INFA_STATUS p_union_eofNotification( INFA_CT_PARTITION_HANDLE partition,
                                     INFA_CT_INPUTGROUP_HANDLE group)
```



```

{
    INFA_CTLogMessageM( eESL_LOG,
        "union_demo: An input group received an EOF notification");

    return INFA_SUCCESS;
}

/*****
    Function: p_union_dataBdryNotification

    Description: Notification that a transaction has ended. The data
    boundary type can either be commit or rollback.
    Return INFA_FAILURE if the session should fail as a result of
    seeing this notification, INFA_SUCCESS otherwise.

    Input: partition - the handle for the partition for the notification
           transactionType - commit or rollback
    Output: None
    *****/

INFA_STATUS p_union_dataBdryNotification ( INFA_CT_PARTITION_HANDLE partition,
                                           INFA_CT_DATABDRY_TYPE transactionType)
{
    /* Do nothing */
    return INFA_SUCCESS;
}

/* Helper functions */

/*****
    Function: validateProperties

    Description: Validate that the transformation has all properties expected
    by a union transformation, such as at least one input group, and only
    one output group. Return INFA_FAILURE if the session should fail since the
    transformation was invalid, INFA_SUCCESS otherwise.

    Input: partition - the handle for the partition
    Output: None
    *****/

INFA_STATUS validateProperties(const INFA_CT_PARTITION_HANDLE* partition)
{
    const INFA_CT_INPUTGROUP_HANDLE* inputGroups = NULL;
    const INFA_CT_OUTPUTGROUP_HANDLE* outputGroups = NULL;
    size_t nNumInputGroups = 0, nNumOutputGroups = 0;
    const INFA_CT_INPUTPORT_HANDLE** allInputGroupsPorts = NULL;
    const INFA_CT_OUTPUTPORT_HANDLE* outputGroupPorts = NULL;
    size_t nNumPortsInOutputGroup = 0;
    size_t i = 0, nTempNumInputPorts = 0;

    /* Get the input and output group handles */
    inputGroups = INFA_CTGetChildrenHandles(partition[0],
                                           &nNumInputGroups,
                                           INPUTGROUPTYPE);

    outputGroups = INFA_CTGetChildrenHandles(partition[0],
                                           &nNumOutputGroups,
                                           OUTPUTGROUPTYPE);

    /* 1. Number of input groups must be >= 2 and number of output groups must
    * be equal to one. */
    if (nNumInputGroups < 1 || nNumOutputGroups != 1)
    {
        INFA_CTLogMessageM( eESL_ERROR,
            "UnionDemo: There must be at least two input groups "
            "and only one output group");
        return INFA_FAILURE;
    }
}

```

```

/* 2. Verify that the same number of ports are in each group (including
 * output group). */
outputGroupPorts = INFA_CTGetChildrenHandles(outputGroups[0],
                                              &nNumPortsInOutputGroup,
                                              OUTPUTPORTTYPE);

/* Allocate an array for all input groups ports */
allInputGroupsPorts = malloc(sizeof(INFA_CT_INPUTPORT_HANDLE*) *
                              nNumInputGroups);

for (i = 0; i < nNumInputGroups; i++)
{
    allInputGroupsPorts[i] = INFA_CTGetChildrenHandles(inputGroups[i],
                                                        &nTempNumInputPorts,
                                                        INPUTPORTTYPE);

    if ( nNumPortsInOutputGroup != nTempNumInputPorts)
    {
        INFA_CTLogMessageM( eESL_ERROR,
                            "UnionDemo: The number of ports in all input and "
                            "the output group must be the same.");
        return INFA_FAILURE;
    }
}

free(allInputGroupsPorts);

/* 3. Datatypes of ports in input group 1 must match data types of all other
 * groups.
 * TODO:*/
return INFA_SUCCESS;
}

```

手順 4。モジュールの構築

モジュールの構築は Windows または UNIX プラットフォームで行うことができます。

以下の表に、モジュールを構築する場合の各プラットフォームでのライブラリファイル名を示します。

プラットフォーム	モジュールファイル名
Windows	<module_identifier>.dll
AIX	lib<module_identifier>.a
Linux	lib<module_identifier>.so
Solaris	lib<module_identifier>.so

Windows でのモジュールの構築

Windows では、Microsoft Visual C++を使用してモジュールを構築します。

Windows でモジュールを構築するには：

1. Visual C++を起動します。
2. [ファイル] - [新規] をクリックします。
3. [新規作成] ダイアログボックスで、[プロジェクト] タブをクリックし、[Win32 Dynamic-Link Library] オプションを選択します。
4. 場所を入力します。

共有体の例では、<クライアントインストールディレクトリ>/TX/UnionDemo を入力します。

5. プロジェクト名を入力します。

カスタムトランスフォーメーションで指定したモジュール名をプロジェクト名として使用する必要があります。共有体の例では、UnionDemo を入力します。

6. [OK] をクリックします。

Visual C++は、プロジェクトのコンポーネントの定義を行うウィザードを表示します。

7. ウィザードでは、[空の DLL プロジェクト] を選択して、[終了] をクリックします。[新規プロジェクト情報] ダイアログボックスで [OK] をクリックします。

Visual C++は、指定されたディレクトリにプロジェクトファイルを作成します。

8. [プロジェクト] - [プロジェクトへの追加] - [ファイル] をクリックします。

9. 1 つ上のディレクトリレベルへ移動します。このディレクトリにはユーザーが作成した手続きが含まれています。すべての.c ファイルを選択して [OK] をクリックします。

共有体の例では、以下のファイルを追加します。

- m_UnionDemo.c
- p_Union.c

10. [プロジェクト] - [設定] をクリックします。

11. [C/C++] タブをクリックしてから、[カテゴリ] フィールドで [プリプロセッサ] を選択します。

12. [インクルードファイルのパス] フィールドに、以下のパスを入力して [OK] をクリックします。

..; <PowerCenter_install_dir>\extproc\include\ct

13. Build > Build <module_name>.dll をクリックするか、F7 キーを押してプロジェクトを構築します。

Visual C++は DLL を作成し、プロジェクトディレクトリ下のデバッグディレクトリまたはリリースディレクトリに格納します。

UNIX でのモジュールの構築

UNIX では、任意の C コンパイラを使用してモジュールを構築します。

UNIX でモジュールを構築するには：

1. Designer が作成したすべての C ファイルとメイクファイルを UNIX マシンにコピーします。

注: 統合サービスマシンとは別のマシンに共有ライブラリを構築する場合は、その構築用マシンの以下のディレクトリにもファイルをコピーする必要があります。

<PowerCenter_install_dir>\ExtProc\include\ct

共有体の例では、<クライアントインストールディレクトリ>/TX/UnionDemo にすべてのファイルをコピーします。

2. INFA_HOME 環境変数を統合サービスのインストールディレクトリに設定します。

注: INFA_HOME 環境変数に間違ったディレクトリパスを指定すると、統合サービスは起動しません。

3. 以下の表のコマンドを入力し、プロジェクトを作成します。

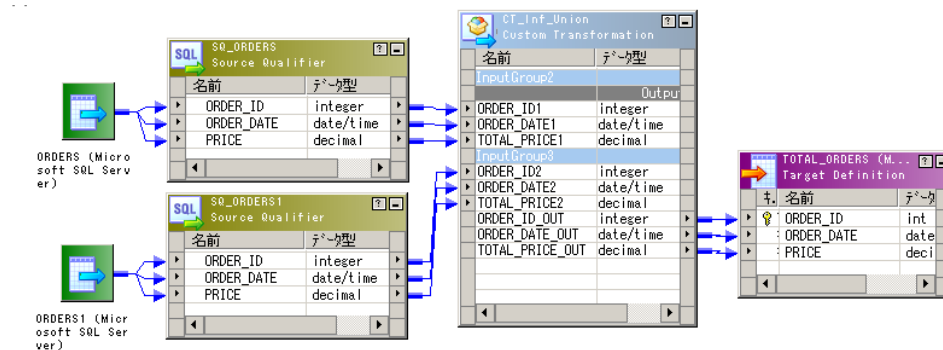
UNIX バージョン	コマンド
AIX (32 ビット)	make -f makefile.aix
AIX (64 ビット)	make -f makefile.aix64
Linux	make -f makefile.linux
Solaris	make -f makefile.sol

手順 5. マッピングの作成

Mapping Designer で、カスタムトランスフォーメーションを使用するマッピングを作成します。

このマッピングでは、同じポートとデータタイプを持つ 2 つのソースが、カスタムトランスフォーメーション内の 2 つの入力グループに接続されています。カスタムトランスフォーメーションは、両方のソース行を取り込み、そのすべてを 1 つの出力グループを通じて出力します。出力グループは、入力グループと同じポートとデータ型を持ちます。

共有体の例では、以下の図に示すようなマッピングを作成します。



手順 6. ワークフローでのセッションの実行

Integration Service でセッションを実行すると、カスタムトランスフォーメーションで指定されている実行場所ので共有ライブラリまたは DLL が検索されます。

ワークフローでセッションを実行するには：

1. Workflow Manager でワークフローを作成します。
2. ワークフローでこのマッピング用のセッションを作成します。
3. 共有ライブラリまたは DLL を実行場所のディレクトリにコピーします。
4. セッションを含むワークフローを実行します。

Integration Service はプロシージャに関連付けられているカスタムトランスフォーメーションをロードするときに、DLL または共有ライブラリをロードし、定義されているプロシージャを呼び出します。

第 4 章

カスタムトランスフォーメーション関数

この章では、以下の項目について説明します。

- [カスタムトランスフォーメーション関数の概要, 85 ページ](#)
- [関数リファレンス, 86 ページ](#)
- [行に関する作業, 89 ページ](#)
- [生成済み関数, 90 ページ](#)
- [API 関数, 96 ページ](#)
- [配列ベース API 関数, 120 ページ](#)

カスタムトランスフォーメーション関数の概要

カスタムトランスフォーメーションは、Designer の外部に作成した手続きと連携して動作し、PowerCenter の機能を拡張します。カスタムトランスフォーメーション関数を使って、カスタムトランスフォーメーションに関連付けられている手続き内のトランスフォーメーションロジックを開発することができます。

PowerCenter は、生成済み関数および API 関数と呼ばれる 2 つの関数のセットを提供しています。

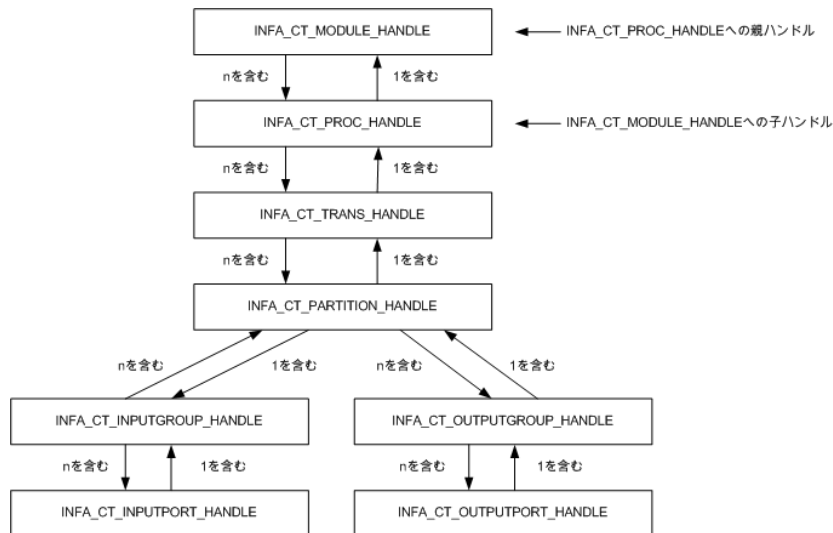
Integration Service は、生成済み関数を使ってプロシージャとのインタフェースをとります。カスタムトランスフォーメーションを作成してソースコードファイルを生成すると、Designer はそのファイルに生成済み関数を取り込みます。手続きコードで API 関数を使って、トランスフォーメーションロジックを開発します。

プロシージャコードを作成するときに、Integration Service から行をブロック単位で受け取るとか、一度に 1 行ずつ受け取るとかを設定できます。行をブロック単位で受け取って処理すると、プロシージャのパフォーマンスを高めることができます。

ハンドルに関する作業

ほとんどの関数は、INFA_CT_PARTITION_HANDLE といったハンドルに関連付けられています。このような関数の最初のパラメータは、関数が効力を及ぼすハンドルです。カスタムトランスフォーメーションのハンドルは、相互に階層リレーションを持ちます。親ハンドルは、子ハンドルに対して 1:n のリレーションを持ちます。

以下の図は、カスタムトランスフォーメーションのハンドルを示しています。



以下の表では、カスタムトランスフォーメーションのハンドルについて説明します。

ハンドル名	説明
INFA_CT_MODULE_HANDLE	DLL または共有ライブラリを表します。エクスターナルプロシージャは、それ自身の共有ライブラリまたは DLL 内のモジュールハンドルにのみアクセスできます。他の共有ライブラリまたは DLL 内のモジュールハンドルにはアクセスできません。
INFA_CT_PROC_HANDLE	共有ライブラリまたは DLL 内の特定の手続きを表します。 複数のカスタムトランスフォーメーションが参照するプロシージャに影響を与える関数を記述する必要がある場合、このハンドルを使用できます。
INFA_CT_TRANS_HANDLE	セッション内の特定のカスタムトランスフォーメーションインスタンスを表します。
INFA_CT_PARTITION_HANDLE	特定のカスタムトランスフォーメーションインスタンス内の特定のパーティションを表します。
INFA_CT_INPUTGROUP_HANDLE	パーティション内の入力グループを表します。
INFA_CT_INPUTPORT_HANDLE	パーティション内の入力グループ内の入力ポートを表します。
INFA_CT_OUTPUTGROUP_HANDLE	パーティション内の出力グループを表します。
INFA_CT_OUTPUTPORT_HANDLE	パーティション内の出力グループ内の出力ポートを表します。

関数リファレンス

カスタムトランスフォーメーション関数には、生成済み関数と API 関数があります。

以下の表に、カスタムトランスフォーメーションの生成済み関数を示します。

関数	説明
m_<モジュール名>_moduleInit()	モジュール初期化関数。
p_<手続き名>_proclnit()	手続き初期化関数。
p_<手続き名>_partitionInit()	パーティション初期化関数。
p_<手続き名>_inputRowNotification()	入力行通知関数。
p_<手続き名>_dataBdryNotification()	データ境界通知関数。
p_<手続き名>_eofNotification()	エンドオブファイル通知関数。
p_<手続き名>_partitionDeinit()	パーティション初期化解除関数。
p_<手続き名>_procedureDeinit()	手続き初期化解除関数。
m_<モジュール名>_moduleDeinit()	モジュール初期化解除関数。

以下の表に、カスタムトランスフォーメーションの API 関数を示します。

関数	説明
INFA_CTSetDataAccessMode()	データアクセスモード設定関数。
INFA_CTGetAncestorHandle()	上位ハンドル取得関数。
INFA_CTGetChildrenHandles()	子ハンドル取得関数。
INFA_CTGetInputPortHandle()	入力ポートハンドル取得関数。
INFA_CTGetOutputPortHandle()	出力ポートハンドル取得関数。
INFA_CTGetInternalProperty<データタイプ>()	内部プロパティ取得関数。
INFA_CTGetAllPropertyNamesM()	MBCS モードでの全プロパティ名取得関数。
INFA_CTGetAllPropertyNamesU()	Unicode モードでの全プロパティ名取得関数。
INFA_CTGetExternalProperty<データタイプ>M()	MBCS での外部プロパティ取得関数。
INFA_CTGetExternalProperty<データタイプ>U()	Unicode での外部プロパティ取得関数。
INFA_CTRebindInputDataType()	入力ポートデータタイプの再関連付け関数。
INFA_CTRebindOutputDataType()	出力ポートデータタイプの再関連付け関数。
INFA_CTGetData<データタイプ>()	データ取得関数。
INFA_CTSetData()	データ設定関数。

関数	説明
INFA_CTGetIndicator()	インジケータ取得関数。
INFA_CTSetIndicator()	インジケータ設定関数。
INFA_CTGetLength()	長さ取得関数。
INFA_CTSetLength()	長さ設定関数。
INFA_CTSetPassThruPort()	パススルーポート設定関数。
INFA_CTOutputNotification()	出力通知関数。
INFA_CTDataBdryOutputNotification()	データ境界出力通知関数。
INFA_CTGetErrorMsgU()	Unicode でのエラーメッセージ取得関数。
INFA_CTGetErrorMsgM()	MBCS でのエラーメッセージ取得関数。
INFA_CTLogMessageU()	Unicode でのセッションログへのメッセージログ関数。
INFA_CTLogMessageM()	MBCS でのセッションログへのメッセージログ関数。
INFA_CTIncrementErrorCount()	エラーカウントインクリメント関数。
INFA_CTIsterminateRequested()	終了要求の確認関数。
INFA_CTBlockInputFlow()	入力グループブロック関数。
INFA_CTUnblockInputFlow()	入力グループブロック解除関数。
INFA_CTSetUserDefinedPtr()	ユーザー定義ポインタ設定関数。
INFA_CTGetUserDefinedPtr()	ユーザー定義ポインタ取得関数。
INFA_CTChangeStringMode()	文字列モード変更関数。
INFA_CTSetDataCodePageID()	データコードページ ID 設定関数。
INFA_CTGetRowStrategy()	行ストラテジ取得関数。
INFA_CTSetRowStrategy()	行ストラテジ設定関数。
INFA_CTChangeDefaultRowStrategy()	トランスフォーメーションのデフォルトの行ストラテジを変更します。

以下の表に、カスタムトランスフォーメーションの配列ベースの関数を示します。

関数	説明
INFA_CTAGetInputRowMax()	最大入力行数取得関数。
INFA_CTAGetOutputRowMax()	最大出力行数取得関数。

関数	説明
INFA_CTASetOutputRowMax()	最大出力行数設定関数。
INFA_CTAGetNumRows()	行数取得関数。
INFA_CTASetNumRows()	行数設定関数。
INFA_CTAIsRowValid()	行有効検証関数。
INFA_CTAGetData<データタイプ>()	データ取得関数。
INFA_CTAGetIndicator()	インジケータ取得関数。
INFA_CTASetData()	データ設定関数。
INFA_CTAGetRowStrategy()	行ストラテジ取得関数。
INFA_CTASetRowStrategy()	行ストラテジ設定関数。
INFA_CTASetInputErrorRowM()	MBCS に対する入力エラー行設定関数。
INFA_CTASetInputErrorRowU()	Unicode に対する入力エラー行設定関数。

行に関する作業

Integration Service ではカスタムトランスフォーメーションプロシージャに対して 1 行ずつ渡したり、配列内の行をブロック単位で渡したりすることができます。手続きコードを作成して、手続きが 1 行ずつ受け取るか、ブロック単位で受け取るかを指定できます。ブロック単位で手続きが行を受け取ると、パフォーマンスが向上します。

- Integration Service やプロシージャが行う関数呼び出しの回数を減らすことができます。Integration Service による入力行通知関数の呼び出し回数が減り、プロシージャによる出力通知関数の呼び出し回数が減ります。
- データに対するメモリアクセススペースの局所性を高めることができます。
- 手続きコードを作成し、データの各行ではなく、ブロック単位にアルゴリズムを実行するようにできます。

デフォルトでは、手続きは一度に 1 行のデータを受け取ります。ブロック単位で行を受け取るようにするには、INFA_CTSetDataAccessMode()関数を使用してデータアクセスモードを配列ベースに変更する必要があります。データアクセスモードが配列ベースの場合、配列ベースのデータ操作関数と行更新方式関数を使用してデータへのアクセスとデータの出力を行う必要があります。データアクセスモードが行ベースの場合、行ベースのデータ操作関数と行更新方式関数を使用してデータへのアクセスとデータの出力を行う必要があります。

すべての配列ベース関数には、接頭語として INFA_CTA が使用されています。その他のすべての関数では接頭語として INFA_CT が使用されています。

手続きコードを作成してブロック単位で行にアクセスするには、以下の手順に従ってください。

1. 手続きの初期化中に INFA_CTSetDataAccessMode()を呼び出して、データアクセスモードを配列ベースに変更します。

2. パッシブなカスタムトランスフォーメーションを作成する場合は、プロシージャの初期化中に `INFA_CTSetPassThruPort()` を呼び出し、入出力ポート用のデータを渡すこともできます。
データブロックがカスタムトランスフォーメーションプロシージャに渡されると、Integration Service は `p_<プロシージャ名>_inputRowNotification()` をデータブロックごとに呼び出します。この関数内で残りの手順を実行します。
3. 入力行通知関数の入力グループハンドルを使用して `INFA_CTGetNumRows()` を呼び出し、現在のブロック内の行数を確認します。
4. 入力ポートハンドルを使用して `INFA_CTGetData<データタイプ>()` 関数を呼び出し、ブロック内の特定の行のデータを取得します。
5. `INFA_CTSetData` を呼び出し、ブロック内の行を出力します。
6. `INFA_CTOutputNotification()` を呼び出す前に、`INFA_CTSetNumRows()` を呼び出し、このプロシージャがブロック内に出力する行数を Integration Service に通知します。
7. `INFA_CTOutputNotification()` を呼び出します。

行ベースおよび配列ベースのデータアクセスモードに関するルールとガイドライン

手続きコードを作成し、行ベースまたは配列ベースのデータアクセスモードを使用するときには、以下の規則およびガイドラインに注意してください。

- 行ベースモードの場合、入力行通知関数に `INFA_ROWERROR` を返し、その関数で入力データ行にエラーが発生したことを示します。この場合、Integration Service は内部エラーカウントを増分します。
- 配列ベースモードの場合、入力行通知関数に `INFA_ROWERROR` を返さないようにします。Integration Service は、致命的なエラーとしてこれを処理します。ブロック内の行にエラーがあることを示す必要がある場合は、`INFA_CTSetInputErrorRowM()` 関数または `INFA_CTSetInputErrorRowU()` 関数を呼び出します。
- 行ベースモードでは、Integration Service は有効な行のみをプロシージャに渡します。
- 配列ベースモードでは、削除された行、フィルタリングされた行、エラーのある行など、入力ブロック内に無効な行が含まれている場合があります。`INFA_CTIsRowValid()` を呼び出して、ブロック内の行が有効かどうかを判断します。
- 配列ベースモードの場合、パッシブなカスタムトランスフォーメーションについては `INFA_CTSetNumRows()` を呼び出さないようにします。この関数は、アクティブなカスタムトランスフォーメーションで呼び出すことができます。
- 配列ベースモードの場合、`INFA_CTOutputNotification()` を 1 度だけ呼び出します。
- 配列ベースモードの場合、`INFA_CTSetPassThruPort()` はパッシブなカスタムトランスフォーメーションでしか呼び出すことができません。
- 配列ベースモードの場合、パッシブなカスタムトランスフォーメーションについては、エラー行も含めて出力ブロック内のすべての行を出力する必要があります。

生成済み関数

Designer を使ってプロシージャコードを生成すると、`m_<モジュール名>.c` および `p_<プロシージャ名>.c` ファイルに生成済み関数と呼ばれる一連の関数がインクルードされます。Integration Service は、生成済み関数

を使ってプロシージャとのインタフェースをとります。Integration Service は、セッション実行時にマッピング内の各ターゲットロード順グループに対して、これらの生成済み関数を以下の順で呼び出します。

1. 初期化関数
2. 通知関数
3. 初期化解除関数

初期化関数

Integration Service は最初に初期化関数を呼び出します。初期化関数を使用して、カスタムトランスフォーマーセッションにデータを渡す前に Integration Service に実行させる処理を記述します。初期化関数内にコードを記述すると、Integration Service はこれらの処理をモジュール、プロシージャ、またはパーティションに対して 1 回だけ実行するため、プロセスのオーバーヘッドが削減されます。

Designer は以下の初期化関数を生成します。

- m_<モジュール名>_moduleInit()
- p_<手続き名>_procInit()
- p_<手続き名>_partitionInit()

モジュール初期化関数

Integration Service はセッションの初期化に際して、セッション実行前のタスクを実行する前に、m_<モジュール名>_moduleInit()関数を呼び出します。Integration Service は他の関数を呼び出す前に、モジュールに対してこの関数を 1 回呼び出します。

Integration Service に、モジュールのロード時に特定の処理を実行させるには、その処理をこの関数に含めておく必要があります。たとえば、モジュール内の手続きからアクセスするグローバルな構造体を作成するコードを記述できます。

以下の構文を使用します。

```
INFA_STATUS m_<module_name>_moduleInit(INFA_CT_MODULE_HANDLE module);
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/出力	説明
module	INFA_CT_MODULE_HANDLE	入力	モジュールハンドル。

戻り値のデータタイプは INFA_STATUS です。戻り値には INFA_SUCCESS および INFA_FAILURE を使用します。関数が INFA_FAILURE を返すと、Integration Service はセッションに失敗します。

手続き初期化関数

Integration Service はセッションの初期化に際して、セッション実行前の作業を実行する前とモジュール初期化関数を実行した後に、p_<プロシージャ名>_procInit()関数を呼び出します。Integration Service は、モジュール内の各プロシージャに対してこの関数を 1 回ずつ呼び出します。

Integration Service に特定のプロシージャに関する処理を行わせる場合、この関数にコードを記述します。また、この手続き初期化関数に、API 関数のナビゲーション関数やプロパティ関数を入力することもできます。

以下の構文を使用します。

```
INFA_STATUS p_<proc_name>_procInit(INFA_CT_PROCEDURE_HANDLE procedure);
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
procedure	INFA_CT_PROCEDURE_HANDLE	入力	手続きハンドル。

戻り値のデータタイプは INFA_STATUS です。戻り値には INFA_SUCCESS および INFA_FAILURE を使用します。関数が INFA_FAILURE を返すと、Integration Service はセッションに失敗します。

パーティション初期化関数

Integration Service は、データをカスタムトランスフォーメーションに渡す前に、p_<プロシージャ名>_partitionInit()関数を呼び出します。Integration Service は、カスタムトランスフォーメーションインスタンスの各パーティションに対してこの関数を 1 回ずつ呼び出します。

カスタムトランスフォーメーションのパーティションをデータが通過する前に Integration Service に特定のプロセスを実行させるには、実行したい処理をこの関数に含めておく必要があります。

以下の構文を使用します。

```
INFA_STATUS p_<proc_name>_partitionInit(INFA_CT_PARTITION_HANDLE transformation);
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
トランスフォーメーション	INFA_CT_PARTITION_HANDLE	入力	パーティションハンドル。

戻り値のデータタイプは INFA_STATUS です。戻り値には INFA_SUCCESS および INFA_FAILURE を使用します。関数が INFA_FAILURE を返すと、Integration Service はセッションに失敗します。

注: カスタムトランスフォーメーションが各パーティションごとにスレッドを 1 つずつ必要とする場合は、パーティション初期化関数にスレッド特有の操作を含めることができます。

通知関数

Integration Service は、カスタムトランスフォーメーションにデータ行を渡すときに通知関数を呼び出します。

Designer は以下の通知関数を生成します。

- p_<手続き名>_inputRowNotification()
- p_<手続き名>_dataBdryRowNotification()
- p_<手続き名>_eofNotification()

注: カスタムトランスフォーメーションが各パーティションごとにスレッドを 1 つずつ必要とする場合は、通知関数にスレッド特有の操作を含めます。

入力行通知関数

Integration Service は、カスタムトランスフォーメーションに行または行ブロックを渡すときに、p_<プロシージャ名>_inputRowNotification()関数を呼び出します。この関数は、入力グループおよびパーティションが、入力グループハンドルおよびパーティションハンドルを通じてデータを受け取ったことを通知します。

以下の構文を使用します。

```
INFA_ROWSTATUS p_<proc_name>_inputRowNotification(INFA_CT_PARTITION_HANDLE Partition,  
INFA_CT_INPUTGROUP_HANDLE group);
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
partition	INFA_CT_PARTITION_HANDLE	入力	パーティションハンドル。
グループ	INFA_CT_INPUTGROUP_HANDLE	入力	入力グループハンドル。

戻り値のデータタイプは INFA_ROWSTATUS です。戻り値には以下の値を使います。

- **INFA_ROWSUCCESS**。関数がデータ行の処理に成功したことを示します。
- **INFA_ROWERROR**。関数がデータ行に関してエラーに遭遇したことを示します。この場合、Integration Service は内部エラーカウントを増分します。データアクセスモードが行のときは、この値のみを返します。

配列ベースモードで入力行通知関数が INFA_ROWERROR を返した場合、Integration Service はこれを致命的なエラーとして処理します。ブロック内の行にエラーがあることを示す必要がある場合は、INFA_CTASetInputErrorRowM()関数または INFA_CTASetInputErrorRowU()関数を呼び出します。
- **INFA_FATALERROR**。関数がデータ行またはデータブロックに関して致命的エラーに遭遇したことを示します。このとき、Integration Service はセッションに失敗します。

データ境界通知関数

Integration Service は、コミット行またはロールバック行をパーティションに渡す前に、p_<プロシージャ名>_dataBdryNotification()関数を呼び出します。

以下の構文を使用します。

```
INFA_STATUS p_<proc_name>_dataBdryNotification(INFA_CT_PARTITION_HANDLE transformation, INFA_CTDataBdryType  
dataBoundaryType);
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
トランスフォーメーション	INFA_CT_PARTITION_HANDLE	入力	パーティションハンドル。
dataBoundaryType	INFA_CTDataBdryType	入力	Integration Service は、dataBoundaryType パラメータに以下のいずれかの値を使用します。 - eBT_COMMIT - eBT_ROLLBACK

戻り値のデータタイプは INFA_STATUS です。戻り値には INFA_SUCCESS および INFA_FAILURE を使用します。関数が INFA_FAILURE を返すと、Integration Service はセッションに失敗します。

エンドオブファイル通知関数

Integration Service は、入力グループ内のパーティションに最後の行を渡した後で、p_<プロシージャ名>_eofNotification()関数を呼び出します。

以下の構文を使用します。

```
INFA_STATUS p_<proc_name>_eofNotification(INFA_CT_PARTITION_HANDLE transformation, INFA_CT_INPUTGROUP_HANDLE group);
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
トランスフォーメーション	INFA_CT_PARTITION_HANDLE	入力	パーティションハンドル。
グループ	INFA_CT_INPUTGROUP_HANDLE	入力	入力グループハンドル。

戻り値のデータタイプは INFA_STATUS です。戻り値には INFA_SUCCESS および INFA_FAILURE を使用します。関数が INFA_FAILURE を返すと、Integration Service はセッションに失敗します。

初期化解除関数

Integration Service は、カスタムトランスフォーメーションに関するデータを処理した後で、初期化解除関数を呼び出します。初期化解除関数を使用して、Integration Service がカスタムトランスフォーメーションにすべてのデータ行を渡した後に行う処理を記述します。

Designer は以下の初期化解除関数を生成します。

- p_<手続き名>_partitionDeinit()
- p_<手続き名>_procDeinit()
- m_<モジュール名>_moduleDeinit()

注: カスタムトランスフォーメーションがパーティションごとにスレッドを 1 つずつ必要とする場合は、初期化関数および初期化解除関数にスレッド特有の操作を含めることができます。

パーティション初期化解除関数

Integration Service は、p_<プロシージャ名>_eofNotification()関数または p_<プロシージャ名>_abortNotification()関数を呼び出した後で、p_<プロシージャ名>_partitionDeinit()関数を呼び出します。Integration Service は、カスタムトランスフォーメーションの各パーティションに対してこの関数を 1 回ずつ呼び出します。

以下の構文を使用します。

```
INFA_STATUS p_<proc_name>_partitionDeinit(INFA_CT_PARTITION_HANDLE partition);
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
partition	INFA_CT_PARTITION_HANDLE	入力	パーティションハンドル。

戻り値のデータタイプは INFA_STATUS です。戻り値には INFA_SUCCESS および INFA_FAILURE を使用します。関数が INFA_FAILURE を返すと、Integration Service はセッションに失敗します。

注: カスタムトランスフォーメーションが各パーティションごとにスレッドを 1 つずつ必要とする場合は、パーティション初期化解除関数にスレッド特有の操作を含めることができます。

手続き初期化解除関数

Integration Service は、マッピング内でこのプロシージャを使用する各カスタムトランスフォーメーションインスタンスのすべてのパーティションに対して、p_<プロシージャ名>_partitionDeinit()関数を呼び出した後で p_<プロシージャ名>_procDeinit()関数を呼び出します。

以下の構文を使用します。

```
INFA_STATUS p_<proc_name>_procDeinit(INFA_CT_PROCEDURE_HANDLE procedure, INFA_STATUS sessionStatus);
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
procedure	INFA_CT_PROCEDURE_HANDLE	入力	手続きハンドル。
sessionStatus	INFA_STATUS	入力	Integration Service は、sessionStatus パラメータに対して以下のいずれかの値を使用します。 <ul style="list-style-type: none">- INFA_SUCCESS。セッションが成功したことを示します。- INFA_FAILURE。セッションが失敗したことを示します。

戻り値のデータタイプは INFA_STATUS です。戻り値には INFA_SUCCESS および INFA_FAILURE を使用します。関数が INFA_FAILURE を返すと、Integration Service はセッションに失敗します。

モジュール初期化解除関数

Integration Service は、セッション実行後のタスクを実行した後で、m_<モジュール名>_moduleDeinit()関数を呼び出します。PowerCenter Server は他の関数をすべて呼び出した後で、モジュールに対してこの関数を 1 回呼び出します。

以下の構文を使用します。

```
INFA_STATUS m_<module_name>_moduleDeinit(INFA_CT_MODULE_HANDLE module, INFA_STATUS sessionStatus);
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
module	INFA_CT_MODULE_HANDLE	入力	モジュールハンドル。
sessionStatus	INFA_STATUS	入力	Integration Service は、sessionStatus パラメータに対して以下のいずれかの値を使用します。 - INFA_SUCCESS。セッションが成功したことを示します。 - INFA_FAILURE。セッションが失敗したことを示します。

戻り値のデータタイプは INFA_STATUS です。戻り値には INFA_SUCCESS および INFA_FAILURE を使用します。関数が INFA_FAILURE を返すと、Integration Service はセッションに失敗します。

API 関数

PowerCenter には、トランスフォーメーションロジックの開発に使用できる一連の API 関数が用意されています。Designer は、ソースコードファイルを生成するときに、ソースコードに生成済み関数を取り込みます。トランスフォーメーションロジックを実装するには、コードに API 関数を追加します。プロシージャは、API 関数を使って Integration Service とのインタフェースをとります。API 関数は手続きの C ファイル内に記述する必要があります。オプションとして、モジュールの C ファイルを記述することもできます。

Informatica には、以下の API 関数のグループが用意されています。

- データアクセスモード設定
- ナビゲーション
- プロパティ
- データタイプの再関連付け
- データ操作（行ベースモード）
- パススルーポートの設定
- 出力通知
- データ境界出力通知
- エラー
- セッションログメッセージ
- エラーカウンターの増分
- 終了要求検査
- ブロック
- ポインタ
- 文字列モード変更
- データコードページ設定
- 行ストラテジ（行ベースモード）

- デフォルト行ストラテジ変更

Informatica は配列ベースの API 関数も提供しています。

データアクセスモード設定関数

デフォルトでは、Integration Service はデータを一度に 1 行ずつカスタムトランスフォーメーションプロセスに渡します。ただし、データアクセスモードを配列ベースに変更するには、INFA_CTSetDataAccessMode()関数を使用します。データアクセスモードを配列ベースに設定すると、Integration Service は、複数の行を配列内のブロックとしてプロセスに渡します。

データアクセスモードを配列ベースに設定した場合には、配列ベースバージョンのデータ操作関数と行更新方式関数を使用する必要があります。行ベースのデータ操作関数または行更新方式関数を使用している場合に配列モードに切り替えると、予想外の結果が生じます。たとえば、DLL や共有ライブラリがクラッシュすることがあります。

これらの関数は、手続き初期化関数でのみ使用できます。

手続きコードにこの関数を使用しない場合、データアクセスモードは行ベースになります。ただし、データアクセスモードを行ベースにする場合は、この関数を組み込んでアクセスモードを行ベースに設定してください。

以下の構文を使用します。

```
INFA_STATUS INFA_CTSetDataAccessMode( INFA_CT_PROCEDURE_HANDLE procedure, INFA_CT_DATA_ACCESS_MODE mode );
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
procedure	INFA_CT_PROCEDURE_HANDLE	入力	手続き名。
mode	INFA_CT_DATA_ACCESS_MODE	入力	データアクセスモード。 モードパラメータには以下の値を使います。 - eDA_ROW - eDA_ARRAY

ナビゲーション関数

ナビゲーション関数は、手続きでハンドル階層内を移動する場合に使用します。

PowerCenter には、以下のナビゲーション関数が用意されています。

- INFA_CTGetAncestorHandle()
- INFA_CTGetChildrenHandles()
- INFA_CTGetInputPortHandle()
- INFA_CTGetOutputPortHandle()

上位ハンドル取得関数

INFA_CTGetAncestorHandle()関数は、手続きで、所定のハンドルの親ハンドルにアクセスする場合に使用します。

以下の構文を使用します。

```
INFA_CT_HANDLE INFA_CTGetAncestorHandle(INFA_CT_HANDLE handle, INFA_CTHandleType returnHandleType);
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
handle	INFA_CT_HANDLE	入力	ハンドル名。
returnHandleType	INFA_CTHandleType	入力	返されるハンドルの種類。 returnHandleType パラメータには以下の値を使います。 <ul style="list-style-type: none">- PROCEDURETYPE- TRANSFORMATIONTYPE- PARTITIONTYPE- INPUTGROUPTYPE- OUTPUTGROUPTYPE- INPUTPORTTYPE- OUTPUTPORTTYPE

handle パラメータには、手続きでアクセスしたい親のハンドルを指定します。関数内で有効なハンドルが指定されている場合、Integration Service は INFA_CT_HANDLE を返します。そうでない場合は NULL 値を返します。

コンパイル時にエラーが発生しないように、ハンドル名を戻り値に設定するよう手続きを記述する必要があります。

たとえば、次のようなコードを入力できます。

```
INFA_CT_MODULE_HANDLE module = INFA_CTGetAncestorHandle(procedureHandle, INFA_CT_HandleType);
```

子ハンドル取得関数

INFA_CTGetChildrenHandles()関数は、手続きで、所定のハンドルの子ハンドルにアクセスする場合に使用します。

以下の構文を使用します。

```
INFA_CT_HANDLE* INFA_CTGetChildrenHandles(INFA_CT_HANDLE handle, size_t* pnChildrenHandles, INFA_CTHandleType returnHandleType);
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
handle	INFA_CT_HANDLE	入力	ハンドル名。
pnChildrenHandles	size_t*	Output	Integration Service は、子ハンドルの配列を返します。pnChildrenHandles パラメータは、配列内の子ハンドルの数を示します。
returnHandleType	INFA_CTHandleType	入力	returnHandleType パラメータには以下の値を使います。 <ul style="list-style-type: none">- PROCEDURETYPE- TRANSFORMATIONTYPE- PARTITIONTYPE- INPUTGROUPTYPE- OUTPUTGROUPTYPE- INPUTPORTTYPE- OUTPUTPORTTYPE

handle パラメータには、手続きでアクセスしたい子のハンドルを指定します。関数内で有効なハンドルが指定されている場合、Integration Service は INFA_CT_HANDLE* を返します。そうでない場合は NULL 値を返します。

コンパイル時にエラーが発生しないように、ハンドル名を戻り値に設定するよう手続きを記述する必要があります。

たとえば、次のようなコードを入力できます。

```
INFA_CT_PARTITION_HANDLE partition = INFA_CTGetChildrenHandles(procedureHandle, pnChildrenHandles,  
INFA_CT_PARTITION_HANDLE_TYPE);
```

ポートハンドル取得関数

Integration Service は、INFA_CT_INPUTPORT_HANDLE を入力ポートおよび入出力ポートに関連付け、INFA_CT_OUTPUTPORT_HANDLE を出力ポートおよび入出力ポートに関連付けます。

PowerCenter には、以下のポートハンドル取得関数が用意されています。

- **INFA_CTGetInputPortHandle()**。この関数は、入出力ポートの出力ポートハンドルがわかっていて、入力ポートハンドルが必要な場合に使用します。

以下の構文を使用します。

```
INFA_CTINFA_CT_INPUTPORT_HANDLE INFA_CTGetInputPortHandle(INFA_CT_OUTPUTPORT_HANDLE outputPortHandle);
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
outputPortHandle	INFA_CT_OUTPUTPORT_HANDLE	入力	出力ポートハンドル。

- **INFA_CTGetOutputPortHandle()**。この関数は、入出力ポートの入力ポートハンドルがわかっていて、出力ポートハンドルが必要な場合に使用します。

以下の構文を使用します。

```
INFA_CT_OUTPUTPORT_HANDLE INFA_CTGetOutputPortHandle(INFA_CT_INPUTPORT_HANDLE inputPortHandle);
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
inputPortHandle	INFA_CT_INPUTPORT_HANDLE	入力	入力ポートハンドル。

ポートハンドル取得関数を入力ポートまたは出力ポートに対して使用すると、Integration Service は NULL を返します。

プロパティ関数

プロパティ関数は、手続きでカスタムトランスフォーメーションのプロパティにアクセスする場合に使用します。プロパティ関数は、カスタムトランスフォーメーションの以下のタブのプロパティにアクセスします。

- ポート
- プロパティ
- 初期化プロパティ
- メタデータ エクステンション
- ポート属性定義

以下のプロパティ関数は、初期化関数で使用します。

- INFA_CTGetInternalProperty<データタイプ>()
- INFA_CTGetAllPropertyNamesM()
- INFA_CTGetAllPropertyNamesU()
- INFA_CTGetExternalProperty<データタイプ>M()
- INFA_CTGetExternalProperty<データタイプ>U()

内部プロパティ取得関数

PowerCenter には、カスタムトランスフォーメーションの「ポート」タブに指定されているポート属性、および「プロパティ」タブに指定されているプロパティにアクセスする関数が用意されています。

Integration Service は、各ポートおよびプロパティ属性をプロパティ ID と関連付けます。プロシージャ内では、プロパティ ID を指定して、その属性に指定されている値にアクセスする必要があります。handle パラメータには、ハンドル階層のハンドル名を指定します。ハンドル名が無効な場合、Integration Service はセッションに失敗します。

プロシージャでプロパティにアクセスする場合は、以下の関数を使用します。

- **INFA_CTGetInternalPropertyStringM()**。指定されたプロパティ ID に対する MBCS の文字列型の値にアクセスします。

以下の構文を使用します。

```
INFA_STATUS INFA_CTGetInternalPropertyStringM( INFA_CT_HANDLE handle, size_t propId, const char**  
psPropValue );
```

- **INFA_CTGetInternalPropertyStringU()**。指定されたプロパティ ID に対する Unicode の文字列型の値にアクセスします。

以下の構文を使用します。

```
INFA_STATUS INFA_CTGetInternalPropertyStringU( INFA_CT_HANDLE handle, size_t propId, const INFA_UNICHAR**  
psPropValue );
```

- **INFA_CTGetInternalPropertyInt32()**。指定されたプロパティ ID に対する整数型の値にアクセスします。

以下の構文を使用します。

```
INFA_STATUS INFA_CTGetInternalPropertyInt32( INFA_CT_HANDLE handle, size_t propId, INFA_INT32*  
pnPropValue );
```

- **INFA_CTGetInternalPropertyBool()**。指定されたプロパティ ID に対する Boolean 型の値にアクセスします。

以下の構文を使用します。

```
INFA_STATUS INFA_CTGetInternalPropertyBool( INFA_CT_HANDLE handle, size_t propId, INFA_BOOLEAN*  
pbPropValue );
```

- **INFA_CTGetInternalPropertyINFA_PTR()**。指定されたプロパティ ID に対する値へのポインタにアクセスします。

以下の構文を使用します。

```
INFA_STATUS INFA_CTGetInternalPropertyINFA_PTR( INFA_CT_HANDLE handle, size_t propId, INFA_PTR*  
pvPropValue );
```

戻り値のデータタイプは INFA_STATUS です。戻り値には INFA_SUCCESS および INFA_FAILURE を使用します。

ポートおよびプロパティ属性のプロパティ ID

以下の表は、カスタムトランスフォーメーションにおけるポートおよびプロパティ属性のプロパティ ID を示しています。それぞれの表には、プロパティ関数のハンドルでアクセスできるカスタムトランスフォーメーションハンドルとプロパティ ID が一覧表示されています。

以下の表に、INFA_CT_MODULE_HANDLE のプロパティ ID を示します。

ハンドルプロパティ ID	データタイプ	説明
INFA_CT_MODULE_NAME	String	モジュール名を指定します。
INFA_CT_SESSION_INFA_VERSION	String	Informatica のバージョンを指定します。
INFA_CT_SESSION_CODE_PAGE	Integer	Integration Service のコードページを指定します。
INFA_CT_SESSION_DATAMOVEMENT_MODE	Integer	データ移動モードを指定します。 Integration Service で返される値は次のいずれかです。 - eASM_MBCS - eASM_UNICODE
INFA_CT_SESSION_VALIDATE_CODEPAGE	ブール	Integration Service が強制的にコードページの検証を実行するかどうかを指定します。
INFA_CT_SESSION_PROD_INSTALL_DIR	String	Integration Service のインストールディレクトリを指定します。
INFA_CT_SESSION_HIGH_PRECISION_MODE	ブール	セッションが高精度に設定されているかどうかを指定します。

ハンドルプロパティ ID	データタイプ	説明
INFA_CT_MODULE_RUNTIME_DIR	String	DLL または共有ライブラリの実行時ディレクトリを指定します。
INFA_CT_SESSION_IS_UPD_STR_ALLOWED	ブール	トランスフォーメーションで [アップデートストラテジトランスフォーメーション] プロパティが選択されているかどうかを指定します。
INFA_CT_TRANS_OUTPUT_IS_REPEATABLE	Integer	セッションを実行するたびにカスタムトランスフォーメーションが同じ順序でデータを生成するかどうかを指定します。 Integration Service で返される値は次のいずれかです。 <ul style="list-style-type: none"> - eOUTREPEAT_NEVER = 1 - eOUTREPEAT_ALWAYS = 2 - eOUTREPEAT_BASED_ON_INPUT_ORDER = 3
INFA_CT_TRANS_FATAL_ERROR	ブール	カスタムトランスフォーメーションが致命的エラーを引き起こしたかどうかを指定します。Integration Service で返される値は次のいずれかです。 <ul style="list-style-type: none"> - INFA_TRUE - INFA_FALSE

以下の表に、INFA_CT_PROC_HANDLE のプロパティ ID を示します。

ハンドルプロパティ ID	データタイプ	説明
INFA_CT_PROCEDURE_NAME	String	カスタムトランスフォーメーションの手続き名を指定します。

以下の表に、INFA_CT_TRANS_HANDLE のプロパティ ID を示します。

ハンドルプロパティ ID	データタイプ	説明
INFA_CT_TRANS_INSTANCE_NAME	String	カスタムトランスフォーメーションのインスタンス名を指定します。
INFA_CT_TRANS_TRACE_LEVEL	Integer	トレースレベルを指定します。Integration Service で返される値は次のいずれかです。 <ul style="list-style-type: none"> - eTRACE_TERSE - eTRACE_NORMAL - eTRACE_VERBOSE_INIT - eTRACE_VERBOSE_DATA
INFA_CT_TRANS_MAY_BLOCK_DATA	ブール	現在のセッションで、Integration Service がプロシージャによる入力データのブロックを許可しているかどうかを指定します。

ハンドルプロパティ ID	データタイプ	説明
INFA_CT_TRANS_MUST_BLOCK_DATA	ブール	カスタムトランスフォーメーションの [Inputs Must Block] プロパティが選択されているかどうかを指定します。
INFA_CT_TRANS_ISACTIVE	ブール	カスタムトランスフォーメーションがアクティブとパッシブのどちらのトランスフォーメーションかを指定します。
INFA_CT_TRANS_ISPARTITIONABLE	ブール	このカスタムトランスフォーメーションを使用するセッションをパーティション化できるかどうかを指定します。
INFA_CT_TRANS_IS_UPDATE_STRATEGY	ブール	カスタムトランスフォーメーションがアップデートストラテジトランスフォーメーションのように動作するかどうかを指定します。
INFA_CT_TRANS_DEFAULT_UPDATE_STRATEGY	Integer	デフォルトの更新方法を指定します。 <ul style="list-style-type: none"> - eDUS_INSERT - eDUS_UPDATE - eDUS_DELETE - eDUS_REJECT - eDUS_PASSTHROUGH
INFA_CT_TRANS_NUM_PARTITIONS	Integer	このカスタムトランスフォーメーションを使用するセッション内のパーティションの数を指定します。
INFA_CT_TRANS_DATACODEPAGE	Integer	Integration Service がカスタムトランスフォーメーションにデータを渡す場合のコードページを指定します。 カスタムトランスフォーメーションから異なるコードページのデータにアクセスする場合に、データコードページ設定関数を使用します。
INFA_CT_TRANS_TRANSFORM_SCOPE	Integer	カスタムトランスフォーメーション内の [Transformation Scope] を指定します。 Integration Service で返される値は次のいずれかです。 <ul style="list-style-type: none"> - eTS_ROW - eTS_TRANSACTION - eTS_ALLINPUT
INFA_CT_TRANS_GENERATE_TRANSACTION	ブール	【トランザクションの生成】 プロパティが有効かどうかを指定します。 Integration Service で返される値は次のいずれかです。 <ul style="list-style-type: none"> - INFA_TRUE - INFA_FALSE

ハンドルプロパティ ID	データタイプ	説明
INFA_CT_TRANS_OUTPUT_IS_REPEATABLE	Integer	セッションを実行するたびにカスタムトランスフォーマーションが同じ順序でデータを生成するかどうかを指定します。Integration Service で返される値は次のいずれかです。 <ul style="list-style-type: none"> - eOUTREPEAT_NEVER = 1 - eOUTREPEAT_ALWAYS = 2 - eOUTREPEAT_BASED_ON_INPUT_ORDER = 3
INFA_CT_TRANS_FATAL_ERROR	ブール	カスタムトランスフォーマーションが致命的エラーを引き起こしたかどうかを指定します。Integration Service で返される値は次のいずれかです。 <ul style="list-style-type: none"> - INFA_TRUE - INFA_FALSE

以下の表に、INFA_CT_INPUT_GROUP_HANDLE および INFA_CT_OUTPUT_GROUP_HANDLE のプロパティ ID を示します。

ハンドルプロパティ ID	データタイプ	説明
INFA_CT_GROUP_NAME	String	グループ名を指定します。
INFA_CT_GROUP_NUM_PORTS	Integer	グループ内のポート数を指定します。
INFA_CT_GROUP_ISCONNECTED	ブール	グループ内のすべてのポートが別のトランスフォーマーションに接続されているかどうかを指定します。
INFA_CT_PORT_NAME	String	ポート名を指定します。
INFA_CT_PORT_CDATATYPE	Integer	ポートのデータタイプを指定します。Integration Service で返される値は次のいずれかです。 <ul style="list-style-type: none"> - eINFA_CTYPE_SHORT - eINFA_CTYPE_INT32 - eINFA_CTYPE_CHAR - eINFA_CTYPE_RAW - eINFA_CTYPE_UNICHAR - eINFA_CTYPE_TIME - eINFA_CTYPE_FLOAT - eINFA_CTYPE_DOUBLE - eINFA_CTYPE_DECIMAL18_FIXED - eINFA_CTYPE_DECIMAL28_FIXED - eINFA_CTYPE_INFA_CTDATETIME
INFA_CT_PORT_PRECISION	Integer	ポートの精度を指定します。
INFA_CT_PORT_SCALE	Integer	ポートの位取りを指定します（該当する場合）。

ハンドルプロパティ ID	データタイプ	説明
INFA_CT_PORT_IS_MAPPED	ブール	マッピング内でポートが他のトランスフォーメーションにリンクされているかどうかを指定します。
INFA_CT_PORT_STORAGE_SIZE	Integer	ポートのデータの内部格納サイズを指定します。格納サイズは、ポートのデータタイプによって異なります。
INFA_CT_PORT_BOUND_DATATYPE	Integer	ポートのデータタイプを指定します。ポートを再関連付けしてデフォルト以外のデータタイプを指定する場合に、INFA_CT_PORT_CDATATYPE の代わりに使用します。

以下の表に、INFA_CT_INPUTPORT_HANDLE および INFA_CT_OUTPUT_HANDLE のプロパティ ID を示します。

ハンドルプロパティ ID	データタイプ	説明
INFA_CT_PORT_NAME	String	ポート名を指定します。
INFA_CT_PORT_CDATATYPE	Integer	<p>ポートのデータタイプを指定します。Integration Service で返される値は次のいずれかです。</p> <ul style="list-style-type: none"> - eINFA_CTYPE_SHORT - eINFA_CTYPE_INT32 - eINFA_CTYPE_CHAR - eINFA_CTYPE_RAW - eINFA_CTYPE_UNICHAR - eINFA_CTYPE_TIME - eINFA_CTYPE_FLOAT - eINFA_CTYPE_DOUBLE - eINFA_CTYPE_DECIMAL18_FIXED - eINFA_CTYPE_DECIMAL28_FIXED - eINFA_CTYPE_INFA_CTDATETIME
INFA_CT_PORT_PRECISION	Integer	ポートの精度を指定します。
INFA_CT_PORT_SCALE	Integer	ポートの位取りを指定します（該当する場合）。
INFA_CT_PORT_IS_MAPPED	ブール	マッピング内でポートが他のトランスフォーメーションにリンクされているかどうかを指定します。

ハンドルプロパティ ID	データタイプ	説明
INFA_CT_PORT_STORAGE_SIZE	Integer	ポートのデータの内部格納サイズを指定します。格納サイズは、ポートのデータタイプによって異なります。
INFA_CT_PORT_BOUNDDATATYPE	Integer	ポートのデータタイプを指定します。ポートを再関連付けしてデフォルト以外のデータタイプを指定する場合に、INFA_CT_PORT_CDATATYPE の代わりに使用します。

全外部プロパティ名の取得（MBCS または Unicode）

PowerCenter には、カスタムトランスフォーメーションの [メタデータエクステンション] タブ、[初期化プロパティ] タブ、および [ポート属性定義] タブで定義されているプロパティ名にアクセスする 2 つの関数が用意されています。

手続きでプロパティ名にアクセスする場合には以下の関数を使用します。

- **INFA_CTGetAllPropertyNamesM()**。MBCS でのプロパティ名にアクセスします。

以下の構文を使用します。

```
INFA_STATUS INFA_CTGetAllPropertyNamesM(INFA_CT_HANDLE handle, const char*const** paPropertyNames, size_t* pnProperties);
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/出力	説明
handle	INFA_CT_HANDLE	入力	ハンドル名を指定します。
paPropertyNames	const char*const**	アウトプット	プロパティ名を指定します。Integration Service は、プロパティ名の配列を MBCS で返します。
pnProperties	size_t*	Output	配列内のプロパティの数を示します。

- **INFA_CTGetAllPropertyNamesU()**。Unicode でのプロパティ名にアクセスします。

以下の構文を使用します。

```
INFA_STATUS INFA_CTGetAllPropertyNamesU(INFA_CT_HANDLE handle, const INFA_UNICHAR*const** pasPropertyNames, size_t* pnProperties);
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
handle	INFA_CT_HANDLE	入力	ハンドル名を指定します。
paPropertyNames	const INFA_UNICHAR*const**	アウトプ ット	プロパティ名を指定します。 Integration Service は、プロパティ名 の配列を Unicode で返します。
pnProperties	size_t*	Output	配列内のプロパティの数を示します。

戻り値のデータタイプは INFA_STATUS です。戻り値には INFA_SUCCESS および INFA_FAILURE を使用します。

外部プロパティの取得（MBCS または Unicode）

PowerCenter には、カスタムトランスフォーメーションの [メタデータエクステンション] タブ、[初期化プロパティ] タブ、または [ポート属性定義] タブで定義されたプロパティの値にアクセスする関数が用意されています。

手続きで値にアクセスする場合は、関数でプロパティ名を指定する必要があります。プロパティ名にアクセスするには、INFA_CTGetAllPropertyNamesM()関数または INFA_CTGetAllPropertyNamesU()関数を使用します。handle パラメータには、ハンドル階層のハンドル名を指定します。ハンドル名が無効な場合、Integration Service はセッションに失敗します。

注: 初期化プロパティをメタデータエクステンションと同じ名前前で定義した場合、Integration Service はメタデータエクステンションの値を返します。

手続きでプロパティの値にアクセスする場合には以下の関数を使用します。

- **INFA_CTGetExternalProperty<datatype>M()**。MBCS でのプロパティの値にアクセスします。

以下の表に、構文を示します。

構文	プロパティ のデータ タイプ
INFA_STATUS INFA_CTGetExternalPropertyStringM(INFA_CT_HANDLE handle, const char* sPropName, const char** psPropValue);	String
INFA_STATUS INFA_CTGetExternalPropertyINT32M(INFA_CT_HANDLE handle, const char* sPropName, INFA_INT32* pnPropValue);	Integer
INFA_STATUS INFA_CTGetExternalPropertyBoolM(INFA_CT_HANDLE handle, const char* sPropName, INFA_BOOLEAN* pbPropValue);	ブール

- **INFA_CTGetExternalProperty<datatype>U()**。Unicode でのプロパティの値にアクセスします。

以下の表に、構文を示します。

構文	プロパティ のデータ タイプ
<code>INFA_STATUS INFA_CTGetExternalPropertyStringU(INFA_CT_HANDLE handle, INFA_UNICHAR* sPropName, INFA_UNICHAR** psPropValue);</code>	String
<code>INFA_STATUS INFA_CTGetExternalPropertyStringU(INFA_CT_HANDLE handle, INFA_UNICHAR* sPropName, INFA_INT32* pnPropValue);</code>	Integer
<code>INFA_STATUS INFA_CTGetExternalPropertyStringU(INFA_CT_HANDLE handle, INFA_UNICHAR* sPropName, INFA_BOOLEAN* pbPropValue);</code>	ブール

戻り値のデータタイプは INFA_STATUS です。戻り値には INFA_SUCCESS および INFA_FAILURE を使用します。

データタイプの再関連付け関数

PowerCenter では、ポートをデフォルトデータタイプとは異なるデータタイプに関連付けることができます。手続きでデフォルトデータタイプ以外のデータタイプのデータにアクセスする場合は、データタイプ再関連付け関数を使います。ポートは互換性のあるデータと関連付ける必要があります。

これらの関数は、初期化関数でのみ使用できます。

出力ポートまたは入出力ポートにデータタイプの再関連付けを行う場合は、以下の規則に注意してください。

- データ操作関数を使用して、そのポートにデータとインジケータを設定する必要があります。行ベースモードでは INFA_CTSetData() 関数と INFA_CTSetIndicator() 関数を使用し、配列ベースモードでは INFA_CTASetData() 関数を使用します。
- 出力ポートに対しては INFA_CTSetPassThruPort() 関数を呼び出さないでください。

以下の表に、互換性のあるデータタイプを示します。

デフォルトのデータタイプ	互換性があるデータタイプ
文字	Unichar
Unichar	文字
日付	INFA_DATETIME 以下の構文を使用します。 <pre>struct INFA_DATETIME { int nYear; int nMonth; int nDay; int nHour; int nMinute; int nSecond; int nNanoSecond; }</pre>

デフォルトのデータタイプ	互換性があるデータタイプ
Dec18	Char、Unichar
Dec28	Char、Unichar

PowerCenter には、以下のデータタイプ再関連付け関数が用意されています。

- **INFA_CTRebindInputDataType()**。入力ポートの再関連付けを行います。以下の構文を使用します。
INFA_STATUS INFA_CTRebindInputDataType(INFA_CT_INPUTPORT_HANDLE portHandle, INFA_CDATATYPE datatype);
- **INFA_CTRebindOutputDataType()**。出力ポートの再関連付けを行います。以下の構文を使用します。
INFA_STATUS INFA_CTRebindOutputDataType(INFA_CT_OUTPUTPORT_HANDLE portHandle, INFA_CDATATYPE datatype);
以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
portHandle	INFA_CT_OUTPUTPORT_HANDLE	入力	出力ポートハンドル。
データ型	INFA_CDATATYPE	入力	<p>ポートに再関連付けするデータ型。 datatype パラメータには以下の値を使います。</p> <ul style="list-style-type: none"> - eINFA_CTYPE_SHORT - eINFA_CTYPE_INT32 - eINFA_CTYPE_CHAR - eINFA_CTYPE_RAW - eINFA_CTYPE_UNICHAR - eINFA_CTYPE_TIME - eINFA_CTYPE_FLOAT - eINFA_CTYPE_DOUBLE - eINFA_CTYPE_DECIMAL18_FIXED - eINFA_CTYPE_DECIMAL28_FIXED - eINFA_CTYPE_INFA_CTDATETIME

戻り値のデータタイプは INFA_STATUS です。戻り値には INFA_SUCCESS および INFA_FAILURE を使用します。

データ操作関数（行ベースモード）

Integration Service が入力行通知関数を呼び出すときに、データ行またはデータブロックにアクセスできることをプロシージャに対して通知します。ただし、入力ポートからデータを取得し、そのデータを変更し、出力ポートにデータを設定するには、入力行通知関数内でデータ操作関数を使う必要があります。データアクセスモードが行ベースの場合は、行ベースのデータ操作関数を使用します。

INFA_CTGetData<データタイプ>()関数を使用して入力ポートからデータを取得し、INFA_CTSetData()関数を使用して出力ポートにデータを設定します。データを取得する前に、ポートにあるのが NULL 値または空の文字列かどうかを手続きで検査したい場合は、INFA_CTGetIndicator()関数または INFA_CTGetLength()関数を使用します。

PowerCenter には、以下のデータ操作関数が用意されています。

- INFA_CTGetData<データタイプ>()
- INFA_CTSetData()
- INFA_CTGetIndicator()
- INFA_CTSetIndicator()
- INFA_CTGetLength()
- INFA_CTSetLength()

データ取得関数（行ベースモード）

INFA_CTGetData<データタイプ>()関数を使って、関数が指定するポートのデータを取得します。

手続きでアクセスしたいポートのデータタイプに合わせて、関数の名前を変更する必要があります。

以下の表に、INFA_CTGetData<データタイプ>()関数の構文と戻り値のデータタイプを示します。

構文	戻り値のデータタイプ
<code>void* INFA_CTGetDataVoid(INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	戻り値へのデータ void ポインタ
<code>char* INFA_CTGetDataStringM(INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	String (MBCS)
<code>IUNICHAR* INFA_CTGetDataStringU(INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	String (Unicode)
<code>INFA_INT32 INFA_CTGetDataINT32(INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	Integer
<code>double INFA_CTGetDataDouble(INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	ダブル
<code>INFA_CT_RAWDATE INFA_CTGetDataDate(INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	Raw date
<code>INFA_CT_RAWDEC18 INFA_CTGetDataRawDec18(INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	Decimal BLOB (精度 18)
<code>INFA_CT_RAWDEC28 INFA_CTGetDataRawDec28(INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	Decimal BLOB (精度 28)
<code>INFA_CT_DATETIME INFA_CTGetDataDateTime(INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	Datetime

データ設定関数（行ベースモード）

INFA_CTSetData()関数は、手続きで値を出力ポートに渡す場合に使います。

以下の構文を使用します。

```
INFA_STATUS INFA_CTSetData(INFA_CT_OUTPUTPORT_HANDLE dataHandle, void* data);
```

戻り値のデータタイプは INFA_STATUS です。戻り値には INFA_SUCCESS および INFA_FAILURE を使用します。

注: INFA_CTSetPassThruPort()関数を入出力ポートに使用する場合には、そのポートに対してデータまたはインジケータを設定しないでください。

インジケータ関数（行ベースモード）

手続きで入力ポート用のインジケータを取得する、または出力ポートにインジケータを設定する場合にインジケータ関数を使用します。ポートのインジケータは、データが有効か、NULL か、切り詰められているかを示します。

PowerCenter には、以下のインジケータ関数が用意されています。

- **INFA_CTGetIndicator()**。入力ポートのインジケータを取得します。以下の構文を使用します。

```
INFA_INDICATOR INFA_CTGetIndicator(INFA_CT_INPUTPORT_HANDLE dataHandle);
```

戻り値のデータタイプは INFA_INDICAOR です。INFA_INDICAOR には以下の値を使います。

- **INFA_DATA_VALID**。データが有効であることを示します。
- **INFA_NULL_DATA**。NULL 値を示します。
- **INFA_DATA_TRUNCATED**。データが切り詰められていることを示します。

- **INFA_CTSetIndicator()**。出力ポートにインジケータを設定します。以下の構文を使用します。

```
INFA_STATUS INFA_CTSetIndicator(INFA_CT_OUTPUTPORT_HANDLE dataHandle, INFA_INDICATOR indicator);
```

以下の表に、この関数の引数を示します。

引数	データ型	入出力	説明
dataHandle	INFA_CT_OUTPUTPORT_HANDLE	入力	出力ポートハンドル。
indicator	INFA_INDICATOR	入力	出力ポートのインジケータ値。以下の値を使用します。 <ul style="list-style-type: none">- INFA_DATA_VALID。データが有効であることを示します。- INFA_NULL_DATA。NULL 値であることを示します。- INFA_DATA_TRUNCATED。データが切り詰められていることを示します。

戻り値のデータタイプは INFA_STATUS です。戻り値には INFA_SUCCESS および INFA_FAILURE を使用します。

注: INFA_CTSetPassThruPort()関数を入力ポートに使用する場合には、そのポートに対してデータまたはインジケータを設定しません。

長さ関数

長さ関数は、手続きで文字列またはバイナリの入力ポートの長さにアクセスする場合、あるいはバイナリまたは文字列の出力ポートの長さを設定する場合に使用します。

以下の長さ関数を使用します。

- **INFA_CTGetLength()**。この関数は、文字列またはバイナリポートにのみ使用できます。Integration Service は、末尾のスペースも含む文字数として長さを返します。以下の構文を使用します。

```
INFA_UINT32 INFA_CTGetLength(INFA_CT_INPUTPORT_HANDLE dataHandle);
```

戻り値のデータタイプは INFA_UINT32 です。戻り値の範囲は 0 から 2GB までです。

- **INFA_CTSetLength()**。カスタムトランスフォーメーションにバイナリまたは文字列の出力ポートが含まれる場合、この関数を使用して末尾のスペースを含むデータの長さを設定する必要があります。文字列またはバイナリのポートに設定する長さが、そのポートの精度を超えていないことを確認します。ポートの精度

を超える長さを設定すると、予想外の結果が生じます。たとえば、セッションが失敗するおそれがあります。

以下の構文を使用します。

```
INFA_STATUS INFA_CTSetLength(INFA_CT_OUTPUTPORT_HANDLE dataHandle, IUINT32 length);
```

戻り値のデータタイプは INFA_STATUS です。戻り値には INFA_SUCCESS および INFA_FAILURE を使用します。

パススルーポート設定関数

INFA_CTSetPassThruPort()関数は、Integration Service が入力ポートから取得したデータを変更せずに出力ポートに渡す場合に使用します。INFA_CTSetPassThruPort()関数を使用すると、Integration Service は入力行通知関数を呼び出したときに出力ポートにデータを渡します。

パススルーポート設定関数を使用するときには、以下の規則とガイドラインに注意してください。

- この関数は初期化関数でのみ使用します。
- 手続きにこの関数を使用する場合には、INFA_CTSetData()関数、INFA_CTSetLength 関数、INFA_CTSetIndicator()関数または INFA_CTASetData()関数を使用して出力ポートにデータを渡さないでください。
- 行ベースモードでは、トランスフォーメーション範囲が [Row] の場合にのみ、この関数を使用できます。トランスフォーメーション範囲が [トランザクション] または [すべての入力] の場合、この関数は INFA_FAILURE を返します。
- 行ベースモードでは、この関数を使用して所定の入力行に複数の行を出力した場合、入力ポートから渡されたデータが各出力行に含まれます。
- 配列ベースモードでは、パッシブなカスタムトランスフォーメーションに対してのみ、この関数を使用できます。

入力ポートと出力ポートで、データタイプ、精度、および位取りが同じである必要があります。

INFA_CTSetPassThruPort()関数で指定した入力ポートと出力ポートで、データタイプ、精度、または位取りが一致しない場合、Integration Service はセッションに失敗します。

以下の構文を使用します。

```
INFA_STATUS INFA_CTSetPassThruPort(INFA_CT_OUTPUTPORT_HANDLE outputport, INFA_CT_INPUTPORT_HANDLE inputport)
```

戻り値のデータタイプは INFA_STATUS です。戻り値には INFA_SUCCESS および INFA_FAILURE を使用します。

出力通知関数

プロシージャから Integration Service に行を出力する必要がある場合、INFA_CTOutputNotification()関数を使用します。この関数は、アクティブなカスタムトランスフォーメーションの場合にのみ含めます。パッシブなカスタムトランスフォーメーションの場合、入力行通知関数が戻り値を取得したときに、プロシージャによって行が Integration Service に出力されます。パッシブなカスタムトランスフォーメーションの場合にプロシージャがこの関数を呼び出すと、Integration Service によってこの関数が無視されます。

注: トランスフォーメーション範囲が [Row] の場合、入力行通知関数にのみ、この関数を使用できます。この関数をほかのどこかに使用した場合は、失敗が返されます。

以下の構文を使用します。

```
INFA_ROWSTATUS INFA_CTOutputNotification(INFA_CT_OUTPUTGROUP_HANDLE group);
```


以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
グループ	INFA_CT_OUTPUT_GROUP_HANDLE	入力	出力グループハンドル。

戻り値のデータタイプは INFA_ROWSTATUS です。戻り値には以下の値を使います。

- **INFA_ROWSUCCESS**。関数がデータ行の処理に成功したことを示します。
- **INFA_ROWERROR**。関数がデータ行に関してエラーに遭遇したことを示します。この場合、Integration Service は内部エラーカウントを増分します。
- **INFA_FATALERROR**。関数がデータ行に関して致命的エラーに遭遇したことを示します。このとき、Integration Service はセッションに失敗します。

注: プロシージャコードが INFA_CTOutputNotification()関数を呼び出す場合は、出力ポートハンドルのすべてのポインタが有効なデータを指していることを確認する必要があります。ポインタが有効なデータを指していない場合、Integration Service は予期せずシャットダウンすることがあります。

データ境界出力通知関数

手続きでコミットまたはロールバックトランザクションを出力する場合に、INFA_CTDataBdryOutputNotification()関数を使用します。

この関数を使う場合は、カスタムトランスフォーメーションで「トランザクションの生成」プロパティを選択する必要があります。このプロパティを選択していない場合、Integration Service はセッションに失敗します。

以下の構文を使用します。

```
INFA_STATUS INFA_CTDataBdryOutputNotification(INFA_CT_PARTITION_HANDLE handle, INFA_CTDataBdryType dataBoundaryType);
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
handle	INFA_CT_PARTITION_HANDLE	入力	ハンドル名。
dataBoundaryType	INFA_CTDataBdryType	入力	トランザクションタイプ。 dataBoundaryType パラメータには以下の値を使います。 <ul style="list-style-type: none">- eBT_COMMIT- eBT_ROLLBACK

戻り値のデータタイプは INFA_STATUS です。戻り値には INFA_SUCCESS および INFA_FAILURE を使用します。

エラー関数

エラー関数を使ってプロシージャエラーにアクセスします。Integration Service は最新のエラーを返します。

PowerCenter には、以下のエラー関数が用意されています。

- **INFA_CTGetErrorMsgM()**。エラーメッセージを MBCS で取得します。以下の構文を使用します。
`const char* INFA_CTGetErrorMsgM();`
- **INFA_CTGetErrorMsgU()**。エラーメッセージを Unicode で取得します。以下の構文を使用します。
`const IUNICHAR* INFA_CTGetErrorMsgU();`

セッションログメッセージ関数

セッションログメッセージ関数は、手続きでセッション内のメッセージを Unicode または MBCS でログに記録する場合に使用します。

PowerCenter には、以下のセッションログメッセージ関数が用意されています。

- **INFA_CTLogMessageU()**。メッセージを Unicode でログに記録します。
以下の構文を使用します。
`void INFA_CTLogMessageU(INFA_CT_ErrorSeverityLevel errorseverityLevel, INFA_UNICHAR* msg)`
以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
errorSeverityLevel	INFA_CT_ErrorSeverityLevel	入力	Integration Service がセッションログに書き込むエラーメッセージの重大度。errorSeverityLevel パラメータには以下の値を使います。 <ul style="list-style-type: none">- eESL_LOG- eESL_DEBUG- eESL_ERROR
msg	INFA_UNICHAR*	入力	メッセージのテキストを Unicode で引用符で囲んで入力します。

- **INFA_CTLogMessageM()**。メッセージを MBCS でログに記録します。
以下の構文を使用します。
`void INFA_CTLogMessageM(INFA_CT_ErrorSeverityLevel errorSeverityLevel, char* msg)`
以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
errorSeverityLevel	INFA_CT_ErrorSeverityLevel	入力	Integration Service がセッションログに書き込むエラーメッセージの重大度。errorSeverityLevel パラメータには以下の値を使います。 <ul style="list-style-type: none">- eESL_LOG- eESL_DEBUG- eESL_ERROR
msg	char*	入力	メッセージのテキストを MBCS で引用符で囲んで入力します。

エラーカウントインクリメント関数

INFA_CTIncrementErrorCount()関数は、セッションのエラーカウントをインクリメントするときに使用します。

以下の構文を使用します。

```
INFA_STATUS INFA_CTIncrementErrorCount(INFA_CT_PARTITION_HANDLE transformation, size_t nErrors, INFA_STATUS* pStatus);
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/出力	説明
トランスフォーメーション	INFA_CT_PARTITION_HANDLE	入力	パーティションハンドル。
nErrors	size_t	入力	Integration Service では、指定されたトランスフォーメーションインスタンスに対してエラーカウントが nErrors ずつ増分されます。
pStatus	INFA_STATUS*	入力	エラーカウントがエラーしきい値を超えると、Integration Service は pStatus パラメータに対して INFA_FAILURE を使用し、セッションは失敗します。

戻り値のデータタイプは INFA_STATUS です。戻り値には INFA_SUCCESS および INFA_FAILURE を使用します。

終了要求検査関数

INFA_CTIsTerminated()関数は、PowerCenter クライアントが Integration Service にセッションの終了を要求したかどうかをプロシージャでチェックする場合に使用します。手続きに時間のかかる処理が含まれる場合に、この関数を使用できます。

以下の構文を使用します。

```
INFA_CTTerminateType INFA_CTIsTerminated(INFA_CT_PARTITION_HANDLE handle);
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/出力	説明
handle	INFA_CT_PARTITION_HANDLE	入力	パーティションハンドル。

戻り値のデータタイプは INFA_CTTerminateType です。Integration Service で返される値は次のいずれかです。

- **eTT_NOTTERMINATED**。PowerCenter クライアントがセッションの終了を要求していないことを示します。
- **eTT_ABORTED**。Integration Service がセッションを強制終了したことを示します。
- **eTT_STOPPED**。Integration Service がセッションに失敗したことを示します。

ブロック関数

カスタムトランスフォーメーションに複数の入力グループが含まれる場合、入力グループへの入力データをブロックするコードを書くことができます。

ブロック関数を使用する際は、下記の規則に注意してください。

- 最大で $n-1$ 個の入力グループをブロックできます。
- 既にブロックされている入力グループはブロックできません。
- 別の入力グループと同じソースからデータを受け取る入力グループはブロックできません。
- 既にブロック解除されている入力グループをブロック解除することはできません。

PowerCenter には、以下のブロック関数が用意されています。

- **INFA_CTBlockInputFlow()**。手続きで入力グループをブロックします。

以下の構文を使用します。

```
INFA_STATUS INFA_CTBlockInputFlow(INFA_CT_INPUTGROUP_HANDLE group);
```

- **INFA_CTUnblockInputFlow()**。手続きで入力グループのブロックを解除します。

以下の構文を使用します。

```
INFA_STATUS INFA_CTUnblockInputFlow(INFA_CT_INPUTGROUP_HANDLE group);
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
グループ	INFA_CT_INPUTGROUP_HANDLE	入力	入力グループハンドル。

戻り値のデータタイプは INFA_STATUS です。戻り値には INFA_SUCCESS および INFA_FAILURE を使用します。

ブロックの確認

プロシージャコードで INFA_CTBlockInputFlow() 関数および INFA_CTUnblockInputFlow() 関数を使用する際は、Integration Service でカスタムトランスフォーメーションによる入力データのブロックが許可されるかどうかをプロシージャがチェックすることを検査します。これを実行するには、INFA_CTGetInternalPropertyBool() 関数を使用して INFA_CT_TRANS_MAY_BLOCK_DATA propID の値をチェックします。

INFA_CT_TRANS_MAY_BLOCK_DATA propID の値が FALSE の場合、手続きはブロック関数を使用できないか、致命的なエラーを返してセッションを中止するかのいずれかでなければなりません。

Integration Service でカスタムトランスフォーメーションによるデータのブロックが許可されない場合、プロシージャコードでブロック関数を使用すると、Integration Service はセッションに失敗する可能性があります。

ポインタ関数

Integration Service でオブジェクトまたは構造体へのポインタを作成し、アクセスできるようにする場合は、ポインタ関数を使用します。

PowerCenter には、以下のポインタ関数が用意されています。

- **INFA_CTGetUserDefinedPtr()**。実行時に手続きでオブジェクトまたは構造体へのポインタにアクセスすることを可能にします。

以下の構文を使用します。

```
void* INFA_CTGetUserDefinedPtr(INFA_CT_HANDLE handle)
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
handle	INFA_CT_HANDLE	入力	ハンドル名。

- **INFA_CTSetUserDefinedPtr()**。オブジェクトまたは構造体と、Integration Service が提供する任意のハンドルとの関連付けをプロシージャで実行できるようにします。処理のオーバーヘッドを削減するには、初期化関数にこの関数を組み込みます。

以下の構文を使用します。

```
void INFA_CTSetUserDefinedPtr(INFA_CT_HANDLE handle, void* pPtr)
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
handle	INFA_CT_HANDLE	入力	ハンドル名。
pPtr	void*	入力	ユーザポインタ。

INFA_CT_HANDLE を有効なハンドルで置き換える必要があります。

文字列モード変更関数

Integration Service が Unicode モードで動作している場合、デフォルトでは UCS-2 でプロシージャにデータを渡します。ASCII モードで動作している場合は、デフォルトでは ASCII でデータを渡します。手続きのデフォルトの文字列モードを変更したい場合は、INFA_CTChangeStringMode()関数を使用します。デフォルトの文字列モードを MBCS に変更すると、Integration Service は Integration Service コードページでデータを渡します。コードページを変更する場合は INFA_CTSetDataCodePageID()関数を使用します。

プロシージャに INFA_CTChangeStringMode()関数が含まれる場合、Integration Service によってこの特定のプロシージャを使用するカスタムトランスフォーメーションごとに全ポートの文字列モードが変更されます。

文字列モード変更関数は、初期化関数で使用します。

以下の構文を使用します。

```
INFA_STATUS INFA_CTChangeStringMode(INFA_CT_PROCEDURE_HANDLE procedure, INFA_CTStringMode stringMode);
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
procedure	INFA_CT_PROCEDURE_HANDLE	入力	手続きハンドル名。
stringMode	INFA_CTStringMode	入力	Integration Service で使用する文字列モードを指定します。stringMode パラメータには以下の値を使用します。 <ul style="list-style-type: none">- eASM_UNICODE。Integration Service が ASCII モードで動作しており、プロシージャから Unicode でデータにアクセスする場合に使用します。- eASM_MBCS。Integration Service が Unicode モードで動作しており、プロシージャから MBCS でデータにアクセスする場合に使用します。

戻り値のデータタイプは INFA_STATUS です。戻り値には INFA_SUCCESS および INFA_FAILURE を使用します。

データコードページ設定関数

Integration Service からカスタムトランスフォーメーションへのデータの受け渡しに Integration Service コードページ以外のコードページを使用する場合、INFA_CTSetDataCodePageID()を使用します。

データコードページ設定関数は、手続き初期化関数で使用します。

以下の構文を使用します。

```
INFA_STATUS INFA_CTSetDataCodePageID(INFA_CT_TRANSFORMATION_HANDLE transformation, int dataCodePageID);
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
トランスフォーメーション	INFA_CT_TRANSFORMATION_HANDLE	入力	トランスフォーメーションハンドル名。
dataCodePageID	整数型	入力	Integration Service がデータを渡すときに使用するコードページを指定します。 dataCodePageID パラメータの有効な値については、『 <i>管理者ガイド</i> 』の「コードページ」を参照してください。

戻り値のデータタイプは INFA_STATUS です。戻り値には INFA_SUCCESS および INFA_FAILURE を使用します。

行更新方式関数（行ベースモード）

行更新方式関数によって、各行の更新方式へのアクセスおよび設定が行えます。

PowerCenter には、以下の行更新方式関数が用意されています。

- **INFA_CTGetRowStrategy()**。手続きで行の更新方式を取得します。

以下の構文を使用します。

```
INFA_STATUS INFA_CTGetRowStrategy(INFA_CT_INPUTGROUP_HANDLE group, INFA_CTUpdateStrategy updateStrategy);
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
グループ	INFA_CT_INPUTGROUP_HANDLE	入力	入力グループハンドル。
updateStrategy	INFA_CT_UPDATESTRATEGY	入力	入力ポートの Update Strategy。 Integration Service は、次の値を使用します。 - eUS_INSERT = 0 - eUS_UPDATE = 1 - eUS_DELETE = 2 - eUS_REJECT = 3

- **INFA_CTSetRowStrategy()**。各行の更新方式を設定します。INFA_CTChangeDefaultRowStrategy 関数を上書きします。

以下の構文を使用します。

```
INFA_STATUS INFA_CTSetRowStrategy(INFA_CT_OUTPUTGROUP_HANDLE group, INFA_CT_UPDATESTRATEGY updateStrategy);
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
グループ	INFA_CT_OUTPUTGROUP_HANDLE	入力	出力グループハンドル。
updateStrategy	INFA_CT_UPDATESTRATEGY	入力	出力ポートに対して設定する Update Strategy。以下の値を使用します。 - eUS_INSERT = 0 - eUS_UPDATE = 1 - eUS_DELETE = 2 - eUS_REJECT = 3

戻り値のデータタイプは INFA_STATUS です。戻り値には INFA_SUCCESS および INFA_FAILURE を使用します。

デフォルトの行ストラテジ変更関数

デフォルトでは、トランスフォーメーション範囲が [Row] の場合、カスタムトランスフォーメーションの行ストラテジはパススルーです。トランスフォーメーション範囲が [トランザクション] または [すべての入力] の場合、デフォルトの行ストラテジは [Treat Source Rows As] セッションプロパティの値と同じです。

たとえば、マッピング内にアップデートストラテジトランスフォーメーションに続いてトランスフォーメーション範囲が [Row] のカスタムトランスフォーメーションがあるとします。アップデートストラテジトランスフォーメーションは、行に更新、挿入、または削除のフラグを設定します。カスタムトランスフォーメーション

ンでは行ストラテジがパススルーであるため、Integration Service によって行を渡されたカスタムトランスフォーメーションは、そのフラグを保持します。

ただし、PowerCenter のカスタムトランスフォーメーションでは、行ストラテジを変更できます。INFA_CTChangeDefaultRowStrategy()関数を使って、トランスフォーメーションレベルのデフォルトの行ストラテジを変更します。例えば、カスタムトランスフォーメーションのデフォルトの行ストラテジを挿入に変更すると、Integration Service はこのトランスフォーメーションを通過するすべての行のフラグを挿入に設定します。

注: セッションがデータドリブンモードでない場合、Integration Service は INFA_FAILURE を返します。

以下の構文を使用します。

```
INFA_STATUS INFA_CTChangeDefaultRowStrategy(INFA_CT_TRANSFORMATION_HANDLE transformation,  
INFA_CT_DefaultUpdateStrategy defaultUpdateStrategy);
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
トランスフォーメーション	INFA_CT_TRANSFORMATION_HANDLE	入力	トランスフォーメーションハンドル。
defaultUpdateStrategy	INFA_CT_DefaultUpdateStrategy	入力	Integration Service がカスタムトランスフォーメーションに対して使用する行ストラテジを指定します。 - eDUS_PASSTHROUGH。行のフラグを単純に設定します。 - eDUS_INSERT。行のフラグを挿入に設定します。 - eDUS_UPDATE。行のフラグを更新に設定します。 - eDUS_DELETE。行のフラグを削除に設定します。

戻り値のデータタイプは INFA_STATUS です。戻り値には INFA_SUCCESS および INFA_FAILURE を使用します。

配列ベース API 関数

配列ベース関数は、データアクセスモードを配列ベースに変更するときに使用する API 関数です。

Informatica には、以下の配列ベースの API 関数のグループが用意されています。

- 最大行数
- 行の数
- 行有効検査
- データ操作（配列ベースモード）
- 行ストラテジ
- 入力エラー行の設定

最大行数関数

デフォルトでは、Integration Service で入力ブロックおよび出力ブロックにおける最大行数が許可されます。ただし、出力ブロックの許容最大行数は変更できます。

INFA_CTAGetInputNumRowsMax()関数および INFA_CTAGetOutputNumRowsMax()関数を使用して、入力ブロックおよび出力ブロックの最大行数を決定します。手続きでバッファが必要な場合、これらの関数が返す値を使用してバッファサイズを決定します。

INFA_CTASetOutputRowMax()関数を使用して、出力ブロックの最大行数を設定できます。手続きにもっと大きなバッファや小さなバッファを使用させる場合に、この関数を使用する場合があります。

これらの関数は、初期化関数のみで呼び出すことができます。

PowerCenter には、ブロック内の最大行数を決定し、設定するための以下の関数が用意されています。

- **INFA_CTAGetInputNumRowsMax()**。この関数を使用して、入力ブロック内の許容最大行数を決定します。

以下の構文を使用します。

```
IINT32 INFA_CTAGetInputRowMax( INFA_CT_INPUTGROUP_HANDLE inputgroup );
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
inputgroup	INFA_CT_INPUTGROUP_HANDLE	入力	入力グループハンドル。

- **INFA_CTAGetOutputNumRowsMax()**。この関数を使用して、入力ブロック内の許容最大行数を決定します。

以下の構文を使用します。

```
IINT32 INFA_CTAGetOutputRowMax( INFA_CT_OUTPUTGROUP_HANDLE outputgroup );
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
outputgroup	INFA_CT_OUTPUTGROUP_HANDLE	入力	出力グループハンドル。

- **INFA_CTASetOutputRowMax()**。この関数を使用して、出力ブロック内の許容最大行数を設定します。

以下の構文を使用します。

```
INFA_STATUS INFA_CTASetOutputRowMax( INFA_CT_OUTPUTGROUP_HANDLE outputgroup, INFA_INT32 nRowMax );
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
outputgroup	INFA_CT_OUTPUTGROUP_HANDLE	入力	出力グループハンドル。
nRowMax	INFA_INT32	入力	出力ブロックの許容最大行数。 正の数値を入力してください。 この関数は、ゼロを含めて正 の数値以外を入力すると、致 命的エラーを返します。

行数関数

行数関数を使用して、指定した入力グループまたは出力グループに対して、入力ブロック内の行数を決定する、または出力ブロック内の行数を設定します。

PowerCenter で提供されている行数関数は、次のとおりです。

- **INFA_CTAGetNumRows()**。入力ブロック内の行数を決定できます。

以下の構文を使用します。

```
INFA_INT32 INFA_CTAGetNumRows( INFA_CT_INPUTGROUP_HANDLE inputgroup );
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
inputgroup	INFA_CT_INPUTGROUP_HANDLE	入力	入力グループハンドル。

- **INFA_CTASetNumRows()**。出力ブロック内の行数を設定できます。この関数を呼び出してから、出力通知関数を呼び出してください。

以下の構文を使用します。

```
void INFA_CTASetNumRows( INFA_CT_OUTPUTGROUP_HANDLE outputgroup, INFA_INT32 nRows );
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
outputgroup	INFA_CT_OUTPUTGROUP_HANDLE	入力	出力ポートハンドル。
nRows	INFA_INT32	入力	出力ブロック内に定義する行数。 正の数値を入力してください。 正数以外を指定すると、 Integration Service は出力通知関 数に失敗します。

行有効検証関数

ブロック内の一部の行は削除、またはフィルタリングされたか、エラー行である可能性があります。INFA_CTIsRowValid()関数を使用して、ブロック内の行が有効かどうかを判断します。行が有効な場合、この関数は INFA_TRUE を返します。

以下の構文を使用します。

```
INFA_BOOLEAN INFA_CTIsRowValid( INFA_CT_INPUTGROUP_HANDLE inputgroup, INFA_INT32 iRow);
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
inputgroup	INFA_CT_INPUTGROUP_HANDLE	入力	入力グループハンドル。
iRow	INFA_INT32	入力	ブロック内の行のインデックス番号。 インデックスはゼロベースです。 この手続きが渡すのはデータブロック 内にあるインデックス番号のみである ことを確認する必要があります。無効 な値を渡すと、Integration Service は 突然シャットダウンします。

データ操作関数（配列ベースモード）

Integration Service は、p_<プロシージャ名>_inputRowNotification()関数を呼び出して、プロシージャがデータの行またはブロックにアクセスできることをプロシージャに通知します。ただし、配列ベースモードの場合に入力ポートからデータを取得し、そのデータを変更し、出力ポートにデータを設定するには、入力行通知関数内で配列ベースのデータ操作関数を使う必要があります。

INFA_CTGetData<データタイプ>()関数をインクルードして入力ポートからデータを取得し、INFA_CTSetData()関数を使用して出力ポートにデータを設定します。データを取得する前に、ポートにあるのが NULL 値または空の文字列かどうかを手続きで検査したい場合は、INFA_CTGetIndicator()関数を使用します。

PowerCenter には、配列ベースのアクセスモード用に以下のデータ操作関数が用意されています。

- INFA_CTGetData<データタイプ>()
- INFA_CTGetIndicator()
- INFA_CTSetData()

データ取得関数（配列ベースモード）

INFA_CTGetData<データタイプ>()関数を使って、関数が指定するポートのデータを取得します。手続きでアクセスしたいポートのデータタイプに合わせて、関数の名前を変更する必要があります。Integration Service は、配列ベースのデータ取得関数にデータの長さを渡します。

以下の表に、INFA_CTGetData<データタイプ>()関数の構文と戻り値のデータタイプを示します。

構文	戻り値のデータタイプ
<code>void* INFA_CTGetDataVoid(INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow, INFA_UINT32* pLength);</code>	戻り値へのデータ void ポインタ
<code>char* INFA_CTGetDataStringM(INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow, INFA_UINT32* pLength);</code>	String (MBCS)
<code>IUNICHAR* INFA_CTGetDataStringU(INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow, INFA_UINT32* pLength);</code>	String (Unicode)
<code>INFA_INT32 INFA_CTGetDataINT32(INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow);</code>	Integer
<code>double INFA_CTGetDataDouble(INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow);</code>	ダブル
<code>INFA_CT_RAWDATETIME INFA_CTGetDataRawDate(INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow);</code>	Raw date
<code>INFA_CT_DATETIME INFA_CTGetDataDateTime(INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow);</code>	Datetime
<code>INFA_CT_RAWDEC18 INFA_CTGetDataRawDec18(INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow);</code>	Decimal BLOB (精度 18)
<code>INFA_CT_RAWDEC28 INFA_CTGetDataRawDec28(INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow);</code>	Decimal BLOB (精度 28)

インジケータ取得関数（配列ベースモード）

入力ポートに NULL 値があるかどうかを手続きに確認させる場合に、インジケータ取得関数を使用します。

以下の構文を使用します。

```
INFA_INDICATOR INFA_CTGetIndicator( INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow );
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
inputport	INFA_CT_INPUTPORT_HANDLE	入力	入力ポートハンドル。
iRow	INFA_INT32	入力	ブロック内の行のインデックス番号。 インデックスはゼロベースです。 この手続きが渡すのはデータブロック 内にあるインデックス番号のみである ことを確認する必要があります。無効 な値を渡すと、Integration Service は 突然シャットダウンします。

戻り値のデータタイプは INFA_INDICAOR です。INFA_INDICAOR には以下の値を使います。

- **INFA_DATA_VALID.**データが有効であることを示します。

- **INFA_NULL_DATA**。NULL 値を示します。
- **INFA_DATA_TRUNCATED**。データが切り詰められていることを示します。

データ設定関数（配列ベースモード）

データ設定関数は、手続きで値を出力ポートに渡す場合に使います。指定した出力ポートのデータ、データ長（該当する場合）、インジケータを設定できます。出力ポートの長さまたはインジケータは個別の関数を使用して設定できません。

以下の構文を使用します。

```
void INFA_CTASetData( INFA_CT_OUTPUTPORT_HANDLE outputport, INFA_INT32 iRow, void* pData, INFA_UINT32 nLength, INFA_INDICATOR indicator);
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
outputport	INFA_CT_OUTPUTPORT_HANDLE	入力	出力ポートハンドル。
iRow	INFA_INT32	入力	ブロック内の行のインデックス番号。インデックスはゼロベースです。 この手続きが渡すのはデータブロック内にあるインデックス番号のみであることを確認する必要があります。無効な値を渡すと、Integration Service は突然シャットダウンします。
pData	void*	入力	データへのポインタ。
nLength	INFA_UINT32	入力	ポートの長さ。文字列またはバイナリのポートのみに使用します。 関数が正確なデータの長さを渡すことを確認する必要があります。関数が間違った長さを渡した場合、出力通知関数はこのポートに対して失敗を返します。 文字列またはバイナリポートに設定した長さが、そのポートの精度を超えていないことを確認します。ポートの精度を超える長さを設定すると、予想外の結果が生じます。たとえば、セッションが失敗するおそれがあります。
indicator	INFA_INDICATOR	入力	出力ポートのインジケータ値。以下の値を使用します。 - INFA_DATA_VALID。データが有効であることを示します。 - INFA_NULL_DATA。NULL 値であることを示します。 - INFA_DATA_TRUNCATED。データが切り詰められていることを示します。

行更新方式関数（配列ベースモード）

配列ベースの更新方式関数によって、ブロック内の各行のアップデートストラテジに対するアクセスおよび設定が行えます。

PowerCenter には、以下の行更新方式関数が用意されています。

- **INFA_CTAGetRowStrategy()**。プロシージャでブロック内の行のアップデートストラテジを取得します。

以下の構文を使用します。

```
INFA_CT_UPDATESTRATEGY INFA_CTAGetRowStrategy( INFA_CT_INPUTGROUP_HANDLE inputgroup, INFA_INT32 iRow);
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
inputgroup	INFA_CT_INPUTGROUP_HANDLE	入力	入力グループハンドル。
iRow	INFA_INT32	入力	ブロック内の行のインデックス番号。 インデックスはゼロベースです。 この手続きが渡すのはデータブロック 内にあるインデックス番号のみである ことを確認する必要があります。無効 な値を渡すと、Integration Service は 突然シャットダウンします。

- **INFA_CTASetRowStrategy()**。ブロック内の行のアップデートストラテジを設定します。

以下の構文を使用します。

```
void INFA_CTASetRowStrategy( INFA_CT_OUTPUTGROUP_HANDLE outputgroup, INFA_INT32 iRow,  
INFA_CT_UPDATESTRATEGY updateStrategy );
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
outputgroup	INFA_CT_OUTPUTGROUP_HANDLE	入力	出力グループハンドル。
iRow	INFA_INT32	入力	ブロック内の行のインデックス番号。 インデックスはゼロベースです。 この手続きが渡すのはデータブロック 内にあるインデックス番号のみである ことを確認する必要があります。無効な 値を渡すと、Integration Service は突 然シャットダウンします。
updateStrategy	INFA_CT_UPDATESTRATEGY	入力	ポートの Update Strategy。以下の 値を使用します。 - eUS_INSERT = 0 - eUS_UPDATE = 1 - eUS_DELETE = 2 - eUS_REJECT = 3

入力エラー行設定関数

配列ベースのアクセスモードを使用する場合、入力行通知関数に `INFA_ROWERROR` を返すことはできません。その代わりに、入力エラー行設定関数を使用して特定の入力行にエラーがあることを Integration Service に通知します。

PowerCenter には、配列ベースモードの以下の入力行設定関数が用意されています。

- **INFA_CTASetInputErrorRowM()**。入力ブロックの行にエラーがあることを Integration Service に通知して、MBCS エラーメッセージをセッションログに出力できます。

以下の構文を使用します。

```
INFA_STATUS INFA_CTASetInputErrorRowM( INFA_CT_INPUTGROUP_HANDLE inputGroup, INFA_INT32 iRow, size_t nErrors, INFA_MBCSCHAR* sErrMsg );
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/出力	説明
inputGroup	INFA_CT_INPUTGROUP_HANDLE	入力	入力グループハンドル。
iRow	INFA_INT32	入力	ブロック内の行のインデックス番号。インデックスはゼロベースです。 この手続きが渡すのはデータブロック内にあるインデックス番号のみであることを確認する必要があります。無効な値を渡すと、Integration Service は突然シャットダウンします。
nErrors	size_t	入力	このパラメータを使用して、この入力行が原因となって発生したエラーの数を特定します。
sErrMsg	INFA_MBCSCHAR*	入力	関数が出力するエラーメッセージを含む MBCS 文字列。NULL 値で終わる文字列を入力する必要があります。 このパラメータはオプションです。この引数を使用すると、行エラーのログを有効にしている場合でも、Integration Service はセッションログにメッセージを出力します。

- **INFA_CTASetInputErrorRowU()**。入力ブロックの行にエラーがあることを Integration Service に通知して、Unicode エラーメッセージをセッションログに出力できます。

以下の構文を使用します。

```
INFA_STATUS INFA_CTASetInputErrorRowU( INFA_CT_INPUTGROUP_HANDLE inputGroup, INFA_INT32 iRow, size_t nErrors, INFA_UNICHAR* sErrMsg );
```

以下の表に、この関数の引数を示します。

引数	データ型	入力/ 出力	説明
inputGroup	INFA_CT_INPUTGROUP_HANDLE	入力	入力グループハンドル。
iRow	INFA_INT32	入力	ブロック内の行のインデックス番号。 インデックスはゼロベースです。 この手続きが渡すのはデータブロック 内にあるインデックス番号のみで あることを確認する必要があります。 無効な値を渡すと、Integration Service は突然シャットダウンしま す。
nErrors	size_t	入力	このパラメータを使用して、この出 力行が原因となって発生したエラー の数を特定します。
sErrMsg	INFA_UNICHAR*	入力	関数が出力するエラーメッセージを 含む Unicode 文字列。NULL 値で終 わる文字列を入力する必要があります。 このパラメータはオプションです。 この引数を使用すると、行エラーの ログを有効にしている場合でも、 Integration Service はセッションログ にメッセージを出力します。

第 5 章

データマスキングトランスフォーメーション

この章では、以下の項目について説明します。

- [データマスキングトランスフォーメーション, 130](#) ページ
- [マスキングプロパティ, 130](#) ページ
- [キーマスキング, 132](#) ページ
- [置換マスキング, 134](#) ページ
- [依存マスキング, 138](#) ページ
- [ランダムマスキング, 139](#) ページ
- [マスキングルールの適用, 141](#) ページ
- [式マスキング, 145](#) ページ
- [特殊マスク形式, 147](#) ページ
- [社会保障番号のマスキング, 147](#) ページ
- [クレジットカード番号のマスキング, 148](#) ページ
- [電話番号マスキング, 149](#) ページ
- [電子メールアドレスマスキング, 149](#) ページ
- [社会保険番号のマスキング, 151](#) ページ
- [IP アドレスマスキング, 151](#) ページ
- [URL アドレスマスキング, 152](#) ページ
- [デフォルト値ファイル, 152](#) ページ
- [データマスキングトランスフォーメーションのセッションプロパティ, 153](#) ページ
- [データマスキングトランスフォーメーションのルールとガイドライン, 153](#) ページ

データマスキングトランスフォーメーション

データマスキングトランスフォーメーションを使用して、機密性が高い実稼働データを、プロダクション環境以外での実際のテストデータに変換します。データマスキングトランスフォーメーションは、カラムごとに設定されたマスキングルールに基づいてソースデータを変更します。

ソフトウェア開発、テスト、トレーニング、およびデータマイニング用にマスクされたデータを作成します。マスクされたデータ内のデータリレーションシップを保持し、データベーステーブル間の参照整合性を保持することができます。データマスキングトランスフォーメーションは、パッシブトランスフォーメーションです。

データマスキングトランスフォーメーションでは、ポートに設定したソースのデータ型およびマスキングの種類に基づいてマスキングルールが提供されます。文字列の場合は、文字列内で置き換える文字およびマスク内で適用する文字を制限できます。数と日付の場合は、マスクされたデータの数の範囲を指定できます。範囲は、元の数に対する固定偏差またはパーセント偏差に基づいて設定できます。統合サービスでは、マスキングルールで設定したロケールに基づいて文字を置き換えます。

データマスキングトランスフォーメーションには、以下のタイプのマスキングを適用できます。

- **キーマスキング**。同じソースデータ、マスキングルール、およびシード値に対して確定的な結果を生成します。
- **置換マスキング**。データのカラムを、ディクショナリ内の似ているが関連のないデータに置き換えます。
- **依存マスキング**。1つのソースカラムの値を、別のソースカラムの値に基づいて置換します。
- **シャッフルマスキング**。1つのデータカラムの値を、同じカラムの別の値に置換します。
- **ランダムマスキング**。同じソースデータとマスキングルールに対して再現可能でないランダムな結果を生成します。
- **式マスキング**。データを変更またはデータを作成するための式をポートに適用します。
- **特殊マスク形式**。特殊マスク形式を一般的なタイプのセンシティブデータに適用します。社会保障番号、社会保険番号、クレジットカード番号、電話番号、URL アドレス、電子メールアドレス、または IP アドレスをマスクすることができます。

データマスキングトランスフォーメーションを使用するには、適切なライセンスが必要です。

関連項目：

- [「ポートのデフォルト値」 \(ページ 39\)](#)
- [「マスキングルールの適用」 \(ページ 141\)](#)
- [「デフォルト値ファイル」 \(ページ 152\)](#)

マスキングプロパティ

入力ポートを定義し、[マスキングプロパティ] タブの各ポートのマスキングプロパティを設定します。選択できるマスキングのタイプは、ポートのデータタイプによって異なります。マスキングのタイプを選択すると、Designer はそのマスキングのタイプのマスキングルールを表示します。

ロケール

ロケールは、データに含まれる文字の言語と地域を識別します。一覧からロケールを選択します。データマスキングトランスフォーメーションによって、選択したロケールの文字で文字データがマスクされます。ソースデータには、選択したロケールと互換性のある文字が含まれている必要があります。

ロケールが一覧にない場合、類似または一致するコードページを持つロケールを選択します。ロケールに Unicode を選択することはできません。

マスキングのタイプ

マスキングのタイプは、選択したカラムに適用されるデータマスキングのタイプです。以下のマスキングのタイプのいずれかを選択します。

- キーマスキング。同じソースデータ、マスキングルール、およびシード値に対して確定的な結果を生成します。
- 置換マスキング。データのカラムを、ディクショナリ内の似ているが関連のないデータに置き換えます。
- 依存マスキング。1つのソースカラムの値を、別のソースカラムの値に基づいて置換します。
- ランダムマスキング。同じソースデータとマスキングルールに対してランダムな結果を生成します。
- 式マスキング。データを変更またはデータを作成するための式をポートに適用します。
- 特殊マスク形式。特殊マスク形式を一般的なタイプのセンシティブデータに適用します。社会保障番号、社会保険番号、クレジットカード番号、電話番号、URL アドレス、電子メールアドレス、または IP アドレスをマスクすることができます。
- 置換。データのカラムを、ディクショナリ内の似ているが関連のないデータに置き換えます。
- マスキングなし。Data Masking トランスフォーメーションでは、ソースデータは変更されません。

デフォルトは、[マスキングなし] です。

再現可能な出力

再現可能な出力は、データマスキングトランスフォーメーションから返される、一貫した値のセットです。

再現可能な出力は確定的な値を返します。例えば、名のカラムに対して再現可能な出力を設定するとします。このデータマスキングトランスフォーメーションをワークフローに含めるたびに、同じマスクされた値が返されます。

再現可能なマスキングは、すべてのデータマスキングタイプに設定できます。再現可能なマスキングを設定するには、**[再現可能な出力]** をクリックして、**[シード値]** を選択します。

シード

シード値は、マスクされた値を生成する開始ポイントです。

Data Masking トランスフォーメーションでは、1~1,000 の範囲の乱数であるデフォルトのシード値が作成されます。別のシード値を入力することも、マッピングパラメータ値を適用することもできます。別のソースデータから同じマスクデータ値を返すには、カラムに同じシード値を適用します。例えば、4つのテーブルに同じ Cust_ID カラムがあり、それらすべてを同じマスクされた値に出力したい場合、4つのカラムすべてを同じシード値に設定します。

マッピングパラメータ

マッピングパラメータを使用してシード値を定義できます。トランスフォーメーションに追加するシード値ごとに、マッピングパラメータを作成します。マッピングパラメータ値は、1~1,000 の数になります。

カラムに対してデータマスキングを設定するときに、シードに対してマッピングパラメータを選択します。Designer に、マッピングパラメータのリストが表示されます。リストからマッピングパラメータを選択します。セッションを実行する前に、セッションのパラメータファイルでマッピングパラメータ値を変更できます。

データマスキングトランスフォーメーションを作成する前に、マッピングパラメータを作成します。シード値をパラメータ化することを選択し、マッピングにマッピングパラメータが含まれていない場合、エラーが表示されます。削除済みのマッピングパラメータを参照するマスキングルールを含むポートを選択した場合、Designer はそのポートに対して新しいランダムシード値を生成します。シード値はマッピングパラメータではありません。マッピングパラメータが削除され、Designer が新しいシード値を作成することを示すメッセージが表示されます。

Integration Service は、以下の場合にデフォルトのシード値を適用します。

- マッピングパラメータオプションがカラムに対して選択されているが、セッションにパラメータファイルがない場合。
- マッピングパラメータを削除した場合。
- マッピングパラメータシード値が 1~1,000 の場合。

Integration Service では、デフォルト値のファイルからマスクされた値が適用されます。デフォルト値のファイルを編集してデフォルト値を変更できます。

デフォルト値のファイルは、以下の場所にある XML ファイルです。

```
<PowerCenter Installation Directory>\infa_shared\SrcFiles\defaultValue.xml
```

シードの名前-値のペアは以下のとおりです。

```
default_seed = "500".
```

デフォルト値のファイルのシード値が 1~1,000 ではない場合、Integration Service はシードに値 725 を割り当て、セッションログにメッセージを書き込みます。

関連項目：

- [「デフォルト値ファイル」 \(ページ 152\)](#)

関連出力ポート

関連出力ポートは、入力ポートに関連付けられた出力ポートです。データマスキングトランスフォーメーションでは、各入力ポートに対して出力ポートが作成されます。ポートの命名規則は、out_<ポート名>です。関連出力ポートは読み取り専用ポートです。

キーマスキング

キーマスキングに対して設定されたカラムは、ソース値とシード値が同じ場合に、マスクされた確定的なデータを返します。データマスキングトランスフォーメーションは、カラムに対して一意の値を返します。

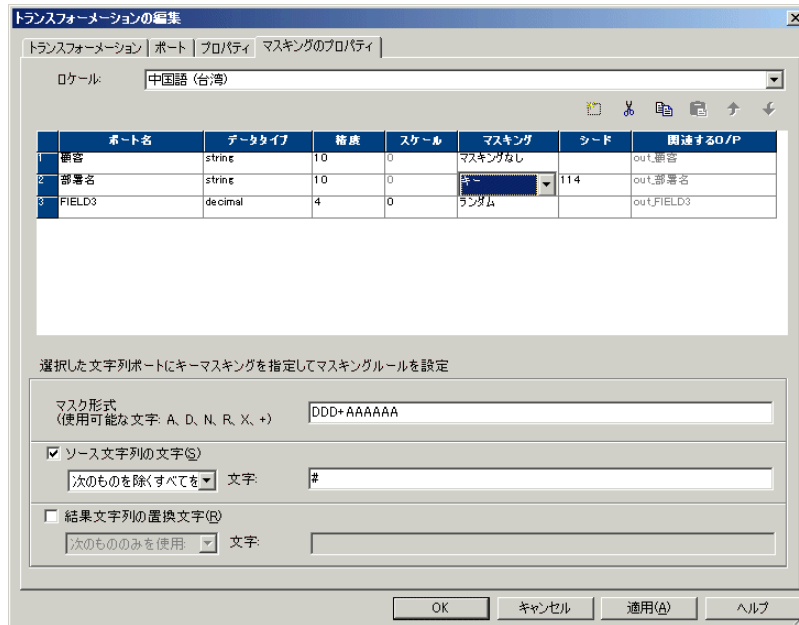
キーマスキングに対してカラムを設定すると、データマスキングトランスフォーメーションによって、そのカラムのシード値が生成されます。異なるデータマスキングトランスフォーメーションで再現可能なデータを生成するようにシード値を変更できます。例えば、キーマスキングを設定して、参照整合性を強制します。あるテーブルのプライマリキーと別のテーブルの外部キーをマスクするには、同じ seed 値を使用します。

データマスキングトランスフォーメーションによって返されるデータのフォーマットに影響を与えるマスキングルールを定義できます。文字列値と数値のマスクには、キーマスキングを使用します。

文字列値のマスキング

文字列のキーマスキングを設定すると、再現可能な出力が生成されます。マスクフォーマットを設定して、出力文字列に含まれる各文字に対する制限を定義します。ソース文字列の文字と結果文字列の置換文字を設定して、マスクするソース文字とマスクに使用する文字を定義します。

以下の図に、文字列データタイプのキーマスキングプロパティを示します。



キーマスキング文字列値には、以下のマスキングルールを設定できます。

- **シード**。シード値を適用し、カラムに対して確定的なマスクされた値を生成します。以下のいずれかのオプションを選択します。
 - **値**。デフォルトのシード値を受け入れるか、1~1,000 の数を入力します。
 - **マッピングパラメータ**。マッピングパラメータを使ってシード値を定義します。Designer では、マッピングに対して作成したマッピングパラメータのリストが表示されます。マッピングパラメータをリストから選択してシード値として使用します。
- **マスク形式**。置換する文字の種類を入力データの文字ごとに定義します。各文字の種類を英文字、数字、または英数字に制限できます。
- **ソース文字列の文字**。ソース文字列内のマスク対象文字を定義します。例えば、入力データに出現する各シャープ記号（#）をマスクできます。[ソース文字列の文字] が空白の場合は、すべての入力文字がマスクされます。データマスキングトランスフォーメーションは、ソース文字列の文字数が結果文字列の文字数より少ない場合、一意のデータが返されないことがあります。
- **結果文字列の文字**。対象文字列内の文字を、[結果文字列の文字] で定義された文字に置き換えます。例えば、各マスクに英文字の大文字をすべて含めるには、以下の文字を入力します。

ABCDEFGHIJKLMNOPQRSTUVWXYZ

数値のマスキング

決定性出力が生成されるようにするには、数値ソースデータにキーマスキングを設定します。カラムに対して数値キーマスキングを設定する場合、カラムにランダムなシード値を割り当てます。データマスキングトラン

スフォーメーションによってソースデータがマスクされる場合、seed を必要とするマスキングアルゴリズムが適用されます。

同じソース値が別のカラムに出現した場合に再現可能な結果が生成されるようにするには、カラムの seed 値を変更します。たとえば、2つのテーブル間でプライマリーキーと外部キーのリレーションが維持されるようにします。この場合、各データマスキングトランスフォーメーションで、プライマリーカラムの seed 値と外部キーカラムの seed 値として同じ seed 値を入力します。データマスキングトランスフォーメーションによって、同じ数値に対して確定的な結果が生成されます。これにより、この2つのテーブル間で参照整合性が維持されるようになります。

日時の値のマスキング

日時の値に対してキーマスキングを設定できる場合、データマスキングトランスフォーメーションでシードとしてランダムな数値が必要になります。カラム間で再現可能な日時の値を返すには、別のカラムのシード値に一致するようにシードを変更できます。

データマスキングトランスフォーメーションは、キーマスキングで 1753～2400 の日付をマスクできます。ソース年がうるう年の場合、データマスキングトランスフォーメーションは同じくうるう年の年を返します。ソース月に 31 日が含まれる場合、データマスキングトランスフォーメーションは 31 日を含む月を返します。ソース月が 2 月の場合、データマスキングトランスフォーメーションは 2 月を返します。

データマスキングトランスフォーメーションは、常に有効な日付を生成します。

関連項目：

- [「日付範囲」 \(ページ 144\)](#)

置換マスキング

置換マスキングでは、データのカラムを似ているが関連のないデータに置き換えることができます。置換マスキングを使用して、プロダクションデータを現実的なテストデータに置き換えます。置換マスキングを設定する場合は、代替値が含まれるディクショナリを定義します。

データマスキングトランスフォーメーションでは、設定したディクショナリでのルックアップが実行されます。データマスキングトランスフォーメーションでは、ソースデータがディクショナリから取得したデータに置き換えられます。ディクショナリファイルには、文字列データ、日時の値、整数、および浮動小数点数を含めることができます。日時の値は次の形式で入力します。

mm/dd/yyyy

データは、再現可能な値または再現不可能な値に置き換えることができます。再現可能な値を選択すると、データマスキングトランスフォーメーションで、同じソースデータおよびシード値に対する確定的な結果が生成されます。データを確定的な結果に置き換えるには、シード値を設定する必要があります。Integration Service では、ソースの格納テーブルと、再現可能なマスキングに使用するマスキング値が保持されます。

データの複数のカラムを同じディクショナリ行のマスキング値に置き換えることができます。1つの入力カラムに対して置換マスキングを設定します。同じディクショナリ行からマスキングデータを受け取る他のカラムに対しては、依存データマスキングを設定します。

ディクショナリ

ディクショナリは、ファイル内の各行の置換データが含まれているフラットファイルまたはリレーショナルテーブルです。データマスキングトランスフォーメーションはディクショナリ行を取得するための数値を生成し

ます。データマスキングトランスフォーメーションでは、再現可能な置換マスキング用にハッシュキーが生成されます。再現不可能なマスキング用には、乱数が生成されます。追加のルックアップ条件を設定できます。

データマスキングトランスフォーメーションでは、ディクショナリを設定して、複数のポートをマスクすることができます。

以下の例に、ファーストネームと性別が含まれるフラットファイルディクショナリを示します。

```
SNO,GENDER,FIRSTNAME
1,M,Adam
2,M,Adeel
3,M,Adil
4,F,Alice
5,F,Alison
```

このディクショナリでは、行の最初のフィールドはシリアル番号、2 番目のフィールドは性別です。性別をルックアップ条件として追加できます。統合サービスでは、ハッシュキーを使用して行がディクショナリから取得され、ソースデータ内の性別と一致する性別の行が検出されます。

ディクショナリを作成する場合は、以下のルールとガイドラインに従います。

- フラットファイルディクショナリの第 1 行には、各レコードのフィールドを識別するためにカラムラベルを設定する必要があります。フィールドは、カンマで区切ります。1 行目にカラムラベルが含まれていない場合、統合サービスでは最初の行のフィールド値がカラム名として処理されます。
- フラットファイルディクショナリは、\$PMLookupFileDir ルックアップファイルディレクトリに配置されている必要があります。デフォルトでは、このディクショナリは次の場所にインストールされます。
<PowerCenter_Installation_Directory>\server\infa_shared\LkpFiles
- Windows でフラットファイルディクショナリを作成し、UNIX マシンにコピーする場合は、ファイルが UNIX に適した形式であることを確認します。例えば、Windows と UNIX では、行末マーカに異なる文字を使用します。
- 複数のポートに対して置換マスキングを設定する場合、すべてのリレーショナルディクショナリは、同じデータベーススキーマ内に配置する必要があります。
- フラットファイルディクショナリの連続行のバッファ長は 600 文字以下にする必要があります。
- セッションプロパティでは、ディクショナリタイプまたは置換ディクショナリ名を変更することはできません。

格納テーブル

データマスキングトランスフォーメーションには、セッションごとに再現可能な置換用に格納テーブルが維持されています。格納テーブルの行には、ソースカラムおよびマスクされた値ペアが含まれています。データマスキングトランスフォーメーションでは、再現可能な代替値で値がマスクされるたびに、ディクショナリ名、ロケール、カラム名、入力値、およびシードで格納テーブルの検索が行われます。行が見つかり、格納テーブルからマスクされた値が返されます。行が見つからない場合、データマスキングトランスフォーメーションはハッシュキーを使用してディクショナリから行を取得します。

格納テーブル内のディクショナリ名の形式は、フラットファイルディクショナリとリレーショナルディクショナリとは異なります。フラットファイルディクショナリの名前は、ファイル名によって識別されます。リレーショナルディクショナリの名前の構文は次のとおりです。

```
<Connection object>_<dictionary table name>
```

Informatica は、リレーショナル格納テーブルを作成するために実行できるスクリプトを提供します。スクリプトは以下の場所に格納されています。

```
<PowerCenter Client installation directory>\client\bin\Extensions\DataMasking
```

ディクショナリには、Sybase、Microsoft SQL Server、IBM DB2、および Oracle の各データベース用のスクリプトが含まれています。各スクリプトの名前は、Substitution_<データベースタイプ>です。SQL 文およびプライマリキー制約を設定する場合は、別のデータベースでテーブルを作成できます。

ストレージ内のデータの暗号化を解除した後、同じシード値とディレクトリを使用して同じカラムを暗号化する場合、置換マスキング用の格納テーブルを暗号化する必要があります。

置換マスキング用の格納テーブルの暗号化

格納テーブルは、トランスフォーメーション言語エンコード機能を使用して暗号化できます。ストレージの暗号化を有効にした場合は、格納テーブルを暗号化する必要があります。

1. IDM_SUBSTITUTION_STORAGE 格納テーブルをソースとして使用し、マッピングを作成します。
2. データマスキングトランスフォーメーションを作成します。
3. 入力値とマスクされた値ポートに、式マスキング方法を適用します。
4. INPUTVALUE ポートに対して次の式を使用します。
`Enc_Base64(AES_Encrypt(INPUTVALUE, Key))`
5. MASKEDVALUE ポートに対して次の式を使用します。
`Enc_Base64(AES_Encrypt(MASKEDVALUE, Key))`
6. ポートをターゲットにリンクします。

置換マスキングプロパティ

置換マスキングには、以下のマスキングルールを設定できます。

- **再現可能な出力。** セッションごとに確定的な結果を返します。データマスキングトランスフォーメーションは、マスクされた値をストレージテーブルに格納します。
- **シード。** シード値を適用し、カラムに対して確定的なマスクされた値を生成します。以下のいずれかのオプションを選択します。
 - **値。** デフォルトのシード値を受け入れるか、1~1,000 の数を入力します。
 - **マッピングパラメータ。** マッピングパラメータを使ってシード値を定義します。Designer では、マッピングに対して作成したマッピングパラメータのリストが表示されます。マッピングパラメータをリストから選択してシード値として使用します。
 - **一意の出力。** データマスキングトランスフォーメーションに一意の入力値に対して一意の出力値を作成させます。同じ出力値に対して 2 つの入力値はマスクされません。ディクショナリに、一意の出力を有効にするのに十分な一意の行が必要です。
一意の出力を無効にすると、データマスキングトランスフォーメーションが一意の出力値に対する入力値をマスクしない場合があります。ディクショナリにいくつかの行が含まれている場合があります。
- **ディクショナリ情報。** 置換データの値を含むフラットファイルまたはリレーショナルテーブルを設定します。
 - **リレーショナルテーブル。** ディクショナリがデータベーステーブル内にある場合は、リレーショナルテーブルを選択します。データベースおよびテーブルを設定するには、[テーブルの選択] をクリックします。
 - **フラットファイル。** ディクショナリがカンマ区切りのフラットファイル内にある場合は、[フラットファイル] を選択します。ファイルを参照および選択するには、[フラットファイルを選択] をクリックします。
 - **ディクショナリ名。** 選択したフラットファイル名またはリレーショナルテーブル名を表示します。
 - **出力カラム。** データマスキングトランスフォーメーションに返すカラムを選択します。
- **ルックアップ条件。** 置換マスキングで使用するディクショナリ行の適正を評価するには、ルックアップ条件を設定します。ルックアップ条件は、SQL クエリの WHERE 句に似ています。ルックアップ条件を設定する場合、ソース内のカラムとディクショナリ内のカラムを比較します。
たとえば、ファーストネームをマスクするとします。ソースデータおよびディクショナリには、ファーストネームの列と性別の列があります。それぞれの女性のファーストネームをディクショナリの女性の名前で置

き換えるという条件を追加できます。ルックアップ条件は、ソース内の性別とディクショナリ内の性別を比較します。

- **入力ポート**。ルックアップで使用するソースデータカラム。
- **ディクショナリカラム**。入力ポートと比較するディクショナリカラム。

リレーショナルディクショナリ

リレーショナルテーブルをディクショナリとして選択した場合、そのディクショナリが含まれるデータベースを設定します。テーブルを定義すると、Designer がテーブルに接続し、テーブル内にカラムが表示されます。

データベースを選択するには、データマスキングトランスフォーメーションの「マスキングプロパティ」タブで「テーブルの選択」をクリックします。

「テーブル名の選択」ダイアログボックスで、以下のフィールドを設定します。

- **ODBC データソース**。ディクショナリが含まれるデータベースに接続する ODBC データソース。
- **ユーザー名**。データベースに接続するデータベースユーザー名。テーブルを表示するには、ユーザー名がデータベースに対する権限を持っている必要があります。
- **オーナー名**。ユーザー名がディクショナリテーブルのオーナーではない場合は、テーブルのオーナーを入力します。
- **パスワード**。データベースユーザー名のパスワード。
- **指定テーブルの検索**。オプション。ダイアログボックスに表示されるテーブルの数を制限します。

「接続」をクリックすると、Designer がデータソースに接続し、テーブルが表示されます。テーブルのリストをスクロールし、ディクショナリに使用するテーブルを選択します。「マスキングプロパティ」タブの「ディクショナリ名」フィールドにテーブル名が表示されます。

接続要件

置換マスキングで格納テーブルおよびリレーショナルディクショナリを使用する場合は、それらに関するデータベース接続を設定する必要があります。

Workflow Manager でデータベース接続を設定します。セッションプロパティの「マッピング」タブで、IDM_Dictionary 接続および IDM_Storage 接続のリレーショナル接続オブジェクトを選択します。

置換マスキングのルールおよびガイドライン

置換マスキングに使用されるルールおよびガイドラインは、次のとおりです。

- 一意の再現可能な置換マスクの格納テーブルが存在しない場合、セッションは失敗します。
- ディクショナリに行が含まれていない場合は、データマスキングトランスフォーメーションがエラーメッセージを返します。
- データマスキングトランスフォーメーションで格納テーブル内にロケール、ディクショナリ、シードを持つ入力値が見つかった場合、その行がディクショナリ内にすでに存在しない場合でもマスク値が取得されます。
- 接続オブジェクトの削除またはディクショナリの変更を行った場合、格納テーブルが切り詰められます。そうでない場合、予期しない結果になる可能性があります。
- ディクショナリ内の値の数がソースデータ内の一意の値の数よりも少ない場合、データマスキングトランスフォーメーションでは一意の再現可能な値でデータをマスクすることができません。データマスキングトランスフォーメーションはエラーメッセージを返します。

依存マスキング

依存マスキングは、ソースデータの複数のカラムを、同じディクショナリ行からのデータに置き換えます。

例えば、データマスキングトランスフォーメーションが複数のカラムに対して置換マスキングを実行すると、マスクされたデータは非現実的なフィールドの組み合わせを含む可能性があります。複数の入力カラムに関して同じディクショナリ行からのデータに置換するために、依存マスキングを設定することができます。マスクされたデータは、「New York, New York」、「Chicago, Illinois」などの有効な組み合わせを受け取ります。

依存マスキングを設定する場合、最初に置換マスキングの入力カラムを設定します。置換カラムに依存するその他の入力カラムを設定します。例えば、置換マスキング用の郵便番号カラムを選択し、その郵便番号カラムに依存する市と州を選択します。依存マスキングは、置き換えられた市と州の値が、その郵便番号カラムに対して有効であることを保証します。

注: 最初に置換マスキングのカラムを設定せずに依存マスキングのカラムを設定することはできません。

依存マスキングのカラムを設定する場合、以下のマスキングルールを設定します。

依存カラム

置換マスキングに対して設定した入力カラムの名前。データマスキングトランスフォーメーションは、そのカラムのマスキングルールを使用して、ディクショナリルールの置換データを受け取ります。置換マスキングに対して設定するカラムは、ディクショナリからマスクされたデータを受け取るためのキーカラムになります。

出力カラム

依存マスキングで設定するカラムの値が含まれるディクショナリカラムの名前。

依存マスキングの例

データマスキングディクショナリに、以下の値を持つ住所行が含まれているとします。

```
SNO,STREET,CITY,STATE,ZIP,COUNTRY
1,32 Apple Lane,Chicago,IL,61523,US
2,776 Ash Street,Dallas,TX,75240,US
3,2229 Big Square,Atleeville,TN,38057,US
4,6698 Cowboy Street, Houston,TX,77001,US
```

ソースデータを、住所ディクショナリからの、市、州、および郵便番号の有効な組み合わせでマスクする必要があります。

置換マスキングの ZIP ポートを設定します。

ZIP ポート用のマスキングルールを以下のように入力します。

ルール	値
ディクショナリファイルタイプ	フラットファイルまたはリレーショナルテーブル
辞書名	Address.dic
出力カラム	ZIP

依存マスキングの City ポートを設定します。

City ポート用のマスキングルールを以下のように入力します。

ルール	値
依存カラム	ZIP
出力カラム	City

依存マスキングの State ポートを設定します。

State ポート用のマスキングルールを以下のように入力します。

ルール	値
依存カラム	ZIP
出力カラム	State

データマスキングトランスフォーメーションが郵便番号をマスクすると、ディクショナリ行から郵便番号の正しい市と州が返されます。

再現可能な依存マスキング

複数の入力フィールドを設定して、同じディクショナリ行からマスクされたデータを受け取る場合、置換マスキング用のフィールドを決定する必要があります。再現可能なマスキングの場合、ソースデータを一意に識別するカラムを選択します。データマスキングトランスフォーメーションは、キーフィールドを使用し、再現可能なマスキング用のストレージにすでに行が存在するかどうかを確認します。

注: ソースデータにプライマリキーがある場合、置換マスキング用のプライマリキーカラムを設定できます。依存マスキング用に別のカラムを設定します。

例えば、従業員データをマスクする場合、置換マスキング用に EmployeeID を設定し、依存マスキング用に FirstName と LastName を設定します。

再現可能なマスキングを設定すると、以下の行は異なるマスクされたデータを受け取ります。

EmployeeID	FirstName	LastName
111	John	Jones
222	Radhika	Jones

再現可能な依存マスキング用のキーフィールドとして LastName を選択すると、姓が Jones であるすべての行が同じマスクデータを受け取ります。

ランダムマスキング

ランダムマスキングでは、非決定性のマスクされたデータがランダムに生成されます。データマスキングトランスフォーメーションでは、異なる行に同じソース値が出現する場合に、異なる値が返されます。データマスキングトランスフォーメーションによって返されるデータのフォーマットに影響を与えるマスキングルールを定義できます。ランダムマスキングでは、数値、文字列値、および日付値がマスクされます。

数値のマスキング

数値データをマスクする場合は、カラムの出力データ範囲を設定できます。データマスキングトランスフォーメーションでは、ポート精度に応じて、範囲の上限と下限の間の値が返されます。範囲を定義するには、範囲の上限と下限を設定するか、元のソース値に対する偏差に基づくブラー範囲を設定します。

数値データには、以下のマスキングパラメータを設定できます。

範囲

出力値の範囲を定義します。データマスキングトランスフォーメーションでは、値の上限と下限の間にある数値データが返されます。

ブラー範囲

ソースデータに対する固定偏差またはパーセント偏差に基づく範囲として、出力値の範囲を定義します。データマスキングトランスフォーメーションでは、ソースデータの値に近い数値データが返されます。範囲とブラー範囲を両方設定することもできます。

文字列値のマスキング

ランダムマスキングを設定すると、文字列カラムにランダムな出力が生成されます。出力文字列に含まれる各文字の制限を設定するには、マスク形式を定義します。ソース文字列の文字と結果文字列の置き換え文字を設定するには、マスクするソース文字とマスクに使用する文字を定義します。

文字列ポートには、以下のマスキングルールを適用できます。

範囲

文字列長の上限と下限を設定します。データマスキングトランスフォーメーションでは、文字列長の上限と下限の範囲内でランダムに構成された文字列が返されます。

マスク形式

置き換える文字の種類を入力データの文字ごとに定義します。各文字の種類を英文字、数字、または英数字に制限できます。

ソース文字列の文字

ソース文字列内のマスク対象文字を定義します。例えば、入力データに出現する各シャープ記号（#）をマスクできます。[ソース文字列の文字] が空白の場合は、すべての入力文字がマスクされます。

結果文字列の置換文字

対象文字列内の文字を、[結果文字列の文字] で定義された文字に置き換えます。例えば、各マスクに英文字の大文字 A～Z を含めるには、以下の文字を入力します。

ABCDEFGHIJKLMNOPQRSTUVWXYZ

日付値のマスキング

日付値をランダムマスキングでマスクするには、出力日の範囲を設定するか、偏差を選択します。偏差を設定する場合は、ブラー対象となる日付部分を選択します。選択できるのは年、月、日、時、分、または秒です。データマスキングトランスフォーメーションは、設定した範囲内の日付を返します。

日時の値をマスクする場合は、以下のマスキングルールを設定できます。

範囲

選択した日時の値に対して返す値の上限と下限を設定します。

ブラー

日付の単位に適用する偏差に基づいて日付をマスクします。データマスキングトランスフォーメーションでは、偏差の範囲内の日付が返されます。ブラーできるのは、年、月、日、時、分、または秒です。適用する低偏差と高偏差を選択します。

マスキングルールの適用

マスキングルールは、ソースのデータタイプに応じて適用します。[マスキングプロパティ] タブでカラムのプロパティをクリックすると、Designer によって、ポートのデータタイプに基づくマスキングルールが表示されます。

以下の表に、マスキングのタイプとソースのデータタイプに応じて設定できるマスキングルールの説明を示します。

マスキングのタイプ	ソースのデータタイプ	マスキングルール	説明
ランダムおよびキー	String	マスク形式	出力文字列内の各文字を英文字、数字、または英数字に制限するマスクです。
ランダムおよびキー	String	ソース文字列の文字	マスクするソース文字セット、またはマスク対象から除外するソース文字セットです。
ランダムおよびキー	String	結果文字列の置換文字	マスクに含める文字セットまたはマスクから除外する文字セットです。
ランダム	Numeric String 日付/時刻	範囲	出力値の範囲です。 <ul style="list-style-type: none">- 数値。データマスキングトランスフォーメーションでは、値の上限と下限の間にある数値データが返されます。- 文字列。上限と下限の文字列長の範囲内で、ランダムな文字で構成される文字列を返します。- 日付/時刻。上限と下限の範囲内で日時を返します。
ランダム	Numeric 日付/時刻	ブラー	ソースデータに対する固定偏差またはパーセント偏差の範囲の出力値です。データマスキングトランスフォーメーションでは、ソースデータの値に近いデータが返されます。日時のカラムは固定偏差である必要があります。

マスク形式

出力カラム内の各文字を英文字、数字、または英数字に制限するには、マスク形式を設定します。以下の文字を使用して、マスク形式を定義します。

A, D, N, X, +, R

注: マスク形式には、大文字を使用します。マスク文字として小文字を入力すると、データマスキングトランスフォーメーションによって大文字に変換されます。

以下の表では、マスク形式文字について説明します。

文字	説明
A	英文字。例えば、ASCII 文字 a～z、A～Z です。
D	数字 0～9。データマスキングトランスフォーメーションは、数字 0～9 以外の文字に対して「X」を返します。
N	英数字。例えば、ASCII 文字 a～z、A～Z、および 0～9 です。
X	任意の文字。たとえば、英文字や記号です。
+	マスキングなし。
R	残りの文字。R は、文字列内のその他の文字に、任意の種類の文字を使用できることを示します。R は、マスクの最後の文字にする必要があります。

たとえば、部署名のフォーマットが以下のとおりであるとします。

nnn-<department_name>

最初の 3 文字を数値に限定し、部署名を英文字に限定し、ダッシュを出力に残すマスクを設定できます。この場合、マスクフォーマットを以下のように設定します。

DDD+AAAAAAAAAAAAAAAA

データマスキングトランスフォーメーションによって、最初の 3 文字が数字に置き換えられます。4 番目の文字は置き換えられません。残りの文字は、英文字に置き換えられます。

マスクフォーマットの定義がない場合、各ソース文字は任意の文字に置き換えられます。マスクフォーマットが入力文字列より長い場合は、マスクフォーマットの余分な文字が無視されます。マスクフォーマットがソース文字列より短い場合、データマスキングトランスフォーメーションは残りの文字を「R」でマスクします。

注: 範囲オプションを使用してマスクフォーマットを設定することはできません。

ソース文字列の文字

ソース文字列の文字とは、マスクするように選択した文字列、またはマスクしないように選択したソース文字のことです。ソース文字列内での文字の位置は、重要ではありません。ソース文字では、大文字と小文字が区別されます。

設定できる文字数に制限はありません。[文字] が空白の場合は、カラム内のすべてのソース文字列が置き換えられます。

ソース文字列の文字について、以下のいずれかのオプションを選択します。

指定文字のみマスク

データマスキングトランスフォーメーションでは、ソースに含まれる文字のうち、ソース文字列の文字として設定されている文字がマスクされます。たとえば、文字 A、B、および c を入力すると、ソースデータに出現した A、B、または c が別の文字に置き換えられます。A、B、または c ではないソース文字は置き換えられません。マスクでは、大文字と小文字が区別されます。

指定文字以外をすべてマスク

ソース文字列に出現したソース文字列の文字を除いて、すべての文字をマスクします。たとえば、フィルタソース文字 "-" を入力し、[指定文字以外をすべてマスク] を選択した場合は、文字 "-" がソースデータに出現しても置き換えられません。それ以外のソース文字は変更されます。

ソース文字列の例

ソースファイルに「[従属]」という名前のカラムがあるとします。「[従属]」カラムには、カンマで区切られた複数の名前が含まれています。「[従属]」カラムをマスクし、名前を区切るカンマをテストデータで保持する必要があります。

「[従属]」カラムに対して、「[ソース文字列の文字]」を選択します。「[マスクしない]」を選択し、対象外の文字として","を入力します。引用符は入力しないでください。

データマスキングトランスフォーメーションによって、ソース文字列内のカンマ以外の文字がすべて置き換えられます。

結果文字列の置換文字

結果文字列の置換文字は、マスクされたデータに含める置き換え文字として選択した文字です。結果文字列の置換文字を設定すると、データマスキングトランスフォーメーションによって、ソース文字列内の文字が結果文字列の置換文字に置き換えられます。異なる入力値に対して同じ出力が生成されないようにするには、設定する置換文字の範囲を広くするか、マスクするソース文字の数を少なくします。文字列内での文字の位置は、重要ではありません。

結果文字列の置換文字について、以下のいずれかのオプションを選択します。

指定文字のみ使用

結果文字列の置換文字として定義した文字のみを使用してソースをマスクします。例えば、文字 A、B、および c を入力すると、ソースカラムの各文字が A、B、または c に置き換えられます。"horse"という単語は、"BACBA"などに置き換えられます。

指定文字以外をすべて使用

結果文字列の置換文字として定義した文字以外を使用してソースをマスクします。例えば、結果文字列の置換文字として A、B、および c を入力すると、マスクされたデータに文字 A、B、または c は出現しません。

結果文字列の置換文字の例

「[従属]」カラムに含まれるすべてのカンマをセミコロンに置き換えるには、以下のタスクを完了します。

1. ソース文字列の文字としてカンマを設定し、「[指定文字のみマスク]」を選択します。
データマスキングトランスフォーメーションでは、「[従属]」カラムにカンマが出現した場合にのみ、カンマがマスクされます。
2. 結果文字列の置換文字としてセミコロンを設定し、「[指定文字のみ使用]」を選択します。
データマスキングトランスフォーメーションでは、「[従属]」カラムにカンマが出現するたびにカンマがセミコロンに置き換えられます。

範囲

数値、日付、または文字列データの範囲を定義します。数値または日付値の範囲を定義すると、データマスキングトランスフォーメーションによって、ソースデータが上限値と下限値の範囲内の値でマスクされます。文字列の範囲を設定する場合は、文字列長の範囲を設定します。

文字列の範囲

ランダム文字列マスキングを設定すると、データマスキングトランスフォーメーションによって、ソース文字列の長さとは異なる長さの文字列が生成されます。オプションで、文字列長の上限と下限を設定できます。文

字列長の上限および下限として入力する値は、正の整数である必要があります。長さは、ポート精度以下である必要があります。

数値の範囲

数値カラムの上限値と下限値を設定します。上限値は、ポート精度以下である必要があります。デフォルトの範囲は、1 からポート精度長までです。

日付範囲

日時の値の上限値と下限値を設定します。上限と下限の各フィールドには、デフォルトの日付の上限と下限が表示されます。デフォルトの日付形式は、MM/DD/YYYY HH24:MI:SS です。上限の日付は、下限の日付より後である必要があります。

ブラー

ブラーでは、ソースデータ値に対する固定偏差またはパーセント偏差の範囲の出力値が生成されます。元の値に近いランダムな値が返されるようにする場合は、ブラーを設定します。ブラーの対象は、数値および日付値です。

数値のブラー

ソース数値のブラー方法として、固定偏差またはパーセント偏差を選択します。低ブラー値は、ソース値より小さい値に関する偏差です。高ブラー値は、ソース値より大きい値に関する偏差です。どちらの値もゼロ以上である必要があります。データマスキングトランスフォーメーションによって返されるマスクされたデータで、数値データは定義した値の範囲内の値に置き換えられます。

以下の表に、入力ソース値が 66 の場合のブラーの範囲値に応じたマスキング結果を示します。

ブラーの種類	低	高	結果
固定長	0	10	66～76
固定長	10	0	56～66
固定長	10	10	56～76
Percent	0	50	66～99
Percent	50	0	33～66
Percent	50	50	33～99

日付値のブラー

ブラーを設定すると、ソース日付に対する偏差で日付をマスクできます。偏差の適用対象の日付単位を選択します。年、月、日、または時を選択できます。ソース日付単位の上下の偏差を定義するには、上限と下限を入力します。データマスキングトランスフォーメーションは、偏差を適用し、偏差の範囲内の日付を返します。

例えば、マスクされた日付をソース日付の 2 年以内に制限するには、単位として年を選択します。高低の境界として 2 を入力します。ソースデータが 02/02/2006 の場合、データマスキングトランスフォーメーションは 02/02/2004 から 02/02/2008 の日付を返します。

デフォルトのブラー単位は年です。

式マスキング

式マスキングは、データを変更または新しいデータを作成するための式をポートに適用します。式マスキングを設定するときは、式エディタで式を作成します。式を構築するための入力ポートおよび出力ポート、関数、変数、演算子を選択します。

複数のポートからのデータを連結して別のポートの値を作成できます。例えば、ログイン名を作成する必要があるとします。ソースには名と姓のカラムがあります。ルックアップファイルから名と姓をマスクします。データマスキングトランスフォーメーションで、Login という別のポートを作成します。Login ポートで、名の最初の文字を姓と連結するための式を設定します。

```
SUBSTR(FIRSTNM,1,1)||LASTNM
```

ポートの式マスキングを設定する場合、ポート名はデフォルトで式として表示されます。式エディタにアクセスするには、[開く] ボタンをクリックします。式エディタには、式マスキングに対して設定されていない入力ポートと出力ポートが表示されます。ある式からの出力を別の式に対する入力として使用することはできません。出力ポート名を手動で式に追加した場合、予期しない結果が発生することがあります。

ポイントアンドクリックインタフェースを使用して関数、ポート、変数、および演算子を選択することで、式を作成するときのエラーを減らすことができます。

式を作成するときは、その式が、ポートのデータタイプに一致する値を返すことを確認します。式ポートのデータタイプが数値であり、式のデータタイプが同じではない場合、データマスキングトランスフォーメーションではゼロが返されます。式ポートのデータタイプが文字列であり、式のデータタイプが同じではない場合、データマスキングトランスフォーメーションでは NULL 値が返されます。

再現可能な式マスキング

複数のテーブルでソースカラムが出現し、各テーブルのカラムを同じ値でマスクする必要がある場合、再現可能な式マスキングを設定します。

再現可能な式マスキングを設定すると、データマスキングトランスフォーメーションによって式の結果が格納テーブルに保存されます。別のソーステーブルで同じカラムが出現した場合、データマスキングトランスフォーメーションは式からマスク値を返すのではなく、格納テーブルからマスク値を返します。

ディクショナリ名

再現可能な式マスキングを設定する場合、ディクショナリ名を入力する必要があります。ディクショナリ名は、複数のデータマスキングトランスフォーメーションで、同じソース値から同じマスク値を生成できるようにするキーです。各データマスキングトランスフォーメーションで同じディクショナリ名を定義します。ディクショナリ名には、任意のテキストを指定できます。

格納テーブル

ストレージテーブルには、セッション間の再現可能な式マスキングの結果が含まれます。格納テーブルの行には、ソース列およびマスクされた値ペアが含まれています。式マスキング用の格納テーブルは、置換マスキング用の格納テーブルとは別のテーブルです。

データマスキングトランスフォーメーションでは、再現可能な式で値がマスクされるたびに、ディクショナリ名、ロケール、列名、および入力値によって格納テーブルの検索が行われます。格納テーブル内に行が見つかったら、格納テーブルからマスクされた値が返されます。データマスキングトランスフォーメーションで行が見つからない場合、その列に対して式からマスク値が生成されます。

ストレージ内のデータの暗号化を解除した後、同じディクショナリ名をキーとして使用する場合は、式マスキング用の格納テーブルを暗号化する必要があります。

式マスキング用の格納テーブルの暗号化

格納テーブルは、トランスフォーメーション言語エンコード機能を使用して暗号化できます。ストレージの暗号化を有効にした場合は、格納テーブルを暗号化する必要があります。

1. IDENTITY_EXPRESSION_STORAGE 格納テーブルをソースとして使用し、マッピングを作成します。
2. データマスキングトランスフォーメーションを作成します。
3. マスクされた値ポートに式マスキング方法を適用します。
4. MASKEDVALUE ポートに対して次の式を使用します。
`Enc_Base64(AES_Encrypt(MASKEDVALUE, Key))`
5. ポートをターゲットにリンクします。

例

例えば、従業員テーブルに以下のカラムが含まれているとします。

FirstName
LastName
LoginID

データマスキングトランスフォーメーションでは、FirstName と LastName を組み合わせた式を使用して LoginID がマスクされます。式マスクを再現可能に設定します。再現可能なマスキングのキーとしてディクショナリ名を入力します。

Computer_Users テーブルには LoginID が含まれていますが、FirstName 列または LastName 列は含まれていません。

Dept
LoginID
Password

Computer_Users テーブル内の LoginID を同じ LoginID で従業員としてマスクするには、LoginID カラム用の式マスキングを設定します。再現可能なマスキングを有効にし、LoginID 従業員テーブルに対して定義した同じディクショナリ名を入力します。Integration Service では、格納テーブルから LoginID 値が取得されます。

Integration Service で LoginID の格納テーブル内の行が見つからない場合に使用するデフォルトの式を作成します。Computer_Users テーブルには FirstName カラムまたは LastName カラムがないので、式によって生成される LoginID にはあまり意味がありません。

格納テーブルスクリプト

Informatica は、格納テーブルを作成するために実行できるスクリプトを提供します。スクリプトは以下の場所に格納されています。

<PowerCenter installation directory>\client\bin\Extensions\DataMasking

ディクショナリには、Sybase、Microsoft SQL Server、IBM DB2、および Oracle の各データベース用のスクリプトが含まれています。各スクリプトの名前は、<Expression_<データベースタイプ>です。

式マスキングのルールとガイドライン

式マスキングには以下のルールおよびガイドラインを使用します。

- ある式からの出力を別の式に対する入力として使用することはできません。出力ポート名を手動で式に追加した場合、予期しない結果が発生することがあります。
- 式を作成する際は、ポイントアンドクリックを使用します。ポイントアンドクリックインタフェースを使用して関数、ポート、変数、および演算子を選択することで、式を作成するときのエラーを減らすことができます。

- 再現可能なマスキング用にデータマスキングトランスフォーメーションが設定されており、格納テーブルが存在しない場合、Integration Service はソースデータをデフォルト値で置換します。

関連項目：

- [「式エディタの使用」 \(ページ 33\)](#)

特殊マスク形式

以下の種類のマスクは、元のデータの形式を保持します。

- 社会保障番号
- クレジットカード番号
- 電話番号
- URL アドレス
- 電子メールアドレス
- IP アドレス
- 社会保険番号

データマスキングトランスフォーメーションは機密データを、現実的な形式のマスクされた値に置き換えます。例えば、SSN をマスクすると、データマスキングトランスフォーメーションにより形式は正しいものの有効ではない SSN が返されます。

ソースデータの形式またはデータタイプがマスクとして無効の場合、Integration Service では、デフォルトのマスクがデータに適用されます。Integration Service では、デフォルト値のファイルからマスクされた値が適用されます。デフォルト値のファイルを編集してデフォルト値を変更できます。

関連項目：

- [「デフォルト値ファイル」 \(ページ 152\)](#)

社会保障番号のマスキング

データマスキングトランスフォーメーションは、社会保障庁の最新の High Group List に基づいて、有効でない社会保障番号を生成します。High Group List には、社会保障局が発行した有効な番号が記載されています。データマスキングトランスフォーメーションでは、以下の場所の High Group List にアクセスします。

<Installation Directory>\infa_shared\SrcFiles\highgroup.txt

データマスキングトランスフォーメーションでは、ハイグループ履歴リストにない SSN 番号が生成されます。社会保障庁では、ハイグループ履歴リストを毎月更新しています。このリストの最新バージョンは、以下の場所からダウンロードします。

<http://www.socialsecurity.gov/employer/ssns/highgroup.txt>

社会保障番号（SSN）形式

データマスキングトランスフォーメーションは、9桁の数字を含む SSN 形式を受け付けます。この数字は任意の文字で区切ることができます。例えば、データマスキングトランスフォーメーションは以下の形式を受け付けます。+54-*9944\$#789-,*()”。

地域コードの要件

データマスキングトランスフォーメーションは、ソースと同じ形式で有効でない社会保障番号を返します。SSN の最初の 3 桁では地域コードが定義されます。地域コードはマスクされません。グループ番号とシリアル番号はマスクされます。ソース SSN には有効な地域コードが含まれている必要があります。データマスキングトランスフォーメーションは、ハイグループリストから地域コードを検索し、マスクされたデータとして適用できる未使用の番号の範囲を判断します。SSN が無効な場合、ソースデータはマスクされません。

再現可能な社会保障番号のマスキング

社会保障番号には、再現可能なマスキングを設定できます。社会保障番号に対して再現可能なマスキングを設定するには、[再現可能な出力] をクリックして、[シード値] または [マッピングパラメータ] を選択します。

[シード値] を選択すると、Designer では乱数がシードとして割り当てられます。異なるソースデータで同じ社会保障番号を生成するには、各データマスキングトランスフォーメーションのシード値を変更して、他のトランスフォーメーションの社会保障番号のシード値に一致させます。マッピングでデータマスキングトランスフォーメーションを定義している場合は、シード値に対するマッピングパラメータを設定できます。

データマスキングトランスフォーメーションでは、再現可能なマスキングが設定された確定的な社会保障番号が返されます。データマスキングトランスフォーメーションでは、社会保障庁が発行した有効な社会保障番号が返されないため、一意のすべての社会保障番号を返すことはできません。

クレジットカード番号のマスキング

クレジットカードマスキングは、組み込みのマスク形式を適用してクレジットカード番号を隠蔽します。複数のクレジットカード番号形式を入力できます。または、クレジットカード発行者を上書きすることができます。

PowerCenter Integration Service では、有効なクレジットカード番号をマスクする場合、論理的に有効なクレジットカード番号が生成されます。ソースクレジットカード番号は 13～19 桁である必要があります。入力クレジットカード番号に、クレジットカード業界のルールに基づく有効なチェックサム値がある必要があります。

ソースクレジットカード番号には、数字、スペース、およびハイフンを使用できます。クレジットカード番号に無効な文字が含まれている場合、または長さが正しくない場合、Integration Service によって、セッションログにエラーが書き込まれます。ソースデータが無効の場合、PowerCenter Integration Service によって、デフォルトのクレジットカード番号マスクが適用されます。

クレジットカードマスキングを設定して、クレジットカード番号の最初の 6 桁を維持または置換できます。これらの桁の組み合わせによって、クレジットカード発行者を特定します。クレジットカード発行者を変える場合は、別のクレジットカード発行者を指定できます。以下のクレジットカード発行者を指定できます。

- American Express
- Discover
- JCB
- MasterCard

- Visa
- 任意

[任意] を選択すると、データマスキングトランスフォーメーションはすべてのクレジットカード発行者の組み合わせを返します。クレジットカード発行者をマスクすると、データマスキングトランスフォーメーションは非固有値を返します。

例えば、CUSTOMER テーブルに以下のクレジットカード番号があるとします。

```
2131 0000 0000 0008
5500 0000 0000 0004
6334 0000 0000 0004
```

クレジットカード発行者を 1 つ選択した場合、クレジットカード番号は最後の桁以外を共有します。有効なクレジットカード番号を生成するには、データマスキングトランスフォーメーションが各クレジットカードの最後の桁を同じ値に設定する必要があります。

電話番号マスキング

データマスキングトランスフォーメーションでは、元の電話番号のフォーマットが変更されることなく、電話番号がマスクされます。例えば、電話番号(408)382 0658 は、(607)256 3106 などとしてマスクされます。

ソースデータには、数字、スペース、ハイフン、およびかっこを使用できます。Integration Service では、英文字または特殊文字はマスクされません。

データマスキングトランスフォーメーションは、string、integer、bigint データをマスクできます。

電子メールアドレスマスキング

データマスキングトランスフォーメーションを使用し、文字列値が含まれる電子メールアドレスをマスクします。データマスキングトランスフォーメーションを使用すると、電子メールアドレスをランダムな ASCII 文字でマスクすることも、電子メールアドレスを現実的な電子メールアドレスに置き換えることもできます。

電子メールアドレスには次のタイプのマスキングを適用できます。

標準の電子メールマスキング

データマスキングトランスフォーメーションは、電子メールアドレスをマスクするときにランダムな ASCII 文字を返します。例えば、Georgesmith@yahoo.com が KtrlupQAPyk@vdSKh.BIC などとしてマスクされます。デフォルトは標準のマスキングです。

詳細電子メールマスキング

データマスキングトランスフォーメーションは、トランスフォーメーション出力ポートまたはディクショナリカラムから派生した他の現実的な電子メールアドレスを使用して電子メールアドレスをマスクします。

詳細電子メールマスキング

詳細電子メールマスキングを使用すると、電子メールアドレスを他の現実的な電子メールアドレスでマスクすることができます。データマスキングトランスフォーメーションでは、ディクショナリカラムまたはトランスフォーメーション出力ポートから電子メールアドレスを作成します。

電子メールアドレスのローカル部分は、マッピング出力ポートから作成できます。または、リレーショナルテーブルカラムやフラットファイルカラムから電子メールアドレスのローカル部分を作成することもできます。

データマスキングトランスフォーメーションでは、電子メールアドレスのドメイン名を定数値またはドメインディクショナリ内のランダム値から作成できます。

詳細電子メールマスキングは、次のオプションに基づいて作成できます。

マッピングに基づいた電子メールアドレス

電子メールアドレスは、データマスキングトランスフォーメーション出力ポートに基づいて作成できます。名と姓のカラムのためのトランスフォーメーション出力ポートを選択します。データマスキングトランスフォーメーションは、名と姓の長さに指定された値に基づいて、名、姓、またはこれらの両方をマスクします。

ディクショナリに基づいた電子メールアドレス

電子メールアドレスは、ディクショナリのカラムに基づいて作成できます。ディクショナリは、リレーショナルテーブルでもフラットファイルでもかまいません。

名と姓のディクショナリカラムを選択します。データマスキングトランスフォーメーションは、名と姓の長さに指定された値に基づいて、名、姓、またはこれらの両方をマスクします。式を設定して電子メールアドレスをマスクすることもできます。式を設定すると、データマスキングトランスフォーメーションが名または姓のカラムの長さに基づいてマスクすることはありません。

詳細電子メールアドレスマスキングタイプの設定パラメータ

詳細電子メールアドレスマスキングを設定する際には、設定パラメータを指定します。

以下のパラメータを指定できます。

区切り文字

電子メールアドレス内の名と姓を区切るために、ドット、ハイフン、アンダースコアなどの区切り文字を選択できます。電子メールアドレス内の名と姓を区切らない場合は、区切り文字を空欄のままにします。

【名】 カラム

電子メールアドレス内の名をマスクするデータマスキングトランスフォーメーション出力ポートまたはディクショナリカラムを選択します。

【姓】 カラム

電子メールアドレス内の姓をマスクするデータマスキングトランスフォーメーション出力ポートまたはディクショナリカラムを選択します。

【名】 カラムまたは 【姓】 カラムの長さ

【名】 カラムと 【姓】 カラムでマスクする文字列の長さを制限します。例えば、入力データの名が Timothy で、姓が Smith であるとし、【名】 カラムの長さを 5、【姓】 カラムの長さを 1 に選択し、区切り文字にはドットを選択したとします。この設定では、データマスキングトランスフォーメーションによって次の電子メールアドレスが生成されます。

```
timot.s@<domain_name>
```

DomainName

ドメイン名には、gmail.com のような一定の値を使用できます。または、ドメイン名の一覧が含まれている別のディクショナリファイルを指定することもできます。ドメインディクショナリは、フラットファイルでもリレーショナルテーブルでもかまいません。

詳細電子メールアドレスマスキングタイプの式

ディクショナリカラムを選択して電子メールアドレスを作成するには、式機能を使用できます。

詳細電子メールアドレスマスキングタイプの式を設定するには、電子メールアドレスを形成するカラムをディクショナリから選択します。続いて、ディクショナリカラム、トランスフォーメーションポート、またはこの両方を式に設定できます。

ディクショナリポートは、式エディタに次の構文で一覧表示されます。

```
<Input string mapping port>_Dictionary_<dictionary column name>
```

社会保険番号のマスキング

このデータマスキングトランスフォーメーションでは、9桁の社会保険番号がマスクされます。番号は任意の文字で区切ることができます。

番号に区切り文字が含まれない場合、マスクされた番号に区切り文字は含まれません。それ以外の場合、マスクされた番号は以下の形式になります。

XXX-XXX-XXX

SIN の開始桁

マスクされた SIN の最初の桁を定義できます。

【開始桁】を有効にして、桁を数字で入力します。データマスキングトランスフォーメーションを実行すると、ここで入力した桁より上をマスクした SIN 番号が作成されます。

再現可能な SIN 番号

再現可能な SIN 値を返すようにデータマスキングトランスフォーメーションを設定することができます。再現可能な SIN マスキングに対応するようポートを設定すると、データマスキングトランスフォーメーションでは、ソース SIN 値とシード値が同じ場合に、マスクされた確定的なデータが返されます。

再現可能な SIN 番号を返すには、【再現可能な値】を有効にしてシード番号を入力します。データマスキングトランスフォーメーションは、各 SIN に対して一意の値を返します。

IP アドレスマスキング

データマスキングトランスフォーメーションでは、IP アドレスをマスクする場合、ピリオドで区切られた4つの数による別の IP アドレスとしてマスクされます。最初の数はネットワークを表します。ネットワーク番号は、ネットワークの範囲内でマスクされます。

クラス A の IP アドレスはクラス A の IP アドレスにマスクされ、10.x.x.x アドレスは 10.x.x.x アドレスにマスクされます。クラスとプライベートネットワークアドレスはマスクされません。例えば、11.12.23.34 は 75.32.42.52 に、10.23.24.32 は 10.61.74.84 に、それぞれマスクされます。

注: 多数の IP アドレスをマスクした場合に、IP アドレスのクラスやプライベートネットワークがマスクされないために、データマスキングトランスフォーメーションで非固有値が戻されることがあります。

URL アドレスマスキング

データマスキングトランスフォーメーションでは、'://'文字列を検索し、その右側の部分文字列を解析することによって URL が解析されます。ソース URL には、'://'文字列が含まれている必要があります。ソース URL には、数字と英文字を使用できます。

URL のプロトコル部分はマスクされません。URL が http://www.yahoo.com の場合は、http://MgL.aHjCa.VsD/が返されます。データマスキングトランスフォーメーションは、有効でない URL を生成する可能性があります。

注: URL の場合は、常に ASCII 文字が返されます。

デフォルト値ファイル

ソースデータの形式またはデータタイプがマスクとして無効の場合、Integration Service では、デフォルトのマスクがデータに適用されます。Integration Service では、デフォルト値のファイルからマスクされた値が適用されます。デフォルト値のファイルを編集してデフォルト値を変更できます。

デフォルト値のファイルは、以下の場所にある XML ファイルです。

<PowerCenter Installation Directory>\infa_shared\SrcFiles\defaultValue.xml

defaultValue.xml ファイルには、以下の名前と値のペアが用意されています。

```
<?xml version="1.0" standalone="yes" ?>
<defaultValue
default_char = "X"
default_digit = "9"
default_date = "11/11/1111 00:00:00"
default_email = "abc@xyz.com"
default_ip = "99.99.9.999"
default_url = "http://www.xyz.com"
default_phone = "999999999"
default_ssn = "999-99-9999"
default_cc = "9999 9999 9999 9999"
default_seed = "500"
/>
```


データマスキングトランスフォーメーションのセッションプロパティ

データマスキングトランスフォーメーションのセッションプロパティを設定して、パフォーマンスを向上させることができます。

以下のセッションプロパティを設定します。

キャッシュサイズ

メインメモリのディクショナリキャッシュのサイズ。パフォーマンスを向上させるには、メモリサイズを増やします。推奨される最小サイズは、100,000 レコードに対して 32 MB です。デフォルトは 8MB です。

キャッシュディレクトリ

ディクショナリキャッシュの場所。そのディレクトリに対する書き込み権限が必要です。デフォルトは \$PMCacheDir です。

共有ストレージテーブル

データマスキングトランスフォーメーションのインスタンス間でのストレージテーブルの共有を可能にします。共有ストレージテーブルは、データマスキングトランスフォーメーションのインスタンスで、データベース接続、シード値、およびロケールに対して同じディクショナリカラムが使用されているときに有効にします。共有ストレージテーブルは、同じデータマスキングトランスフォーメーション内の 2 つのポートで、接続、シード、およびロケールに対して同じディクショナリカラムが使用されているときにも有効にできます。データマスキングトランスフォーメーションまたはポートでディクショナリカラムが共有されていないときは、共有ストレージテーブルを無効にします。デフォルトでは無効になっています。

ストレージのコミット間隔

ストレージテーブルに一度にコミットする行数。パフォーマンスを向上させるには、この値を増やします。コミット間隔は、共有ストレージテーブルを設定しないときに設定します。デフォルトは 100,000 です。

ストレージの暗号化

IDM_SUBSTITUTION_STORAGE、IDM_EXPRESSION_STORAGE などのストレージテーブルを暗号化します。暗号化ストレージプロパティを有効にする前に、ストレージテーブル内のデータが暗号化されていること確認してください。ストレージテーブルを暗号化しない場合は、このオプションを無効にします。デフォルトでは無効になっています。

ストレージの暗号化キー

データマスキングトランスフォーメーションは、ストレージ暗号化キーに基づいてストレージを暗号化します。同一のデータマスキングトランスフォーメーションインスタンスの各セッションの実行には、同じ暗号化キーを使用します。

データマスキングトランスフォーメーションのルールとガイドライン

データマスキングトランスフォーメーションを設定する場合には、以下のルールおよびガイドラインに従います。

- データマスキングトランスフォーメーションは NULL 値をマスクしません。ソースデータに NULL 値が含まれている場合、データマスキングトランスフォーメーションは NULL 値を返します。NULL 値を置換するには、入力ポートにユーザー定義のデフォルト値を指定できるアップストリームトランスフォーメーションを追加します。

- ソースデータの形式またはデータタイプがマスクとして無効の場合、Integration Service では、デフォルトのマスクがデータに適用されます。Integration Service では、デフォルト値のファイルからマスクされた値が適用されます。
- データマスキングトランスフォーメーションは、ソースと同じ形式と地域コードを持つ、無効な社会保障番号を返します。社会保障庁が特定の地域の社会保障番号の半数以上を発行している場合、データマスキングトランスフォーメーションはキーマスキングによって一意の無効な社会保障番号を返すことができない場合があります。

第 6 章

データマスキングの例

この章では、以下の項目について説明します。

- [名前と住所のルックアップファイル, 155 ページ](#)
- [ルックアップトランスフォーメーションを使用したデータの置換, 155 ページ](#)
- [式トランスフォーメーションを使用したデータのマスキング, 159 ページ](#)

名前と住所のルックアップファイル

データマスキングのインストールには、データのマスキングに使用する名前、住所、および会社名を含むフラットファイルがいくつか含まれます。ルックアップトランスフォーメーションを設定して、ランダムな姓と名、および住所をこれらのファイルから取得できます。

データマスキングサーバーコンポーネントのインストール時に、インストーラによって、以下のファイルが `server\infa_shared\LkpFiles` フォルダに配置されています。

- **Address.dic**。住所を含むフラットファイルです。各レコードには、シリアル番号、番地、市区町村名、都道府県名、郵便番号、および国が含まれています。
- **Company_names.dic**。会社名を含むフラットファイルです。各レコードには、シリアル番号と会社名が含まれています。
- **Firstnames.dic**。名（ファーストネーム）を含むフラットファイルです。各レコードには、シリアル番号、名、および性別コードが含まれています。
- **Surnames.dic**。姓（ラストネーム）を含むフラットファイルです。各レコードには、シリアル番号と姓が含まれています。

ルックアップトランスフォーメーションを使用したデータの置換

カラムのデータを、似ているが関連のないデータに置き換えることができます。置換は、センシティブデータを本物のように見えるデータにマスクする方法として効果的です。

以下の例は、複数のルックアップトランスフォーメーションを設定して、テストデータを取得し、それをソースデータと置き換える方法を示しています。CUSTOMERS_PROD テーブルのセンシティブなフィールドをマスクするデータマスキングマッピングを作成します。

この例には、以下の種類のマスキングが含まれています。

- ルックアップテーブルを使用した名前と住所の置換
- キーマスキング
- ブラー
- 特殊マスク形式

注: この例は、client\samples フォルダからリポジトリにインポートできる M_CUSTOMERS_MASKING.xml のマッピングを示しています。

センシティブデータは、Customers_Prod という顧客データベーステーブルに含まれています。セキュリティを保ちながら、テストシナリオで顧客データを使用する必要があります。この場合、各カラムのデータをマスクし、テストデータを Customers_Test というターゲットテーブルに書き込みます。

Customers_Prod テーブルには、以下のカラムが含まれています。

カラム	データタイプ
CustID	Integer
FullName	String
住所	String
電話番号	String
ファックス	String
CreatedDate	日付
電子メール	String
SSN	String
CreditCard	String

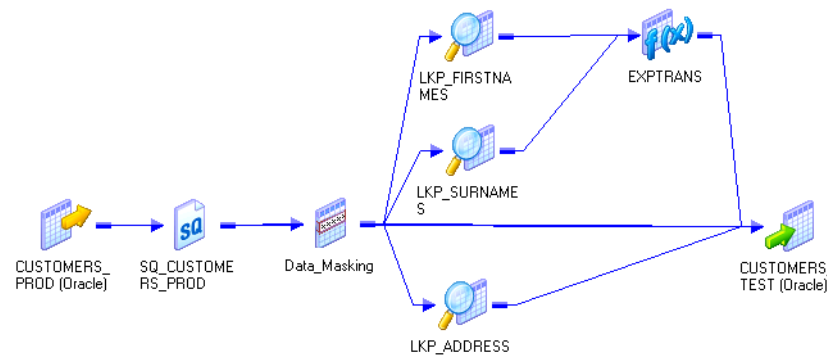
置き換える値をディクショナリファイルからルックアップするマッピングを作成できます。データマスキングトランスフォーメーションによって、顧客データがディクショナリファイルの値でマスクされます。ファイルには、名ファイル、姓ファイル、および住所ファイルが含まれています。

以下の表に server\infra_shared\LkpFiles フォルダに用意されているファイルを示します。

ファイル	レコード数	フィールド	説明
Firstnames.dic	21,000	SNO、Gender、Firstname	名（ファーストネーム）のアルファベット順の一覧です。シリアル番号は 1～21,000 です。Gender は、それぞれが男性名か女性名かを示します。
Surnames.dic	81,000	SNO、Surname	姓（ラストネーム）のアルファベット順の一覧です。シリアル番号は 1～81,000 です。
Address.dic	13,000	SNO、Street、City、State、Zip、Country	完結した住所の一覧です。シリアル番号は 1～13,000 です。

注: Firstnames.dic ファイルには Gender カラムが用意されているため、これを使用してルックアップソース ファイルを男女別に作成できます。男性名は男性名で、女性名は女性名であり、それぞれマスクする必要がある場合は、ルックアップ条件で Gender カラムを使用します。

以下の図は、インポート可能なマッピングを示しています。



マッピングには、ソースおよびターゲットとともに以下のトランスフォーメーションが含まれます。

- **ソース修飾子。** 顧客データをデータマスキングトランスフォーメーションに渡します。 CustID カラムをトランスフォーメーションの複数のポートに渡します。
 - **CustID。** 顧客番号。
 - **Randid1。** 名のルックアップ用に生成される乱数。
 - **Randid2。** 姓のルックアップ用に生成される乱数。
 - **Randid3。** 住所のルックアップ用に生成される乱数。
- **データマスキングトランスフォーメーション。** 置換用の姓、名、住所をルックアップするための乱数を生成します。また、特殊マスクフォーマットを電話番号、ファックス、電子メールアドレス、およびクレジットカード番号に適用します。データマスキングトランスフォーメーションは、以下のカラムをマスクします。

入力ポート	マスキングのタイプ	マスキングルール	説明	出力先
CustID	キー	seed = 934	CustID はプライマリキーカラムです。再現可能で確定的な乱数を使用してマスクする必要があります。	Customers_Test
Randid1	ランダム	範囲 下限 = 0 上限 = 21000	LKUP_Firstnames トランスフォーメーションで名のルックアップで使用する乱数。	LKUP_Firstnames
Randid2	ランダム	範囲 下限 = 0 上限 = 13000	LKUP_Surnames トランスフォーメーションで姓のルックアップで使用する乱数。	LKUP_Surnames
Randid3	ランダム	範囲 下限 = 0 上限 = 81000	LKUP_Address トランスフォーメーションで住所のルックアップで使用する乱数。	LKUP_Address

入力ポート	マスキング のタイプ	マスキングルール	説明	出力先
電話番号	電話番号	-	ソース電話番号と同じフォーマットの電話番号。	Customers_Test
ファックス	電話番号	-	ソース電話番号と同じフォーマットの電話番号。	Customers_Test
CreatedDate	ランダム	ブラー 単位 = 年 下限 = 1 上限 = 1	ソース年から 1 年以内のランダムな日付。	Customers_Test
電子メール	電子メールアドレス	-	元のアドレスと同じフォーマットの電子メールアドレス。	Customers_Test
SSN	SSN	-	highgroup.txt ファイルに含まれていない SSN。	Customers_Test
CreditCard	クレジットカード	-	最初の 6 桁がソースと同じで、有効なチェックサムを持つクレジットカード番号。	Customers_Test

- LKUP_Firstnames.** Firstnames.dic に対するフラットファイルルックアップを実行します。このトランスフォーメーションでは、シリアル番号が乱数 Randid1 と同じレコードを取得します。ルックアップ条件は、以下のとおりです。
SNO = out_RANDID1
LKUP_Firstnames トランスフォーメーションによって、マスクされた名が Exptrans 式トランスフォーメーションに渡されます。
 - LKUP_Surnames.** Surnames.dic に対するフラットファイルルックアップが実行されます。このトランスフォーメーションによって、シリアル番号が乱数 Randid2 と同じレコードが取得されます。
LKUP_Surnames トランスフォーメーションによって、マスクされた姓が Exptrans 式トランスフォーメーションに渡されます。
 - Exptrans.** 姓と名を組み合わせるフルネームが返されます。式トランスフォーメーションによって、フルネームが Customers_Test ターゲットに渡されます。
姓と名を組み合わせる式は、以下のとおりです。
FIRSTNAME || ' ' || SURNAME
 - LKUP_Address.** Address.dic に対するフラットファイルルックアップが実行されます。このトランスフォーメーションによって、シリアル番号が乱数 Randid3 と同じレコードが取得されます。ルックアップトランスフォーメーションによって、アドレスの各カラムがターゲットに渡されます。
- Customer_Test テーブルは、テスト環境で使用できます。

式トランスフォーメーションを使用したデータのマスクング

式トランスフォーメーションをデータマスクングトランスフォーメーションと組み合わせて使用すると、カラムのいずれかをマスクした後での2つのカラム間の関係を保持できます。

例えば、保険契約の開始日と終了日を含む顧客情報をマスクする場合、マスクされた各レコードで契約期間を保持するとします。その場合は、データマスクングトランスフォーメーションを使用して、終了日以外のすべてのデータをマスクします。次に、式トランスフォーメーションを使用して、契約期間を計算し、マスクされた開始日に加算します。

この例には、以下の種類のマスクングが含まれています。

- キー
- 日付のブラー
- 数のブラー
- マスクフォーマット指定

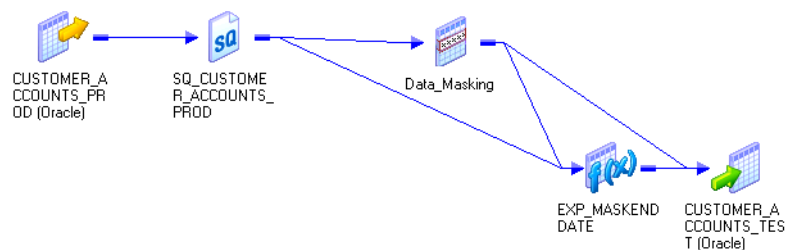
注: この例は、client\samples フォルダからリポジトリにインポートできる M_CUSTOMER_ACCOUNTS_MASKING.xml マッピングです。

センシティブデータは、Customers_Prod という顧客データベーステーブルに含まれています。この場合、各カラムのデータをマスクし、テストデータを Customers_Test というターゲットテーブルに書き込みます。

以下の Customer_Accounts_Prod カラムをマスクします。

カラム	データ型
AcctID	String
CustID	Integer
Balance	ダブル
StartDate	Datetime
EndDate	Datetime

以下の図は、インポート可能なマッピングを示しています。



マッピングには、ソースおよびターゲットとともに以下のトランスフォーメーションが含まれます。

- **ソース修飾子。** AcctID、CustID、Balance、および Start_Date がデータマスクングトランスフォーメーションに渡されます。また、Start_Date カラムと End_Date カラムが式トランスフォーメーションに渡されます。

- **データマスキングトランスフォーメーション。** End_Date 以外のすべてのカラムがマスクされます。データマスキングトランスフォーメーションによって、マスクされたカラムがターゲットに渡されます。また、契約の開始日、終了日、およびマスクされた開始日が式トランスフォーメーションに渡されます。

データマスキングトランスフォーメーションによって、以下のカラムがマスクされます。

入力ポート	マスキングのタイプ	マスキングルール	説明	出力先
AcctID	ランダム	マスクフォーマット AA+DDDDD 結果文字列の置換文字 ABCDEFGHIJKLMNOPQRSTUVWXYZ	最初の 2 文字は大文字の英文字です。3 番目の文字はダッシュで、マスクされません。残りの 5 文字は数字です。	Customer_Account_Test のターゲット
CustID	キー	seed = 934	seed は 934 です。 CustID マスクは確定的です。	Customer_Account_Test のターゲット
Balance	ランダム	ブラー Percent 下限 = 10 上限 = 10	マスクされた Balance は、ソースバランスの 10 パーセントの範囲内になります。	Customer_Account_Test のターゲット
Start_Date	ランダム	ブラー 単位 = 年 下限 = 2 上限 = 2	マスクされた Start_Date はソース日付から 2 年以内になります。	Customer_Account_Test のターゲット Exp_MaskEndDateTransformation

- **式トランスフォーメーション。** マスクされた終了日が計算されます。まず、開始日と終了日の差が計算されます。次に、マスクされた開始日に結果が加算されて、マスクされた終了日が決定されます。

マスクされた終了日を生成する式は、以下のとおりです。

```
DIFF = DATE_DIFF(END_DATE, START_DATE, 'DD')
out_END_DATE = ADD_TO_DATE(out_START_DATE, 'DD', DIFF)
```

式トランスフォーメーションによって、out_END_DATE がターゲットに渡されます。

第 7 章

式トランスフォーメーション

この章では、以下の項目について説明します。

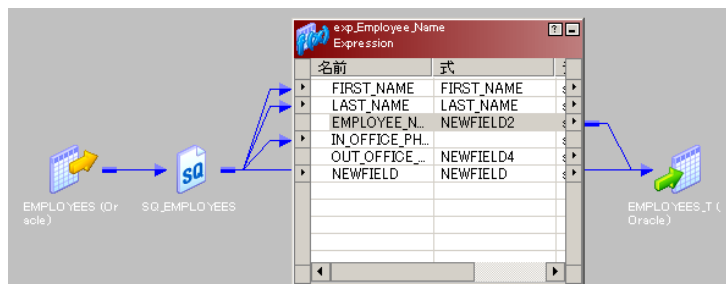
- [式トランスフォーメーションの概要, 161 ページ](#)
- [式トランスフォーメーションコンポーネント, 161 ページ](#)
- [ポートの設定, 162 ページ](#)
- [式トランスフォーメーションの作成, 163 ページ](#)

式トランスフォーメーションの概要

式トランスフォーメーションは、単一行の値を計算する場合に使用します。例えば、従業員の給与を調整したり、名前と姓を連結したり、文字列を数値に変換したりすることが可能です。また、式トランスフォーメーションを用いて条件文をテストした後で、結果をターゲットやその他のトランスフォーメーションに渡すこともできます。式トランスフォーメーションは、パッシブトランスフォーメーションです。

式トランスフォーメーションを使用して、非集計計算を実行します。合計値や平均値など、複数の行を対象とした計算を行うには、アグリゲータトランスフォーメーションを使用してください。

以下の図は、式トランスフォーメーションを使って EMPLOYEES テーブルから取得した従業員の姓と名前を連結する、単純なマッピングを示します。



式エディタで設定した式は評価できます。

式トランスフォーメーションコンポーネント

式トランスフォーメーションは、Transformation Developer または Mapping Designer で作成できます。

式トランスフォーメーションには、次のタブがあります。

- **【トランスフォーメーション】**。トランスフォーメーションの名前および説明を入力します。式トランスフォーメーションの命名規則は、「EXP_トランスフォーメーション名」です。また、トランスフォーメーションを再利用可能にすることもできます。
- **ポート**。ポートを作成して設定します。
- **プロパティ**。セッションログファイルに記録されたトランザクション詳細の量を決定するトレースレベルを設定します。
- **メタデータエクステンション**。エクステンション名、データタイプ、精度、および値を指定します。再利用可能なメタデータエクステンションも、作成することができます。

ポートの設定

[ポート] タブでは、ポートの作成および編集が可能です。

[ポート] タブでは、以下のコンポーネントを設定します。

- **ポート名**。ポートの名前。
- **データタイプ、精度、およびスケール**。ポートごとにデータタイプを設定し、精度およびスケールを設定します。
- **ポートタイプ**。ポートは、入力ポート、出力ポート、入出力ポート、または変数ポートのいずれかです。入力ポートはデータを受け取り、出力ポートはデータを渡します。入出力ポートはデータを変更せずに渡します。変数ポートはデータを一時的に格納し、いくつかの行に値を格納できます。
- **式**。式を入力するには、式エディタを使用します。式は、SQL に似た関数を含むトランスフォーメーション言語を使って計算を行います。
- **デフォルト値と説明**。ポート用のデフォルト値を設定し、説明を追加します。

値の計算

式トランスフォーメーションを使用して単一行の値を計算するには、以下のポートを含める必要があります。

- **入力ポートまたは入出力ポート**。計算に使用される値を提供します。たとえば、注文の合計価格を計算する場合は、入力ポートまたは入出力ポートを 2 つ作成します。1 つのポートで単価が提供され、もう 1 つのポートで注文個数が提供されます。
- **出力ポート**。式の戻り値を提供します。出力ポートの設定オプションとして式を入力します。また、ポートごとにデフォルト値を設定することもできます。

出力ポートごとに式を作成しておくことで、単一の式トランスフォーメーションで複数の式の入力が可能になります。たとえば、国税や地方税、社会保障費や健康保険料といった、各従業員からの数種の源泉税額を計算したい場合があります。これらの計算すべてに従業員の給料および源泉項目が必須で、さらに該当税率が必要な場合もあるため、給料および源泉項目用に入出力ポートを作成し、各計算用に別個の出力ポートを作成することをお勧めします。

式トランスフォーメーションの作成

以下の手順に従って、式トランスフォーメーションを作成します。

1. Mapping Designer でマッピングを開きます。
2. [トランスフォーメーション] - [作成] をクリックします。式トランスフォーメーションを選択します。
3. 名前を入力し、[Done] をクリックします。
4. ソース修飾子またはその他のトランスフォーメーションからポートを選択し、式トランスフォーメーションにドラッグして追加します。
また、トランスフォーメーションを開いて、ポートを手動で作成することもできます。
5. タイトルバーをダブルクリックし、[ポート] タブをクリックします。出力ポートおよび変数ポートは、トランスフォーメーション内で作成することができます。
6. ポートのデータタイプ、精度、および位取りを、式の戻り値に一致するように割り当てます。
7. [式] セクションで、出力ポートまたは変数ポートを選択し、式エディタを開きます。
8. 式を入力します。
9. [検証] をクリックして、式の構文を検証します。
10. 右側のペインの [テスト式] セクションで、式の条件の最新の変更を反映するには、[更新] をクリックします。
11. 右側のペインに、式内で使用される入力ポートのサンプル値を入力します。
12. 式を評価するには、[評価] をクリックします。
13. [OK] をクリックします。
14. [トランスフォーメーション] タブで、再利用可能トランスフォーメーションを作成します。
注: いったんトランスフォーメーションを再利用可能にした後は、ソース修飾子または他のトランスフォーメーションからポートをコピーできなくなります。ポートは、トランスフォーメーション内で手動で作成できます。
15. [プロパティ] タブで、トレースレベルを設定します。
16. [メタデータエクステンション] タブで、メタデータエクステンションを追加します。
17. [OK] をクリックします。
18. 出力ポートをダウンストリームトランスフォーメーションまたはターゲットに接続します。

第 8 章

エクスターナルプロシージャトランスフォーメーション

この章では、以下の項目について説明します。

- [エクスターナルプロシージャトランスフォーメーションの概要, 164 ページ](#)
- [エクスターナルプロシージャトランスフォーメーションのプロパティの設定, 166 ページ](#)
- [COM プロシージャの作成, 168 ページ](#)
- [Informatica エクスターナルプロシージャの作成, 175 ページ](#)
- [エクスターナルプロシージャの配布, 184 ページ](#)
- [作成にあたっての注意, 185 ページ](#)
- [初期化プロパティでのサービスプロセス変数, 191 ページ](#)
- [エクスターナルプロシージャのインタフェース, 192 ページ](#)

エクスターナルプロシージャトランスフォーメーションの概要

エクスターナルプロシージャトランスフォーメーションは、Designer インタフェースの外部に作成した手続きと共に動作し、PowerCenter の機能を拡張します。

標準のトランスフォーメーションでも広範囲のオプションをサポートしていますが、PowerCenter が提供する機能を拡張したい場合があります。たとえば、式トランスフォーメーションやフィルタトランスフォーメーションなどの標準トランスフォーメーションの機能の範囲では、必要とする動作を実現できない場合があります。また経験豊富なプログラマであれば、マッピングで必要な式トランスフォーメーションを作成する代わりに、動的リンクライブラリ（DLL）や UNIX 共有ライブラリを使って複雑な関数を作成したい場合もあるでしょう。

こうした拡張性を実現するにあたっては、PowerCenter に組み込まれている Transformation Exchange (TX) という動的呼び出しインタフェースを使用できます。TX を用いて、Informatica エクスターナルプロシージャトランスフォーメーションを作成し、既に作成済みのエクスターナルプロシージャと関連付けることができます。エクスターナルプロシージャトランスフォーメーションは 2 種類のエクスターナルプロシージャと関連付けることができます。

- COM エクスターナルプロシージャ（Windows でだけ利用可能）
- Informatica エクスターナルプロシージャ（Windows、AIX、Linux、Solaris で利用可能）

TX を使用するには、C、C++、または Visual Basic に精通している必要があります。

エクスターナルプロシージャでは、マルチスレッドコードを使用できます。

コードページの互換性

Integration Service が ASCII モードで動作している場合、エクスターナルプロシージャは 7 ビット ASCII のデータを処理できます。Integration Service が Unicode モードで動作している場合、エクスターナルプロシージャは Integration Service のコードページと相互互換性のあるデータを処理できます。

エクスターナルプロシージャ DLL または共有ライブラリにマルチバイト文字が含まれている場合は、Integration Service を Unicode モードで動作するように設定します。Integration Service からの入力文字列を変換し、マルチバイト文字を含む出力文字列を作成するためには、エクスターナルプロシージャが Integration Service と同じコードページを使用する必要があります。

エクスターナルプロシージャ DLL または共有ライブラリに ASCII 文字のみが含まれている場合は、Integration Service を ASCII モードまたは Unicode モードで動作するように設定します。

エクスターナルプロシージャとエクスターナルプロシージャトランスフォーメーション

TX には 2 つのコンポーネントがあります。エクスターナルプロシージャとエクスターナルプロシージャトランスフォーメーションです。

エクスターナルプロシージャは Integration Service とは別個に存在します。これは、ユーザーがトランスフォーメーションを定義するために記述した C、C++、または Visual Basic のコードで構成されています。このコードはコンパイルされて DLL または共有ライブラリにリンクされ、Integration Service によって実行時にロードされます。エクスターナルプロシージャはエクスターナルプロシージャトランスフォーメーションに「関連付け」られています。

エクスターナルプロシージャトランスフォーメーションは Designer で作成します。このオブジェクトは Informatica リポジトリに常駐し、いくつかの目的を実現します。

1. エクスターナルプロシージャトランスフォーメーションには下記のエクスターナルプロシージャを記述したメタデータが含まれています。Integration Service は、このメタデータを通じてエクスターナルプロシージャの「宣言」（パラメータの数とデータタイプ、戻り値の型）を確認します。
2. エクスターナルプロシージャトランスフォーメーションにより、エクスターナルプロシージャをマッピング内で参照することができます。エクスターナルプロシージャトランスフォーメーションのインスタンスをマッピングに追加することにより、このトランスフォーメーションに関連付けられたエクスターナルプロシージャを呼び出します。

注: コネクトされているか、またはコネクトされていないエクスターナルプロシージャを作成できます。

3. Informatica エクスターナルプロシージャを作成した場合に、エクスターナルプロシージャトランスフォーメーションは Informatica エクスターナルプロシージャスタブの作成に必要な情報を提供します。

エクスターナルプロシージャトランスフォーメーションのプロパティ

Transformation Developer で再利用可能なエクスターナルプロシージャトランスフォーメーションを作成し、トランスフォーメーションのインスタンスをマッピングに追加します。Mapping Designer または Mapplet Designer 内でエクスターナルプロシージャトランスフォーメーションを作成することはできません。

エクスターナルプロシージャトランスフォーメーションは、入力行ごとに出力行を 1 行返す場合と、1 行も返さない場合があります。

エクスターナルプロシージャトランスフォーメーションの [プロパティ] タブの [Module/Programmatic Identifier] および [Procedure Name] フィールドには、ASCII 文字のみを入力します。これらのフィールド内にマルチバイト文字を入力することはできません。エクスターナルプロシージャトランスフォーメーションの [ポート] タブでは、ポート名は ASCII 文字でのみ入力します。エクスターナルプロシージャトランスフォーメーションのポート名をマルチバイト文字で入力することはできません。

COM プロシージャと Informatica エクスターナルプロシージャ

以下の表に、COM と Informatica のエクスターナルプロシージャの違いを示します。

	COM	Informatica
テクノロジー	COM テクノLOGYを使用	Informatica 独自のテクノロジーを使用
オペレーティングシステム	Windows のみで動作	Integration Service をサポートしているすべてのプラットフォーム（Windows、AIX、HP、Linux、Solaris）で動作
言語	C、C++、VC++、VB、Perl、VJ++	C++のみ

BankSoft の例

以降の節では、BankSoft という架空の企業の例を見ながら、COM 手続きと Informatica 手続きの作成方法について説明します。この BankSoft の例では、エクスターナルプロシージャの作成方法と呼び出し方法について説明するために、FV という財務関数を使用しています。FV は、定期的な支払い額と一定の利率に基づいて投資額の推移を計算する手続きです。

エクスターナルプロシージャトランスフォーメーションのプロパティの設定

「プロパティ」タブで、トランスフォーメーションのプロパティを設定します。

以下の表で、エクスターナルプロシージャトランスフォーメーションのプロパティについて説明します。

プロパティ	説明
タイプ	External Procedure のタイプ。次のタイプを使用します。 <ul style="list-style-type: none">- COM- Informatica デフォルトは「Informatica」です。
モジュール/プログラム識別子	モジュール名は、External Procedure を含む DLL（Windows の場合）または共有オブジェクト（UNIX の場合）の基本となる名前です。プラットフォーム上の DLL または共有オブジェクトの名前は、モジュール名によって決定されます。 ASCII 文字のみを入力してください。 プログラマティック識別子（ProgID）は、クラスの論理名です。Designer では、ProgID によって COM クラスを参照します。内部では、クラスはたとえば次のような数値 CLSID で識別されます。 {33B17632-1D9F-11D1-8790-0000C044ACF9} ProgID の標準的な形式は「プロジェクト.クラス [バージョン]」です。 ASCII 文字のみを入力してください。
プロシージャ名	外部手続きの名前。ASCII 文字のみを入力してください。

プロパティ	説明
実行時位置	<p>DLL または共有ライブラリの格納場所。エクスターナルプロシージャセッションを実行する統合サービスノードへの相対パスを入力します。[Runtime Location] を「\$PMExtProcDir」に設定すると、統合サービスはプロセス変数\$PMExtProcDirで指定したディレクトリを調べ、ライブラリの場所を確認します。</p> <p>このプロパティが空白の場合、統合サービスは、統合サービスノードで定義されている環境変数を使用して DLL または共有ライブラリの位置を探します。</p> <p>[Runtime Location] としてパスを直接記述することもできます。ただし、パスは個々のマシンに固有なものであるため、この方法はお勧めできません。</p> <p>統合サービスノードで定義されている実行時位置または環境変数に、すべての DLL または共有ライブラリをコピーする必要があります。DLL、共有ライブラリ、または参照されるファイルが見つからない場合、統合サービスは手続きのロードに失敗します。</p> <p>デフォルトは\$PMExtProcDir です。</p>
トレースレベル	<p>セッションログにレポートされるトランザクション詳細の量。以下のトレースレベルを使用します。</p> <ul style="list-style-type: none"> - Terse - ノーマル - Verbose Initialization - Verbose Data <p>デフォルトは [Normal] です。</p>
パーティション化可能	<p>このトランスフォーメーションを使用するパイプライン内に複数のパーティションを作成できるかどうかを指定します。使用する値は次のとおりです。</p> <ul style="list-style-type: none"> - No : トランスフォーメーションはパーティション化できません。同一パイプライン内のこのトランスフォーメーションおよびその他のトランスフォーメーションは、1つのパーティションに含まれる必要があります。 - Locally : トランスフォーメーションをパーティション化することはできますが、同じノード上のパイプラインですべてのパーティションが実行される必要があります。BAPI/RFC トランスフォーメーションの様々なパーティションがメモリ内のオブジェクトを共有する必要がある場合、[Locally] を選択します。 - Across Grid : トランスフォーメーションをパーティション化することができ、各パーティションは異なるノードに配分されます。 <p>デフォルトは [No]。</p>
出力が再現可能	<p>トランスフォーメーションが行を生成する順序がセッション間で同じかどうかを示します。統合サービスが最後のチェックポイントからセッションを再開できるのは、出力が再現可能で決定性のある場合です。使用する値は次のとおりです。</p> <ul style="list-style-type: none"> - Always。入力データの順序がセッションの実行ごとに異なる場合でも、出力データの順序は常に同じです。 - Based On Input Order。すべての入力グループの入力データの順序がセッションの実行の度に常に同じである場合は、再現可能なデータを作成します。入力グループの入力データが順序付けられていない場合、出力は順序付けられません。 - Never。出力データの順序はセッションの実行ごとに異なります。トランスフォーメーションが再現可能なデータを作成しない場合、最後のチェックポイントから再開するようにリカバリを設定することはできません。 <p>デフォルトは [Based On Input Order] です。</p>
出力が確定的かどうか	<p>トランスフォーメーションが、セッションの実行ごとに一貫した出力データを生成するかどうかを指定します。このトランスフォーメーションを使用するセッションでリカバリを実行するには、このプロパティを有効にする必要があります。デフォルトでは無効になっています。</p>

警告: トランスフォーメーションを繰り返し可能で一意に定まるものとして設定する場合は、データが繰り返し可能で一意に定まることを保証する必要があります。セッションとリカバリで同じデータが生成されないトランスフォーメーションを使用してセッションをリカバリしようとすると、リカバリプロセスを実行した結果、データが破損する可能性があります。

以下の表に、各プラットフォームで実行時位置として DLL または共有オブジェクトの検索に使用される環境変数の説明を示します。

表 1. 環境変数

オペレーティングシステム	環境変数
Windows	PATH
AIX	LIBPATH
HPUX	SHLIB_PATH
Linux	LD_LIBRARY_PATH
Solaris	LD_LIBRARY_PATH

COM プロシージャの作成

Microsoft Visual C++または Visual Basic を用いて COM エクスターナルプロシージャの作成を行うことができます。以下では、Visual C++を使用した COM エクスターナルプロシージャの作成方法と、Visual Basic を使用した COM エクスターナルプロシージャの作成方法について説明します。

COM プロシージャの作成手順

COM エクスターナルプロシージャを作成するには、以下の手順に従います。

1. Microsoft Visual C++または Visual Basic を使用してプロジェクトを作成します。
2. *IDispatch* インタフェースでクラスを定義します。
3. インタフェースにメソッドを追加します。このメソッドは、Integration Service の内部から呼び出されるエクスターナルプロシージャです。
4. クラスをコンパイルして、DLL にリンクします。
5. クラスをローカル Windows レジストリに登録します。
6. COM 手続きを Transformation Developer にインポートします。
7. COM 手続きを使ってマッピングを作成します。
8. マッピングを使ってセッションを作成します。

COM エクスターナルプロシージャのサーバータイプ

Integration Service は、サーバータイプがダイナミックリンクライブラリであるインプロセス COM サーバーのみをサポートしています。その目的は性能の向上にあります。大量のデータを処理する場合には、データを同一のプロセスで処理した方が、同一マシン上またはリモートマシン上の別のプロセスに転送して処理するよりも効率的です。

Visual C++による COM プロシージャの作成

C++プログラム開発者なら、Visual C++のバージョン 5.0 以降を使用して、COM 手続きを作成することができます。まず初めに、プロジェクトを作成します。

関連項目：

- [「エクスターナルプロシージャの配布」](#) (ページ 184)
- [「既存の C/C++ライブラリまたは Visual Basic 関数に対するラッパークラス」](#) (ページ 187)

手順 1. ATL COM AppWizard プロジェクトの作成

1. Visual C++を起動し、[ファイル] - [新規] をクリックします。
2. ダイアログが表示されるので、[プロジェクト] タブを選択します。
3. プロジェクト名と場所を入力します。
BankSoft の例では、プロジェクト名として「*COM_VC_Banksoft*」、ディレクトリとして「*c:\COM_VC_Banksoft*」と入力してください。
4. プロジェクトリストボックスで *[ATL COM AppWizard]* オプションを選択し、[OK] をクリックします。
Visual C++で COM プロジェクトを作成するのに使用するウィザードが表示されます。
5. [サーバータイプ] を [ダイナミックリンクライブラリ] に設定し、*[MFC のサポート]* オプションにチェックマークを付けて、[終了] をクリックします。
ウィザードの最終ページが表示されます。
6. [OK] をクリックして、Visual C++に戻ります。
7. 新しいプロジェクトにクラスを追加します。
8. ウィザードの次ページで [OK] をクリックします。Developer Studio によって基本プロジェクトファイルが作成されます。

手順 2. プロジェクトに ATL オブジェクトを追加

1. ワークスペースウィンドウで [Class View] タブを選択し、ツリー項目の *[COM_VC_BankSoft.BSoftFin]* クラスを右クリックしてから、表示されるローカルメニューで [ATL オブジェクトの新規作成] を選択します。
2. 左側のリストボックスの [オブジェクト] を反転表示させ、オブジェクト種別のリストから [シンプルオブジェクト] を選択します。
3. [次へ] をクリックします。
4. [ショートネーム] フィールドで、作成するクラスの簡略名を指定します。
BankSoft の例では、「*BSoftFin*」という名前を使用します。これは、架空の企業である BankSoft のための財務関数を作成するからです。[ショートネーム] フィールドに名前を入力すると、ウィザードは他のフィールドに名前の候補を表示します。
5. クラスのプログラマティック識別子を入力します。
BankSoft の例では、[ProgID] (プログラマティック識別子) フィールドを「*COM_VC_BankSoft.BSoftFin*」に変更します。
プログラマティック識別子 (ProgID) は、人間がクラスを識別するための名前です。内部では、クラスはたとえば次のような数値 CLSID で識別されます。
{33B17632-1D9F-11D1-8790-0000C044ACF9}

ProgID の標準的な形式は「プロジェクト.クラス [バージョン]」です。Designer では、ProgID によって COM クラスを参照します。

6. [アトリビュート] タブを選択し、スレッドモデルを [フリー] に、インタフェースを [デュアル] に、アグリゲーションを [いいえ] に設定します。
7. [OK] をクリックします。

これで基本のクラス定義ができたので、メソッドを追加することができます。

手順 3. クラスに必要なメソッドを追加

1. ワークスペースウィンドウの [Classes View] タブに戻ります。
2. ツリー表示を展開します。
BankSoft の例では、*COM_VC_BankSoft* を展開します。
3. 新しく追加したクラスを右クリックします。
BankSoft の例では、*IBSoftFin* ツリー項目を右クリックします。
4. メニューから [メソッドの追加] を選択します。メソッドの名前を入力します。
BankSoft の例では、「FV」と入力します。
5. [パラメータ] フィールドに、メソッドの宣言を入力します。

FV では、以下のように入力します。

```
[in] double Rate,  
[in] long nPeriods,  
[in] double Payment,  
[in] double PresentValue,  
[in] long PaymentType,  
[out, retval] double* FV
```

この宣言は、Microsoft インタフェース記述言語 (MIDL) で記述されています。MIDL の詳細については、MIDL 言語のリファレンスを参照してください。以下の点に注意してください。

- **[in]** : パラメータが入力パラメータであることを示します。
- **[out]** : パラメータが出力パラメータであることを示します。
- **[out, retval]** : パラメータがメソッドの戻り値であることを示します。

また、[out] パラメータはすべて、参照によって渡されます。BankSoft の例では、パラメータ FV は Double です。

6. [OK] をクリックします。
Developer Studio によって、追加したメソッドのスタブがプロジェクトに追加されます。

手順 4. メソッドスタブにインプリメンテーションを記述

1. BankSoft の例では、ワークスペースウィンドウの [Class View] タブに戻り、COM_VC_BankSoft クラス項目を展開します。
2. CBSoftFin 項目を展開します。
3. 上記の項目の下にある IBSoftFin 項目を展開します。
4. FV 項目を右クリックし、[定義の表示] を選択します。
5. 編集ウィンドウでカーソルを TODO コメントの次の行に置き、以下のコードを追加します。

```
double v = pow((1 + Rate), nPeriods);  
*FV = -(  
    (PresentValue * v) +  
    (Payment * (1 + (Rate * PaymentType))) * ((v - 1) / Rate)  
);
```

pow 関数を参照しているので、ファイルの先頭で他のすべてのインクルード文のあとに、以下の前処理文を追加する必要があります。

```
#include <math.h>
```

最後に DLL を構築します。DLL を構築すると、COM 手続きが自動的に Windows レジストリに登録されます。

手順 5. プロジェクトの構築

1. [ビルド] メニューをプルダウンします。
2. [リビルド] を選択します。

Developer Studio がプロジェクトを構築する間、以下のメッセージが出力されます。

```
-----Configuration: COM_VC_BankSoft - Win32 Debug-----
```

```
Performing MIDL step
```

```
Microsoft (R) MIDL Compiler Version 3.01.75
```

```
Copyright (c) Microsoft Corp 1991-1997. All rights reserved.
```

```
Processing .\COM_VC_BankSoft.idl
```

```
COM_VC_BankSoft.idl
```

```
Processing C:\msdev\VC\INCLUDE\oidl.idl
```

```
oidl.idl
```

```
Processing C:\msdev\VC\INCLUDE\objidl.idl
```

```
objidl.idl
```

```
Processing C:\msdev\VC\INCLUDE\unknwn.idl
```

```
unknwn.idl
```

```
Processing C:\msdev\VC\INCLUDE\wtypes.idl
```

```
wtypes.idl
```

```
Processing C:\msdev\VC\INCLUDE\ocidl.idl
```

```
ocidl.idl
```

```
Processing C:\msdev\VC\INCLUDE\oleidl.idl
```

```
oleidl.idl
```

```
Compiling resources...
```

```
Compiling...
```

```
StdAfx.cpp
```

```
Compiling...
```

```
COM_VC_BankSoft.cpp
```

```
BSoftFin.cpp
```

```
Generating Code...
```

```
Linking...
```

```
Creating library Debug/COM_VC_BankSoft.lib and object Debug/COM_VC_BankSoft.exp
```

Registering ActiveX Control...

RegSvr32: DllRegisterServer in .\Debug\COM_VC_BankSoft.dll succeeded.
COM_VC_BankSoft.dll - 0 error(s), 0 warning(s)

Visual C++はプロジェクトにファイルをコンパイルし、ファイルを COM_VC_BankSoft.DLL という名前の動的リンクライブラリ (DLL) にリンクし、COM (ActiveX) クラス COM_VC_BankSoft.BSoftFin をローカルレジストリに登録します。

登録されたコンポーネントは、当該ホスト上で動作する Integration Service からアクセスできます。

手順 6. COM 手続きをリポジトリに登録

1. Transformation Developer を開きます。
2. [トランスフォーメーション] - [エクスターナルプロシージャのインポート] を選択します。
[外部 COM 手続きのメソッドのインポート] ダイアログボックスが表示されます。
3. [参照] をクリックします。
4. 作成した COM DLL を選択し、[OK] をクリックします。
Banksoft の例では、「COM_VC_Banksoft.DLL」を選択します。
5. [メソッドの選択] ツリー表示で、クラスノード（ここでは BSoftFin）を展開します。
6. メソッドを展開します。
7. 目的のメソッド（ここでは FV）を選択し、[OK] をクリックします。
Designer はエクスターナルプロシージャトランスフォーメーションを作成します。
8. エクスターナルプロシージャトランスフォーメーションを開き、[プロパティ] タブを選択します。
[モジュール/プログラム識別子] フィールドおよび [プロシージャ名] フィールドに ASCII 文字を入力します。
9. このトランスフォーメーションには以下のポートがあります。
10. ポート名を入力します。
11. [OK] をクリックします。

手順 7. マッピングのターゲットとソースの作成

ソーステーブルを作成して、このテーブルにサンプルデータを書き込むには、以下の SQL 文を使用します。

```
create table FVInputs(  
    Rate float,  
    nPeriods int,  
    Payment float,  
    PresentValue float,  
    PaymentType int  
)  
  
insert into FVInputs values (.005,10,-200.00,-500.00,1)  
insert into FVInputs values (.01,12,-1000.00,0.00,0)  
insert into FVInputs values (.11/12,35,-2000.00,0.00,1)  
insert into FVInputs values (.005,12,-100.00,-1000.00,1)
```

ターゲットテーブルを作成するには、以下の SQL 文を使用します。

```
create table FVOutputs(  
    FVin_ext_proc float,  
)
```

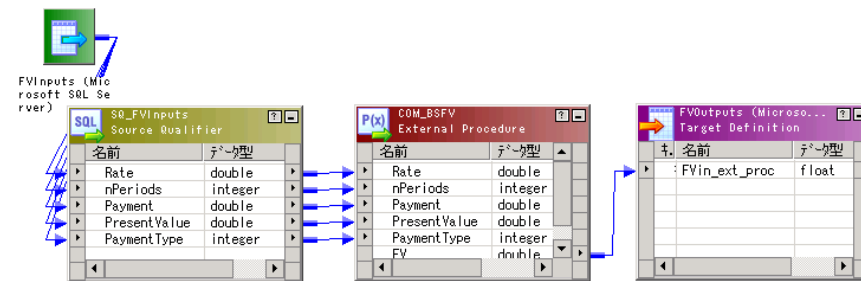
Source Analyzer と Target Designer を使用して、COM_BSFV トランスフォーメーションを作成したフォルダと同じフォルダに FVInputs と FVOutputs をインポートします。

手順 8. エクスターナルプロシージャトランスフォーメーションのテスト用マッピングの作成

ここで、エクスターナルプロシージャトランスフォーメーションをテストするためのマッピングを作成します。

1. Mapping Designer で「Test_BSFP」という名前のマッピングを作成します。
2. FVInputs ソーステーブルをマッピング内にドラッグします。
3. FVOutputs ターゲットテーブルをマッピング内にドラッグします。
4. COM_BSFP トランスフォーメーションをマッピング内にドラッグします。
5. エクスターナルプロシージャトランスフォーメーションポートに適切なソース修飾子トランスフォーメーションポートを接続します。
6. エクスターナルプロシージャトランスフォーメーションの FV ポートを FVIn_ext_proc ターゲットコラムに接続します。
7. マッピングの検証を行ってから保存します。

以下の図は完全なマッピングを示しています。



手順 9. Integration Service の開始

Integration Service を開始します。このサービスは、COM コンポーネントを登録したホストと同じホストで開始する必要があります。

手順 10. ワークフローの実行によるマッピングのテスト

Integration Service がワークフロー内でセッションを実行する場合、以下の関数を実行します。

- COM 実行時機能を使用して DLL をロードし、当該クラスのインスタンスを作成します。
- COM IDispatch インタフェースを使用して、マッピングを通過する各行について 1 回ずつ、定義されたエクスターナルプロシージャを呼び出します。

注: 複数のメソッドを持つ複数のクラスを、単一のプロジェクト内で定義できます。これらのメソッドはそれぞれ、エクスターナルプロシージャとして呼び出すことができます。

ワークフローを実行してマッピングをテストするには：

1. Workflow Manager で、Test_BSFP マッピングから s_Test_BSFP セッションを作成します。
2. s_Test_BSFP セッションを含むワークフローを作成します。
3. ワークフローを実行します。Integration Service は、レジストリから COM_VC_BankSoft.BSoftFin クラスのエントリを検索します。このエントリの情報を用いて、Integration Service は当該クラスを含む DLL の場所を決定できます。Integration Service は DLL をロードし、当該クラスのインスタンスを作成して、ソーステーブルの各行について FV 関数を呼び出します。

ワークフローが終了すると、FVOutputs テーブルには以下の結果が含まれているはずです。

FVIn_ext_proc

2581.403374

12682.503013

82846.246372

2301.401830

Visual Basic による COM プロシージャの作成

Microsoft Visual Basic は、COM 手続きを作成するための別の開発環境を提供します。Basic 言語の文法と規則は異なっていますが、大まかな作成手順は Visual C++ の COM 手続きの場合と同じです。

関連項目：

- [「エクスターナルプロシージャの配布」 \(ページ 184\)](#)
- [「既存の C/C++ ライブラリまたは Visual Basic 関数に対するラッパークラス」 \(ページ 187\)](#)

手順 1. 単一クラスの Visual Basic プロジェクトの作成

1. Visual Basic を起動し、[ファイル] - [新しいプロジェクト] を選択します。
2. 表示されたダイアログボックスで、プロジェクト種別として ActiveX DLL を選択し、[OK] をクリックします。

Visual Basic は「*Project1*」という名前のプロジェクトを新たに作成します。

プロジェクトウィンドウが表示されない場合は、Ctrl+R キーを押すか、[表示] - [プロジェクトエクスプローラ] を選択してください。

プロパティウィンドウが表示されない場合は、F4 キーを押すか、[表示] - [プロパティ] を選択してください。

3. 新プロジェクトのプロジェクトエクスプローラウィンドウで、プロジェクトを右クリックし、表示されるメニューから [Project1 のプロパティ] を選択してください。
4. 新しいプロジェクトの名前を入力します。
プロジェクトウィンドウで、「*Project1*」を選択し、プロパティウィンドウで名前を「*COM_VB_BankSoft*」に変更します。

手順 2. プロジェクト名とクラス名の変更

1. プロジェクトエクスプローラで、ツリーコントロールのルート項目である [プロジェクト - Project1] を選択します。プロパティウィンドウにプロジェクトのプロパティが表示されます。
2. プロパティウィンドウで [全体] タブを選択し、[オブジェクト名] プロパティを *COM_VB_BankSoft* に変更します。これにより、プロジェクトエクスプローラのルート項目の名前が「*COM_VB_BankSoft (COM_VB_BankSoft)*」に変更されます。
3. プロジェクトエクスプローラで *COM_VB_BankSoft (COM_VB_BankSoft)* 項目を展開します。
4. クラスモジュール項目を展開します。
5. *Class1 (Class1)* 項目を選択します。クラスのプロパティがプロパティウィンドウに表示されます。
6. プロパティウィンドウで [全体] タブを選択し、[オブジェクト名] プロパティを *BSoftFin* に変更します。

プロジェクトの名前とクラスを変更することで、ユーザーが作成するクラスのプログラマティック識別子が「COM_VB_BankSoft.BSoftFin」であることを指定します。この ProgID を使って、Designer 内部でこのクラスを参照します。

手順 3. クラスにメソッドを追加

ポインタをコードウィンドウ内に置いて、以下のテキストを入力します。

```
Public Function FV( _  
    Rate As Double, _  
    nPeriods As Long, _  
    Payment As Double, _  
    PresentValue As Double, _  
    PaymentType As Long _  
) As Double  
    Dim v As Double  
    v = (1 + Rate) ^ nPeriods  
    FV = -( _  
        (PresentValue * v) + _  
        (Payment * (1 + (Rate * PaymentType))) * ((v - 1) / Rate) _  
    )  
End Function
```

この Visual Basic FV 関数は、[「Visual Basic による COM プロシージャの作成」 \(ページ 174\)](#)の C++ FV 関数と同じ働きをします。

手順 4. プロジェクトの構築

プロジェクトを構築するには：

1. [ファイル] メニューから、[COM_VB_BankSoft.DLL の作成] を選択します。ダイアログボックスが表示され、ファイルの場所を入力するように求められます。
2. ファイルの場所を入力して、[OK] をクリックします。

Visual Basic はソースコードをコンパイルし、指定した場所に COM_VB_BankSoft.DLL を作成します。また、クラス COM_VB_BankSoft.BSoftFin をローカルレジストリに登録します。

登録されたコンポーネントは、当該ホスト上で動作する Integration Service からアクセスできます。

Informatica エクスターナルプロシージャの作成

32 ビットまたは 64 ビットの Integration Service マシンで実行されるエクスターナルプロシージャを作成できます。Informatica スタイルのエクスターナルプロシージャの作成にあたっては、以下の手順を実行します。

1. Transformation Developer で、エクスターナルプロシージャトランスフォーメーションを作成する。
エクスターナルプロシージャトランスフォーメーションは手続きの宣言を定義します。ポート名、データタイプ、およびポートのタイプ（入力または出力）は、エクスターナルプロシージャの宣言に一致しなければなりません。

2. エクスターナルプロシージャのテンプレートコードを作成する。

このコマンドを実行すると、Designer はエクスターナルプロシージャトランスフォーメーションからの情報を使用して、数種類の C++ソースコードファイルおよびメイクファイルを作成します。このうち 1 つのソースコードファイルには、トランスフォーメーションで宣言を定義した関数の「スタブ」が含まれています。

3. コードを変更して、手続きロジックを追加する。スタブにインプリメンテーションを記述した後、C++コンパイラを使用してコンパイルしたソースコードファイルを DLL または共有ライブラリにリンクします。

Integration Service は、Informatica プロシージャと関連付けられたエクスターナルプロシージャトランスフォーメーションを見つけると、DLL または共有ライブラリをロードし、定義したエクスターナルプロシージャを呼び出します。

4. ライブラリを構築して、それを Integration Service マシンにコピーします。
5. エクスターナルプロシージャトランスフォーメーションを使ってマッピングを作成する。
6. ワークフローでセッションを実行します。

BankSoft の例は、この機能を実装する方法を示しています。

手順 1. エクスターナルプロシージャトランスフォーメーションの作成

1. Transformation Developer を開き、エクスターナルプロシージャトランスフォーメーションを作成します。

2. トランスフォーメーションを開き、名前を入力します。

BankSoft の例では、「EP_extINF_BSFV」と入力します。

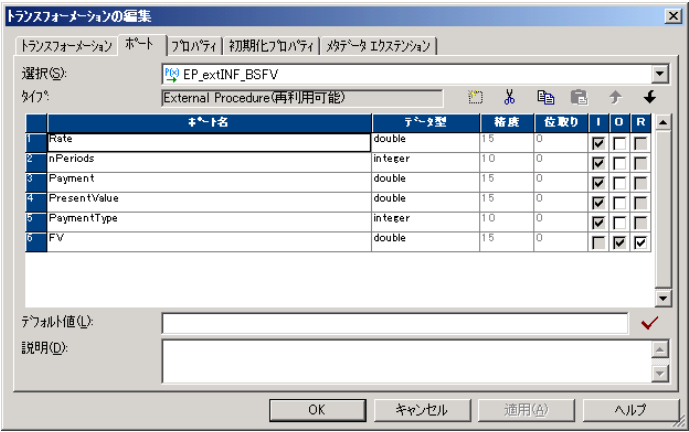
3. [ポート] タブで、定義するプロシージャに渡す各引数について、ポートを作成します。

必ず、正しいデータタイプを使用してください。

以下の表に、ポートを示します。

ポート名	データ型	精度	スケール	入出力	再利用可能
率	double	15	0	入力	いいえ
n 期間	integer	10	0	入力	いいえ
支払い	double	15	0	入力	いいえ
現在価値	double	15	0	入力	いいえ
支払タイプ	integer	10	0	入力	いいえ
FV	double	15	0	出力	はい

次の BankSoft の例は、エクスターナルプロシージャトランスフォーメーションの例を示しています。

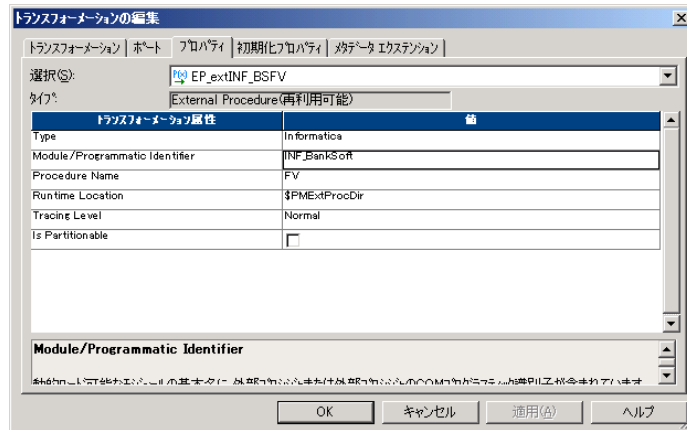


最後のポートの FV は、手続きから戻り値を取得します。

4. [プロパティ] タブを選択し、プロシージャを Informatica プロシージャとして設定します。
この BankSoft の例では、次のプロパティを設定します。

トランスフォーメーション属性	値
タイプ	Informatica
モジュール/プログラム識別子	INF_BankSoft
プロシージャ名	FV
実行時位置	\$PMExtProcDir
トレースレベル	標準
パーティション化可能	いいえ
出力が再現可能	入力順による
出力は確定的	いいえ

次の BankSoft の例は、Informatica プロシージャの例を示しています。



注: モジュール/プログラム識別子について:

以下の表に、モジュール名によって DLL または共有オブジェクトの名前がどのように決定されるかを、各プラットフォームについて示します。

オペレーティングシステム	モジュール識別子	ライブラリファイル名
Windows	INF_BankSoft	INF_BankSoft.DLL
AIX	INF_BankSoft	libINF_BankSoftshr.a
HPUX	INF_BankSoft	libINF_BankSoft.sl
Linux	INF_BankSoft	libINF_BankSoft.so
Solaris	INF_BankSoft	libINF_BankSoft.so.1

5. [OK] をクリックします。

これで、手続きを呼び出すエクスターナルプロシージャトランスフォーメーションが作成できました。次の手順では C++ ファイルを生成します。

手順 2.C++ファイルの生成

エクスターナルプロシージャトランスフォーメーションを作成したあとは、コードを生成します。Designer はファイル名を小文字で生成します。これは、UNIX マッピングのドライブで作成するファイルは常に小文字のファイル名を持つからです。生成されるファイルには以下の規則が適用されます。

- **ファイル名。** TX モジュールファイルの名前の先頭には「tx」が付きます。
- **モジュールクラス名。** 生成コードには、TX プロシージャを含むモジュールについてのクラス宣言が含まれます。TX モジュールクラスの名前の先頭には「Tx」が付きます。たとえばエクスターナルプロシージャトランスフォーメーションのモジュール名が Mymod である場合、そのクラス名は TxMymod となります。

エクスターナルプロシージャのコードを生成するには：

1. トランスフォーメーションを選択し、[トランスフォーメーション] - [外部プロシージャコードの生成] を選択します。
2. 作成した手続きの名前の横にあるチェックボックスにチェックマークを付けます。
BankSoft の例では、[INF_BankSoft.FV] を選択します。

3. ファイルの生成先ディレクトリを指定し、[生成] をクリックします。

Designer は、指定したディレクトリの下に INF_BankSoft というサブディレクトリを作成します。

Designer で作成した各エクスターナルプロシージャトランスフォーメーションでは、モジュールと手続き名を指定する必要があります。Designer は共通のモジュール名を共有するすべてのトランスフォーメーションについて、単一のディレクトリにコードを生成します。1 つのディレクトリにコードを作成すると、単一の共有ライブラリが作成されます。

Designer は以下のファイルを生成します。

- **tx<モジュール名>.h**。エクスターナルプロシージャモジュールクラスを定義します。このクラスは、ベースクラス TINFExternalModule60 から派生したクラスです。このクラスのデータメンバは生成したコードに定義されません。しかし、新しいデータメンバおよびメソッドを追加することができます。
- **tx<モジュール名>.cpp**。エクスターナルプロシージャモジュールクラスを実装します。InitDerived() メソッドを拡張して、追加した新規データメンバの初期設定をインクルードすることができます。Integration Service が派生クラスの InitDerived() メソッドを呼び出すのは、ベースクラスの Init() メソッドが正しく完了したときだけです。

このファイルは、モジュール内のすべてのエクスターナルプロシージャトランスフォーメーションの宣言を定義します。これらの宣言を変更すると、Designer で定義したエクスターナルプロシージャトランスフォーメーション間に不整合が生じます。したがって、宣言を変更してはなりません。

このファイルにはまた、C 関数の CreateExternalModuleObject が含まれ、このファイルで定義したコンストラクタを用いて、エクスターナルプロシージャモジュールのオブジェクトを作成します。Integration Service は、コンストラクタを直接呼び出すのではなく、CreateExternalModuleObject を呼び出します。

- **<プロシージャ名>.cpp**。Designer はこのモジュール内の各エクスターナルプロシージャについて、このファイルを作成します。このファイルに含まれるコードにより、データのクレンジングやフィルタリングといった手続きロジックが実現されます。データのクレンジングでは、コードを作成して入力ポートから値を読み込み、出力ポートの値を生成します。フィルタリングでは、コードを作成して、INF_NO_OUTPUT_ROW を返すことにより、出力行の生成を抑止します。
- **stdafx.h**。UNIX システムでの構築に使用されるスタブファイル。各*.cpp ファイルで、このファイルがインクルードされます。Windows システムでは、Designer が生成したファイルの代わりに、Visual Studio が生成する stdafx.h ファイルを使用する必要があります。
- **version.cpp**。この実装のバージョン番号が入っている小さなファイルです。以前のリリースでは、エクスターナルプロシージャの実装は別の方法で行われていました。このファイルを使用すると、Integration Service でエクスターナルプロシージャモジュールのバージョンの判別が可能になります。
- **makefile.aix、makefile.aix64、makefile.hp、makefile.hp64、makefile.hpparisc64、makefile.linux、makefile.sol**。UNIX プラットフォーム用のメイクファイル。32 ビットプラットフォームの場合は、makefile.aix、makefile.hp、makefile.linux、makefile.sol を使用します。64 ビット AIX プラットフォームには makefile.aix64 を使用し、64 ビット HP-UX (Itanium) プラットフォームには makefile.hp64 を使用します。

例 1

BankSoft の例では、Designer は下記のファイルを生成します。

- **txinf_banksoft.h**。モジュールクラス TxINF_BankSoft と、エクスターナルプロシージャの FV の宣言が含まれています。
- **txinf_banksoft.cpp**。モジュールクラス TxINF_BankSoft のコードが含まれています。
- **fv.cpp**。プロシージャ FV のコードが含まれています。
- **version.cpp**。TX バージョンに戻ります。
- **stdafx.h**。UNIX でのコンパイルに必要です。Windows では、Visual Studio によって stdafx.h が生成されます。

- **readme.txt**。全般的なヘルプ情報が含まれています。

例 2

2つのエクスターナルプロシージャトランスフォーメーション（手続き名が「Myproc1」と「Myproc2」で、モジュール名は共に「Mymod」）を作成した場合、Designer は以下のファイルを生成します。

- **txmymod.h**。モジュールクラス TxMymod と、エクスターナルプロシージャの Myproc1 および Myproc2 の宣言が含まれます。
- **txmymod.cpp**。モジュールクラス TxMymod のコードが含まれます。
- **myproc1.cpp**。プロシージャ Myproc1 のコードが含まれます。
- **myproc2.cpp**。プロシージャ Myproc2 のコードが含まれます。
- **version.cpp**。
- **stdafx.h**。
- **readme.txt**。

手順 3. メソッドスタブにインプリメンテーションを記述

最後に、手続きのコーディングを行います。

1. 手続きに対して生成された<手続き名>.cpp という名前のスタブファイルを開きます。
BankSoft の例では、fv.cpp を開いて、TxINF_BankSoft::FV 手続きのコーディングを行います。
2. 手続きの C++コードを入力します。

FV 手続きのインプリメンテーションには、以下のコードを入力します。

```
INF_RESULT TxINF_BankSoft::FV()
{
    // Input port values are mapped to the m_pInParamVector array in
    // the InitParams method. Use m_pInParamVector[i].IsValid() to check
    // if they are valid. Use m_pInParamVector[i].GetLong or GetDouble,
    // etc. to get their value. Generate output data into m_pOutParamVector.
    // TODO: Fill in implementation of the FV method here.
    ostringstream ss;
    char* s;
    INF_BOOLEAN bVal;
    double v;
    TINFPParam* Rate = &m_pInParamVector[0];
    TINFPParam* nPeriods = &m_pInParamVector[1];
    TINFPParam* Payment = &m_pInParamVector[2];
    TINFPParam* PresentValue = &m_pInParamVector[3];
    TINFPParam* PaymentType = &m_pInParamVector[4];
    TINFPParam* FV = &m_pOutParamVector[0];
    bVal =
        INF_BOOLEAN(
            Rate->IsValid() &&
            nPeriods->IsValid() &&
            Payment->IsValid() &&
            PresentValue->IsValid() &&
            PaymentType->IsValid()
        );
    if (bVal == INF_FALSE)
    {
        FV->SetIndicator(INF_SQL_DATA_NULL);
        return INF_SUCCESS;
    }
}
```

```

v = pow((1 + Rate->GetDouble()), (double)nPeriods->GetLong());
FV->SetDouble(
    -(
        (PresentValue->GetDouble() * v) +
        (Payment->GetDouble() *
            (1 + (Rate->GetDouble() * PaymentType->GetLong())) *
            ((v - 1) / Rate->GetDouble())
        )
    );
ss << "The calculated future value is: " << FV->GetDouble() <<ends;
s = ss.str();
(*m_pfnMessageCallback)(E_MSG_TYPE_LOG, 0, s);
(*m_pfnMessageCallback)(E_MSG_TYPE_ERR, 0, s);
delete [] s;
return INF_SUCCESS;
}

```

Designer は、引数や戻り値などの関数プロファイルを生成します。コメントに示すように、関数内に実際のコードを入力する必要があります。POW 関数を参照し、ostrstream 変数を定義しているので、前処理文も含めなければなりません。

Windows では：

```

#include <math.h>
#include <strstream> using namespace std;

```

UNIX では include 文は次のようになります。

```

#include <math.h>
#include <strstream.h>

```

3. 変更したファイルを保存します。

手順 4. モジュールの構築

Windows では、Visual C++を使用して DLL をコンパイルします。

Windows でのモジュールの構築

Windows で DLL を作成する手順

1. Visual C++を起動します。
2. [ファイル] - [新規] をクリックします。
3. [新規作成] ダイアログボックスで、[プロジェクト] タブをクリックし、[MFC AppWizard (DLL)] オプションを選択します。
4. 場所を入力します。

BankSoft の例では、c:\pmclient\tx にファイルを生成したとすれば、「c:\pmclient\tx\INF_BankSoft」と入力します。

5. プロジェクト名を入力します。

エクスターナルプロシージャトランスフォーメーションに指定したモジュール名と同じ名前にする必要があります。BankSoft の例では、INF_BankSoft になります。

6. [OK] をクリックします。

Visual C++のウィザードに従って、プロジェクトのコンポーネントをすべて定義します。

7. ウィザードで、[MFC の拡張 DLL] (共有 MFC DLL を使用) をクリックします。
8. [終了] をクリックします。

ウィザードは数種類のファイルを生成します。

9. [プロジェクト] - [プロジェクトへの追加] - [ファイル] をクリックします。

10. 1つ上のディレクトリレベルへ移動します。このディレクトリにはユーザーが作成したエクスターナルプロシージャが含まれています。.cpp ファイルをすべて選択します。
BankSoft の例では、以下のファイルを追加します。
 - fv.cpp
 - txinf_banksoft.cpp
 - version.cpp
11. [プロジェクト] - [設定] をクリックします。
12. [C/C++] タブをクリックしてから、[カテゴリ] フィールドで [プリプロセッサ] を選択します。
13. [インクルードファイルのパス] フィールドに「`..; <pmserver のインストールディレクトリ>\extproc\include`」を入力します。
14. [リンク] タブをクリックし、[カテゴリ] フィールドで [全般] を選択します。
15. [オブジェクト/ライブラリモジュール] フィールドに「`<pmserver のインストールディレクトリ>\bin\pmtx.lib`」を入力します。
16. [OK] をクリックします。
17. [ビルド] - [ビルド INF_BankSoft.dll] を選択するか、F7 キーを押してプロジェクトを構築します。
コンパイラは DLL を作成し、プロジェクトディレクトリ下のデバッグディレクトリまたはリリースディレクトリに格納します。

UNIX でのモジュールの構築

UNIX で共有ライブラリを構築するには：

1. PowerCenter クライアントツールに直接アクセスできない場合、必要なファイルをすべて、UNIX マシンの共有ライブラリにコピーする必要があります。
たとえば BankSoft 手続きでは、FTP などの機能を使用して、INF_BankSoft ディレクトリからすべてのファイルを UNIX マシンにコピーします。
2. 環境変数 INFA_HOME を PowerCenter のインストールディレクトリに設定します。
警告: INFA_HOME 環境変数に間違ったディレクトリパスを指定すると、統合サービスは起動しません。
3. コマンドを入力して、プロジェクトを作成します。
以下に示すとおり、入力するコマンドは UNIX のバージョンによって異なります。

UNIX バージョン	コマンド
AIX (32 ビット)	make -f makefile.aix
AIX (64 ビット)	make -f makefile.aix64
Linux	make -f makefile.linux
Solaris	make -f makefile.sol

手順 5. マッピングの作成

Mapping Designer で、このエクスターナルプロシージャトランスフォーメーションを使用するマッピングを作成します。

手順 6. セッションの実行

セッションを実行すると、Integration Service は実行時位置として指定したディレクトリを検索し、手順 4 で構築したライブラリ（DLL）を探します。セッションプロパティにおける [実行時位置] プロパティのデフォルト値は「\$PMEExtProcDir」です。

セッションを実行するには：

1. Workflow Manager でワークフローを作成します。
2. ワークフローでこのマッピング用のセッションを作成します。
ヒント: または、Task Developer で再利用可能なセッションを作成し、ワークフローで使用することもできます。
3. ライブラリ（DLL）を [実行時位置] のディレクトリにコピーします。
4. セッションを含むワークフローを実行します。

Windows におけるモジュールのデバッグバージョンによるセッションの実行

Informatica は Windows 用の PowerCenter を、エクスターナルプロシージャトランスフォーメーションライブラリのリリースビルド（pmtx.dll）およびデバッグビルド（pmtxdbg.dll）と共に出荷します。これらのライブラリは、サーバーの bin ディレクトリにインストールされます。

手順 4 でモジュールのリリースバージョンを構築した場合、ワークフローでセッションを実行するとエクスターナルプロシージャトランスフォーメーションライブラリのリリースビルド（pmtx.dll）が使用されます。以下のタスクを実行する必要はありません。

手順 4 でモジュールのデバッグバージョンを構築した場合、以下の手順に従って、エクスターナルプロシージャトランスフォーメーションライブラリのデバッグビルド（pmtxdbg.dll）を使用してください。

モジュールのデバッグバージョンを使用してセッションを実行するには：

1. Workflow Manager でワークフローを作成します。
2. ワークフローでこのマッピング用のセッションを作成します。
再利用可能なセッションを Task Developer で作成し、ワークフローで使用することもできます。
3. ライブラリ（DLL）を [実行時位置] のディレクトリにコピーします。
4. エクスターナルプロシージャトランスフォーメーションライブラリのデバッグビルドを使用するには：
 - 名前を変更するか、サーバーの bin ディレクトリから移動させ、pmtx.dll を保存します。
 - 名前を pmtxdbg.dll から pmtx.dll へ変更する。
5. セッションを含むワークフローを実行します。
6. エクスターナルプロシージャトランスフォーメーションライブラリのリリースビルドをデフォルトライブラリに戻すには：
 - 名前を pmtx.dll から元の pmtxdbg.dll に戻す。
 - 元の pmtx.dll ファイルをサーバーの bin ディレクトリに戻すか、または名前を変更します。

注: Windows 上でモジュールのデバッグバージョンによってこのセッションを含むワークフローを実行する場合、元の pmtx.dll ファイルを元の名前および場所に戻してから、非デバッグセッションを実行する必要があります。

エクスターナルプロシージャの配布

一連のエクスターナルプロシージャを作成し、Integration Service を実行している複数のサーバーで、これらのエクスターナルプロシージャを利用できるようにするとします。そのための方法は、エクスターナルプロシージャの種類と、このエクスターナルプロシージャを構築したオペレーティングシステムに応じて異なります。

また、同じ手順で社外の顧客にエクスターナルプロシージャを配布することもできます。

COM プロシージャの配布

プロジェクトを構築するときに、Visual Basic および Visual C++ は COM クラスをローカルレジストリに登録します。登録されたクラスは、DLL をコンパイルしたマシンで動作している Integration Service からアクセスできます。例えば、プロジェクトを HOST1 上で構築すると、そのプロジェクトのクラスはすべて HOST1 のレジストリに登録され、HOST1 で動作している Integration Service からアクセスできるようになります。ここで、これらの各クラスを、HOST2 で動作する Integration Service にアクセスできるようにするとします。そうするためには、各クラスを HOST2 のレジストリに登録する必要があります。

Visual Basic が提供するセットアッププログラム作成ユーティリティを使用すると、COM クラスを Windows マシンにインストールして、そのマシンのレジストリに登録することができます。Visual C++ では、そのようなユーティリティはありませんが、クラスの登録は簡単に行えます。

Visual Basic の COM プロシージャの配布

Visual Basic の COM プロシージャを配布する手順

1. DLL を構築したら、Visual Basic を終了し、Visual Basic Application Setup ウィザードを起動します。
2. ウィザードの最初のパネルはスキップします。
3. 2 パネル目では、プロジェクトの場所を指定し、[セットアッププログラムの作成] オプションを選択します。
4. 3 パネル目で、使用する配布方法を選択します。
5. 次のパネルでは、セットアップファイルを書き込みたいディレクトリを指定します。
単純な ActiveX コンポーネントの場合は、このままウィザードの最後のパネルまで進みます。それ以外の場合には、ファイルの種別や配布方法に応じて、情報を追加する必要がある場合があります。
6. 最後のパネルで、[終了] をクリックします。

Visual Basic は、DLL のセットアッププログラムを作成します。Integration Service が動作している Windows マシンで、このセットアッププログラムを実行します。

Visual Basic の COM プロシージャの手動配布

Visual C++/Visual Basic の COM プロシージャを手動で配布する手順

1. 配布先の新しい Windows マシン上のディレクトリに DLL をコピーします。
2. この Windows マシンにログオンし、DOS プロンプトを開きます。
3. DLL を格納したディレクトリに移動して、次のコマンドを実行します。

```
REGSVR32 project_name.DLL
```

project_name は作成した DLL の名前です。BankSoft の例では、プロジェクト名は「COM_VC_BankSoft.DLL」です。または「COM_VB_BankSoft.DLL」です。

このコマンドラインプログラムにより、DLL と、これに含まれる COM クラスが登録されます。

Informatica モジュールの配布

エクスターナルプロシージャはリポジトリ間で配布することができます。

リポジトリ間でエクスターナルプロシージャを配布するには：

1. エクスターナルプロシージャを含む DLL または共有オブジェクトを、Integration Service がアクセスできるマシン上のディレクトリに移動します。
2. Designer クライアントツールを使用して、エクスターナルプロシージャトランスフォーメーションを元のリポジトリから配布先リポジトリにコピーします。

エクスターナルプロシージャトランスフォーメーションを XML ファイルにエクスポートして、配布先リポジトリにインポートすることもできます。

作成にあたっての注意

この節では、COM および Informatica エクスターナルプロシージャの作成に関して、ここまで説明した以外のガイドラインや情報をいくつか提供します。

COM データタイプ

Visual C++または Visual Basic のいずれかを使用して COM プロシージャを作成する場合は、使用する COM データタイプが、Integration Service でのデータの読み込みと変換に使用される内部データタイプに対応している必要があります。Integration Service が、エクスターナルプロシージャトランスフォーメーションのポートと、そのトランスフォーメーションが呼び出すプロシージャの引数（または戻り値）との間でデータタイプのマッピングを行う場合、データタイプの一致が重要になります。

以下の表に、Visual C++のデータタイプとトランスフォーメーションのデータタイプとの比較を示します。

Visual C++の COM データタイプ	トランスフォーメーションデータタイプ
VT_I4	Integer
VT_UI4	Integer
VT_R8	ダブル
VT_BSTR	String
VT_DECIMAL	Decimal
VT_DATE	日付/時刻

以下の表に、Visual Basic のデータタイプとトランスフォーメーションのデータタイプとの比較を示します。

Visual Basic の COM データタイプ	トランスフォーメーションデータタイプ
Long	Integer
ダブル	ダブル

Visual Basic の COM データタイプ	トランスフォーメーションデータタイプ
String	String
Decimal	Decimal
日付	日付/時刻

データタイプが正確に一致しないと、Integration Service は変換を試みる場合があります。例えば、ポートに割り当てられたデータタイプが Integer、該当する引数のデータタイプが BSTR であった場合、Integration Service は Integer 値を BSTR に変換しようと試みます。

行レベルのプロシージャ

エクスターナルプロシージャトランスフォーメーションはすべて、トランスフォーメーションに渡された単一行からの値を使用して、手続きを呼び出します。単一の手続き呼び出しで複数の行からの値を使用することはできません。たとえば、集計関数 SUM または AVG と同等のものを手続き呼び出し内にコーディングすることはできません。この意味で、エクスターナルプロシージャはすべて「状態を持たない」ものでなければなりません。

プロシージャからの戻り値

プロシージャの呼び出しを行うと、Integration Service は、このプロシージャ内にコーディングされた戻り値とは別に、追加の戻り値を取得します。この戻り値は、Integration Service によるプロシージャの呼び出しが成功したかどうかを示します。

COM 手続きの場合、この戻り値は HRESULT という型を持ちます。

Informatica 手続きの場合、INF_RESULT という型を持ちます。戻り値が S_OK/INF_SUCCESS の場合、Integration Service のよるプロシージャの呼び出しは成功です。エクスターナルプロシージャの成功または失敗を明示する適切な値を返す必要があります。Informatica 手続きは以下の 4 つの値を返します。

- **INF_SUCCESS**。エクスターナルプロシージャの行処理に成功しました。Integration Service は行をマッピング内の次のトランスフォーメーションへ渡します。
- **INF_NO_OUTPUT_ROW**。Integration Service は、エクスターナルプロシージャロジックにより現在の行を書き込みません。これは、エラーではありません。INF_NO_OUTPUT_ROW を使って行をフィルタリングする場合、エクスターナルプロシージャトランスフォーメーションはフィルタトランスフォーメーションと同じように振舞います。
注: エクスターナルプロシージャで INF_NO_OUTPUT_ROW を使用する場合、エクスターナルプロシージャトランスフォーメーションは必ず、エクスターナルプロシージャトランスフォーメーションからのみ行を受け取るトランスフォーメーションに接続するようにします。
- **INF_ROW_ERROR**。トランスフォーメーションエラーに相当します。Integration Service は現在の行をスキップしますが、*n* 回のエラーで停止するようにセッションを設定していない限り、次の行を処理します。
- **INF_FATAL_ERROR**。ABORT()関数呼び出しに相当します。Integration Service はセッションを中断し、以降の行の処理を行いません。

プロシージャの呼び出しにおける例外

Integration Service は、エクスターナルプロシージャトランスフォーメーションで COM または Informatica プロシージャを呼び出すときに発生するほとんどの例外に対応できます。例えば、プロシージャの呼び出しによってゼロ除算エラーが発生した場合、Integration Service はこの例外を検出します。

Integration Service がプロシージャの呼び出しによって発生したエラーを検出できない場合もあります。Integration Service はインプロセス COM サーバーだけをサポートして、Informatica プロシージャはすべて共有ライブラリおよび DLL に格納されるため、エクスターナルプロシージャの実行コードは Integration Service と同じメモリ内のアドレス空間に存在します。したがって、エクスターナルプロシージャのコードが Integration Service のメモリを上書きすることで Integration Service が停止する場合があります。COM プロシージャまたは Informatica プロシージャが原因で Integration Service が停止した場合は、ソースコードを見直してメモリアクセス上の問題を解決します。

プロシージャのためのメモリ管理

Informatica 手続きで使用するデータタイプはすべて固定長であるため、Informatica エクスターナルプロシージャのメモリ管理上の問題はありません。COM 手続きの場合、メモリの割り当てが必要なのは、手続きからの [out] パラメータが BSTR データタイプを使用している場合だけです。この場合には、この手続きを呼び出すたびにメモリを割り当てる必要があります。セッションの間、Integration Service は関数の呼び出し後にメモリを解放します。

既存の C/C++ライブラリまたは Visual Basic 関数に対するラッパークラス

BankSoft に C 関数または C++関数のライブラリがあり、これらの関数を Integration Service に組み込む必要があると想定します。さらに、このライブラリには BankSoft 固有の FV 関数である PreExistingFV が含まれているとします。そのような関数を PowerCenter Server に組み込むための一般的な方法は、COM エクスターナルプロシージャの場合も PowerCenter エクスターナルプロシージャの場合も同じです。Visual Basic でも同様な方法が利用できます。必要なのは、既存の Visual Basic 関数か、Visual Basic がアクセスできるオブジェクトのメソッドを呼び出すことです。

エラーメッセージとトレースメッセージの生成

[「手順 4. モジュールの構築」 \(ページ 181\)](#) の Informatica エクスターナルプロシージャ TxINF_BankSoft::FV のインプリメンテーションには以下のコードが含まれています。

```
ostrstream ss;

char* s;

...

ss << "The calculated future value is: " << FV->GetDouble() << ends;

s = ss.str();

(*m_pfnMessageCallback)(E_MSG_TYPE_LOG, 0, s);

(*m_pfnMessageCallback)(E_MSG_TYPE_ERR, 0, s);

delete [] s;
```

Integration Service が Tx<MODNAME>型のオブジェクトを作成するときは、そのコンストラクタに対して、エラーやデバッグのメッセージをセッションログに書き込む際に使用できるコールバック関数へのポインタを渡します。（Tx<MODNAME>コンストラクタ用のコードは、Tx.cpp ファイルにあります。）このポインタは

Tx<MODNAME>メンバ変数 m_pfnMessageCallback に格納されています。このポインタの型は、\$PMExtProcDir/include/infemmsg.h ファイルの typedef に定義されています。

```
typedef void (*PFN_MESSAGE_CALLBACK)(  
    enum E_MSG_TYPE eMsgType,  
    unsigned long Code,  
    char* Message  
);
```

また、このファイルには、列挙型 E_MSG_TYPE が定義されています。

```
enum E_MSG_TYPE {  
    E_MSG_TYPE_LOG = 0,  
    E_MSG_TYPE_WARNING,  
    E_MSG_TYPE_ERR  
};
```

コールバック関数の eMsgType を「E_MSG_TYPE_LOG」と指定すると、このコールバック関数はログメッセージをセッションログに書き込みます。「E_MSG_TYPE_ERR」を指定すると、このコールバック関数はエラーメッセージをセッションログに書き込みます。「E_MSG_TYPE_WARNING」を指定すると、このコールバック関数は警告メッセージをセッションログに書き込みます。これらのメッセージを使用して、Informatica エクスターナルプロシージャを容易にデバッグできます。

COM エクスターナルプロシージャのデバッグにあたっては、Visual Basic または C++ クラス内から使用できる出力機能を利用できます。たとえば Visual Basic では、MsgBox を使用して、各行の計算結果を印刷できます。言うまでもなく、デバッグ中にこれを行うのは小さなサンプルデータについてです。実稼動に移る前に、必ず MsgBox を削除してください。

注: Visual Basic または C++ のクラス内から出力機能を使用する場合は、その前に以下の値を必ずレジストリに追加してください。

1. Windows レジストリに以下のエントリを追加します。

```
\HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\PowerMart\Parameters\MiscInfo\RunInDebugMode=Yes
```

このオプションは、Integration Service をサービスではなく通常のアプリケーションとして起動します。Integration Service のデバッグは、実行中の Integration Service サービスに対するデバッグ特権を変更することなく行えます。

2. PMSERVER.EXE コマンドを使用して、Integration Service をコマンドラインから起動します。

これで、Integration Service がデバッグモードで動作します。

デバッグが終了したら、必ずこのエントリをレジストリから削除するか、「RunInDebugMode」を「No」に設定してください。そうしないと、PowerCenter をサービスとして起動しようとしても、起動しません。

1. Integration Service を停止し、前に追加したレジストリエントリを以下の設定に変更します。

```
\HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\PowerMart\Parameters\MiscInfo\RunInDebugMode=No
```

2. Integration Service を Windows サービスとして再起動します。

TINFPParam クラスとインジケータ

<プロシージャ名>メソッドは、2つのパラメータ配列を使って入出力パラメータにアクセスします。各配列要素は TINFPParam データタイプです。TINFPParam データタイプは、任意の Informatica 内部データタイプを保持できる「不定」のデータ構造体として使用される C++ クラスです。TINFPParam* 型のパラメータ内の実際のデータは、Get<Type> および Set<Type> というメンバ関数でアクセスします。ここで <Type> は、いずれかの

Informatica 内部データタイプです。TINFParm はまた、各パラメータのインジケータを取得、設定するためのメソッドを備えています。

必ず、エクスターナルプロシージャの呼び出し時にこのインジケータを確認し、また終了時には、このインジケータを設定してください。呼び出し時には、すべての出力パラメータのインジケータが INF_SQL_DATA_NULL に明示的に設定されるので、このインジケータをリセットしないでエクスターナルプロシージャから戻る場合と、すべての出力パラメータについて NULL を取得することになります。TINFParm クラスはまた、特定のパラメータのメタデータを取得する関数をサポートしています。TINFParm クラスのメンバ関数すべての詳細については、tx/include ディレクトリの infemdef.h インクルードファイルを参照してください。

Informatica エクスターナルプロシージャが COM エクスターナルプロシージャよりも優れている点の 1 つは、Informatica エクスターナルプロシージャがインジケータの操作を直接サポートしているということです。したがって、入力パラメータが NULL かどうかを確認できるほか、出力パラメータを NULL に設定することができます。COM はインジケータをサポートしていません。したがって、COM スタイルエクスターナルプロシージャを実行する行に NULL があると、その行の処理ができません。この欠点を解消するには、Designer のデフォルト値機能を使用します。ただし、COM 関数から NULL を受け取ることはできません。

接続されていないエクスターナルプロシージャトランスフォーメーション

エクスターナルプロシージャトランスフォーメーションのインスタンスをマッピングに追加する場合、パイプラインの一部として接続するのか、未接続のままとしておくのかを選択できます。接続されたエクスターナルプロシージャトランスフォーメーションは、行がこれを通過するたびに、COM または Informatica 手続きを呼び出します。

コネクトされていないエクスターナルプロシージャトランスフォーメーションから戻り値を取得するには、以下の構文の式で呼び出します。

`:EXT.transformation_name(arguments)`

行がこの式を含むトランスフォーメーションを通過すると、Integration Service はエクスターナルプロシージャトランスフォーメーションに関連付けられたプロシージャを呼び出します。式はエクスターナルプロシージャトランスフォーメーションの戻りポートを通じてプロシージャの戻り値を取得します。このポートには、結果(R)オプションが設定されています。

COM および Informatica モジュールの初期化

エクスターナルプロシージャには、初期設定が必要なものがあります。この初期設定は、エクスターナルプロシージャの種類に応じて、2 つの方法のいずれかでを行います。

- **Informatica スタイルのエクスターナルプロシージャの初期化。** Tx<MODNAME>クラスは、エクスターナルプロシージャを含み、初期化関数 Tx<MODNAME>::InitDerived も含みます。この初期化関数の宣言は、Integration Service では一般的なもので、3 つのパラメータから構成されます。

- **nInitProps。** このパラメータは初期化関数に対して、渡される初期化プロパティの数を知らせます。
- **プロパティ。** このパラメータは、初期化プロパティの名前を表す nInitProp 個の文字列の配列です。
- **Values。** このパラメータは、初期化プロパティの値を表す nInitProp 個の文字列の配列です。

Integration Service は、まずベースクラスの Init()関数を呼び出します。Init()関数が正常に完了すると、ベースクラスは Tx<MODNAME>::InitDerived()関数を呼び出します。

Integration Service は、Tx<MODNAME>オブジェクトを作成してから、初期化関数を呼び出します。Tx<MODNAME>::InitDerived()関数の中で、初期化プロパティを解釈し、それらを使用してエクスターナルプロシージャを初期化する部分は、エクスターナルプロシージャの作成者が提供する必要があります。オブジェクトが作成されて初期化されると、Integration Service は各行についてオブジェクトに対するエクスターナルプロシージャを呼び出すことができます。

- **COM スタイルのエクスターナルプロシージャの初期化。**エクスターナルプロシージャを含むオブジェクト（EP オブジェクト）には、初期化関数は含まれていません。その代わりに、別のオブジェクト（CF オブジェクト）が、EP オブジェクトのクラスファクトリとして使用されます。CF オブジェクトは、EP オブジェクトを作成できるメソッドを備えています。

CF オブジェクトメソッドの宣言は、そのタイプライブラリから決定されます。Integration Service は CF オブジェクトを作成してから EP オブジェクトを作成するためのメソッドをそのオブジェクトで呼び出し、必要なパラメータをこのメソッドに渡します。これには、メソッドの宣言が以下の 2 つの要素から構成される必要があります。1 つは、その型がタイプライブラリから決定できる一連の入力パラメータ、もう 1 つは、IUnknown**もしくは IDispatch**（またはそのいずれかを指す VARIANT*）である単一の出力パラメータです。

入力パラメータは EP オブジェクトの初期設定に必要な値を保持し、出力パラメータは初期設定されたオブジェクトを受け取ります。出力パラメータは [out] または [out, retval] の属性を持つことができます。したがって、初期設定したオブジェクトは、出力パラメータまたはメソッドの戻り値として返すことができます。入力パラメータに対してサポートされているデータタイプは以下のとおりです。

- COM VC 型
- VT_UI1
- VT_BOOL
- VT_I2
- VT_UI2
- VT_I4
- VT_UI4
- VT_R4
- VT_R8
- VT_BSTR
- VT_CY
- VT_DATE

Designer での初期化プロパティの設定

[トランスフォーメーションの編集] ダイアログボックスの [初期化プロパティ] タブにエクスターナルプロシージャ初期設定プロパティを入力します。このタブに表示されるフィールドは、エクスターナルプロシージャが COM スタイルか Informatica スタイルかに応じて異なります。

COM スタイルのエクスターナルプロシージャトランスフォーメーションの場合、[初期化プロパティ] タブには下記のフィールドが含まれています。

- **[クラスファクトリのプログラム識別子]。**クラスファクトリのプログラム識別子を入力します。
- **[コンストラクタ]。**EP オブジェクトを作成するクラスファクトリのメソッドを指定します。

初期化プロパティは好きな数だけ入力して、COM スタイルおよび Informatica スタイル両方のエクスターナルプロシージャトランスフォーメーションのコンストラクタメソッドに渡すことができます。

新しい初期設定プロパティを入力するには、[追加] をクリックします。[プロパティ] カラムにパラメータ名を入力し、[値] カラムにパラメータの値を入力します。たとえば、次のようなパラメータを入力できます。

パラメータ	値
Param1	abc
Param2	100
Param3	3.17

注: Designer で定義した初期化プロパティとクラスファクトリのコンストラクタメソッドの入力パラメータとの間には、1 対 1 の対応がなければなりません。例えば、コンストラクタに n 個のパラメータがあり、最後のパラメータが初期設定済みオブジェクトを受け取る出力パラメータである場合、コンストラクタメソッドの各入力パラメータに対し、 $n-1$ 個の初期設定プロパティを Designer で定義する必要があります。

また、プロセス変数を初期化プロパティで使用することもできます。

TX に配布されて使用される他のファイル

以下に示すヘッダファイルはパス `$PMExtProcDir/include` の下に配置され、エクスターナルプロシージャのコンパイル時に必要とされます。

- `infconfig.h`
- `infem60.h`
- `infemdef.h`
- `infemmsg.h`
- `infparam.h`
- `infsigtr.h`

以下に示すライブラリファイルはパス `<PMInstallDir>` の下に配置され、エクスターナルプロシージャのリンク時とセッションの実行時に使用されます。

- `libpmtx.a` (AIX)
- `libpmtx.so` (Linux)
- `libpmtx.so` (Solaris)
- `pmtx.dll` および `pmtx.lib` (Windows)

初期化プロパティでのサービスプロセス変数

PowerCenter は、エクスターナルプロシージャトランスフォーメーションの初期化プロパティのリストにおいて、組み込みプロセス変数をサポートしています。プロパティ値に組み込みプロセス変数が含まれている場合、Integration Service は組み込みプロセス変数を展開してからエクスターナルプロシージャライブラリに渡します。これは、移植可能なエクスターナルプロシージャトランスフォーメーションを作成する上で非常に便利です。

以下の表は、トランスフォーメーションのプロパティの「初期化プロパティ」タブにあるエクスターナルプロシージャトランスフォーメーションの初期化プロパティおよび値を示しています。

表 2. エクスターナルプロシージャの初期化プロパティ

プロパティ	値	エクスターナルプロシージャライブラリに渡される展開された値
mytempdir	\$PMTempDir	/tmp
memorysize	5000000	5000000
input_file	\$PMSourceFileDir/file.in	/data/input/file.in
output_file	\$PMTargetFileDir/file.out	/data/output/file.out
extra_var	\$some_other_variable	\$some_other_variable

ワークフローを実行すると、Integration Service はプロパティリストを展開し、エクスターナルプロシージャ初期化関数へ渡します。組み込みプロセス変数\$PMTempDir の値を */tmp*、\$PMSourceFileDir の値を */data/input*、\$PMTargetFileDir の値を */data/output* と想定した場合、[「初期化プロパティでのサービスプロセス変数」](#) (ページ 191) の最後のカラムにはプロパティおよび展開された値の情報が含まれます。最後のプロパティである「\$some_other_variable」は組み込みプロセス変数ではないため、Integration Service では展開されません。

エクスターナルプロシージャのインタフェース

Integration Service は、以下の主な関数をエクスターナルプロシージャで使用します。

- ディスパッチ
- エクスターナルプロシージャ
- プロパティアクセス
- パラメータアクセス
- コードページアクセス
- トランスフォーメーション名アクセス
- Procedure アクセス
- パーティション関連
- トレースレベル

ディスパッチ関数

Integration Service はディスパッチ関数を呼び出し、各入力行をエクスターナルプロシージャモジュールに渡します。ディスパッチ関数は、指定したエクスターナルプロシージャ関数を呼び出します。

エクスターナルプロシージャがトランスフォーメーション内のポートにアクセスする際、入力ポートの場合には m_plnParamVector メンバ変数を、出力ポートの場合には m_pOutParamVector メンバ変数を直接使用します。

宣言

ディスパッチ関数は、インデックスパラメータを 1 つ含む固定宣言を持っています。

```
virtual INF_RESULT Dispatch(unsigned long ProcedureIndex) = 0
```

エクスターナルプロシージャ関数

エクスターナルプロシージャ関数はエクスターナルプロシージャモジュールへの主要な入り口で、エクスターナルプロシージャトランスフォーメーションの 1 つの属性です。ディスパッチ関数は各入力行ごとにエクスターナルプロシージャ関数を呼び出します。エクスターナルプロシージャトランスフォーメーションの場合、エクスターナルプロシージャ関数はエクスターナルプロシージャモジュールからの入出力のために使用します。各入力行について、この関数は IN および IN-OUT ポート値にアクセスでき、OUT および IN-OUT ポート値を設定できます。エクスターナルプロシージャ関数に、入力処理および出力処理ロジックがすべて含まれています。

宣言

エクスターナルプロシージャ関数はパラメータを取りません。入力パラメータ配列は、InitParams() メソッドを介してメンバ変数 m_pInParamVector に渡されて格納されています。配列内の各エントリは、同じ順序でエクスターナルプロシージャトランスフォーメーションの対応する IN ポートおよび IN-OUT ポートに一致します。Integration Service は、ディスパッチ関数を呼び出す前にこのベクトルに値を入れます。

m_pOutParamVector メンバ変数を使って、Dispatch() 関数を返す前に出力行を渡します。

MyExternal Procedure トランスフォーメーションの場合、エクスターナルプロシージャ関数は下記のとおりです。そして入力パラメータは m_pInParamVector メンバ変数の中にあり、出力値は m_pOutParamVector メンバ変数の中にあります。

```
INF_RESULT Tx<ModuleName>::MyFunc()
```

プロパティアクセス関数

プロパティアクセス関数は、エクスターナルプロシージャトランスフォーメーションと関連のある初期化プロパティに関する情報を提供します。初期設定プロパティ名および値は、エクスターナルプロシージャトランスフォーメーションを編集するときに [初期化プロパティ] タブに表示されます。

Informatica は、ベースクラスと TINFConfigEntriesList クラスの両方でプロパティアクセス関数を提供します。TINFConfigEntriesList クラスで、GetConfigEntryName() 関数を使用して初期設定プロパティ名にアクセスし、GetConfigEntryValue() 関数を使用して初期設定値にアクセスします。

宣言

Informatica はベースクラスで下記の関数を提供しています。

```
TINFConfigEntriesList* TINFBaseExternalModule60::accessConfigEntriesList();  
const char* GetConfigEntry(const char* LHS);
```

Informatica は TINFConfigEntriesList クラスで下記の関数を提供しています。

```
const char* TINFConfigEntriesList::GetConfigEntryValue(const char* LHS);  
const char* TINFConfigEntriesList::GetConfigEntryValue(int i);  
const char* TINFConfigEntriesList::GetConfigEntryName(int i);  
const char* TINFConfigEntriesList::GetConfigEntry(const char* LHS)
```

注: TINFConfigEntriesList クラスで初期化プロパティ名と値にアクセスするには、GetConfigEntryName() プロパティアクセス関数と GetConfigEntryValue() プロパティアクセス関数を使用します。

これらの関数は、TX プログラムから呼び出すことができます。TX プログラムは、たとえば `atoi` または `sscanf` を使用して、この文字列値を数値に変換します。下記の例では、「`addFactor`」が初期設定プロパティです。`accessConfigEntriesList()`は TX ベースクラスのメンバ変数であり、定義の必要がありません。

```
const char* addFactorStr = accessConfigEntriesList()-> GetConfigEntryValue("addFactor");
```

パラメータアクセス関数

パラメータのデータタイプによって、異なるパラメータアクセス関数が使用されます。パラメータアクセス関数の `GetDataType` を使用して、パラメータのデータタイプを返します。そして、このデータタイプに対応するパラメータアクセス関数を使用して、パラメータに関する情報を返します。

エクスターナルプロシージャに渡されるパラメータは、データタイプ `TINFPParam*` に属します。ヘッダファイルの `infparam.h` は、関連するアクセス関数を定義します。Designer は、パラメータのデータタイプを示すコメントを含んだスタブコードを生成します。また、対応するエクスターナルプロシージャトランスフォーメーション内のパラメータのデータタイプを決定することもできます。

宣言

エクスターナルプロシージャに渡されるパラメータは、`TINFPParam` クラスのオブジェクトへのポインタです。この固定宣言関数はそのクラスのメソッドであり、パラメータのデータタイプを `enum` 値として返します。

有効なデータタイプは次のとおりです。

- `INF_DATATYPE_LONG`
- `INF_DATATYPE_STRING`
- `INF_DATATYPE_DOUBLE`
- `INF_DATATYPE_RAW`
- `INF_DATATYPE_TIME`

以下の表に、パラメータアクセス関数を示します。

パラメータアクセス関数	説明
<code>INF_DATATYPE GetDataType(void);</code>	パラメータのデータ型を取得します。パラメータのデータ型を使用して、パラメータ値にアクセスするときに使用するデータ型別の関数を決定します。
<code>INF_BOOLEAN IsValid(void);</code>	入力データが有効であるかを確認します。パラメータに <code>string</code> データ型または <code>raw</code> データ型用の切り詰められたデータが含まれる場合、 <code>FALSE</code> を返します。この他、入力データが指定された精度を超えるか入力データが <code>NULL</code> 値である場合も <code>FALSE</code> を返します。
<code>INF_BOOLEAN IsNULL(void);</code>	入力データが <code>NULL</code> であるかを確認します。
<code>INF_BOOLEAN IsInputMapped (void);</code>	このパラメータにデータを渡す入力ポートがトランスフォーメーションに接続されているか確認します。
<code>INF_BOOLEAN IsOutput Mapped (void);</code>	このパラメータからデータを受け取る出力ポートがトランスフォーメーションに接続されているか確認します。
<code>INF_BOOLEAN IsInput(void);</code>	パラメータが入力ポートに対応しているか確認します。
<code>INF_BOOLEAN IsOutput(void);</code>	パラメータが出力ポートに対応しているか確認します。

パラメータアクセス関数	説明
INF_BOOLEAN GetName(void);	パラメータの名前を取得します。
SQLIndicator GetIndicator(void);	パラメータインジケータの値を取得します。IsValid および ISNULL は、この関数の特殊なケースです。この関数は、INF_SQL_DATA_TRUNCATED を返すこともできます。
void SetIndicator(SQLIndicator Indicator);	invalid（無効）、truncated（切り詰め）などの出力パラメータインジケータを設定します。
long GetLong(void);	Long または Integer データ型のパラメータの値を取得します。パラメータのデータ型が Integer または Long であると分かっている場合にのみ、この関数を呼び出します。この関数は、別のデータ型から Long にデータを変換しません。
double GetDouble(void);	Float または Double データ型のパラメータの値を取得します。パラメータのデータ型が Float または Double であると分かっている場合にのみ、この関数を呼び出します。この関数は、別のデータ型から Double にデータを変換しません。
char* GetString(void);	NULL で終わる文字列として、パラメータの値を取得します。パラメータのデータ型が String であると分かっている場合にのみ、この関数を呼び出します。この関数は、別のデータ型から String にデータを変換しません。 ポインタの値は、次の行のデータが読み込まれると変更されます。後で使用するために行の値を格納しておきたい場合は、割り当てられたバッファに文字列を明示的にコピーします。
char* GetRaw(void);	NULL で終わらないバイト配列として、パラメータの値を取得します。パラメータのデータ型が Raw であると分かっている場合にのみ、この関数を呼び出します。この関数は、別のデータ型から Raw にデータを変換しません。
unsigned long GetActualDataLen(void);	GetRaw によって返された配列の現在の長さを取得します。
TINFTime GetTime(void);	Date/Time データ型のパラメータの値を取得します。パラメータのデータ型が Date/Time であると分かっている場合にのみ、この関数を呼び出します。この関数は、別のデータ型から Date/Time にデータを変換しません。
void SetLong(long lVal);	Long データ型の出力パラメータの値を設定します。
void SetDouble(double dblVal);	Double データ型の出力パラメータの値を設定します。
void SetString(char* sVal);	String データ型の出力パラメータの値を設定します。
void SetRaw(char* rVal, size_t ActualDataLen);	NULL で終わらないバイト配列の値を設定します。
void SetTime(TINFTime timeVal);	Date/Time データ型の出力パラメータの値を設定します。

64 ビットの Integration Service でエクスターナルプロシージャを実行する場合、SetInt32 関数または GetInt32 関数のみを使用します。次の関数は使用しないでください。

- GetLong
- SetLong

- GetpLong
- GetpDouble
- GetpTime

2つのパラメータリストを使ってパラメータを渡します。

以下の表に、エクスターナルプロシージャのベースクラスのメンバ変数を示します。

変数	説明
m_nInParamCount	入力パラメータの数。
m_pInParamVector	実際の入力パラメータ配列。
m_nOutParamCount	出力パラメータの数。
m_pOutParamVector	実際の出力パラメータ配列。
注: 入出力として定義されているポートは、両方のパラメータリストに含まれます。	

コードページアクセス関数

Informatica では、Integration Service のコードページを返す 2 つのコードページアクセス関数と、エクスターナルプロシージャが処理するデータのコードページを返す 2 つのコードページアクセス関数が提供されています。Integration Service が Unicode モードで動作している場合、エクスターナルプロシージャプログラムに渡される文字列データにはマルチバイト文字を含めることができます。コードページはエクスターナルプロシージャがマルチバイト文字列をどのように変換するかを決定します。Integration Service が Unicode モードで動作している場合、エクスターナルプロシージャプログラムで処理するデータは Integration Service のコードページと相互互換性がある必要があります。

宣言

エクスターナルプロシージャプログラムで Integration Service のコードページを取得するには、次の関数を使用します。どちらの関数も、等価な情報を返します。

```
int GetServerCodePageID() const;
const char* GetServerCodePageName() const;
```

下記の関数を使って、エクスターナルプロシージャがエクスターナルプロシージャプログラムを通じて処理するデータのコードページを取得します。どちらの関数も、等価な情報を返します。

```
int GetDataCodePageID(); // returns 0 in case of error
const char* GetDataCodePageName() const; // returns NULL in case of error
```

トランスフォーメーション名アクセス関数

Informatica は、エクスターナルプロシージャトランスフォーメーションの名前を返すトランスフォーメーション名アクセス関数を 2 つ提供します。GetWidgetName()関数はトランスフォーメーションの名前を返し、GetWidgetInstanceName()関数はマプレットまたはマッピングの中のトランスフォーメーションインスタンスの名前を返します。

宣言

トランスフォーメーション名アクセス関数が返す `char*` は、Integration Service のコードページ内の MBCS 文字列です。これはデータコードページにはありません。

```
const char* GetWidgetInstanceName() const;
const char* GetWidgetName() const;
```

プロシージャアクセス関数

Informatica は、エクスターナルプロシージャトランスフォーメーションに関連するエクスターナルプロシージャに関する情報を提供する Procedure アクセス関数を 2 つ提供します。GetProcedureName() 関数は、エクスターナルプロシージャトランスフォーメーションの [Procedure Name] フィールドで指定されているエクスターナルプロシージャの名前を返します。GetProcedureIndex() 関数は、エクスターナルプロシージャのインデックスを返します。

宣言

下記の関数を使用して、エクスターナルプロシージャトランスフォーメーションに関連するエクスターナルプロシージャの名前を取得します。

```
const char* GetProcedureName() const;
```

下記の関数を使用して、エクスターナルプロシージャトランスフォーメーションに関連するエクスターナルプロシージャのインデックスを取得します。

```
inline unsigned long GetProcedureIndex() const;
```

パーティション関連の関数

エクスターナルプロシージャのパーティション関連の関数は、複数のパーティションを使用するセッションで使用します。エクスターナルプロシージャトランスフォーメーションを含むセッションをパーティション化すると、Integration Service は各パーティションにこれらのトランスフォーメーションのインスタンスを作成します。例えば、1 つのセッションに対して 5 つのパーティションを定義した場合、Integration Service はセッション実行時に、各エクスターナルプロシージャに対して 5 つのインスタンスを作成します。

宣言

セッションのパーティションの数を取得するには、次の関数を使用します。

```
unsigned long GetNumberOfPartitions();
```

このエクスターナルプロシージャを呼び出したパーティションのインデックスを取得するには、次の関数を使用します。

```
unsigned long GetPartitionIndex();
```

トレースレベル関数

トレースレベル関数は、以下に示すようにセッショントレースレベルを返します。

```
typedef enum
{
    TRACE_UNSET = 0,
    TRACE_TERSE = 1,
    TRACE_NORMAL = 2,
    TRACE_VERBOSE_INIT = 3,
    TRACE_VERBOSE_DATA = 4
} TracingLevelType;
```

宣言

セッショントレースレベルを返すには、次の関数を使用します。

```
TracingLevelType GetSessionTraceLevel();
```

第 9 章

フィルタトランスフォーメーション

この章では、以下の項目について説明します。

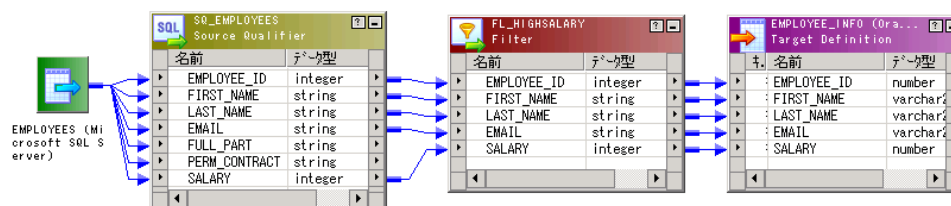
- [フィルタトランスフォーメーションの概要, 199 ページ](#)
- [フィルタトランスフォーメーションのコンポーネント, 200 ページ](#)
- [フィルタ条件, 200 ページ](#)
- [フィルタトランスフォーメーションの作成手順, 201 ページ](#)
- [フィルタトランスフォーメーションに関するヒント, 202 ページ](#)

フィルタトランスフォーメーションの概要

フィルタトランスフォーメーションは、マッピング内の行をフィルタで除外するのに使用します。アクティブトランスフォーメーションとして、フィルタトランスフォーメーションはそれを通過する行の数を変更することができます。フィルタトランスフォーメーションで行が通過を許可されるのは、指定されたフィルタ条件を満たした場合であり、条件を満たさない行は削除されます。データは 1 つ以上の条件に基づいてフィルタできます。フィルタトランスフォーメーションはアクティブなトランスフォーメーションです。

Integration Service で指定の条件を満たすかどうかを基準に評価された各行について、フィルタ条件により TRUE または FALSE が返されます。TRUE を返した各行はトランスフォーメーションを通過し、FALSE を返した各行は除去されてメッセージがセッションログに書き込まれます。

以下のマッピングは、人的資源テーブルのうち従業員データが含まれている行をフィルタトランスフォーメーションに渡します。このフィルタは、給料が\$30,000 以上の従業員の行を通過させます。



複数のトランスフォーメーションからのポートをフィルタトランスフォーメーションに連結することはできません。フィルタの入力ポートは、1 つのトランスフォーメーションからのものでなければなりません。

ヒント: セッションのパフォーマンスを最大限に高めたい場合は、マッピング内のソースのできる限り近くにフィルタトランスフォーメーションを配置してください。破棄しようとしている行がマッピングを通過しないよ

うに、ソースからターゲットへのデータフローの初期段階で不必要なデータをフィルタで除外することができます。

フィルタトランスフォーメーションのコンポーネント

フィルタトランスフォーメーションは、Transformation Developer または Mapping Designer で作成できます。フィルタトランスフォーメーションには、次のタブが含まれています。

- **【トランスフォーメーション】**。トランスフォーメーションの名前および説明を入力します。フィルタトランスフォーメーションの命名規則は、「FIL_トランスフォーメーション名」です。また、トランスフォーメーションを再利用可能にすることもできます。
- **ポート**。ポートを作成して設定します。
- **プロパティ**。行をフィルタリングするフィルタ条件を設定します。フィルタ条件は、式エディタを使用して入力します。[プロパティ] タブで、セッションログファイルに記録されるトランザクション詳細の量を決定するトレースレベルを設定することもできます。
- **メタデータエクステンション**。再利用不可能のメタデータエクステンションを作成して、トランスフォーメーションのメタデータを拡張します。エクステンション名、データタイプ、精度、および値を設定します。メタデータエクステンションをすべてのトランスフォーメーションで利用可能にしたい場合、メタデータエクステンションを再利用可能に格上げすることもできます。

フィルタトランスフォーメーションのポートの設定

[ポート] タブでは、ポートの作成および編集が可能です。

[ポート] タブでは、以下のプロパティを設定できます。

- **ポート名**。ポートの名前。
- **データタイプ、精度、およびスケール**。ポートごとにデータタイプを設定し、精度およびスケールを設定します。
- **ポートタイプ**。ポートはすべて、入出力ポートです。入力ポートはデータを受け取り、出力ポートはデータを渡します。
- **Default values and description**。ポート用のデフォルト値を設定し、説明を追加します。

フィルタ条件

フィルタ条件は、TRUE または FALSE を返す式です。条件は、[プロパティ] タブで利用可能な式エディタを使用して入力します。

単一の値を返す式はすべてフィルタとして使用することができます。たとえば給料が\$30,000 未満の従業員の行を除外したい場合は、次のように条件を入力します。

```
SALARY > 30000
```


AND および OR 論理演算子を使用すると、複数の条件を指定することができます。たとえば給料が\$30,000 未満であるか、または\$100,000 を超える従業員を除外したい場合は、次のように条件を入力します。

`SALARY > 30000 AND SALARY < 100000`

フィルタ条件に定数を入力することもできます。FALSE に該当する値はゼロ (0) です。ゼロ以外の値は TRUE とみなされます。たとえば、数値データタイプの NUMBER_OF_UNITS という名前のポートが、トランスフォーメーションに含まれているとします。NUMBER_OF_UNITS の値が 0 に等しければ FALSE を返すようにフィルタ条件を設定します。値がゼロでなければ、TRUE が返されます。

TRUE または FALSE を式の値として指定する必要はありません。TRUE および FALSE は、設定したすべての条件に対する暗黙の戻り値です。フィルタ条件が NULL として評価されると、その行は FALSE として扱われます。

注: フィルタ条件では大文字と小文字が区別されます。

NULL 値を含む行のフィルタ

NULL 値または空白を含む行をフィルタリングするには、ISNULL 関数と IS_SPACES 関数を使用してポートの値を調べます。たとえば FIRST_NAME ポートで NULL 値を含む行をフィルタで除外したい場合は、次の条件を使用します。

`IIF(ISNULL(FIRST_NAME),FALSE,TRUE)`

この条件は、FIRST_NAME ポートが NULL であれば戻り値として FALSE を返し、その行を無視することを指定しています。NULL が含まれていなければ、行は次のトランスフォーメーションへ渡されます。

フィルタトランスフォーメーションの作成手順

以下の手順に従って、フィルタトランスフォーメーションを作成します。

1. Mapping Designer でマッピングを開きます。
2. [トランスフォーメーション] - [作成] をクリックします。フィルタトランスフォーメーションを選択します。
3. トランスフォーメーションの名前を入力します。[作成] をクリックし、次に [完了] をクリックします。
4. ソース修飾子またはその他のトランスフォーメーションからポートをすべて選択し、フィルタトランスフォーメーションにドラッグして追加します。
5. タイトルバーをダブルクリックし、[ポート] タブをクリックします。トランスフォーメーション内でポートを手動で作成することもできます。
6. [プロパティ] タブをクリックし、フィルタ条件およびトレースレベルを設定します。
7. フィルタ条件の [値] セクションで、式エディタを開きます。
8. 適用したいフィルタ条件を入力します。デフォルトの条件では、TRUE が返されます。
トランスフォーメーションのいずれかの入力ポートの値をこの条件の一部として使用します。ただし、ほかのトランスフォーメーションの出力ポートの値を使用することもできます。
9. 式を入力します。[検証] をクリックして、入力した条件の構文をチェックします。
10. トレースレベルを選択します。
11. [メタデータエクステンション] タブで、メタデータエクステンションを追加します。

フィルタトランスフォーメーションに関するヒント

マッピングの初期段階でフィルタトランスフォーメーションを使用してください。

セッションのパフォーマンスを最大限に高めるには、マッピング内のソースのできる限り近くにフィルタトランスフォーメーションを配置します。無視したい行がマッピングを通過しないように、ソースからターゲットへのデータフローの初期段階で不必要なデータを除外することができます。

ソース修飾子トランスフォーメーションを使用してフィルタリングを行います。

ソース修飾子トランスフォーメーションは、行をフィルタリングするためのもうひとつの方法を提供します。マッピング内で行をフィルタリングする代わりに、ソース修飾子はソースから行を読み込むときにフィルタリングを行います。主な違いは、ソース修飾子ではソースから抽出される行セット、フィルタトランスフォーメーションではターゲットに送られる行セットが制限される点です。ソース修飾子ではマッピングにおいて使用される行の数が減るために、パフォーマンスが向上します。

ただしソース修飾子トランスフォーメーションの場合はリレーショナルソースからの行だけをフィルタリングするのに対し、フィルタトランスフォーメーションの場合はすべての種類のソースからの行をフィルタリングできます。また、データベース内で実行されるため、ソース修飾子トランスフォーメーション内のフィルタ条件に標準 SQL だけが使用されていることに注意してください。フィルタトランスフォーメーションでは、TRUE または FALSE 値を返す任意の文またはトランスフォーメーション関数を使用して、条件を定義することができます。

第 10 章

HTTP トランスフォーメーション

この章では、以下の項目について説明します。

- [HTTP トランスフォーメーションの概要, 203 ページ](#)
- [HTTP トランスフォーメーションの作成, 204 ページ](#)
- [\[プロパティ\] タブの設定, 205 ページ](#)
- [\[HTTP\] タブの設定, 206 ページ](#)
- [例, 213 ページ](#)

HTTP トランスフォーメーションの概要

HTTP トランスフォーメーションによって、HTTP サーバーに接続し、サービスおよびアプリケーションを使用できます。HTTP トランスフォーメーションは、パッシブトランスフォーメーションです。HTTP トランスフォーメーションを含むセッションを実行すると、統合サービスは HTTP サーバーに接続し、トランスフォーメーションの設定に応じて、HTTP サーバーからのデータの取得または HTTP サーバーのデータの更新のリクエストを発行します。

- **HTTP サーバーからのデータの読み取り。**統合サービスが HTTP サーバーからデータを読み取るとき、HTTP サーバーからデータを取得して、そのデータをターゲットまたはマッピング内の後続のトランスフォーメーションに渡します。例えば、HTTP サーバーに接続して現在の在庫データを読み取り、PowerCenter セッション中にデータの計算を実行し、そのデータをターゲットに渡すことができます。
- **HTTP サーバー上のデータの更新。**統合サービスが HTTP サーバーに書き込むとき、HTTP サーバーにデータをポストし、HTTP サーバーのレスポンスをターゲットまたはマッピング内の後続のトランスフォーメーションに渡します。例えば、セッション中に、先行するトランスフォーメーションからスケジュール情報を提供するデータを HTTP サーバーにポストできます。

統合サービスは、先行するトランスフォーメーションまたはソースから HTTP トランスフォーメーションにデータを渡し、HTTP トランスフォーメーションまたはアプリケーション接続内に設定された URL を読み取り、データの読み取りまたは更新の HTTP リクエストを HTTP サーバーに送信します。

リクエストにはヘッダ情報が含まれており、ボディ情報が含まれる場合もあります。ヘッダには、認証パラメータ、HTTP サーバー上のプログラムまたは Web サービスを有効化するコマンド、HTTP リクエスト全体に適用されるその他の情報などが含まれます。ボディには、統合サービスが HTTP サーバーに送信するデータが含まれます。

統合サービスがデータ読み取りのリクエストを送信する場合、HTTP サーバーはリクエストされたデータを含む HTTP レスポンスを返信します。統合サービスは、リクエストされたデータを後続のトランスフォーメーションまたはターゲットに送信します。

統合サービスがデータ更新のリクエストを送信する場合、HTTP サーバーは受信したデータを書き込み、更新成功の HTTP レスポンスを返します。HTTP トランスフォーメーションは、応答コード 200、201、202 を成功と見なします。HTTP トランスフォーメーションでは、他のレスポンスコードはすべて失敗と見なします。失敗コードとして認識された応答コードが HTTP サーバーから HTTP トランスフォーメーションに渡されると、セッションログにエラーが表示されます。統合サービスは、HTTP レスポンスを後続のトランスフォーメーションまたはターゲットに送信します。

HTTP レスポンスのヘッダ用の HTTP トランスフォーメーションを設定できます。HTTP レスポンスのボディデータは、HTTPOUT 出力ポートを通過します。

認証

HTTP トランスフォーメーションは、以下の形式の認証を使用します。

- **ベーシック**。暗号化されないユーザー名およびパスワードに基づく認証です。
- **ダイジェスト**。暗号化されたユーザー名およびパスワードに基づく認証です。
- **NTLM**。暗号化されたユーザー名、パスワード、およびドメインに基づく認証です。

HTTP サーバーへの接続

HTTP トランスフォーメーションを設定するときに、接続用の URL を設定できます。また、Workflow Manager で HTTP 接続オブジェクトを作成することもできます。以下の場合、HTTP アプリケーション接続を設定してください。

- HTTP サーバーが認証を要求した場合。
- 接続タイムアウトを設定する場合。
- HTTP トランスフォーメーションのベース URL を上書きする場合。

HTTP トランスフォーメーションの作成

HTTP トランスフォーメーションは、Transformation Developer または Mapping Designer で作成します。HTTP トランスフォーメーションには、以下のタブがあります。

- **[トランスフォーメーション]**。トランスフォーメーションの名前および説明を設定します。
- **Ports**。トランスフォーメーションの入力ポートおよび出力ポートを表示します。[ポート] タブでポートの追加や編集を行うことはできません。[HTTP] タブでヘッダグループにポートを追加すると、Designer によって [ポート] タブにポートが作成されます。
- **Properties**。[プロパティ] タブでは、HTTP トランスフォーメーションのプロパティを設定します。
- **[HTTP]**。[HTTP] タブでは、メソッド、ポート、および URL を設定します。

HTTP トランスフォーメーションを作成するには：

1. Transformation Developer または Mapping Designer で、[トランスフォーメーション] - [作成] をクリックします。
2. HTTP トランスフォーメーションを選択します。
3. トランスフォーメーションの名前を入力します。
4. [作成] をクリックします。

HTTP トランスフォーメーションがワークスペースに表示されます。

5. [完了] をクリックします。
6. トランスフォーメーションのタブを設定します。

[プロパティ] タブの設定

HTTP トランスフォーメーションは、カスタムトランスフォーメーションを使用して作成されます。カスタムトランスフォーメーションの一部のプロパティは、HTTP トランスフォーメーションには適用されないか、または設定できません。

以下の表に、設定可能な HTTP トランスフォーメーションのプロパティを示します。

オプション	説明
実行時位置	<p>DLL または共有ライブラリの格納場所。デフォルトは \$PMExtProcDir です。HTTP トランスフォーメーションのセッションを実行する Integration Service ノードへの相対パスを入力します。</p> <p>このプロパティが空白の場合、Integration Service は、Integration Service で定義されている環境変数を使用して DLL または共有ライブラリの場所を探します。</p> <p>Integration Service ノードで定義されている実行時位置または環境変数に、すべての DLL または共有ライブラリをコピーする必要があります。DLL、共有ライブラリ、または参照されるファイルが見つからない場合、Integration Service は手続きのロードに失敗します。</p>
トレースレベル	<p>トランスフォーメーションのセッションログに表示される情報の詳細度。デフォルトは [Normal] です。</p>
パーティション化可能	<p>このトランスフォーメーションを使用するパイプラインで、複数のパーティションを作成できるかどうかを指定します。</p> <ul style="list-style-type: none">- いいえ。トランスフォーメーションはパーティション化できません。同一パイプライン内のこのトランスフォーメーションおよびその他のトランスフォーメーションは、1 つのパーティションに含まれる必要があります。- ローカルで。トランスフォーメーションをパーティション化することはできますが、同じノード上のパイプラインですべてのパーティションが実行される必要があります。Csutom トランスフォーメーションの別のパーティションがメモリ内のオブジェクトを共有する必要がある場合、[ローカルで] を選択します。- グリッドをまたがる。トランスフォーメーションをパーティション化することができ、各パーティションは異なるノードに配分されます。 <p>デフォルトは [No]。</p>
出力が再現可能	<p>出力データの順序をセッションの実行ごとに一致させるかどうかを指定します。</p> <ul style="list-style-type: none">- Never。出力データの順序はセッションの実行ごとに異なります。- 入力順による。入力データの順序がセッションの実行ごとに一致している場合、出力順序をセッションの実行ごとに一致させます。- Always。入力データの順序がセッションの実行ごとに異なる場合でも、出力データの順序は常に同じです。 <p>デフォルトは [入力順による] です。</p>

オプション	説明
パーティションごとに1つのスレッドを要求します	Integration Service によってプロシージャの各パーティションが1つのスレッドで処理される場合に指定します。
出力が確定的かどうか	トランスフォーメーションが、セッションの実行ごとに一貫した出力データを生成するかどうかを指定します。このトランスフォーメーションを使用するセッションでリカバリを実行するには、このプロパティを有効にします。デフォルトでは有効になっています。

警告: トランスフォーメーションを繰り返し可能で一意に定まるものとして設定する場合は、データが繰り返し可能で一意に定まることを保証する必要があります。セッションとリカバリで同じデータが生成されないトランスフォーメーションを使用してセッションをリカバリしようとすると、リカバリプロセスを実行した結果、データが破損する可能性があります。

[HTTP] タブの設定

[HTTP] タブでは、HTTP サーバーからデータを読み取るか、または HTTP サーバーにデータを書き込むようにトランスフォーメーションを設定できます。[HTTP] タブでは、以下の情報を設定します。

- **メソッドを選択します。** データを読み取るか、HTTP サーバーにデータを書き込むか、部分更新を実行するか、データブロック全体を置換するか、または HTTP サーバーからデータを削除するかに基づいて、GET、POST、SIMPLE POST、SIMPLE PATCH、SIMPLE PUT、または SIMPLE DELETE メソッドを選択します。
- **グループおよびポートの設定。** 入力ポートおよび出力ポートを設定することによって、HTTP リクエスト/レスポンスのボディとヘッダの詳細を管理します。特殊文字を使用するポート名を設定することもできます。
- **ベース URL の設定。** 接続先の HTTP サーバーのベース URL を設定します。

メソッドの選択

トランスフォーメーションで定義するグループおよびポートは、選択するメソッドによって異なります。HTTP サーバーからデータを読み取るには、GET メソッドを選択します。HTTP サーバーにデータを書き込むには、POST または SIMPLE POST メソッドを選択します。

以下の表で、各メソッドについて説明します。

メソッド	説明
GET	HTTP サーバからデータを読み取ります。
POST	複数の入力ポートからのデータを HTTP サーバに書き込みます。
SIMPLE POST	1つの入力ポートからのデータを1つのデータブロックとして HTTP サーバに書き込みます。
SIMPLE PATCH	1つの入力ポートから部分データをリソースに対するパッチとして更新します。

メソッド	説明
SIMPLE PUT	リソースを置換または書き込みます。
SIMPLE DELETE	HTTP サーバーからリソースを削除します。

グループおよびポートの設定

HTTP トランスフォーメーションに追加するポートは、選択するメソッドおよびグループによって異なります。HTTP トランスフォーメーションは、以下のグループを使用します。

- **出力。** HTTP 応答のボディデータが含まれます。応答は、HTTP サーバーから後続のトランスフォーメーションまたはターゲットに渡されます。デフォルトで、1 つの出力ポート (HTTPOUT) が含まれます。出力グループにポートを追加することはできません。HTTPOUT 出力ポートの精度を変更できます。
- **入力。** HTTP リクエストのボディデータが含まれます。HTTP サーバーに接続する最終 URL を生成するために Designer が使用するメタデータも含まれます。HTTP サーバーにデータを書き込むために、入力グループによってボディ情報が HTTP サーバーに渡されます。デフォルトで、1 つの入力ポートが含まれます。
- **ヘッダ。** リクエストおよび応答のヘッダデータが含まれます。Integration Service が HTTP リクエストを送信すると、ヘッダ情報が HTTP サーバーに渡されます。ヘッダグループに追加するポートによって、HTTP ヘッダのデータが渡されます。ヘッダグループにポートを追加すると、ポートは Designer によって [ポート] タブの入力グループおよび出力グループに追加されます。デフォルトでは、ポートは含まれません。すべてのメソッドで、HTTP リクエストのヘッダ情報用にヘッダグループを使用できます。

注: HTTP トランスフォーメーションを通過するデータは、文字列データタイプである必要があります。文字列データには、HTML や XML など、HTTP 通信で一般的なマークアップ言語が含まれます。

GET メソッド

HTTP サーバーからデータを読み取ります。HTTP リクエストのメタデータを定義するには、入力グループを使用し、HTTP サーバーの最終 URL を構築するために Designer が使用する入力ポートを追加します。

以下の表では、GET メソッドのグループおよびポートについて説明します。

要求/ レスポンス	グループ	説明
リクエスト	入力	Designer は、入力ポートの名前および値を使用して、最終 URL を生成します。
リクエスト	ヘッダ	HTTP リクエストの入出力ポートを設定できます。Designer はヘッダグループに追加されたポートに基づいて、入力グループおよび出力グループにポートを追加します。 <ul style="list-style-type: none"> - 入力グループ。ヘッダグループからの入力および入出力ポートに基づいて、入力ポートを作成します。 - 出力グループ。ヘッダグループからの入出力ポートに基づいて、出力ポートを作成します。

要求/ レスポンス	グループ	説明
レスポンス	ヘッダ	HTTP レスポンスの出力ポートおよび入出力ポートを設定できます。Designer はヘッダグループに追加されたポートに基づいて、入力グループおよび出力グループにポートを追加します。 <ul style="list-style-type: none"> - 入力グループ。ヘッダグループからの入出力ポートに基づいて、入力ポートを作成します。 - 出力グループ。ヘッダグループからの出力ポートおよび入出力ポートに基づいて、出力ポートを作成します。
レスポンス	アウトプット	HTTP レスポンスのすべてのボディデータが HTTPOUT 出力ポートを通過します。

POST メソッド

複数の入力ポートからのデータを HTTP サーバーに書き込みます。HTTP リクエストのメタデータを定義するには、HTTP リクエストのボディを定義するデータの入力グループを使用します。

以下の表では、POST メソッドのポートについて説明します。

要求/ レスポンス	グループ	説明
リクエスト	入力	複数のポートを入力グループに追加できます。ヘッダグループに追加されたポートに基づいて、HTTP リクエストのボディデータは 1 つ以上の入力ポートを通過できます。
リクエスト	ヘッダ	HTTP リクエストの入出力ポートを設定できます。Designer はヘッダグループに追加されたポートに基づいて、入力グループおよび出力グループにポートを追加します。 <ul style="list-style-type: none"> - 入力グループ。ヘッダグループからの入力および入出力ポートに基づいて、入力ポートを作成します。 - 出力グループ。ヘッダグループからの入出力ポートに基づいて、出力ポートを作成します。
レスポンス	ヘッダ	HTTP レスポンスの出力ポートおよび入出力ポートを設定できます。Designer はヘッダグループに追加されたポートに基づいて、入力グループおよび出力グループにポートを追加します。 <ul style="list-style-type: none"> - 入力グループ。ヘッダグループからの入出力ポートに基づいて、入力ポートを作成します。 - 出力グループ。ヘッダグループからの出力ポートおよび入出力ポートに基づいて、出力ポートを作成します。
レスポンス	アウトプット	HTTP レスポンスのすべてのボディデータが HTTPOUT 出力ポートを通過します。

SIMPLE POST メソッド

POST メソッドの簡略化されたバージョンです。1 つの入力ポートからのデータを 1 つのデータブロックとして HTTP サーバーに書き込みます。HTTP リクエストのメタデータを定義するには、HTTP リクエストのボディを定義するデータの入力グループを使用します。

以下の表では、SIMPLE POST メソッドのポートについて説明します。

要求/レスポンス	グループ	説明
リクエスト	入力	1つの入力ポートを追加できます。HTTP リクエストのボディデータは、1つの入力ポートを通過できます。
リクエスト	ヘッダ	HTTP リクエストの入出力ポートを設定できます。Designer はヘッダグループに追加されたポートに基づいて、入力グループおよび出力グループにポートを追加します。 <ul style="list-style-type: none">- 入力グループ。ヘッダグループからの入力および入出力ポートに基づいて、入力ポートを作成します。- 出力グループ。ヘッダグループからの入出力ポートに基づいて、出力ポートを作成します。
レスポンス	ヘッダ	HTTP レスポンスの出力ポートおよび入出力ポートを設定できます。Designer はヘッダグループに追加されたポートに基づいて、入力グループおよび出力グループにポートを追加します。 <ul style="list-style-type: none">- 入力グループ。ヘッダグループからの入出力ポートに基づいて、入力ポートを作成します。- 出力グループ。ヘッダグループからの出力ポートおよび入出力ポートに基づいて、出力ポートを作成します。
レスポンス	アウトプット	HTTP レスポンスのすべてのボディデータが HTTPOUT 出力ポートを通過します。

SIMPLE PATCH メソッド

1つの入力ポートから部分データをリソースに対するパッチとして更新します。1つの入力ポートからのデータを完全または部分のデータブロックとして HTTP サーバーに書き込みます。HTTP リクエストのメタデータを定義するには、HTTP リクエストのボディを定義するデータの入力グループを使用します。

以下の表では、SIMPLE PATCH メソッドのポートについて説明します。

要求/レスポンス	グループ	説明
リクエスト	入力	1つの入力ポートを追加できます。HTTP リクエストのボディデータは、1つの入力ポートを通過できます。
リクエスト	ヘッダ	HTTP リクエストの入出力ポートを設定できます。Designer はヘッダグループに追加されたポートに基づいて、入力グループおよび出力グループにポートを追加します。 <ul style="list-style-type: none">- 入力グループ。ヘッダグループからの入力および入出力ポートに基づいて、入力ポートを作成します。- 出力グループ。ヘッダグループからの入出力ポートに基づいて、出力ポートを作成します。

要求/レスポンス	グループ	説明
レスポンス	ヘッダ	HTTP レスポンスの出力ポートおよび入出力ポートを設定できます。Designer はヘッダグループに追加されたポートに基づいて、入力グループおよび出力グループにポートを追加します。 <ul style="list-style-type: none"> - 入力グループ。ヘッダグループからの入出力ポートに基づいて、入力ポートを作成します。 - 出力グループ。ヘッダグループからの出力ポートおよび入出力ポートに基づいて、出力ポートを作成します。
レスポンス	アウトプット	HTTP レスポンスのすべてのボディデータが HTTPOUT 出力ポートを通過します。

SIMPLE PUT メソッド

リソースを置換または書き込みます。1つの入力ポートからのデータを1つのデータブロックとして HTTP サーバーに書き込みます。

データが存在しない場合、SIMPLE PUT メソッドでデータを送信します。データが存在する場合、SIMPLE PUT メソッドで1つの入力ポートからのデータを1つのデータブロックとして HTTP サーバーに書き込みます。

HTTP リクエストのメタデータを定義するには、HTTP リクエストのボディを定義するデータの入力グループを使用します。

以下の表では、SIMPLE PUT メソッドのポートについて説明します。

要求/レスポンス	グループ	説明
リクエスト	入力	1つの入力ポートを追加できます。HTTP リクエストのボディデータは、1つの入力ポートを通過できます。
リクエスト	ヘッダ	HTTP リクエストの入出力ポートを設定できます。Designer はヘッダグループに追加されたポートに基づいて、入力グループおよび出力グループにポートを追加します。 <ul style="list-style-type: none"> - 入力グループ。ヘッダグループからの入力および入出力ポートに基づいて、入力ポートを作成します。 - 出力グループ。ヘッダグループからの入出力ポートに基づいて、出力ポートを作成します。
レスポンス	ヘッダ	HTTP レスポンスの出力ポートおよび入出力ポートを設定できます。Designer はヘッダグループに追加されたポートに基づいて、入力グループおよび出力グループにポートを追加します。 <ul style="list-style-type: none"> - 入力グループ。ヘッダグループからの入出力ポートに基づいて、入力ポートを作成します。 - 出力グループ。ヘッダグループからの出力ポートおよび入出力ポートに基づいて、出力ポートを作成します。
レスポンス	アウトプット	HTTP レスポンスのすべてのボディデータが HTTPOUT 出力ポートを通過します。

SIMPLE DELETE メソッド

HTTP サーバーからリソースを削除します。要求本文が必要な場合、SIMPLE DELETE メソッドで HTTP サーバーに対する1つのデータブロックとして1つの入力ポートからのデータを削除します。HTTP リクエストのメ

タデータを定義するには、入力グループを使用し、HTTP サーバーの最終 URL を構築するために Designer が使用する入力ポートを追加します。

以下の表では、SIMPLE DELETE メソッドのポートについて説明します。

要求/ レスポンス	グループ	説明
リクエスト	入力	Designer は、入力ポートの名前および値を使用して、最終 URL を生成します。
リクエスト	ヘッダ	HTTP リクエストの入出力ポートを設定できます。Designer はヘッダグループに追加されたポートに基づいて、入力グループおよび出力グループにポートを追加します。 <ul style="list-style-type: none">- 入力グループ。ヘッダグループからの入力および入出力ポートに基づいて、入力ポートを作成します。- 出力グループ。ヘッダグループからの入出力ポートに基づいて、出力ポートを作成します。
レスポンス	ヘッダ	HTTP レスポンスの出力ポートおよび入出力ポートを設定できます。Designer はヘッダグループに追加されたポートに基づいて、入力グループおよび出力グループにポートを追加します。 <ul style="list-style-type: none">- 入力グループ。ヘッダグループからの入出力ポートに基づいて、入力ポートを作成します。- 出力グループ。ヘッダグループからの出力ポートおよび入出力ポートに基づいて、出力ポートを作成します。
レスポンス	アウト プット	HTTP レスポンスのすべてのボディデータが HTTPOUT 出力ポートを通過します。

HTTP 名の追加

Designer では、ポート名にダッシュ (-) などの特殊文字を使用できません。ポート名に特殊文字を使用する必要がある場合は、ポート名を上書きするように HTTP 名を設定できます。たとえば、Content-type という名前の入力ポートが必要な場合、ポートに ContentType という名前を付け、HTTP 名として Content-Type を入力できます。

URL の設定

メソッドを選択し、入力ポートおよび出力ポートを設定した後、URL を設定する必要があります。ベース URL を入力すると、Designer によって最終 URL が生成されます。GET または SIMPLE DELETE メソッドを選択した場合、最終 URL にはベース URL と入力グループのポート名に基づくパラメータが含まれます。SIMPLE PATCH、SIMPLE PUT、POST、または SIMPLE POST メソッドを選択した場合、最終 URL はベース URL と同じになります。

マッピングパラメータや変数を使用して、ベース URL を設定できます。たとえば、マッピングパラメータ `$ParamBaseURL` を宣言し、[ベース URL] フィールドにマッピングパラメータ `$$ParamBaseURL` を入力してから、パラメータファイルに `$$ParamBaseURL` を定義します。

HTTP アプリケーション接続を設定するときに URL を指定することもできます。HTTP アプリケーション接続で指定されたベース URL によって、HTTP トランスフォーメーションで指定されたベース URL は上書きされます。

注: HTTP サーバーは、HTTP リクエストを別の HTTP サーバーにリダイレクトできます。この場合、HTTP サーバーは URL を Integration Service に返信し、Integration Service は他の HTTP サーバーへの接続を確立します。Integration Service では、最大 5 つの追加接続を確立できます。

GET メソッドの最終 URL の生成

Designer によって、ベース URL および入力グループのポート名に基づいて、GET メソッドの最終 URL が生成されます。ベース URL に HTTP 引数が付加され、HTTP クエリー文字列の形式で最終 URL が生成されます。クエリー文字列は、疑問符 (?)、およびそれに続く名前／値のペアで構成されます。Designer によって、疑問符、および入力グループに追加した入力ポートの名前と値に対応する名前／値のペアが付加されます。

GET メソッドを選択し、入力ポートを入力グループに追加すると、Designer によって以下のグループおよびポート情報がベース URL に付加され、最終 URL が生成されます。

```
?<input group input port 1 name> = $<input group input port 1 value>
```

最初の入力グループの入力ポートに続く入力ポートごとに、Designer によって以下のグループおよびポート情報が付加されます。

```
& <input group input port n name> = $<input group input port n value>
```

ここで、*n* は入力ポートを表します。

たとえば、ベース URL に `www.company.com` を入力し、ID、EmpName、および Department という入力ポートを入力グループに追加した場合、Designer によって以下の最終 URL が生成されます。

```
www.company.com?ID=$ID&EmpName=$EmpName&Department=$Department
```

最終 URL を編集して、演算子、変数、またはその他の引数を変更または追加できます。HTTP リクエストおよびクエリー文字列の詳細については、<http://www.w3c.org> を参照してください。

ベース URL のパラメータ化

GET メソッドと SIMPLE DELETE メソッドのベース URL をパラメータ化できます。

例えば、次のようなベース URL があるとします。

```
www.informatica.com
```

Designer はベース URL の情報を使用し、次のような最終 URL を生成することができます。

```
www.informatica.com?firstname=1&lastname=2
```

GET または SIMPLE DELETE メソッドの場合、Designer でベース URL をパラメータ化するチェックボックス オプションを選択すると、ベース URL を編集して、次のようにマッピングパラメータまたは変数を使用することができます。

```
www.informatica.com/$firstname$lastname
```

Integration Services によって、ソースファイルの値が HTTP トランスフォーメーションの名および姓の入力ポートに送信され、最終 URL で指定された HTTP サーバーに HTTP リクエストが送信されます。

次のような最終 URL が表示されます。

```
www.informatica.com/12
```

URL での特殊文字

URL の設定時、最終 URL には UTF-8 文字セットでエンコードされた特殊文字が含まれる可能性があります。
次の表に、URL で使用される一部の特殊文字が UTF-8 環境内でどのように設定されるかを示します。

特殊文字	エンコード (UTF-8)
'	%27
(%28
)	%29
*	%2A
スペース	%20

統合サービスレベルまたはセッションレベルで UseURLAsEnteredinCURLEncoding カスタムフラグを使用する場合、ベース URL に入力したとおりの URL (CURL エンコード形式) を渡すことができます。

例

ここでは、それぞれのタイプのメソッドの例について説明します。

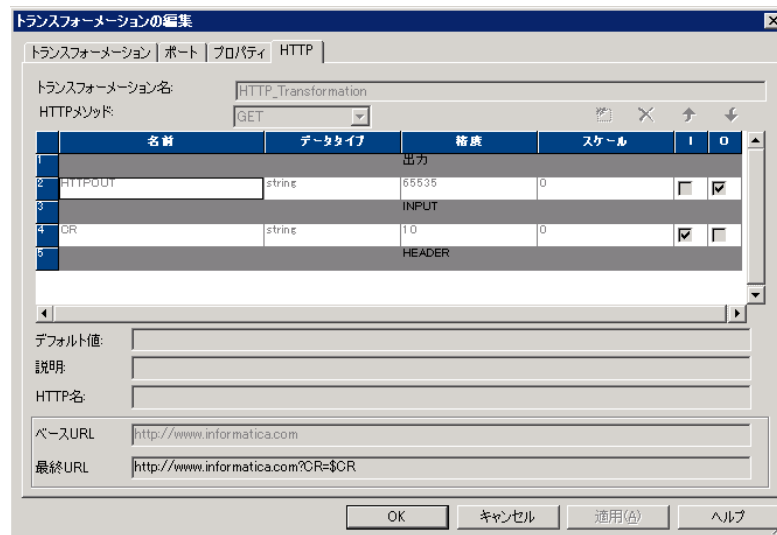
- GET
- POST
- SIMPLE POST
- SIMPLE PATCH
- SIMPLE PUT
- SIMPLE DELETE

GET の例

この例で使用するソースファイルには、以下のデータが含まれます。

78576
78577
78578

以下の図は、GET の例の HTTP トランスフォーメーションの [HTTP] タブを示しています。



Designer によって、ベース URL に疑問符 (?)、入力グループの入力ポート名、ドル記号 (\$)、および再び入力グループの入力ポート名が付加され、以下の最終 URL が生成されます。

http://www.informatica.com?CR=\$CR

Integration Service によって、ソースファイルの値が HTTP トランスフォーメーションの CR 入力ポートに送信され、以下の HTTP リクエストが HTTP サーバーに送信されます。

http://www.informatica.com?CR=78576

http://www.informatica.com?CR=78577

http://www.informatica.com?CR=78578

HTTP サーバーによって HTTP 応答が Integration Service に返信され、Integration Service によって HTTP トランスフォーメーションの出力ポートを介してターゲットにデータが送信されます。

POST の例

この例で使用するソースファイルには、以下のデータが含まれます。

33,44,1
44,55,2
100,66,0

以下の図は、ソースファイルの各フィールドには対応する入力ポートがあることを示しています。

トランスフォーメーションの編集

トランスフォーメーション: HTTP
HTTPメソッド: POST

	名前	データタイプ	精度	スケール	I	O
1			出力			
2	HTTPOUT	string	65535	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>
3			INPUT			
4	num1	string	100	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
5	num2	string	100	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
6	num3	string	100	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
7			HEADER			

デフォルト値:
説明:
HTTP名:
ベースURL:
最終URL:

OK Cancel Apply Help

Integration Service によって、各行の 3 つのフィールドの値が HTTP トランスフォーメーションの入力ポートを介して送信され、最終 URL で指定された HTTP サーバーに HTTP リクエストが送信されます。

SIMPLE POST の例

以下のテキストは、この例で使用される XML ファイルを示します。

```
<?xml version="1.0" encoding="UTF-16LE"?>
<n4:Envelope xmlns:cli="http://localhost:8080/axis/Clienttest1.jws" xmlns:n4="http://schemas.xmlsoap.org/soap/envelope/" xmlns:tns="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <n4:Header>
  </n4:Header>
  <n4:Body n4:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"><cli:smpSource>
    <Metadatainfo xsi:type="xsd:string">smpSourceRequest.Metadatainfo106</Metadatainfo></cli:smpSource>
  </n4:Body>
</n4:Envelope>,capeconnect:Clienttest1services:Clienttest1#smpSource
```

以下の図は、SIMPLE POST の例の HTTP トランスフォーメーションの [HTTP] タブを示しています。

トランスフォーメーションの編集

トランスフォーメーション: HTTP

HTTPメソッド: SIMPLE POST

	名前	データタイプ	精度	スケール	I	O
1			出力			
2	HTTPOUT	string	65535	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>
3			INPUT			
4	SOAPRequest	string	65535	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
5			HEADER			
6	soapaction	string	100	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
7	host	string	100	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

デフォルト値:

説明:

HTTP名:

ベースURL: http://scorpio:8080/axis/services/Clienttest1

最終URL: http://scorpio:8080/axis/services/Clienttest1

OK Cancel Apply Help

Integration Service によって、ソースファイルのボディが入力ポートを介して送信され、最終 URL で指定された HTTP サーバーに HTTP リクエストが送信されます。

SIMPLE PATCH の例

この例で使用するソースファイルには、以下のデータが含まれます。

```
{"firstname": "Raj"}
```


以下の図は、ソースファイルの各フィールドには対応する入力ポートがあることを示しています。

Edit Transformations

Transformation Ports Properties **HTTP**

Static

Transformation Name:

HTTP Method:

	Name	Datatype	Precision	Scale	I	O
1			OUTPUT			
2	HTTPOUT	string	65535	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>
3			INPUT			
4	update	string	1000	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
5			HEADER			
6	ContentType	string	1000	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Default Value:

Description:

HTTP Name:

Base URL:

Final URL:

☐ Parameterize Base URL

OK Cancel Apply Help

ベース URL および最終 URL として \$\$BASEURL1 マッピング変数を含めます。

\$\$BASEURL1 マッピング変数は、次のリンクを示します。

http://avengers5:8181/emp_all/1/

SIMPLE PATCH は、1 つの入力ポートから HTTP サーバーに対する部分データ更新をサポートします。

Integration Service によって、影響を受ける行の値が HTTP トランスフォーメーションの入力ポートを介して送信され、最終 URL で指定された HTTP サーバーに HTTP リクエストが送信されます。

SIMPLE PUT の例

この例で使用するソースファイルには、以下のデータが含まれます。

```
{"emp_id": 10,"firstname": "Raj","lastname": "Kumar","designation": "QA","department": "RND","age": 24}
```

以下の図は、ソースファイルの各フィールドには対応する入力ポートがあることを示しています。

Edit Transformations

Transformation Ports Properties **HTTP**

Static

Transformation Name:

HTTP Method:

	Name	Datatype	Precision	Scale	I	O
1			OUTPUT			
2	HTTPOUT	string	65535	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>
3			INPUT			
4	update	string	1000	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
5			HEADER			
6	contentType	string	100	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Default Value:

Description:

HTTP Name:

Base URL

Final URL

☐ Parameterize Base URL

OK Cancel Apply Help

SIMPLE PUT は 1 つの入力ポートからのデータを 1 つのデータブロックとして HTTP サーバーに書き込みます。Integration Service によって、ソースファイルのボディが HTTP トランスフォーメーションの入力ポートを介して送信され、最終 URL で指定された HTTP サーバーに HTTP リクエストが送信されます。

SIMPLE DELETE の例

この例で使用するソースファイルには、以下のデータが含まれます。

2
1

次の図に、[ベース URL のパラメータ化] オプションを選択した SIMPLE DELETE の例における HTTP トランスフォーメーションの [HTTP] タブを示します。

Edit Transformations

Transformation Ports Properties **HTTP**

Static

Transformation Name: HTTP

HTTP Method: SIMPLE DELETE

	Name	Datatype	Precision	Scale	I	O
1			OUTPUT			
2	HTTPOUT	string	65535	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>
3			INPUT			
4	id	string	10	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
5			HEADER			
6	ContentType	string	10	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Default Value:

Description:

HTTP Name:

Base URL:

Final URL:

☒ Parameterize Base URL

OK Cancel Apply Help

[ベース URL のパラメータ化] チェックボックスを選択すると、Designer はベース URL に入力グループ入力ポート名を付加して最終 URL を生成します。

http://www.avengers5:8181/emp_all/\$id/

Integration Service によって、ソースファイルの値が HTTP トランスフォーメーションの ID 入力ポートに送信され、以下の HTTP リクエストが HTTP サーバーに送信されます。

http://www.avengers5:8181/emp_all/2

http://www.avengers5:8181/emp_all/1

SIMPLE DELETE メソッドによって必要なデータが HTTP サーバーから削除されます。HTTP サーバーによって HTTP 応答が Integration Service に返信され、Integration Service によって HTTP トランスフォーメーションの出力ポートを介してターゲットに更新されたデータが送信されます。

第 11 章

ID 解決トランスフォーメーション

この章では、以下の項目について説明します。

- [ID 解決トランスフォーメーションの概要, 220 ページ](#)
- [トランスフォーメーションの作成と設定, 220 ページ](#)
- [ID 解決トランスフォーメーションのタブ, 223 ページ](#)
- [グループおよびポート, 223 ページ](#)

ID 解決トランスフォーメーションの概要

ID 解決トランスフォーメーションは、Informatica Identity Resolution (IIR) でデータを検索したりマッチングする際に使用できるアクティブなトランスフォーメーションです。PowerCenter Integration Service は、ID 解決トランスフォーメーションで指定した検索定義を使用して、IIR テーブルに存在するデータの検索やマッチングを行います。このトランスフォーメーションの入力ポートと出力ポートは、システム内の入力ビューと出力ビューによって決まります。

ID 解決トランスフォーメーションでマッチングの許容範囲と検索の幅パラメータを設定することにより、マッチング方式と検索レベルを設定します。セッションを実行すると、IIR は候補のレコードを ID 解決トランスフォーメーションに返します。

IIR では、Oracle、DB2/UDB、および Microsoft SQL Server のテーブルに格納されたあらゆるタイプの ID データのオンラインおよびバッチによる検索、マッチング、スクリーニング、リンク付け、および重複の検出を行います。IIR では IIR Search Server を使用して検索およびマッチング操作を行います。

トランスフォーメーションの作成と設定

ID 解決トランスフォーメーションは Transformation Developer で作成します。ID 解決トランスフォーメーションを作成するときは、まず Informatica Identity Resolution Search Server に接続します。Search Server は IIR テーブルへのアクセスを提供します。

接続したら、システムと検索およびマッチングのパラメータを設定します。 **【IR トランスフォーメーションの設定】** ウィンドウで、次の情報を設定します。

- IR Search Server の接続
- システムと検索の設定
- 入力ビューと出力ビュー

検索サーバーの接続

検索サーバーに接続するには、ホスト名とポート番号を指定します。次に、ルールベースの接続文字列を設定します。ルールベースには、IIR がデータの検索および照合に使用するシステムに関する情報が含まれています。ID 解決トランスフォーメーションのルールベース接続では、ルールベースが格納されているデータベースを指定します。

注: トランスフォーメーションの作成後にホスト名またはルールベース接続を変更することはできません。

IR ホスト名

Informatica ID 解決サーバーを実行するマシンのホスト名。

ポート

検索サーバーのポート番号。

ルールベースの接続文字列

検索サーバーへの接続文字列。ルールベース接続は、ODBC または SSA として指定できます。IIR ホストで、ODBC および SSA を設定します。

ODBC

以下の表に、ODBC 接続情報を示します。

接続プロパティ	説明
ルールベース番号	IR ルールベースに指定した番号。
ユーザー名	データベースユーザー名。
パスワード	上記データベースユーザー名のパスワード。
サービス名	odbc.ini ファイルで定義されているサービス名。

SSA

SSA を介して接続するには、エイリアスを指定する必要があります。エイリアスを指定することにより、アプリケーションプログラムから接続文字列を参照できなくなります。IIR は、ルールベース、データベース、およびソース名に関するエイリアスを使用します。

システムと検索の設定

Search Server に接続したら、システムパラメータ、検索パラメータ、一致パラメータを選択します。

これらのパラメータは、トランスフォーメーションの作成後に **[IR 設定]** タブで変更できます。

システム

ルールベース内でシステムを選択します。システムは、IIR ルールベース内の最上位の論理定義です。

検索定義

システムのルールを含む検索定義を選択します。検索定義はシステム内に存在し、レコードの検索、一致、および表示のためのルールが含まれます。

検索範囲

実行する検索の範囲を決定する検索範囲を選択します。

以下の表に、選択可能な検索範囲を示します。

検索範囲	説明
低	短い応答時間で限られた範囲の検索を行います。このオプションは、一致が見つからない可能性が低い場合、高い照合精度が要求されない場合、またはデータ量が多く応答時間が非常に重要である場合に使用します。
通常	データ品質と応答時間のバランスがとれた検索が可能です。このオプションは、オンライントランザクションまたはバッチトランザクション検索で使用します。デフォルトは [通常] です。
高	長い応答時間で詳細な検索を行います。このオプションは、一致が見つからない可能性が高い場合、データ品質が重要である場合、またはデータ量が少なく応答時間が重要でない場合に使用します。

一致許容範囲

候補レコードを選択する一致スキームの範囲を決定する一致許容範囲を選択します。

以下の表に、選択可能な一致許容範囲のオプションを示します。

一致許容範囲	説明
最高	一致候補と可能な処理方法のセットが得られます。処理の応答時間は相対的に長くなります。このオプションは、エラーや差異がある一致を含めて検索する場合に使用します。
保守的	過度でない一致または最高度でない一致が得られます。オンライントランザクションまたはバッチトランザクション検索に使用します。
通常	近似する一致が得られます。バッチシステムで正確な一致が必要な場合に使用します。
ルーズ	通常のレベルに比べて差異が大きい一致が得られます。一致が見つからない可能性が高く、結果について検討できるシステムで使用します。

選択の表示

システムと検索定義を選択した後に、入力ビューと出力ビューを選択できます。

入力ビュー

入力ビュー内のフィールドによって、トランスフォーメーションの入力ポートが決定します。入力ビューには、検索および照合操作に使用するフィールドが含まれています。入力ビューを選択しない場合、トランスフォーメーションでは入力ビューの代わりに IIR Identity Table (IDT) レイアウトが使用されます。

注: システムのビュー定義に、複数のフィールドを 1 つのフィールドに連結する XFORM 句が含まれている場合は、ID 解決トランスフォーメーションのビューを使用できません。ただし、ID 解決トランスフォーメーションの外部で複数のフィールドを連結して、連結された文字列を ID 解決トランスフォーメーションの入力ポートに渡すことはできます。

出力ビュー

出力ビュー内のフィールドによって、トランスフォーメーションの出力ポートが決定します。出力ビューは、Search Server から返された検索結果の書式を設定します。出力ビューを選択しない場合、トランスフォーメーションでは出力ビューの代わりに IIR Identity Table (IDT) レイアウトが使用されます。

ID 解決トランスフォーメーションのタブ

ID 解決トランスフォーメーションには次のタブがあります。

トランスフォーメーション

トランスフォーメーションの説明を設定します。

ポート

入力ビューと出力ビューを表すグループおよびポートを表示します。グループおよびポートは編集できません。

プロパティ

以下の表に、**【プロパティ】** タブで設定できるプロパティを示します。

トランスフォーメーション属性	説明
実行時位置	DLL または共有ライブラリの格納場所。デフォルトは\$PMExtProcDir です。
トレースレベル	トランスフォーメーションのセッションログに表示される情報の詳細度。デフォルトは [Normal] です。
パーティション化可能	このトランスフォーメーションを使用するパイプラインで、複数のパーティションを作成できるかどうかを指定します。 <ul style="list-style-type: none">- いいえ。トランスフォーメーションはパーティション化できません。同一パイプライン内のこのトランスフォーメーションおよびその他のトランスフォーメーションは、1 つのパーティションに含まれる必要があります。- ローカルで。トランスフォーメーションをパーティション化することはできますが、同じノード上のパイプラインですべてのパーティションが実行される必要があります。カスタムトランスフォーメーションの別のパーティションがメモリ内のオブジェクトを共有する必要がある場合は、[ローカルで] を選択します。- グリッドをまたがる。トランスフォーメーションをパーティション化することができ、各パーティションは異なるノードに配分されます。デフォルトは [グリッドをまたがる] です。

IR の設定

このトランスフォーメーションを作成したときに行った設定を表示します。システムおよび検索情報を変更することができます。

グループおよびポート

ID 解決トランスフォーメーションには、入力グループと出力グループが含まれます。入力グループには、検索定義の入力ビュー内のフィールドを表すポートがあります。出力グループには、検索結果を表すポートに加えて、検索定義の出力ビュー内のフィールドを表すポートがあります。

グループおよびポートは、**【ポート】** タブで表示できます。ポートは **【IR の設定】** タブでも表示できます。

入力グループおよびポート

ID 解決トランスフォーメーションには、入力グループが入力ポートと共に含まれます。入力グループ内のポートの数は、選択した入力ビュー内のフィールドの数と一致します。検索に必要なすべての入力ポートを、ID 解決トランスフォーメーションにリンクします。

出力グループおよびポート

出力グループには出力ポートと入出力ポートが含まれます。出力グループには、各 ID 解決トランスフォーメーションのデフォルトポートに加えて、出力ビューのポートが含まれます。

出力ビューに基づく出力ポートには、Search Server からの候補のレコードが含まれます。

以下の表に、デフォルトの出力ポートを示します。

デフォルトの出力ポート	説明
IR_Search_Link	入出力検索リンクポート。このパススルーポートを使用して、入力側のソースデータと結果の出力候補との間のリンクを渡します。
IR_Score_Out	Search Server によって実施された検索操作からのスコアを含む出力ポート。値は 0～100 の正の数です。 例えば、入力「Rob」を含む名前を検索する場合、結果には「Rob」という名前を含むスコアが 100 の候補レコードが含まれる可能性があります。結果には「Bob」という名前を含むスコアが 90 の候補レコードも含まれる可能性があります。Informatica Identity Resolution は、結果に関連付けられたスコアを計算します。
IR_Result_Count	レコードの数を含む出力ポート。

第 12 章

Java トランスフォーメーション

この章では、以下の項目について説明します。

- [Java トランスフォーメーションの概要, 225 ページ](#)
- [\[Java コード\] タブの使用, 228 ページ](#)
- [ポートの設定, 229 ページ](#)
- [Java トランスフォーメーションプロパティの設定, 230 ページ](#)
- [Java コードの開発, 233 ページ](#)
- [Java トランスフォーメーション設定値の設定, 239 ページ](#)
- [Java トランスフォーメーションのコンパイル, 241 ページ](#)
- [コンパイルエラーの修正, 241 ページ](#)

Java トランスフォーメーションの概要

PowerCenter の機能を Java トランスフォーメーションを使用して拡張します。Java トランスフォーメーションは、単純なネイティブのプログラミングインタフェースを提供し、Java プログラミング言語を使用してトランスフォーメーション機能を定義します。Java トランスフォーメーションを使用すると、Java プログラミング言語に関する高度な知識または外部 Java 開発環境がなくても、単純またはやや高度なトランスフォーメーション機能を素早く定義することができます。Java トランスフォーメーションは、パッシブまたはアクティブなトランスフォーメーションにすることができます。

PowerCenter クライアントでは、Java Development Kit (JDK) を使用して Java コードがコンパイルされ、トランスフォーメーションのバイトコードが生成されます。PowerCenter クライアントは、バイトコードを PowerCenter リポジトリに格納します。

Integration Service は Java Runtime Environment (JRE) を使用して、生成されたバイトコードをランタイムで実行します。Integration Service が Java トランスフォーメーションを使用してセッションを実行するときに、Integration Service は JRE を使用してバイトコードを実行し、入力行を処理して出力行を生成します。

Java トランスフォーメーションを作成するには、トランスフォーメーションロジックを定義する Java コードスニペットを記述します。以下のイベントに基づいて、Java トランスフォーメーションのトランスフォーメーション動作を定義します。

- トランスフォーメーションが入力行を受け取ったとき。
- すべての入力行はトランスフォーメーションによって処理されている。
- トランスフォーメーションが、コミットまたはロールバックなどのトランザクションの通知を受け取る。

関連項目：

- [「Java トランスフォーメーション API のリファレンス」 \(ページ 244\)](#)
- [「Java 式」 \(ページ 257\)](#)

Java トランスフォーメーションを定義する手順

Java トランスフォーメーションで、Java コードの記述とコンパイル、およびコンパイル時のエラーの修正を実行するには、以下の手順を実行します。

1. Transformation Developer または Mapping Designer でトランスフォーメーションを作成します。
2. 入出力ポートを設定し、トランスフォーメーション用にグループ化します。Java コードスニペットでは、ポート名は変数として使用します。
3. トランスフォーメーションのプロパティを設定します。
4. トランスフォーメーションのコードエントリタブを使用して、トランスフォーメーションの Java コードを記述およびコンパイルします。
5. トランスフォーメーション用の Java コードのコンパイルエラーを検出および修正します。

アクティブ Java トランスフォーメーションとパッシブ Java トランスフォーメーション

Java トランスフォーメーションを作成するときは、そのタイプをアクティブまたはパッシブとして定義します。

トランスフォーメーションタイプは、設定した後で変更することはできません。

Java トランスフォーメーションは、入力データのそれぞれの行に対して 1 回、**【入力時】【入力行に達したとき】** タブで定義された Java コードを実行します。

Java トランスフォーメーションは、以下のように、トランスフォーメーションタイプに基づいて出力行を処理します。

- パッシブ Java トランスフォーメーションは、トランスフォーメーションのそれぞれの入力行を処理した後、各入力行に対して 1 つの出力行を生成します。
- アクティブ Java トランスフォーメーションは、トランスフォーメーションのそれぞれの入力行に対して複数の出力行を生成します。

各出力行を生成するには、generateRow メソッドを使用します。例えば、トランスフォーメーションに開始日と終了日を表す 2 つの入力ポートが含まれている場合は、generateRow メソッドを使用して、開始日と終了日の間の各日付に対して出力行を生成することができます。

データ型の変換

Java トランスフォーメーションは、Java トランスフォーメーションのポートタイプに基づいて、PowerCenterDeveloper ツールのデータ型を Java データ型に変換します。

Java トランスフォーメーションは、入力行を読み込むと、入力ポートデータ型を Java データ型に変換します。

Java トランスフォーメーションは、出力行を書き込むと、Java データ型を出力ポートデータ型に変換します。

例えば、Java トランスフォーメーションの整数データ型の入力ポートに対しては、以下の処理が行われます。

1. Java トランスフォーメーションは、入力ポートの整数データ型を Java プリミティブデータ型 int に変換します。

2. このトランスフォーメーションで、トランスフォーメーションは入力ポートの値を Java プリミティブデータ型 int として扱います。
3. トランスフォーメーションは、出力行を生成すると、Java プリミティブデータ型 int を整数データ型に変換します。

以下の表に、Java トランスフォーメーションが PowerCenterDeveloper ツールのデータ型を Java プリミティブデータ型および複合データ型にマッピングする方法を示します。

PowerCenter のデータ型	Java データ型
Char	String
バイナリ	byte[]
Long (INT32)	整数型
ダブル	倍精度浮動小数点数型
Decimal	倍精度浮動小数点数型 BigDecimal
BIGINT	長整数型
日付/時刻	BigDecimal long (1970/01/01 00:00:00.000 GMT 以降のミリ秒数)

Developer ツールのデータ型	Java データ型
array*	java.util.List
bigint	long
binary	byte[]
date/time	ナノ秒の処理を有効にした場合は、ナノ秒の精度の BigDecimal ナノ秒の処理を無効にした場合は、ミリ秒の精度の long (1970/01/01 00:00:00.000 GMT 以降のミリ秒数)
decimal	高精度の処理を無効にした場合は、精度が 15 の double 高精度の処理を有効にした場合は、BigDecimal
double	double
integer	整数型
map*	java.util.Map
string	ストリング
struct*	struct フィールド要素のゲッターとセッターによりカスタマイズされた JavaBean クラス

Developer ツールのデータ型	Java データ型
text	String
*Spark エンジンでのみサポートされます。	

Java では、java.util.List、java.util.Map、String、byte[]、BigDecimal の各データ型は、複合データ型になります。double、int、long の各データ型は、プリミティブデータ型になります。

Developer ツールでは、array、struct、map の各データ型は、複合データ型になります。

Java トランスフォーメーションは、プリミティブデータ型の NULL 値をゼロに設定します。**【入力行に達したとき】【入力時】** タブでは、isNull API メソッドおよび setNull API メソッドを使用して、入力ポートの NULL 値を出力ポートの NULL 値に設定できます。例については、[「setNull」 \(ページ 254\)](#)を参照してください。

注: decimal データ型は、高精度が有効なときに BigDecimal にマッピングされます。BigDecimal は、+などの演算子と一緒に使用できません。decimal ポートを使用する式が Java コードに含まれていて、そのポートがいくつかの演算子と一緒に使用された場合、Java コードはコンパイルに失敗します。

注: decimal データ型は、高精度が有効なときに BigDecimal にマッピングされます。BigDecimal は、+などの演算子と一緒に使用できません。decimal ポートまたは decimal データ型の要素を持つ複合ポートを使用する式が Java コードに含まれていて、そのポートがいくつかの演算子と一緒に使用された場合、Java コードはコンパイルに失敗します。

[Java コード] タブの使用

Java コードの定義、コンパイル、およびコンパイルエラーの修正には、[Java コード] タブを使用します。コードスニペットは、コードエントリタブで作成します。

Java コードを定義する場合、以下のタスクを実行できます。

- 静的コード、静的ブロック、インスタンス変数、およびユーザー定義メソッドを定義する。
- Java 式を定義し、トランスフォーメーションロジックを定義する。
- Java トランスフォーメーション API メソッドおよび標準の Java 言語の構文を使用する。
- サードパーティ製の Java API、Java の組み込みパッケージ、または Java のカスタムパッケージをインポートする。Java パッケージの Java コードスニペットを使用できます。

コードスニペットの作成後は、Java コードをコンパイルして、出力ウィンドウでコンパイルの結果を表示することも、Java コード全体を表示することもできます。

[Java コード] タブには、以下のコンポーネントがあります。

- **ナビゲータ**。コードスニペットに、入力ポート、出力ポート、または API を追加します。ナビゲータには、トランスフォーメーションの入力ポートおよび出力ポート、使用できる Java トランスフォーメーション API、およびポートと API 関数の説明の一覧が表示されます。入力ポートおよび出力ポートの説明には、ポート名、タイプ、データタイプ、精度、および位取りが含まれています。API 関数の説明には、構文と API 関数の使用方法が含まれています。

ナビゲータは、そのコードエントリタブでは使用できないポートまたは API 関数を無効化します。たとえば [パッケージのインポート] コードエントリタブから、ポートの追加または API 関数の呼び出しは実行できません。

- **コードウィンドウ。**トランスフォーメーション用の Java コードを作成します。コードウィンドウでは、基本的な Java 構文が強調表示されます。
- **コードエントリタブ。**トランスフォーメーションの動作を定義します。それぞれのコードエントリタブには、関連するコードウィンドウがあります。コードエントリタブに Java コードを入力するには、タブをクリックし、コードウィンドウに Java コードを書き込みます。
- **式の定義リンク。**Java 式の作成に使用する [式の定義] ダイアログボックスを開きます。
- **設定リンク。**[設定] ダイアログボックスが開きます。[設定] ダイアログボックスを使用して、サードパーティ製 Java パッケージおよびカスタム Java パッケージのクラスパスの設定、Decimal データタイプでの高精度 10 進演算の有効化、サブ秒データの処理を行います。PowerCenter クライアントは、Java コードをコンパイルするときにクラスパス内のファイルをインクルードします。
- **コンパイルリンク。**トランスフォーメーション用の Java コードをコンパイルします。Java コンパイラからの出力（エラーメッセージおよび情報メッセージを含む）は、出力ウィンドウに表示されます。
- **コード全体リンク。**コード全体ウィンドウを開くと、Java トランスフォーメーションの完全なクラスコードが表示されます。トランスフォーメーションの完全なコードには、Java トランスフォーメーションクラスのテンプレートに追加されたコードエントリタブの Java コードも含まれます。
- **出力ウィンドウ。**Java トランスフォーメーションクラスのコンパイルの結果を表示します。出力ウィンドウに表示されたエラーメッセージを右クリックすると、Java トランスフォーメーションクラスについて、スニペットコードまたはコード全体でのエラー箇所をコード全体ウィンドウで特定できます。出力ウィンドウでエラーをダブルクリックすると、エラーのソースを検出できます。

ポートの設定

Java トランスフォーメーションには、入力ポート、出力ポート、および入出力ポートがあります。[ポート] タブでは、グループとポートの作成および編集を実行します。ポートのデフォルト値を指定できます。トランスフォーメーションにポートを追加すると、ポート名は Java コードスニペット内で変数として使用できます。

グループとポートの作成

作成した Java トランスフォーメーションには、入力グループおよび出力グループが 1 つずつ含まれています。

Java トランスフォーメーションには、必ず入力グループおよび出力グループが 1 つずつあります。複数の入力グループまたは出力グループを持つトランスフォーメーションは無効です。既存のグループ名を変更するには、グループヘッダを入力します。グループを削除した場合、[入力グループの作成] アイコンまたは [出力グループの作成] アイコンをクリックして新規グループを追加できます。

ポートを作成すると、DesignerDeveloper ツールはそのポートを現在選択されている行またはグループの下に追加します。入力グループの下に表示される入出力ポートは、出力グループにも属します。出力グループの下に表示される入出力ポートは、入力グループにも属します。

デフォルトポート値の設定

Java トランスフォーメーションでは、ポートのデフォルト値を定義できます。

Java トランスフォーメーションは、ポートのデータ型に基づいて、ポートのデフォルト値を適用してポート変数を初期化します。

入出力ポート

Java トランスフォーメーションは、Java コードスニペットに値が割り当てられていない未接続の n 入力または出力ポートの値を初期化します。

Java トランスフォーメーションでは、次の Java データ型に基づいて、ポートを初期化します。

プリミティブデータ型

ポートのデフォルト値を NULL 以外の値に定義した場合、トランスフォーメーションはポート変数の値をそのデフォルト値に初期化します。それ以外の場合、ポート変数の値は 0 に初期化されます。

複合データ型

ポートのデフォルト値を定義した場合、トランスフォーメーションは新規のオブジェクトを作成し、そのオブジェクトをデフォルト値に初期化します。それ以外の場合、トランスフォーメーションはポート変数を NULL に初期化します。例えば、String ポートのデフォルト値を定義した場合、トランスフォーメーションは新規の String オブジェクトを作成し、その String オブジェクトをデフォルト値に初期化します。

注: Java コードで値が NULL の入力ポート変数にアクセスすると、NullPointerException が発生します。

入力ポートをパーティションキーおよびソートキーとして有効化し、ソート方向を割り当てることができます。データ統合サービスは、データをパーティション化し、ソートキーとソート方向によって各パーティション内のデータをソートします。トランスフォーメーションの範囲が [すべての入力] に設定されると、パーティションキーとソートキーは有効になります。

データのパーティション化およびソートには以下のプロパティを使用します。

パーティションキー

1 つのパーティションにグループ化するデータの行数を決定する入力ポート。

ソートキー

各パーティション内のソート基準を決定する入力ポート。

方向

昇順または降順。デフォルトは昇順です。

入出力ポート

Java トランスフォーメーションは、入出力ポートをパススルーポートとして処理します。トランスフォーメーションの Java コードにポートの値を設定しない場合、出力値と入力値は同じになります。Java トランスフォーメーションは、入出力ポートの値を入力ポートの初期化と同じ方法で初期化します。

Java コードに入出力ポートのポート変数の値を設定した場合、Java トランスフォーメーションはこの値を使用して出力行を生成します。入出力ポートの値を設定しない場合、Java トランスフォーメーションは単純データタイプのポート変数の値を 0 に、複合データタイプのポート変数の値を NULL に設定して出力行を生成します。

Java トランスフォーメーションプロパティの設定

Java トランスフォーメーションには、トランスフォーメーションコードおよびトランスフォーメーションのプロパティが両方とも含まれています。Transformation Developer で Java トランスフォーメーションを作成した場合、マッピングでトランスフォーメーションを使用する際にトランスフォーメーションプロパティを上書きできます。

以下の表では、Java トランスフォーメーションのプロパティについて説明します。

プロパティ	説明
言語	トランスフォーメーションコードで使用される言語。ユーザーはこの値を変更することができません。
クラス名	トランスフォーメーション用 Java クラスの名前。ユーザーはこの値を変更することができません。
トレースレベル	トランスフォーメーションのセッションログに表示される情報の詳細度。以下のトレースレベルを使用します。 <ul style="list-style-type: none"> - Terse - ノーマル - Verbose Initialization - Verbose Data デフォルトは [Normal] です。
パーティション化可能	パイプラインに複数のパーティションがあると、このトランスフォーメーションを使用できます。次のオプションを使用します。 <ul style="list-style-type: none"> - いいえ。トランスフォーメーションはパーティション化できません。同一パイプライン内のこのトランスフォーメーションおよびその他のトランスフォーメーションは、1 つのパーティションに含まれる必要があります。データクレンジングなど、トランスフォーメーションによりすべての入力データが一度に処理される場合は、[いいえ] を選択する場合があります。 - ローカルで。トランスフォーメーションをパーティション化することはできませんが、同じノード上のパイプラインですべてのパーティションが実行される必要があります。トランスフォーメーションの異なるパーティションがメモリ内でオブジェクトを共有する必要がある場合、[ローカルで] を選択します。 - グリッドをまたがる。トランスフォーメーションをパーティション化することができ、各パーティションは異なるノードに配分されます。 デフォルトは [No]。
入力はブロック	トランスフォーメーションに関連するプロシージャでは、入力データをブロックすることが必要です。デフォルトでは有効になっています。
アクティブ	トランスフォーメーションは、それぞれの入力行に対して複数の出力行を生成できます。 Java トランスフォーメーションを作成した後は、このプロパティを変更できません。このプロパティを変更する必要がある場合、新しい Java トランスフォーメーションを作成します。
アップデートストラテジトランスフォーメーション	このトランスフォーメーションは、出力行のアップデートストラテジを定義します。アクティブな Java トランスフォーメーションでは、このプロパティを有効にすることができます。 デフォルトでは無効になっています。
トランスフォーメーション範囲	Integration Service が入力データに対してトランスフォーメーションロジックを適用する方法です。次のオプションを使用します。 <ul style="list-style-type: none"> - 行 - トランザクション - すべての入力 パッシブトランスフォーメーションに対しては、このプロパティは必ず [Row] に設定します。アクティブトランスフォーメーションに対しては、デフォルトは [すべての入力] です。

プロパティ	説明
トランザクションの生成	このトランスフォーメーションは、トランザクション行を生成します。アクティブな Java トランスフォーメーションでは、このプロパティを有効にすることができます。 デフォルトでは無効になっています。
出力が再現可能	出力データの順序はセッションの実行ごとに一致しています。 <ul style="list-style-type: none"> - Never。出力データの順序はセッションの実行ごとに異なります。 - 入力順による。入力データの順序がセッションの実行ごとに一致している場合、出力順序をセッションの実行ごとに一致させます。 - Always。入力データの順序がセッションの実行ごとに異なる場合でも、出力データの順序は常に同じです。 アクティブなトランスフォーメーションの場合、デフォルトは [Never] です。パッシブなトランスフォーメーションの場合、デフォルトは [入力順による] です。
パーティションごとに 1 つのスレッドを要求します	1 つのスレッドが、それぞれのパーティションに対するデータを処理します。 ユーザーはこの値を変更することができません。
出力が確定的かどうか	トランスフォーメーションは、セッションを実行するたびに一致する出力データを生成します。このトランスフォーメーションを使用するセッションでリカバリを実行するには、このプロパティを有効にします。 デフォルトでは有効になっています。

警告: トランスフォーメーションを繰り返し可能で一意に定まるものとして設定する場合は、データが繰り返し可能で一意に定まることを保証する必要があります。セッションとリカバリで同じデータが生成されないトランスフォーメーションを使用してセッションをリカバリしようとする、リカバリプロセスを実行した結果、データが破損する可能性があります。

トランザクション制御に関する作業

Java トランスフォーメーションのトランザクション制御は、以下のプロパティを使って制御できます。

- **トランスフォーメーション範囲。** Integration Service が入力データにトランスフォーメーションロジックを適用する方法を指定します。
- **トランザクションの生成。** トランスフォーメーション用の Java コードがトランザクション行を生成し、出力グループに渡すことを示しています。

トランスフォーメーション範囲

Integration Service が入力データにトランスフォーメーションロジックを適用する方法を設定できます。以下の値のいずれかを選択できます。

- **行。** トランスフォーメーションロジックを一度に 1 行のデータに適用します。トランスフォーメーションの結果がデータの単一の行に依存する場合は、[Row] を選択してください。パッシブトランスフォーメーションの場合、必ず [Row] を選択することが必要です。
- **トランザクション。** トランスフォーメーションロジックをトランザクションのすべての行に適用します。トランスフォーメーションの結果が同一トランザクションのすべての行に依存し、他のトランザクションの行には依存していない場合には、[Transaction] を選択します。たとえば、Java コードが単一のトランザクションのデータに対して集計計算を実行する場合に、[Transaction] を選択します。

- **すべての入力。**トランスフォーメーションロジックをすべての入力データに適用します。[すべての入力]を選択すると、Integration Service はトランザクションの境界を削除します。[すべての入力] は、トランスフォーメーションの結果がソース内のデータのすべての行に依存する場合に選択します。たとえば、トランスフォーメーションの Java コードがすべての入力データをソートする場合、[すべての入力] を選択します。

トランザクションの生成

アクティブな Java トランスフォーメーションで Java コードを定義すると、コミット行やロールバック行などのトランザクション行を生成できます。トランザクション行を生成するには、トランスフォーメーションでトランザクション行の生成を有効にします。

トランザクション行を生成するようにトランスフォーメーションを設定すると、Integration Service は Java トランスフォーメーションをトランザクション制御トランスフォーメーションと同様に扱います。マッピング内でトランザクション制御トランスフォーメーションに適用されるルールのほとんどは、Java トランスフォーメーションにも適用されます。たとえば、トランザクション行を生成するように Java トランスフォーメーションを設定した場合、そのトランスフォーメーションを含むパイプラインまたはパイプラインブランチは連結できません。

トランザクション行を生成するように設定された Java トランスフォーメーションを使用するセッションを編集または作成する場合、ユーザー定義コミット用に設定してください。

関連項目：

- [「コミット」 \(ページ 245\)](#)
- [「rollBack」 \(ページ 253\)](#)

アップデートストラテジの設定

マッピングの更新方式を設定するには、アクティブな Java トランスフォーメーションを使用します。更新方式は、以下のレベルで設定できます。

- **Java コード内。**出力行のアップデートストラテジを設定する Java コードを記述できます。Java コードは、挿入、更新、削除、またはリジェクトのフラグを行に設定できます。更新戦略の設定の詳細については、[「setOutRowType」 \(ページ 254\)](#)を参照してください。
- **マッピング内。**挿入、更新、削除、または拒否のフラグを行に設定する場合は、マッピング内の Java トランスフォーメーションを使用します。Java トランスフォーメーション用のアップデートストラテジトランスフォーメーションプロパティを選択します。
- **セッション内。**ソース行をデータドリブンとして扱うようにセッションを設定します。

更新方式を定義するように Java トランスフォーメーションを設定しない場合、またはセッションをデータドリブンとして設定しない場合、Integration Service は出力行にフラグを設定するために Java コードを使用することはありません。その代わりに、Integration Service が出力行に挿入のフラグを設定します。

Java コードの開発

Java トランスフォーメーションの機能を定義する Java コードスニペットを入力するには、コードエントリタブを使用します。コードエントリタブでは、Java パッケージのインポート、Helper コードの記述、Java 式の定義、および特定のトランスフォーメーションイベントにおけるトランスフォーメーションの動作を定義する Java コードの記述を実現するための Java コードを記述できます。コードエントリタブでは、任意の順序でスニペットを開発できます。

Java コードは、以下のコードエントリタブに入力できます。

- **パッケージのインポート**。サードパーティ製の Java パッケージ、ビルトイン Java パッケージ、またはカスタム Java パッケージをインポートします。
- **Helper コード**。[パッケージのインポート] タブ以外のすべてのタブで使用できる変数およびメソッドを定義します。
- **入力行に達したとき**。入力行を受け取ったときのトランスフォーメーションの動作を定義します。
- **データの終わりに達したとき**。すべての入力データを処理したときのトランスフォーメーションの動作を定義します。
- **トランザクションを受け取ったとき**。トランザクション通知を受け取ったときのトランスフォーメーションの動作を定義します。アクティブな Java トランスフォーメーションで使います。
- **Java 式**。PowerCenter の式を呼び出す Java 式を定義します。Java 式は、[Helper コード]、[入力行に達したとき]、[データの終わりに達したとき]、および [On Transaction] のコードエントリタブで使用できます。

入力データへのアクセスおよび出力データの設定は、[入力行に達したとき] タブで実行します。アクティブなトランスフォーメーションの場合、[データの終わりに達したとき] タブおよび [トランザクションを受け取ったとき] タブでも、出力データを設定できます。

Java コードスニペットの作成

Java コードスニペットを作成してトランスフォーメーションの動作を定義するには、[Java コード] コードエントリタブの [コード] [Java コード] ウィンドウを使います。

1. 適切なコードエントリタブをクリックします。

以下の表に、[Java] ビューのコードエントリタブで完了できるタスクを示します。

タブ	説明
インポート	アクティブまたはパッシブな Java トランスフォーメーションに対して、サードパーティ製、組み込み、またはカスタムの Java パッケージをインポートします。パッケージのインポート後、それらのパッケージを他のコードエントリタブで使えます。
ヘルパ	アクティブまたはパッシブな Java トランスフォーメーション内の Java トランスフォーメーションクラスのユーザー定義変数およびメソッドを宣言します。変数およびメソッドを宣言すると、それらを [インポート] タブを除く他のすべてのコードエントリタブで使えます。
入力時	入力行を受け取った際のアクティブまたはパッシブな Java トランスフォーメーションの動作を定義します。このタブで定義した Java コードは、入力行ごとに 1 回実行されます。このタブでは、入出力ポートのデータ、変数、および Java トランスフォーメーション API メソッドにアクセスして使することもできます。
最後	すべての入力データを処理した後のアクティブまたはパッシブな Java トランスフォーメーションの動作を定義します。このタブでは、アクティブなトランスフォーメーションの出力データの設定や、Java トランスフォーメーション API メソッドの呼び出しも行うことができます。

タブ	説明
関数	Java トランスフォーメーションの式を呼び出す関数を、Java プログラミング言語を使用して定義します。例えば、入出力ポートの値や Java トランスフォーメーションの変数の値をルックアップする式を呼び出す関数を定義できます。 【関数】タブで、手動で関数を定義するか、または【新しい関数】をクリックして関数を簡単に定義できる【関数の定義】ダイアログボックスを呼び出します。
最適化インターフェース	初期選択または最適化にプッシュインを定義します。ナビゲータで、最適化方式を選択します。最適化を有効にするには、コードスニペットを更新します。入力ポートおよびフィルタロジックをプッシュする関連付けられた出力ポートを定義します。
コード全体	読み取り専用。このタブでは、Java トランスフォーメーションのクラスコード全体を表示し、コンパイルすることができます。

2. スニペット内の入力または出力カラムの変数にアクセスするには、ナビゲータに表示されたポートの名前をダブルクリックします。ナビゲータに表示された【入力】または【出力】リストを展開し、ポートの名前をダブルクリックします。
3. スニペット内の Java トランスフォーメーション API を呼び出すには、ナビゲータに表示された API の名前をダブルクリックします。必要に応じて、適切な API 入力値を設定してください。ナビゲータに表示された【呼び出し可能な API】リストを展開し、メソッドの名前をダブルクリックします。必要に応じて、メソッドに適切な入力値を設定してください。
4. コードスニペットコードエントリタブのタイプに基づいて、適切な Java コードを書き込みます。
【コード全体】タブの【コード全体】【Java コード】ウィンドウで、Java トランスフォーメーションのクラスコード全体を表示します。

Java パッケージのインポート

【パッケージのインポート】【インポート】タブでは、アクティブまたはパッシブな Java トランスフォーメーションに対して Java パッケージをインポートできます。

サードパーティ製、組み込み、またはカスタムの Java パッケージをインポートできます。Java パッケージのインポート後、インポートされたパッケージを他のコードエントリタブで使用できます。

注: 【パッケージのインポート】【インポート】タブでは、静的変数、インスタンス変数、またはユーザーメソッドの宣言または使用はできません。

PowerCenter ClientDeveloper ツールで、Java トランスフォーメーションを含むメタデータをエクスポートまたはインポートしても、Java トランスフォーメーションが必要とするサードパーティ製またはカスタムのパッケージが格納されている JAR ファイルやクラスファイルはエクスポートまたはインポートされません。

Java トランスフォーメーションを含むメタデータをインポートする場合は、必要なサードパーティ製またはカスタムのパッケージが格納された JAR ファイルまたはクラスファイルを、PowerCenter クライアントおよび Integration Service ノード Developer ツールクライアントおよび Data Integration Service ノードにコピーする必要があります。

例えば Java I/O パッケージをインポートするには、【パッケージのインポート】【インポート】タブに以下のコードを入力します。

```
import java.io.*;
```

標準以外の Java パッケージをインポートするときは、Java トランスフォーメーションの設定でクラスパスにパッケージまたはクラスを追加します。

関連項目：

- [「Java トランスフォーメーション設定値の設定」 \(ページ 239\)](#)

Helper コードの定義

[Helper コード] [ヘルパ] タブで、アクティブ Java トランスフォーメーションまたはパッシブ Java トランスフォーメーション内の Java トランスフォーメーションクラスのユーザー定義変数およびメソッドを宣言できます。

[Helper コード] [ヘルパ] タブで変数およびメソッドを宣言すると、**[パッケージのインポート] [インポート]** タブを除くすべてのコードエントリタブでその変数およびメソッドを使用できます。

[Helper コード] [ヘルパ] タブでは、以下のタイプのコード、変数、およびメソッドを宣言できます。

- 静的コードおよび静的変数。

静的ブロック内では、静的変数および静的コードを宣言できます。マッピング内の再利用可能な Java トランスフォーメーションのすべてのインスタンスおよびセッション内のすべてのパーティションが、静的コードおよび静的変数を共有します。静的コードは、Java トランスフォーメーション内の他のどのコードよりも先に実行されます。

例えば、以下のコードは、マッピング内の Java トランスフォーメーションのすべてのインスタンスに対するエラーしきい値を保存する静的変数を宣言します。

```
static int errorThreshold;
```

この変数を使用してトランスフォーメーションのエラーしきい値を保存すると、そのエラーしきい値にマッピング内の Java トランスフォーメーションの全インスタンスおよびセッション内の任意のパーティションからアクセスできます。

注: 複数のパーティションがあるセッションまたは再利用可能な Java トランスフォーメーションでは、静的変数を同期する必要があります。

- インスタンス変数。

パーティションレベルのインスタンス変数を宣言できます。マッピング内の再利用可能な Java トランスフォーメーションの複数のインスタンスまたはセッション内の複数のパーティションは、インスタンス変数を共有しません。重複を避けるためにプレフィックスを追加してインスタンス変数を宣言し、非プリミティブインスタンス変数を初期化します。

たとえば、以下のコードを使用すると、ブール変数を使用して出力行を生成するかどうかを決定できます。

```
// boolean to decide whether to generate an output row
// based on validity of input
private boolean generateRow;
```

- ユーザー定義の静的メソッドおよびインスタンスメソッド

Java トランスフォーメーションの機能を拡張します。**[Helper コード] [ヘルパ]** タブで宣言された Java メソッドでは、出力変数またはローカルで宣言されたインスタンス変数を使用または変更できます。

[Helper コード] [ヘルパ] タブの Java メソッドからは、入力変数にアクセスできません。

例えば、**[Helper コード] [ヘルパ]** タブの以下のコードを使用して 2 つの整数を追加する関数を宣言します。

```
private int myTXAdd (int num1,int num2)
{
    return num1+num2;
}
```

[入力行に達したとき] タブ

[入力行に達したとき] タブを使用して、Java トランスフォーメーションが入力行を受け取った場合の動作を定義します。このタブの Java コードは、各入力行に対して 1 回ずつ実行されます。[入力行に達したとき] タブの入力行データにのみアクセスできます。

以下の入出力ポートのデータ、変数、およびメソッドは、[入力行に達したとき] タブからアクセスして使用します。

- **入出力ポート変数。**入出力ポートのデータを変数としてアクセスするには、ポートの名前を変数の名前として使用します。たとえば「in_int」が整数の入力ポートである場合、Java 基本データタイプ int で「in_int」変数として参照することで、このポートのデータにアクセスできます。入力ポートおよび出力ポートを変数として宣言する必要はありません。

入力ポート変数に値は割り当てないでください。[入力行に達したとき] タブの入力変数に値を割り当てると、対応するポートの入力データを現在の行では取得できません。

- **インスタンス変数およびユーザー定義メソッド。** [Helper コード] タブで宣言した任意のインスタンス変数、静的変数、またはユーザー定義メソッドを使用します。

たとえば、アクティブ Java トランスフォーメーションに、整数データタイプの 2 つの入力ポート (BASE_SALARY と BONUSSES)、および整数データタイプの 1 つの出力ポート (TOTAL_COMP) があるとします。また [Helper コード] タブで、2 つの整数を加算して結果を返すユーザー定義メソッド (myTXAdd) を作成したとします。この場合、[入力行に達したとき] タブで以下の Java コードを使用し、入力ポートの合計値を出力ポートに割り当てて出力行を生成します。

```
TOTAL_COMP = myTXAdd (BASE_SALARY,BONUSSES);  
generateRow();
```

Java トランスフォーメーションは、入力行を受け取ると 2 つの入力ポート (BASE_SALARY および BONUSSES) の値を加算した値を出力ポート (TOTAL_COMP) に割り当て、出力行を生成します。

- **Java トランスフォーメーション API メソッド。**Java トランスフォーメーションによって提供される API メソッドを呼び出すことができます。

[データの終わりに達したとき] タブ

アクティブな Java トランスフォーメーションまたはパッシブな Java トランスフォーメーションがすべての入力データを処理した場合の Java トランスフォーメーションの動作を定義するには、[データの終わりに達したとき] タブを使用します。[データの終わりに達したとき] タブで出力行を生成するには、トランスフォーメーションのトランスフォーメーション範囲を [トランザクション] または [すべての入力] に設定します。このタブでは、入力ポートの変数にアクセスしたり、値を設定することはできません。

以下の変数およびメソッドは、[データの終わりに達したとき] タブからアクセスして使用します。

- **出力ポート変数。**アクティブな Java トランスフォーメーションで出力データへのアクセスまたは設定を実行する場合、出力ポート名を変数として使用します。
- **インスタンス変数およびユーザー定義メソッド。** [Helper コード] タブで宣言した任意のインスタンス変数またはユーザー定義メソッドを使用します。
- **Java トランスフォーメーション API メソッド。**Java トランスフォーメーションによって提供される API メソッドを呼び出します。commit および rollBackAPI メソッドを使用して、トランザクションを生成します。

たとえば、以下の Java コードを使用して、データの終わりに達したときにセッションログに情報を書き込みます。

```
logInfo("Number of null rows for partition is: " + partCountNullRows);
```

[トランザクションを受け取ったとき] タブ

アクティブな Java トランスフォーメーションでは、[トランザクションを受け取ったとき] タブを使用してアクティブ Java トランスフォーメーションがトランザクション通知を受け取った場合の動作を定義します。[トランザクションを受け取ったとき] タブのコードスニペットは、トランスフォーメーションのトランザクションスコープが [トランザクション] に設定されている場合に限り実行されます。このタブでは、入力ポートの変数にアクセスしたり、値を設定することはできません。

[トランザクションを受け取ったとき] タブでは、以下の出力データ、変数、およびメソッドにアクセスして使用できます。

- **出力ポート変数。**出力データへのアクセスまたは設定を実行する場合、出力ポート名を変数として使用します。
- **インスタンス変数およびユーザー定義メソッド。**[Helper コード] タブで宣言した任意のインスタンス変数またはユーザー定義メソッドを使用します。
- **Java トランスフォーメーション API メソッド。**Java トランスフォーメーションによって提供される API メソッドを呼び出します。commit および rollBackAPI メソッドを使用して、トランザクションを生成します。たとえば以下の Java コードを使用すると、トランスフォーメーションがトランザクションを受け取った後でトランザクションが生成されます。

```
commit();
```

Java コードによるフラットファイルの解析

フラットファイルを解析する Java コードを作成できます。スキーマがさまざまに異なるフラットファイルまたは JMS メッセージから特定のデータカラムを抽出するには、Java コードを使用します。

たとえば、区切りフラットファイルから先頭 2 つのデータカラムを読み込むものとします。区切りフラットファイルからデータを読み込んで、1 つ以上の出力ポートに渡すマッピングを作成します。

マッピングは、以下のようなコンポーネントから構成されています。

- **ソース定義。**ソースは区切りフラットファイルです。フラットファイルの行を文字列として Java トランスフォーメーションに渡すようにソースを設定します。ソースファイルには、以下のデータがあります。

```
1a,2a,3a,4a,5a,6a,7a,8a,9a,10a
1b,2b,3b,4b,5b,6b,7b,8b,9b
1c,2c,3c,4c,5c,6c,7c
1d,2d,3d,4d,5d,6d,7d,8d,9d,10d
```

- **Java トランスフォーメーション。**[Java コード] タブで Java トランスフォーメーションの機能を定義します。

java パッケージをインポートするには、[Java コード] タブの [パッケージのインポート] タブを使用します。[パッケージのインポート] タブで以下のコードを入力します。

```
import java.util.StringTokenizer;
```

文字列から各データ行を読み込んで出力ポートに渡すには、[Java コード] タブの [入力行に達したとき] タブを使用します。先頭 2 つのデータカラムを取得するには、[入力行に達したとき] タブに以下のコードを入力します。

```
StringTokenizer st1 = new StringTokenizer(row, ",");
f1 = st1.nextToken();
f11= st1.nextToken();
```

- **ターゲット定義。**Java トランスフォーメーションからデータ行を受け取るようにターゲットを設定します。マッピングが含まれているワークフローを実行した後、ターゲットにはデータカラムが 2 つ含まれます。ターゲットファイルには、以下のデータがあります。

```
1a,2a
1b,2b
1c,2c
1d,2d
```


Java トランスフォーメーション設定値の設定

Java トランスフォーメーションは、サードパーティ製 Java パッケージおよびカスタム Java パッケージのクラスパスを設定し、Decimal データタイプで高精度 10 進演算を有効化するように設定することができます。また、トランスフォーメーションは、サブ秒データが処理されるように設定することもできます。

クラスパスの設定

[パッケージのインポート] タブで標準以外の Java パッケージをインポートするときは、PowerCenter クライアントおよび Integration Service のクラスパスを、Java パッケージに関連付けられた各 JAR ファイルまたはクラスファイルのディレクトリに設定します。UNIX の場合、クラスパスの各項目を区切るにはコロンのを使用します。Windows の場合、クラスパスの各項目を区切るにはセミコロンを使用します。JAR ファイルまたはクラスファイルは、PowerCenter クライアントノードおよび Integration Service ノード上でアクセス可能でなければなりません。

たとえば、[パッケージのインポート] タブに Java パッケージである converter をインポートし、このパッケージを converter.jar で定義したとします。この場合、Java トランスフォーメーションの Java コードをコンパイルする前に、必ず converter.jar をクラスパスに追加する必要があります。

ビルトイン Java パッケージの場合、クラスパスを設定する必要はありません。たとえば java.io はビルトイン Java パッケージです。java.io をインポートした場合、java.io 用にクラスパスを設定する必要はありません。

Integration Service のクラスパスを設定します。

Integration Service のクラスパスに JAR ファイルまたはクラスファイルのディレクトリを追加できます。Integration Service ノード上でクラスパスを設定するには、次の作業のいずれかを完了します。

- **Java クラスパスセッションプロパティを設定します。** [Java クラスパス] セッションプロパティを使用して、クラスパスを設定します。このクラスパスは、セッションに適用されます。
- **Java SDK クラスパスを設定します。** Java SDK クラスパスは、Informatica Administrator の Integration Service プロパティの [プロセス] タブで設定します。この設定値は、Integration Service 上で実行されているセッションすべてに適用されます。
- **CLASSPATH 環境変数を設定します。** CLASSPATH 環境変数は Integration Service ノード上で設定します。環境変数の設定後、Integration Servicewo を再起動します。これは、Integration Service 上で実行されているセッションすべてに適用されます。

UNIX 上での Integration Service のクラスパスの設定

UNIX で CLASSPATH 環境変数を設定するには：

- ▶ UNIX の C シェル環境では、次のように入力します。
setenv CLASSPATH <classpath>
UNIX の Bourne シェル環境では、次のように入力します。
CLASSPATH = <classpath>
export CLASSPATH

Windows 上での Integration Service のクラスパスの設定

Windows で CLASSPATH 環境変数を設定するには：

- ▶ 環境変数 CLASSPATH を入力し、値をデフォルトのクラスパスに設定します。

PowerCenterDeveloper ツールクライアントのクラスパスの設定

PowerCenter ClientDeveloper ツールクライアントのクラスパスに jar ファイルまたはクラスファイルのディレクトリを追加できます。

PowerCenter ClientDeveloper ツールクライアントが動作しているマシン用にクラスパスを設定するには、次の作業のいずれかを実行します。

- CLASSPATH 環境変数を設定します。CLASSPATH 環境変数は PowerCenter ClientDeveloper ツールクライアントマシン上で設定します。これは、マシン上で実行されている java プロセスすべてに適用されます。
- 再利用不可能な Java トランスフォーメーションの場合は、Java トランスフォーメーション設定の詳細プロパティでクラスパスを設定します。これは、この Java トランスフォーメーションを含むセッションマッピングに適用されます。PowerCenter ClientDeveloper tool クライアントは、Java コードをコンパイルするときにクラスパス内のファイルをインクルードします。

jar ファイルまたはクラスファイルのディレクトリを Java トランスフォーメーションのクラスパスに追加するには、以下の手順を実行します。

1. **【Java コード】【詳細】** タブで、**【設定】** リンクをクリックします **【クラスパス】** の横にある **【値】** カラムの下矢印アイコンをクリックします。
【設定】【クラスパスの編集】 ダイアログボックスが表示されます。
2. クラスパスを追加するには、以下の手順を実行します。
 - a. **【追加】** をクリックします。
【名前を付けて保存】 ウィンドウが表示されます。
 - b. **【名前を付けて保存】** ウィンドウで、jar ファイルの保存先のディレクトリに移動します。
 - c. **【OK】** をクリックします。
【クラスパスの編集】 ダイアログボックスにクラスパスが表示されます。
3. **【クラスパスの追加】** で **【参照】** をクリックし、インポートしたパッケージの jar ファイルまたはクラスファイルのディレクトリを選択します。 **【OK】** をクリックします。
4. **【追加】** をクリックします。
jar ファイルまたはクラスファイルのディレクトリが、トランスフォーメーションに関する jar ファイルおよびクラスファイルのディレクトリの一覧に表示されます。
5. jar ファイルまたはクラスファイルのディレクトリを削除するには、jar ファイルまたはクラスファイルのディレクトリを選択して **【削除】** をクリックします。
ディレクトリの一覧からディレクトリが削除されます。

高精度 10 進演算の有効化

デフォルトでは、Java トランスフォーメーションは Decimal タイプのポートを Double データタイプ（精度 15）に変換します。精度が 15 を超える Decimal データタイプを処理する場合は、**【高精度 10 進演算を有効にする】** を選択して、Decimal ポートを Java の BigDecimal クラスを使用して処理します。

高精度 10 進演算を有効化すると、Decimal ポートを BigDecimal として 28 までの精度で処理できます。Java トランスフォーメーションは、精度が 28 より大きい Decimal データを Double データタイプに変換します。Java トランスフォーメーション式で処理されるデータは、バイナリ、長整数、および文字列です。bigint データは Java トランスフォーメーション式では処理できません。

たとえば、Decimal タイプの入力ポートを持つ Java トランスフォーメーションがあるとします。このトランスフォーメーションは、40012030304957666903 の値を受け取ります。高精度 10 進演算を有効化している場合、このポートの値は表示されたとおりに扱われます。高精度 10 進演算が有効化されていない場合、このポートの値は $4.00120303049577 \times 10^{19}$ になります。

注: Java トランスフォーメーションの高精度 10 進演算を有効にすると、その Java トランスフォーメーションを含むセッションの高精度 10 進演算も有効になります。Java トランスフォーメーションで高精度 10 進演算が有効でも、それがセッションで有効になっていない場合、そのセッションは失敗し、次のエラーが表示される可能性があります。

[ERROR]Failed to bind column with index <index number> to datatype <datatype name>.

サブ秒の処理

Java コードでは、1 秒以下のデータをナノ秒単位まで処理できます。日時の値にナノ秒を使用するよう設定すると、生成された Java コードにより、トランスフォーメーションの Date/Time データタイプが、ナノ秒単位の精度を持つ Java BigDecimal データタイプに変換されます。

デフォルトでは、生成された Java コードによって、トランスフォーメーションの Date/Time データタイプが、ミリ秒単位の精度を持つ Java Long データタイプに変換されます。

Java トランスフォーメーションのコンパイル

PowerCenter Client の Developer ツールでは、Java コンパイラを使用して Java コードをコンパイルしてトランスフォーメーション用のバイトコードを生成します。

Java コンパイラは、Java コードをコンパイルし、コンパイルの結果を【出力】ウィンドウの【結果】ウィンドウのコードエントリタブの【コンパイル】プロパティに表示します。Java コンパイラは、PowerCenter Client の Developer ツールと一緒に java/bin ディレクトリにインストールされます。

Java トランスフォーメーションのコード全体をコンパイルするには、[Java コード] [コード全体] タブの【コンパイル】プロパティの【コンパイル】をクリックします。

作成した Java トランスフォーメーションには、Java トランスフォーメーションの基本的な機能を定義する Java クラスが含まれています。Java クラスのコード全体には、トランスフォーメーションのテンプレートクラスコードに加えて、コードエントリタブで定義した Java コードが格納されています。

Java トランスフォーメーションをコンパイルすると、PowerCenter Client の Developer ツールはコードエントリタブのコードをトランスフォーメーションのテンプレートクラスに追加し、トランスフォーメーションのクラスコード全体を生成します。その後、PowerCenter クライアントデベロッパツールは Java コンパイラを呼び出してクラスコード全体をコンパイルします。Java コンパイラは、トランスフォーメーションをコンパイルし、トランスフォーメーションのバイトコードを生成します。

コンパイルの結果は【出力】【結果】ウィンドウに表示されます。コンパイルの結果を使用して、Java コードエラーを特定および検出します。

注: トランスフォーメーション内で【OK】をクリックしても、Java トランスフォーメーションがコンパイルされます。

コンパイルエラーの修正

アウトプットウィンドウでは、Java トランスフォーメーション用の Java コードエラーの特定、およびそのソースの検出を実行できます。Java トランスフォーメーションエラーは、コードエントリタブでのエラーまたは Java トランスフォーメーションクラスのコード全体のエラーが原因で発生する場合があります。

Java トランスフォーメーションのトラブルシューティングを実行するには：

- **エラーのソースを特定します。**エラーのソースは、トランスフォーメーションの Java スニペットコードまたはクラスコード全体から検出できます。
- **エラーのタイプを特定します。**エラーのタイプを特定するには、アウトプットウィンドウに表示されるコンパイルの結果、およびエラーの場所を使用します。

エラーのソースとタイプを特定してから、コードエントリタブで Java コードを修正し、トランスフォーメーションを再度コンパイルします。

コンパイルエラーのソースの検出

Java トランスフォーメーションをコンパイルすると、アウトプットウィンドウにコンパイルの結果が表示されます。コンパイルエラーを特定するには、コンパイル結果を使用します。アウトプットウィンドウを使用してエラーのソースを特定する場合、PowerCenter クライアントはコードエントリタブまたはコード全体ウィンドウでエラーのソースを強調表示します。

コード全体ウィンドウでは、エラーの特定は可能ですが Java コードは編集できません。コード全体ウィンドウで検出したエラーを修正するには、適切なコードエントリタブでコードを変更します。コード全体ウィンドウは、トランスフォーメーションのクラスコード全体にユーザーコードを追加したことが原因で発生したエラーを表示する場合などでは、コード全体ウィンドウを使用する必要があります。

以下の場所で発生したエラーを特定する場合、アウトプットウィンドウに表示されるコンパイル結果を使用します。

- コードエントリタブ
- コード全体ウィンドウ

コードエントリタブでのエラーの検出

コードエントリタブでエラーのソースを検出するには、アウトプットウィンドウでエラーを右クリックして [コードの一部のエラーの表示] を選択するか、アウトプットウィンドウでエラーをダブルクリックします。PowerCenter クライアントは、適切なコードエントリタブでエラーのソースを強調表示します。

コード全体ウィンドウにおけるエラーの検出

コード全体ウィンドウでエラーのソースを検出するには、アウトプットウィンドウでエラーを右クリックして [コード全体のエラーの表示] を選択するか、アウトプットウィンドウでエラーをダブルクリックします。PowerCenter クライアントは、コード全体ウィンドウでエラーのソースを強調表示します。

コンパイルエラーの原因の特定

コンパイルエラーは、ユーザーコードのエラーが原因で発生する場合があります。

ユーザーコードのエラーは、クラスの非ユーザーコードでのエラーの原因になる可能性もあります。コンパイルエラーは、Java トランスフォーメーションのユーザーコードおよび非ユーザーコードで発生します。

ユーザーコードのエラー

エラーは、コードエントリタブのユーザーコードで発生する可能性があります。ユーザーコードのエラーには、標準 Java 構文および言語のエラーが含まれます。

ユーザーコードのエラーは、PowerCenter ClientDeveloper ツールがコードエントリタブのユーザーコードをクラスコード全体に追加した場合にも発生することがあります。

たとえば、Java トランスフォーメーションには整数データ型の `int1` という名前の入力ポートがあるとしします。クラスのコード全体は、以下のコードで入力ポートの変数を宣言します。

```
int int1;
```

しかし、**【入力行に達したとき】** **【入力時】** タブで同じ変数名を使用すると、Java コンパイラは変数の再宣言としてエラーを発行します。エラーを修正するには、**【入力行に達したとき】** **【入力時】** タブで変数の名前を変更します。

非ユーザーコードのエラー

コードエントリタブのユーザーコードは、非ユーザーコードでのエラーの原因になる場合もあります。

たとえば、Java トランスフォーメーションには整数データ型の `int1` および `out1` という名前の入力ポートと出力ポートがあるとしします。ここで、以下のコードを **【入力行に達したとき】** **【入力時】** コードエントリタブに書き込み、入力ポート `int1` の `interest` を計算して出力ポート `out1` に割り当てます。

```
int interest;
interest = CallInterest(int1); // calculate interest
out1 = int1 + interest;
}
```

トランスフォーメーションをコンパイルすると、PowerCenter ClientDeveloper ツールは **【入力行に達したとき】** **【入力時】** コードエントリタブのコードを、トランスフォーメーションのクラスコード全体に追加します。Java コンパイラが Java コードをコンパイルする際に中括弧が一致していないと、クラスコード全体のメソッドは不完全なまま終了し、Java コンパイラによってエラーが発行されます。

第 13 章

Java トランスフォーメーション API のリファレンス

この章では、以下の項目について説明します。

- [Java トランスフォーメーション API メソッドの概要, 244](#) ページ
- [コミット, 245](#) ページ
- [defineJExpression, 246](#) ページ
- [failSession, 247](#) ページ
- [generateRow, 247](#) ページ
- [getInRowType, 248](#) ページ
- [getMetadata, 249](#) ページ
- [incrementErrorCount, 249](#) ページ
- [invokeJExpression, 250](#) ページ
- [isNull, 251](#) ページ
- [logError, 251](#) ページ
- [logInfo, 252](#) ページ
- [resetNotification, 252](#) ページ
- [rollBack, 253](#) ページ
- [setNull, 254](#) ページ
- [setOutRowType, 254](#) ページ
- [storeMetadata, 255](#) ページ

Java トランスフォーメーション API メソッドの概要

Java トランスフォーメーションの **【Java コード】** タブで、エディタの **【Java】** ビューのコードエントリタブで、API メソッドを Java コードに追加してトランスフォーメーションの動作を定義することができます。

API メソッドをコードに追加するには、コードエントリタブのナビゲータで **【呼び出し可能な API】** リストを展開し、コードに追加するメソッドの名前をダブルクリックします。

また、ナビゲータから Java コードスニペットにメソッドをドラッグするか、Java コードスニペットに API メソッドを手動で入力することもできます。

Java トランスフォーメーションの Java コードに追加できる API メソッドは次のとおりです。

commit

トランザクションを生成します。

defineJExpression

Java 式を定義します。

failSession

エラーメッセージ付きの例外をスローし、セッションマッピングを失敗させます。

generateRow

アクティブな Java トランスフォーメーションの出力行を生成します。

getInRowType

トランスフォーメーションの現在の行の入力タイプを返します。

incrementErrorCount

セッションマッピングのエラーカウントを増やします。

invokeJExpression

defineJExpression メソッドを使用して定義した Java 式を呼び出します。

isNull

入力カラムの NULL 値の有無を確認します。

logError

セッションログにエラーメッセージを書き込みます。

logInfo

セッションログに情報メッセージを書き込みます。

resetNotification

Data Integration Service マシンがリスタートモードで実行されている場合に、マッピングの実行後に Java コードで使用する変数をリセットします。

rollback

ロールバックトランザクションを生成します。

setNull

アクティブまたはパッシブ Java トランスフォーメーションの出力カラムの値を NULL に設定します。

setOutRowType

出力行のアップデートストラテジを設定します。行に挿入、更新、または削除のフラグを設定できます。

コミット

トランザクションを生成します。

commit は、[パッケージのインポート] または [Java 式] の 2 つのコードエントリタブ以外の任意のタブで使用できます。commit は、トランザクションを生成するように設定されたアクティブなトランスフォーメーションでのみ使用できます。トランザクションを生成するように設定されていないアクティブなトランスフォーメーションで commit を使用すると、Integration Service ではエラーが発生してセッションに失敗します。

以下の構文を使用します。

```
commit();
```

以下の Java コードを使用すると、Java トランスフォーメーションが 100 行処理するごとにトランザクションが生成され、rowsProcessed 数が 0 に設定されます。

```
if (rowsProcessed==100) {  
    commit();  
    rowsProcessed=0;  
}
```

defineJExpression

式（式の文字列および入力パラメータを含む）を定義します。defineJExpression メソッドの引数には、式の構文を定義する入力パラメータと文字列値を含む JExprParamMetadata オブジェクトの配列が含まれています。

以下の構文を使用します。

```
defineJExpression(  
    String expression,  
    Object[] paramMetadataArray  
);
```

次の表に、これらのパラメータについて説明します。

パラメータ	タイプ	データ型	説明
式	入力	String	式を表す文字列。
paramMetadataArray	入力	オブジェクト []	式の入力パラメータを含む JExprParamMetadata オブジェクトの配列。

defineJExpression メソッドは、**【インポート】** タブと **【関数】** タブを除く任意のコードエントリタブで Java コードに追加することができます。

defineJExpression メソッドを使用するには、式の入力パラメータを表す JExprParamMetadata オブジェクトの配列をインスタンス化する必要があります。パラメータのメタデータ値を設定し、その配列をパラメータとして defineJExpression メソッドに渡します。

例えば、以下の Java コードでは、2 つの文字列の値をルックアップする式を作成します。

```
JExprParamMetadata params[] = new JExprParamMetadata[2];  
params[0] = new JExprParamMetadata(EDataType.STRING, 20, 0);  
params[1] = new JExprParamMetadata(EDataType.STRING, 20, 0);  
defineJExpression(":lkp.mylookup(x1,x2)",params);
```

注: 式に渡す一連のパラメータには、先頭に文字 x を付けて番号を示す必要があります。例えば、3 つのパラメータを式に渡す場合は、各パラメータに x1、x2、および x3 という名前を付けます。

failSession

エラーメッセージ付きの例外をスローし、セッションマッピングを失敗させます。

以下の構文を使用します。

```
failSession(String errorMessage);
```

次の表に、パラメータを示します。

パラメータ	パラメータの タイプ	データ型	説明
errorMessage	Input	String	エラーメッセージの文字列。

failSession メソッドを使用して、セッションマッピングを終了します。コードエントリタブの try/catch ブロック内では、failSession メソッドを使用しないでください。

【インポート】 **【パッケージのインポート】** および **【関数】** **【Java 式】** タブを除くコードエントリタブでは、Java コードに failSession メソッドを追加できます。

以下の Java コードは、input1 入力ポートに NULL 値が存在するかどうかについてテストする方法と、input1 が NULL の場合はセッションマッピングに失敗することを示しています。

```
if(isNull("input1")) {  
    failSession("Cannot process a null value for port input1.");  
}
```

generateRow

アクティブな Java トランスフォーメーションの出力行を生成します。

以下の構文を使用します。

```
generateRow();
```

generateRow メソッドを呼び出すと、Java トランスフォーメーションは出力ポート変数の現在の値を使用して出力行を生成します。1つの入力行に対応する複数の行を生成するには、各入力行に対して generateRow メソッドを複数回呼び出すことができます。アクティブな Java トランスフォーメーションで generateRow メソッドを使用しない場合、トランスフォーメーションは出力行を生成しません。

【インポート】 **【パッケージのインポート】** および **【関数】** **【Java 式】** タブ以外のすべてのコードエントリタブで、Java コードに generateRow メソッドを追加できます。

generateRow メソッドを呼び出すことができるのは、アクティブトランスフォーメーションのみです。パッシブトランスフォーメーションで generateRow メソッドを呼び出すと、セッション Data Integration Service でエラーが発生します。

以下の Java コードを使用すると、1つの出力行が生成され、出力ポートの値が変更され、別の出力行が生成されます。

```
// Generate multiple rows.  
if(!isNull("input1") && !isNull("input2"))  
{  
    output1 = input1 + input2;  
    output2 = input1 - input2;  
}  
generateRow();  
// Generate another row with modified values.
```

```
output1 = output1 * 2;
output2 = output2 * 2;
generateRow();
```

getInRowType

トランスフォーメーションの現在の行の入力タイプを返します。このメソッドは、挿入、更新、削除、またはリジェクトの値を返します。

以下の構文を使用します。

```
rowType getInRowType();
```

次の表に、パラメータを示します。

パラメータ	パラメータのタイプ	データ型	説明
rowType	アウトプット	String	アップデイトストラテジのタイプを返します。以下のいずれかの値となります。 - DELETE - INSERT - REJECT - UPDATE

【入力時】【入力行に達したとき】 コードエントリタブで、Java コードに getInRowType メソッドを追加できます。

getInRowType メソッドは、アップデイトストラテジを設定するように設定されたアクティブなトランスフォーメーションで使用できます。アップデイトストラテジを設定するように設定されていないアクティブなトランスフォーメーションでこのメソッドを呼び出すと、セッション Data Integration Service でエラーが発生します。

以下の Java コードを使用して、以下のアクションを実行します。

- 現在の入力行タイプを出力行にプロパゲートします。
- input1 入力ポートの値が 100 を超える場合は、出力行タイプを DELETE に設定します。

```
// Set the value of the output port.
output1 = input1;
// Get and set the row type.
String rowType = getInRowType();
setOutRowType(rowType);
// Set row type to DELETE if the output port value is > 100.
if(input1 > 100
    setOutRowType(DELETE);
```


getMetadata

Java トランスフォーメーションのメタデータを実行時に取得します。getMetadata メソッドは、pushFilter 関数でオプティマイザが Java トランスフォーメーションに渡すフィルタ条件など、storeMetadata メソッドを使用して保存するメタデータを取得します。

以下の構文を使用します。

```
getMetadata (String key);
```

次の表に、これらのパラメータについて説明します。

パラメータ	パラメータのタイプ	データ型	説明
key	Input	String	メタデータを特定します。getMetadata メソッドでは、取得するメタデータを決定するキーが使用されます。

getMetadata メソッドは、次のコードエントリタブで Java コードに追加できます。

- ヘルパ
- 入力時
- 最後
- 最適化インタフェース
- 関数

最適化にプッシュインのフィルタ条件を取得するように、getMetadata メソッドを設定することができます。getMetadata メソッドは、格納する各フィルタ条件をオプティマイザから取得できます。

```
// Retrieve a filter condition
String mydata = getMetadata ("FilterKey");
```

incrementErrorCount

セッションのエラーカウントを増やします。エラー数がセッションのエラーしきい値に達すると、セッションマッピングは失敗します。

以下の構文を使用します。

```
incrementErrorCount(int nErrors);
```

次の表に、パラメータを示します。

パラメータ	パラメータのタイプ	データ型	説明
nErrors	入力	Integer	セッションのエラーカウントの増分数。

incrementErrorCount メソッドは、**【インポート】** **【パッケージのインポート】** タブと **【関数】** **【Java 式】** タブを除く任意のコードエントリタブで Java コードに追加することができます。

以下の Java コードでは、トランスフォーメーションの入力ポートが NULL 値の場合にエラーカウントが増加します。

```
// Check if input employee id and name is null.
if (isNull ("EMP_ID_INP") || isNull ("EMP_NAME_INP"))
{
    incrementErrorCount(1);
    // if input employee id and/or name is null, don't generate a output row for this input row
    generateRow = false;
}
```

invokeJExpression

式を呼び出し、式の値を返します。

以下の構文を使用します。

```
(datatype)invokeJExpression(
    String expression,
    Object[] paramMetadataArray);
```

invokeJExpression メソッドの入力パラメータは、式および式の入力パラメータを含むオブジェクトの配列を表す文字列値です。

次の表に、これらのパラメータについて説明します。

パラメータ	パラメータ のタイプ	データ型	説明
式	入力	String	式を表す文字列。
paramMetadataArray	入力	オブジェクト []	式の入力パラメータを含むオブジェクトの配列。

invokeJExpression メソッドは、**【インポート】** タブと **【関数】** **【パッケージのインポート】** タブと **【Java 式】** タブを除く任意のコードエントリタブで Java コードに追加することができます。

invokeJExpression メソッドを使用する場合は、以下のルールおよびガイドラインに従います。

- 戻りデータ型。invokeJExpression メソッドの戻りデータ型はオブジェクトです。関数の戻り値は、適切なデータ型でキャストする必要があります。
Integer、Double、String、および byte [] のデータ型で値を返すことができます。
- 行タイプ。invokeJExpression メソッドの戻り値の行タイプは INSERT です。
戻り値に異なる行タイプを使用するには、高度なインタフェースを使用します。
- NULL 値。パラメータとして NULL 値を渡した場合、または invokeJExpression メソッドの戻り値が NULL の場合、この値は NULL インジケータとして処理されます。
例えば、式の戻り値が NULL で戻りデータ型が String の場合、NULL 値の文字列が返されます。
- Date データ型。Date データ型の入力パラメータは、String データ型に変換する必要があります。
式中の文字列を Date データ型として使用するには、to_date()関数を使用して、文字列を Date データ型に変換します。
また、Date データ型を String データ型として返す式の戻り型をキャストする必要があります。

以下の例は、文字列「John」と「Smith」を連結し、文字列「John Smith」を返します。

```
(String)invokeJExpression("concat(x1,x2)", new Object [] { "John ", "Smith" });
```

注: 式に渡す一連のパラメータには、先頭に文字 x を付けて番号を示す必要があります。例えば、3 つのパラメータを式に渡す場合は、各パラメータに x1、x2、および x3 という名前を付けます。

isNull

入力カラムの値を調べ、NULL 値の有無を確認します。

以下の構文を使用します。

```
Boolean isNull(String strColName);
```

次の表に、パラメータを示します。

パラメータ	パラメータのタイプ	データ型	説明
strColName	Input	String	入力カラムの名前。

isNull メソッドは、**【入力時】【入力行に達したとき】** コードエントリタブで Java コードに追加できます。

以下の Java コードは、SALARY 入力カラムの値が NULL であるかどうかを確認してから totalSalaries インスタンス変数に追加する方法を示しています。

```
// if value of SALARY is not null
if (!isNull("SALARY")) {
    // add to totalSalaries
    TOTAL_SALARIES += SALARY;
}
```

以下の Java コードを使用しても同じ結果が得られます。

```
// if value of SALARY is not null
String strColName = "SALARY";
if (!isNull(strColName)) {
    // add to totalSalaries
    TOTAL_SALARIES += SALARY;
}
```

logError

セッションログにエラーメッセージを書き込みます。

以下の構文を使用します。

```
logError(String msg);
```

次の表に、パラメータを示します。

パラメータ	パラメータのタイプ	データ型	説明
msg	Input	String	エラーメッセージの文字列。

logError メソッドは、**【インポート】** **【パッケージのインポート】** タブと **【関数】** **【Java 式】** タブを除く任意のコードエントリタブで Java コードに追加することができます。

以下の Java コードでは、入力ポートが NULL の場合にエラーがログに記録されます。

```
// check BASE_SALARY
if (isNull("BASE_SALARY")) {
    logError("Cannot process a null salary field.");
}
```

このコードを実行すると、以下のメッセージがセッションログに出力されます。

[JTX_1013] [ERROR] Cannot process a null salary field.

logInfo

セッションログに情報メッセージを書き込みます。

以下の構文を使用します。

```
logInfo(String msg);
```

次の表に、パラメータを示します。

パラメータ	パラメータの タイプ	データ型	説明
msg	Input	String	情報メッセージの文字列。

【インポート】 **【パッケージのインポート】** および **【関数】** **【Java 式】** タブを除くすべてのコードエントリタブで、Java コードに logInfo メソッドを追加できます。

以下の Java コードは、Java トランスフォーメーションがメッセージしきい値である 1,000 行を処理した後に、セッションログにメッセージを書き込む方法を示しています。

```
if (numRowsProcessed == messageThreshold) {
    logInfo("Processed " + messageThreshold + " rows.");
}
```

resetNotification

Data Integration Service マシンがリスタートモードで実行されている場合に、マッピングの実行後に Java コードで使用する変数をリセットします。

リスタートモードでは、Data Integration Service の初期化は解除されませんが、Data Integration Service が要求後にリセットされて次の要求を処理できるようになります。

Java トランスフォーメーションの場合は、resetNotification メソッドを使用すると、マッピングの実行後に Java コードの変数がリセットされます。

以下の構文を使用します。

```
public int resetNotification(IGroup group) {
    return EStatus.value;
}
```

次の表に、これらのパラメータについて説明します。

パラメータ	パラメータのタイプ	データ型	説明
int	出力	EStatus.value	戻り値。value は次の値のいずれかになります。 <ul style="list-style-type: none">- SUCCESS。成功です。- FAILURE。失敗です。- NOIMPL。実装されていません。
group	入力	IGroup	入力グループ。

resetNotification メソッドは、[ヘルパ] タブのコードエントリタブで Java コードに追加できます。

resetNotification メソッドは [呼び出し可能な API] リストには表示されません。

例えば、Java コードで out5_static という名前の静的変数を宣言し、その変数を 1 に初期化するとします。以下の Java コードでは、次のマッピングの実行後に out5_static 変数が 1 にリセットされます。

```
public int resetNotification(IGroup group) {  
    out5_static=1;  
    return EStatus.SUCCESS;  
}
```

このメソッドは必須ではありません。ただし、Data Integration Service をリスタートモードで実行している場合、resetNotification メソッドが実装されていない Java トランスフォーメーションがマッピングに含まれていると、JSDK_42075 警告メッセージがログに出力されます。

rollBack

ロールバックトランザクションを生成します。

rollBack は、[パッケージのインポート] または [Java 式] の 2 つのコードエントリタブ以外の任意のタブで使用できます。rollback は、トランザクションを生成するように設定されたアクティブなトランスフォーメーションでのみ使用できます。トランザクションを生成するように設定されていないアクティブなトランスフォーメーションで rollback を使用すると、Integration Service ではエラーが発生してセッションに失敗します。

以下の構文を使用します。

```
rollBack();
```

以下のコードを使用すると、入力行が不正な状態にある場合にロールバックトランザクションが生成され、セッションに失敗します。それ以外の場合は、処理された行数が 100 のときにトランザクションが生成されます。

```
// If row is not legal, rollback and fail session.  
if (!isRowLegal()) {  
    rollback();  
    failSession("Cannot process illegal row.");  
} else if (rowsProcessed==100) {  
    commit();  
    rowsProcessed=0;  
}
```

setNull

アクティブまたはパッシブな Java トランスフォーメーションの出力カラムの値を NULL に設定します。

以下の構文を使用します。

```
setNull(String strColName);
```

次の表に、パラメータを示します。

パラメータ	パラメータのタイプ	データ型	説明
strColName	Input	String	出力カラムの名前。

setNull メソッドは、アクティブまたはパッシブな Java トランスフォーメーションの出力カラムの値を NULL に設定します。出力カラムを NULL に設定したら、出力行を生成するまで値は変更できません。

【インポート】 **【パッケージのインポート】** および **【関数】** **【Java 式】** タブを除くすべてのコードエントリタブで、Java コードに setNull メソッドを追加できます。

以下の Java コードは、入力カラムの値をチェックし、出力カラムの対応する値を NULL に設定する方法を示しています。

```
// check value of Q3RESULTS input column
if(isNull("Q3RESULTS")) {
    // set the value of output column to null
    setNull("RESULTS");
}
```

また、以下の Java コードを使用して同じ結果を得ることもできます。

```
// check value of Q3RESULTS input column
String strColName = "Q3RESULTS";
if(isNull(strColName)) {
    // set the value of output column to null
    setNull(strColName);
}
```

setOutRowType

出力行の更新方式を設定します。setOutRowType メソッドは、挿入、更新、または削除を実行する行にフラグを設定できます。

setOutRowType は、**【入力行に達したとき】** コードエントリタブでのみ使用できます。setOutRowType は、更新方式を設定するように設定されたアクティブなトランスフォーメーションでのみ使用できます。更新方式を設定するように設定されていないアクティブなトランスフォーメーションで setOutRowType を使用すると、セッションでエラーが発生してセッションに失敗します。

以下の構文を使用します。

```
setOutRowType(String rowType);
```

以下の表に、このメソッドの引数を示します。

引数	データ型	入力/ 出力	説明
rowType	文字列	入力	更新方式のタイプ。INSERT、UPDATE、DELETE を使用できます。

行のタイプが UPDATE または INSERT で、input1 入力ポートの値が 100 より小さい場合は、以下の Java コードを使用して現在の行の入力タイプをプロパゲートします。input1 の値が 100 よりも大きい場合は、出力タイプを DELETE に設定します。

```
// Set the value of the output port.  
  
output1 = input1;  
  
// Get and set the row type.  
  
String rowType = getInRowType();  
setOutRowType(rowType);  
  
// Set row type to DELETE if the output port value is > 100.  
  
if(input1 > 100)  
    setOutRowType(DELETE);
```

storeMetadata

実行時に getMetadata メソッドを使用して取得できる Java トランスフォーメーションのメタデータを格納します。

以下の構文を使用します。

```
storeMetadata (String key String data);
```

次の表に、これらのパラメータについて説明します。

パラメータ	パラメータ のタイプ	データ型	説明
key	Input	String	メタデータを特定します。storeMetadata メソッドでは、メタデータを特定するキーが必要です。キーを任意の文字列として定義します。
data	Input	String	Java トランスフォーメーションのメタデータとして格納するデータ。

storeMetadata メソッドは、次のコードエントリタブで Java コードに追加できます。

- ヘルパ
- 入力時
- 最後
- 最適化インタフェース

- 関数

最適化にプッシュインのフィルタ条件を受け入れるように、アクティブなトランスフォーメーションで `storeMetadata` メソッドを設定することができます。 `storeMetadata` メソッドは、オプティマイザがマッピングから Java トランスフォーメーションにプッシュするフィルタ条件を格納します。

```
// Store a filter condition  
storeMetadata ("FilterKey", condition);
```


第 14 章

Java 式

この章では、以下の項目について説明します。

- [Java 式の概要, 257 ページ](#)
- [\[式の定義\] \[関数の定義\] ダイアログボックスを使用した式の定義, 258 ページ](#)
- [単純なインタフェースに関する作業, 260 ページ](#)
- [高度なインタフェースに関する作業, 262 ページ](#)
- [JExpression クラス API リファレンス, 266 ページ](#)

Java 式の概要

Java トランスフォーメーションの PowerCenter の式を、Java プログラミング言語を使用して呼び出すことができます。

式を使用すると、Java トランスフォーメーションの機能が拡張されます。たとえば、Java トランスフォーメーションの式を呼び出して、入出力ポートの値、あるいは Java トランスフォーメーション変数の値をルックアップできます。

Java トランスフォーメーションの式を呼び出すには、Java コードを生成するか、または Java トランスフォーメーション API メソッドを使用します。式を呼び出して、適切なコードエントリタブでの式の結果を使用します。式を呼び出す Java コードを生成することも、API メソッドを使用して式を呼び出す Java コードを記述することもできます。

以下の表に、Java トランスフォーメーションの式の作成および呼び出しに使用できる方法を示します。

メソッド	説明
[式の定義] [関数の定義] ダイアログボックス	式を呼び出す関数を作成し、式のコードを生成できます。
単純なインタフェース	式を呼び出す 1 つのメソッドを呼び出し、その式の結果を取得できます。
高度なインタフェース	式を定義し、式を呼び出して、その式の結果を使用できます。 オブジェクト指向プログラミングについての知識があり、式の呼び出しをさらに制御したい場合は、高度なインタフェースを使用します。

式の関数タイプ

Java トランスフォーメーションの式は、**式エディタ**を使用するか、**【式の定義】** ダイアログボックスに式を書き込むか、**【関数の定義】** ダイアログボックスを使用するか、または単純/高度なインタフェースを使用して作成できます。

入出力ポート変数、あるいは Java コード内の変数を入力パラメータとして使用する式を入力できます。

【式の定義】 ダイアログボックスを使用する場合、Java トランスフォーメーションで使用する前に**式エディタ**を使用して式を検証できます。

【関数の定義】 ダイアログボックスを使用する場合、Java トランスフォーメーションで使用する前に式を検証できます。

Java トランスフォーメーションでは、以下のタイプの式の関数を呼び出すことができます。

式の関数タイプ	説明
トランスフォーメーション言語関数	一般的な式を扱うように設計された、SQL に似た関数です。
ユーザー定義関数	トランスフォーメーション言語関数に基づいて PowerCenterDeveloper ツールで作成する関数です。
カスタム関数	カスタム関数 API を使用して作成する関数です。

また、未接続のトランスフォーメーションおよび、ビルトイン変数、ユーザー定義マッピングおよびワークフロー変数、および定義済みワークフロー変数も式で使用できます。例えば、未接続のルックアップトランスフォーメーションを式で使用できます。

【式の定義】【関数の定義】 ダイアログボックスを使用した式の定義

Java 式を定義する場合、関数を設定し、式を作成し、式を呼び出すコードを生成します。

関数を定義して式を作成するには、**【式の定義】** **【関数の定義】** ダイアログボックスを使用します。

式関数を作成して Java トランスフォーメーションで式を使用するには、以下の高度なタスクを実行します。

1. 式を呼び出す関数を設定します。関数名、説明、およびパラメータの設定を含みます。関数パラメータは、式を作成する場合に使用します。
2. 式の構文を作成し、式を検証します。
3. 式を呼び出す Java コードを生成します。

生成されたコードが、Transformation Developer の **【Java 式】** コードエントリタブに表示されます。

生成されたコードが、**【関数】** コードエントリタブに表示されます。

Java コードを生成したら、生成した関数を適切なコードエントリタブで呼び出し、単純なインタフェースと高度なインタフェースのどちらを使用するかに基づいて、式を呼び出すか JExpression オブジェクトを取得します。

注: 式の作成時に式を検証する場合、**【式の定義】** **【関数の定義】** ダイアログボックスを使用する必要があります。

手順 1. 関数の設定

式を呼び出す Java 関数の関数名、説明、および入力パラメータを設定します。

関数を設定する場合、以下のルールおよびガイドラインを使用します。

- トランスフォーメーション内に既に存在する Java 関数、または Java の予約語と名前が重複しない、ユニークな関数名を使用します。
- パラメータ名、Java データ型、精度、および位取りを設定する必要があります。入力パラメータは、トランスフォーメーションで Java コードの関数を呼び出す際に渡す値です。
- 式に Date データ型を渡すには、入力パラメータの String データ型を使用します。

Date データ型を返す式の場合、単純なインタフェースでは戻り値を String データ型として、高度なインタフェースでは String データ型または Long データ型として使用できます。

手順 2. 式の作成と検証

式を作成する場合、設定したパラメータを関数で使用します。

式では、トランスフォーメーション言語の関数、カスタム関数、または他のユーザー定義関数も使用できます。式の作成および検証は、**【関数の定義】** ダイアログボックス **【式の定義】** ダイアログボックスまたは **【式エディタ】** ダイアログボックスで実行できます。

手順 3. 式の Java コードの生成

関数と関数パラメータの定義、および式の定義と検証を完了したら、式を呼び出す Java コードを生成できます。

DesignerDeveloper は、生成された Java コードを **【Java 式】** **【関数】** コードエントリタブに格納します。生成した Java コードを使用して、Transformation Developer 内のコードエントリタブの式を呼び出す関数を呼び出します。単純な Java コード、または高度な Java コードを生成できます。

式を呼び出す Java コードを生成した後では、その式の編集および再検証を実行できません。コードを生成した後で式を変更する場合、式を再作成する必要があります。

【式の定義】【関数の定義】 ダイアログボックスを使用した式の作成と Java コードの生成

式を呼び出す関数を作成するには、**【式の定義】** **【関数の定義】** ダイアログボックスを使用します。

式を呼び出す関数を作成するには、以下の手順を実行します。

1. Transformation Developer で、Java トランスフォーメーションを開きます。または、新規の Java トランスフォーメーションを作成します。
2. **【Java コード】** タブで、**【新しい関数】** **【式の定義】** リンクをクリックします。
【式の定義】 **【関数の定義】** ダイアログボックスが表示されます。
3. 関数名を入力します。
4. 必要に応じて、式の説明を入力します。
最大で 2,000 文字まで入力できます。
5. 関数のパラメータ引数を作成します。
パラメータ引数を作成する場合、パラメータ引数の名前、データ型、精度、位取りを設定します。
6. **【エディタの起動】** をクリックし、作成したパラメータを使用して式を作成します。 **【式】** タブで、作成した引数を使用して式を作成します。

7. 式を検証するには、**【検証】** をクリックします。
8. 必要に応じて、**【式】** ボックスに式を入力します。その後、**【検証】** をクリックして式を検証します。
9. 高度なインタフェースを使用して Java コードを生成するには、**【詳細コードの生成】** オプションを選択します。次に、**【生成】** をクリックします。

DesignerDeveloper で、**【Java 式】** **【関数】** コードエントリタブの式を呼び出す関数が生成されます。

Java 式のテンプレート

式の単純または高度な Java コードを使用すると、式の Java コードを生成できます。

式の Java コードは、式のテンプレートに基づいて生成されます。

以下の例は、単純な Java コード用に生成された Java 式のテンプレートを示しています。

```
Object function_name (Java datatype x1[,  
                        Java datatype x2 ...] )  
                        throws SDK Exception  
{  
    return (Object)invokeJExpression( String expression,  
                                      new Object [] { x1[, x2, ... ]} );  
}
```

以下の例は、高度なインタフェースを使用して生成された Java 式のテンプレートを示しています。

```
JExpression function_name () throws SDKException  
{  
    JExprParamMetadata params[] = new JExprParamMetadata[number of parameters];  
    params[0] = new JExprParamMetadata (  
        EDataType.STRING, // data type  
        20, // precision  
        0 // scale  
    );  
    ...  
    params[number of parameters - 1] = new JExprParamMetadata (  
        EDataType.STRING, // data type  
        20, // precision  
        0 // scale  
    );  
    ...  
    return defineJExpression(String expression,params);  
}
```

単純なインタフェースに関する作業

単純なインタフェースで式を呼び出すには、invokeJExpression Java API メソッドを使用します。

invokeJExpression

式を呼び出し、式の値を返します。

以下の構文を使用します。

```
(datatype)invokeJExpression(  
    String expression,  
    Object[] paramMetadataArray);
```

invokeJExpression メソッドの入力パラメータは、式および式の入力パラメータを含むオブジェクトの配列を表す文字列値です。

次の表に、これらのパラメータについて説明します。

パラメータ	パラメータ のタイプ	データ型	説明
式	入力	String	式を表す文字列。
paramMetadataArray	入力	オブジェクト []	式の入力パラメータを含むオブジェクトの配列。

invokeJExpression メソッドは、**【インポート】** タブと **【関数】** **【パッケージのインポート】** タブと **【Java 式】** タブを除く任意のコードエントリタブで Java コードに追加することができます。

invokeJExpression メソッドを使用する場合は、以下のルールおよびガイドラインに従います。

- 戻りデータ型。invokeJExpression メソッドの戻りデータ型はオブジェクトです。関数の戻り値は、適切なデータ型でキャストする必要があります。
Integer、Double、String、および byte [] のデータ型で値を返すことができます。
- 行タイプ。invokeJExpression メソッドの戻り値の行タイプは INSERT です。
戻り値に異なる行タイプを使用するには、高度なインタフェースを使用します。
- NULL 値。パラメータとして NULL 値を渡した場合、または invokeJExpression メソッドの戻り値が NULL の場合、この値は NULL インジケータとして処理されます。
例えば、式の戻り値が NULL で戻りデータ型が String の場合、NULL 値の文字列が返されます。
- Date データ型。Date データ型の入力パラメータは、String データ型に変換する必要があります。
式中の文字列を Date データ型として使用するには、to_date()関数を使用して、文字列を Date データ型に変換します。
また、Date データ型を String データ型として返す式の戻り型をキャストする必要があります。

注: 式に渡す一連のパラメータには、先頭に文字 x を付けて番号を示す必要があります。例えば、3 つのパラメータを式に渡す場合は、各パラメータに x1、x2、および x3 という名前を付けます。

単純なインタフェースの例

【Helper コード】 **【ヘルパ】** および **【入力行に達したとき】** **【入力時】** コードエントリタブで、invokeJExpression API メソッドを使用する式を定義し、呼び出すことができます。

以下の例は、Java トランスフォーメーションの NAME および ADDRESS 入力ポートに対してルックアップを完了する方法、および COMPANY_NAME 出力ポートに戻り値を割り当てる方法を示しています。

【入力行に達したとき】 **【入力時】** コードエントリタブで、以下のコードを入力します。

```
COMPANY_NAME = (String)invokeJExpression(":lkp.my_lookup(X1,X2)", new Object [] {str1 ,str2} );
generateRow();
```

高度なインタフェースに関する作業

高度なインタフェースでは、オブジェクト指向の API メソッドを使用して、式の定義、呼び出し、および結果の取得を行うことができます。

以下の表に、高度なインタフェースで使用可能なクラスおよび API メソッドを示します。

クラスまたは API メソッド	説明
EDatatype クラス	式のデータ型を列挙します。
JExprParamMetadata クラス	式内の各パラメータのメタデータが含まれています。パラメータのメタデータには、データ型、精度、および位取りが含まれます。
defineJExpression API メソッド	式を定義します。PowerCenter 式の文字列およびパラメータが含まれます。
invokeJExpression API メソッド	式を呼び出します。
JExpression クラス	メタデータの作成、呼び出し、式の結果の取得、および戻りデータ型のチェックを実行するメソッドが含まれます。

高度なインタフェースを使用した式の呼び出し

高度なインタフェースを使用して、式の定義、呼び出し、および結果の取得を行うことができます。

1. **[Helper コード]** または **[入力行に達したとき]** **[ヘルパ]** または **[入力時]** コードエントリタブで、式の各パラメータ引数に対する JExprParamMetadata クラスのインスタンスを作成し、メタデータの値を設定します。必要に応じて、defineJExpression メソッドで JExprParamMetadata オブジェクトをインスタンス化できます。
2. defineJExpression メソッドを使用して、式の JExpression オブジェクトを取得します。
3. 適切なコードエントリタブで、invokeJExpression メソッドを使用して式を呼び出します。
4. isResultNull メソッドを使用して、戻り値の結果をチェックします。
5. getResultDataType メソッドまたは getResultMetadata メソッドを使用して、戻り値のデータ型またはメタデータをそれぞれ取得できます。
6. 適切な API メソッドを使用して、式の結果を取得します。getInt メソッド、getDouble メソッド、getStringBuffer メソッド、および getBytes メソッドを使用できます。

高度なインタフェースに関する作業のルールとガイドライン

高度なインタフェースを使用するときは、ルールとガイドラインに注意する必要があります。

以下のルールおよびガイドラインを使用します。

- パラメータとして NULL 値を渡す場合、または式の結果が NULL の場合、その値は NULL インジケータとして処理されます。例えば、式の結果が NULL で戻り値のデータ型が String の場合、NULL 値の文字列が返されます。式の結果は、isResultNull メソッドを使用して確認できます。

- Date データ型の入力パラメータを String データ型に変換すると、式で使えるようになります。式中の文字列を Date データ型として使用するには、to_date()関数を使用して、文字列を Date データ型に変換します。

Date データ型を String データ型または Long データ型として返す式の結果を取得できます。

Date データ型を String データ型として返す式の結果を取得するには、getStringBuffer メソッドを使用します。Date データ型を Long データ型として返す式の結果を取得するには、getLong メソッドを使用します。

EDataType クラス

式で使用される Java データ型を列挙します。式の戻りデータ型を取得、または JExprParamMetadata オブジェクトのパラメータのデータ型を割り当てます。EDataType クラスをインスタンス化する必要はありません。

以下の表に、式で値が列挙される Java データ型を示します。

データ型	列挙された値
INT	1
DOUBLE	2
STRING	3
BYTE_ARRAY	4
DATE_AS_LONG	5

以下の Java コード例は、EDataType クラスを使用して、String データ型を JExprParamMetadata オブジェクトに割り当てる方法を示しています。

```
JExprParamMetadata params[] = new JExprParamMetadata[2];
params[0] = new JExprParamMetadata (
    EDataType.STRING, // data type
    20, // precision
    0 // scale
);
...
```

JExprParamMetadata クラス

式のパラメータを表すオブジェクトをインスタンス化し、パラメータのメタデータを設定します。

入力パラメータのメタデータを設定するには、JExprParamMetadata オブジェクトの配列を defineJExpression メソッドへの入力として使用します。JExprParamMetadata オブジェクトのインスタンスは、**【Java 式】** **【関数】** コードエントリタブまたは defineJExpression で作成できます。

以下の構文を使用します。

```
JExprParamMetadata paramMetadataArray[] = new JExprParamMetadata[numberOfParameters];
paramMetadataArray[0] = new JExprParamMetadata(datatype, precision, scale);
...
paramMetadataArray[numberOfParameters - 1] = new JExprParamMetadata(datatype, precision, scale);;
```

次の表に、引数を示します。

引数	引数のタイプ	引数のデータ型	説明
データ型	入力	EDatatype	パラメータのデータ型。
精度	入力	Integer	パラメータの精度。
位取り	入力	Integer	パラメータの位取り。

たとえば、以下の Java コードを使用して、2 つの JExprParamMetadata オブジェクトの配列を、String データ型、精度 20、位取り 20 でインスタンス化します。

```
JExprParamMetadata params[] = new JExprParamMetadata[2];
params[0] = new JExprParamMetadata(EDatatype.STRING, 20, 0);
params[1] = new JExprParamMetadata(EDatatype.STRING, 20, 0);
return defineJExpression("LKP.LKP_addresslookup(X1,X2)",params);
```

defineJExpression

式（式の文字列および入力パラメータを含む）を定義します。defineJExpression メソッドの引数には、式の構文を定義する入力パラメータと文字列値を含む JExprParamMetadata オブジェクトの配列が含まれています。

以下の構文を使用します。

```
defineJExpression(
    String expression,
    Object[] paramMetadataArray
);
```

次の表に、これらのパラメータについて説明します。

パラメータ	タイプ	データ型	説明
式	入力	String	式を表す文字列。
paramMetadataArray	入力	オブジェクト []	式の入力パラメータを含む JExprParamMetadata オブジェクトの配列。

defineJExpression メソッドは、**【インポート】** タブと **【関数】** タブを除く任意のコードエントリタブで Java コードに追加することができます。

defineJExpression メソッドを使用するには、式の入力パラメータを表す JExprParamMetadata オブジェクトの配列をインスタンス化する必要があります。パラメータのメタデータ値を設定し、その配列をパラメータとして defineJExpression メソッドに渡します。

例えば、以下の Java コードでは、2 つの文字列の値をルックアップする式を作成します。

```
JExprParamMetadata params[] = new JExprParamMetadata[2];
params[0] = new JExprParamMetadata(EDatatype.STRING, 20, 0);
params[1] = new JExprParamMetadata(EDatatype.STRING, 20, 0);
defineJExpression("lkp.mylookup(x1,x2)",params);
```

注: 式に渡す一連のパラメータには、先頭に文字 x を付けて番号を示す必要があります。例えば、3 つのパラメータを式に渡す場合は、各パラメータに x1、x2、および x3 という名前を付けます。

JExpression クラス

式の作成および呼び出しを実行するメソッド、式の値を返すメソッド、および戻りデータ型をチェックするメソッドが含まれています。

以下の表に、JExpression クラスのメソッドを示します。

メソッド名	説明
invoke	式を呼び出します。
getResultDataType	式の結果のデータ型を返します。
getResultMetadata	式の結果のメタデータを返します。
isResultNull	式の結果の結果値をチェックします。
getInt	式の結果の値を Integer データ型で返します。
getDouble	式の結果の値を Double データ型で返します。
getStringBuffer	式の結果の値を String データ型で返します。
getBytes	式の結果の値を byte [] データ型で返します。

関連項目：

- [「JExpression クラス API リファレンス」 \(ページ 266\)](#)

高度なインタフェースの例

高度なインタフェースを使用して、Java トランスフォーメーションのルックアップ式を作成したり呼び出したりすることができます。

次の例は、式を呼び出す関数を作成する方法、および戻り値を取得するための式を呼び出す方法を示しています。この例では、String データ型の 2 つの入力ポート (NAME および COMPANY) の値を myLookup 関数に渡します。myLookup 関数は、ルックアップ式を使用して ADDRESS 出力ポートの値をルックアップします。

注: この例では、LKP_addresslookup という名前のマッピング内に未接続のルックアップトランスフォーメーションが存在すると想定しています。

Transformation Developer の **[Helper コード]** **[ヘルパ]** タブで、以下の Java コードを使用します。

```
JExpression addressLookup() throws SDKException
{
    JExprParamMetadata params[] = new JExprParamMetadata[2];
    params[0] = new JExprParamMetadata (
        EDataType.STRING,    // data type
        50,                  // precision
        0,                   // scale
    );
    params[1] = new JExprParamMetadata (
        EDataType.STRING,    // data type
        50,                  // precision
        0,                   // scale
    );
    return defineJExpression(":LKP.LKP_addresslookup(X1,X2)",params);
}
JExpression lookup = null;
boolean isJExprObjCreated = false;
```

式を呼び出して ADDRESS ポートの値を返すには、**【入力行に達したとき】** **【入力時】** タブで以下の Java コードを使用します。

```
...
if(!isJExprObjCreated)
{
    lookup = addressLookup();
    isJExprObjCreated = true;
}
lookup = addressLookup();
lookup.invoke(new Object [] {NAME,COMPANY}, ERowType.INSERT);
EDatatype addressDataType = lookup.getResultDataType();
if(addressDataType == EDatatype.STRING)
{
    ADDRESS = (lookup.getStringBuffer()).toString();
} else {
    logError("Expression result datatype is incorrect.");
}
...

```

JExpression クラス API リファレンス

JExpression クラスには、式の作成および呼び出しを行う API メソッド、式の値を返す API メソッド、および戻りデータ型をチェックする API メソッドが含まれています。

JExpression クラスには、以下の API メソッドが含まれています。

- getBytes
- getDouble
- getInt
- getLong
- getResultDataType
- getResultMetadata
- getStringBuffer
- 呼び出し
- isResultNull

getBytes

式の結果の値を byte [] データ型で返します。AES_ENCRYPT 関数でデータを暗号化する式の結果を取得します。

以下の構文を使用します。

```
objectName.getBytes();
```

以下の Java コードの例では、JExprEncryptData が JExpression オブジェクトである場合に、AES_ENCRYPT 関数を使用してバイナリデータを暗号化する式の結果を取得します。

```
byte[] newBytes = JExprEncryptData.getBytes();
```

getDouble

式の結果の値を Double データ型で返します。

以下の構文を使用します。

```
objectName.getDouble();
```

以下の Java コードの例では、JExprSalary が JExpression オブジェクトである場合に給与の値を Double データ型として返す式の結果を取得します。

```
double salary = JExprSalary.getDouble();
```

getInt

式の結果の値を Integer データ型で返します。

以下の構文を使用します。

```
objectName.getInt();
```

たとえば、以下の Java コードを使用して、findEmpID が JExpression オブジェクトである場合に社員の ID 番号を整数として返す式の結果を取得します。

```
int empID = findEmpID.getInt();
```

getLong

式の結果の値を Long データ型として返します。Date データ型を使用する式の結果を取得します。

以下の構文を使用します。

```
objectName.getLong();
```

以下の Java コード例を使用して、JExprCurrentDate が JExpression オブジェクトである場合に日付の値を Long データ型として返す式の結果を取得します。

```
long currDate = JExprCurrentDate.getLong();
```

getResultDataType

式の結果のデータ型を返します。EDatatype の値を返します。

以下の構文を使用します。

```
objectName.getResultDataType();
```

以下の Java コードの例では、式を呼び出し、結果のデータ型を dataType 変数に割り当てます。

```
myObject.invoke(new Object[] { NAME,COMPANY }, ERowType INSERT);  
EDatatype dataType = myObject.getResultDataType();
```

getResultMetadata

式の結果のメタデータを返します。getResultMetadata を使用して、式の結果の精度、位取り、およびデータ型を取得し、式の戻り値のメタデータを JExprParamMetadata オブジェクトに割り当てることができます。結果のメタデータを取得するには、getScale、getPrecision、getDataType オブジェクトメソッドを使用します。

以下の構文を使用します。

```
objectName.getResultMetadata();
```

以下の Java コードの例では、myObject の戻り値の位取り、精度、およびデータ型を変数に割り当てます。

```
JExprParamMetadata myMetadata = myObject.getResultMetadata();
int scale = myMetadata.getScale();
int prec = myMetadata.getPrecision();
int datatype = myMetadata.getDataType();
```

注: getDataType オブジェクトメソッドは、データ型の整数値を EDataType に列挙されたとおりに返します。

getStringBuffer

式の結果の値を String データ型で返します。

以下の構文を使用します。

```
objectName.getStringBuffer();
```

以下の Java コード例を使用して、JExprConcat が JExpression オブジェクトである場合に 2 つの連結された文字列を返す式の結果を取得します。

```
String result = JExprConcat.getStringBuffer();
```

呼び出し

式を呼び出します。invoke の引数には、入力パラメータおよび行タイプを定義するオブジェクトが含まれています。invoke メソッドを使用する前に JExpression オブジェクトをインスタンス化する必要があります。行タイプには、ERowType.INSERT、ERowType.DELETE、および ERowType.UPDATE を使用します。

以下の構文を使用します。

```
objectName.invoke(
    new Object[] { param1[, ... paramN ]},
    rowType
);
```

次の表に、引数を示します。

引数	データ型	入力/ アウトプット	説明
objectName	JExpression	入力	JExpression のオブジェクト名です。
parameters	-	入力	式の入力値が含まれるオブジェクトの配列です。

例えば、**【Java 式】【関数】** コードエントリタブで、address_lookup() という名前の関数を作成したとします。この関数は、式を表す JExpression オブジェクトを返します。NAME および COMPANY の入力ポートを使用する式を呼び出すには、以下のコードを使用します。

```
JExpression myObject = address_lookup();
myObject.invoke(new Object[] { NAME, COMPANY }, ERowType.INSERT);
```

isResultNull

式の結果の値をチェックします。

以下の構文を使用します。

```
objectName.isResultNull();
```

以下の Java コード例を使用して式を呼び出し、戻り値が NULL でなかった場合に式の戻り値を ADDRESS 変数に割り当てます。

```
JExpression myObject = address_lookup();
myObject.invoke(new Object[] { NAME,COMPANY }, ERowType INSERT);
if(!myObject.isResultNull()) {
    String address = myObject.getStringBuffer();
}
```

第 15 章

Java トランスフォーメーションの例

この章では、以下の項目について説明します。

- [Java トランスフォーメーションの例の概要, 270 ページ](#)
- [手順 1. マッピングのインポート, 271 ページ](#)
- [手順 2. トランスフォーメーションの作成とポートの設定, 271 ページ](#)
- [手順 3. Java コードの入力, 272 ページ](#)
- [\[パッケージのインポート\] タブ, 272 ページ](#)
- [\[Helper コード\] タブ, 273 ページ](#)
- [\[入力行に達したとき\] タブ, 273 ページ](#)
- [手順 4. Java コードのコンパイル, 275 ページ](#)
- [手順 5. セッションとワークフローの作成, 275 ページ](#)

Java トランスフォーメーションの例の概要

ここで例として示される Java コードを使用すると、アクティブな Java トランスフォーメーションの作成およびコンパイルを実行できます。マッピングの例をインポートして、Java トランスフォーメーションの作成およびコンパイルを実行します。その後、そのマッピングを含むセッションおよびワークフローを作成すると実行できます。

この Java トランスフォーメーションは、架空の企業の従業員データを処理します。また、フラットファイルソースから入力行を読み込み、フラットファイルターゲットに出力行を書き込みます。ソースファイルには、従業員 ID 番号、名前、職位、管理職の ID 番号を含む従業員データが格納されています。

トランスフォーメーションは、指定された従業員の管理職の名前を管理職の ID 番号に基づいて検索し、従業員データを含む出力行を生成します。出力データには、従業員 ID 番号、名前、職位、従業員の管理職の名前が含まれています。ソースデータに従業員の管理職が存在しない場合は、トランスフォーメーションはその従業員が組織階層の最上位に位置している人物であると想定します。

注: トランスフォーメーションロジックは、従業員の職位がソースファイルで降順に配置されていると想定します。

マッピングの例のインポート、Java トランスフォーメーションの作成およびコンパイル、マッピングが含まれるセッションとワークフローの作成を実行するには、以下の手順を実行します。

1. マッピングの例をインポートします。

2. Java トランスフォーメーションを作成し、Java トランスフォーメーションのポートを設定します。
3. トランスフォーメーションの Java コードを適切なコードエントリタブに入力します。
4. Java コードをコンパイルします。
5. セッションとワークフローを作成および実行します。

PowerCenter クライアントのインストールには、この例で利用できるマッピング (m_jtx_hier_useCase.xml) およびフラットファイルソース (hier_data) が含まれています。

関連項目：

- [「Java トランスフォーメーション」 \(ページ 225\)](#)

手順 1。マッピングのインポート

Designer で、マッピングの例のメタデータをインポートします。マッピングの例には、以下のコンポーネントが含まれています。

- **ソース定義およびソース修飾子トランスフォーメーション。** トランスフォーメーションのソースデータを定義するフラットファイルソース定義 (hier_input)。
- **ターゲット定義。** トランスフォーメーションから出力データを受け取るフラットファイルターゲット定義 (hier_data)。

以下の場所から、マッピングのメタデータをインポートできます。

<PowerCenter Client installation directory>\client\bin\m_jtx_hier_useCase.xml

以下の図はマッピングの例を示しています。



手順 2. トランスフォーメーションの作成とポートの設定

Mapping Designer で、Java トランスフォーメーションを作成してポートを設定します。Java コードでは、入出力ポートの名前を変数として使用できます。Java トランスフォーメーションでは、入力グループまたは出力グループ内で入力ポートおよび出力ポートを作成します。Java トランスフォーメーションには、入力グループと出力グループがそれぞれ 1 つずつしか含まれていない場合があります。

Mapping Designer で、アクティブな Java トランスフォーメーションを作成してポートを設定します。この例では、トランスフォーメーションの名前は jtx_hier_useCase として設定されています。

注: この例で Java コードを使用するには、入力ポートと出力ポートでまったく同じ名前を使用する必要があります。

以下の表に、トランスフォーメーションの入力ポートおよび出力ポートを示します。

ポート名	ポートタイプ	データタイプ	精度	スケール
EMP_ID_INP	入力	Integer	10	0
EMP_NAME_INP	入力	String	100	0
EMP_AGE	入力	Integer	10	0
EMP_DESC_INP	入力	String	100	0
EMP_PARENT_EMPID	入力	Integer	10	0
EMP_ID_OUT	出力	Integer	10	0
EMP_NAME_OUT	出力	String	100	0
EMP_DESC_OUT	出力	String	100	0
EMP_PARENT_EMPNAME	出力	String	100	0

手順 3. Java コードの入力

以下のコードエントリタブで、トランスフォーメーション用の Java コードを入力します。

- **パッケージのインポート。** java.util.Map および java.util.HashMap パッケージをインポートします。
- **Helper Code.** Java トランスフォーメーションにおいてデータの状態を追跡するために使用する Map オブジェクト、ロックオブジェクト、およびブール変数が格納されています。
- **入力行に達したとき。** 入力行を受け取る際の Java トランスフォーメーションの動作を定義します。

[パッケージのインポート] タブ

[パッケージのインポート] タブでは、サードパーティ製の Java パッケージ、ビルトイン Java パッケージ、カスタム Java パッケージをインポートします。トランスフォーメーションの例では、Map パッケージおよび HashMap パッケージを使用しています。

[パッケージのインポート] タブに、以下のコードを入力します。

```
import java.util.Map;  
import java.util.HashMap;
```

Designer が、トランスフォーメーションの Java コードにインポート文を追加します。

関連項目：

- [「Java トランスフォーメーション設定値の設定」 \(ページ 239\)](#)

[Helper コード] タブ

[Helper コード] タブで、Java トランスフォーメーションのユーザー定義変数およびメソッドを宣言します。
[Helper コード] タブは、[入力行に達したとき] タブの Java コードで使用する以下の変数を定義します。

- **empMap**。ソースから取得した ID 番号および従業員名を格納する Map オブジェクトです。
- **lock**。パーティション全体で empMap へのアクセスを同期するために使用する Lock オブジェクトです。
- **generateRow**。現在の入力行に対して出力行の生成が必要かどうかを判断するために使用する Boolean 変数です。
- **isRoot**。その従業員が企業の組織図の最上位（ルート）に位置するかどうかを判断するために使用する Boolean 変数です。

[Helper コード] タブに、以下のコードを入力します。

```
// Static Map object to store the ID and name relationship of an
// employee. If a session uses multiple partitions, empMap is shared
// across all partitions.
private static Map <Integer, String> empMap = new HashMap <Integer, String> ();
// Static lock object to synchronize the access to empMap across
// partitions.
private static Object lock = new Object();
// Boolean to track whether to generate an output row based on validity
// of the input data.
private boolean generateRow;
// Boolean to track whether the employee is root.
private boolean isRoot;
```

[入力行に達したとき] タブ

Java トランスフォーメーションは、入力行を受け取ると [入力行に達したとき] タブで Java コードを実行します。この例では、トランスフォーメーションによって出力行を生成することについての可能性が入力行の値に基づいて決定されます。

[入力行に達したとき] タブで、以下のコードを入力します。

```
// Initially set generateRow to true for each input row.
generateRow = true;
// Initially set isRoot to false for each input row.
isRoot = false;
```

```

// Check if input employee id and name is null.
if (isNull ("EMP_ID_INP") || isNull ("EMP_NAME_INP"))
{
    incrementErrorCount(1);
    // If input employee id and/or name is null, don't generate a output
    // row for this input row.
    generateRow = false;
} else {
    // Set the output port values.
    EMP_ID_OUT = EMP_ID_INP;
    EMP_NAME_OUT = EMP_NAME_INP;
}

if (isNull ("EMP_DESC_INP"))
{
    setNull("EMP_DESC_OUT");
} else {
    EMP_DESC_OUT = EMP_DESC_INP;
}

boolean isParentEmpIdNull = isNull("EMP_PARENT_EMPID");

if(isParentEmpIdNull)
{
    // This employee is the root for the hierarchy.
    isRoot = true;
    logInfo("This is the root for this hierarchy.");
    setNull("EMP_PARENT_EMPNAME");
}

synchronized(lock)
{
    // If the employee is not the root for this hierarchy, get the
    // corresponding parent id.
    if(!isParentEmpIdNull)
        EMP_PARENT_EMPNAME = (String) (empMap.get(new Integer (EMP_PARENT_EMPID)));
    // Add employee to the map for future reference.
    empMap.put (new Integer(EMP_ID_INP), EMP_NAME_INP);
}

```

```

}

// Generate row if generateRow is true.
if(generateRow)
    generateRow();

```

手順 4. Java コードのコンパイル

トランスフォーメーションの Java コードをコンパイルするには、Transformation Developer の [コンパイル] をクリックします。出力ウィンドウには、コンパイルの状況が表示されます。Java コードのコンパイルに失敗した場合、コードエントリタブでエラーを修正してから Java コードをコンパイルします。トランスフォーメーションのコンパイルに成功したら、トランスフォーメーションをリポジトリに保存します。

関連項目：

- [「Java トランスフォーメーションのコンパイル」 \(ページ 241\)](#)
- [「コンパイルエラーの修正」 \(ページ 241\)](#)

手順 5. セッションとワークフローの作成

m_jtx_hier_useCase マッピングを使用して、Workflow Manager でマッピングのセッションおよびワークフローを作成します。

セッションを設定すると、以下の場所のソースファイルの例を使用できます。

<PowerCenter Client installation directory>\client\bin\hier_data

データの例

以下のデータは、サンプルソースファイルから抜粋したデータです。

```

1,James Davis,50,CEO,
4,Elaine Masters,40,Vice President - Sales,1
5,Naresh Thiagarajan,40,Vice President - HR,1
6,Jeanne Williams,40,Vice President - Software,1
9,Geetha Manjunath,34,Senior HR Manager,5
10,Dan Thomas,32,Senior Software Manager,6
14,Shankar Rahul,34,Senior Software Manager,6
20,Juan Cardenas,32,Technical Lead,10
21,Pramodh Rahman,36,Lead Engineer,14
22,Sandra Patterson,24,Software Engineer,10
23,Tom Kelly,32,Lead Engineer,10

```

35,Betty Johnson,27,Lead Engineer,14

50,Dave Chu,26,Software Engineer,23

70,Srihari Giran,23,Software Engineer,35

71,Frank Smalls,24,Software Engineer,35

以下のデータは、サンプルターゲットファイルから抜粋したデータです。

1,James Davis,CEO,

4,Elaine Masters,Vice President - Sales,James Davis

5,Naresh Thiagarajan,Vice President - HR,James Davis

6,Jeanne Williams,Vice President - Software,James Davis

9,Geetha Manjunath,Senior HR Manager,Naresh Thiagarajan

10,Dan Thomas,Senior Software Manager,Jeanne Williams

14,Shankar Rahul,Senior Software Manager,Jeanne Williams

20,Juan Cardenas,Technical Lead,Dan Thomas

21,Pramodh Rahman,Lead Engineer,Shankar Rahul

22,Sandra Patterson,Software Engineer,Dan Thomas

23,Tom Kelly,Lead Engineer,Dan Thomas

35,Betty Johnson,Lead Engineer,Shankar Rahul

50,Dave Chu,Software Engineer,Tom Kelly

70,Srihari Giran,Software Engineer,Betty Johnson

71,Frank Smalls,Software Engineer,Betty Johnson

第 16 章

ジョイナトランスフォーメーション

この章では、以下の項目について説明します。

- [ジョイナトランスフォーメーションの概要, 277 ページ](#)
- [ジョイナトランスフォーメーションのプロパティ, 278 ページ](#)
- [ジョイン条件の定義, 280 ページ](#)
- [結合タイプの定義, 280 ページ](#)
- [ソート済み入力の使用, 283 ページ](#)
- [1つのソースからのデータの結合, 285 ページ](#)
- [ソースパイプラインのブロック, 287 ページ](#)
- [トランザクションに関する作業, 288 ページ](#)
- [ジョイナトランスフォーメーションの作成, 290 ページ](#)
- [ジョイナトランスフォーメーションに関するヒント, 291 ページ](#)

ジョイナトランスフォーメーションの概要

ジョイナトランスフォーメーションを使用すると、異なる場所またはファイルシステムにある 2 つの関連する異種ソースからのソースデータを結合できます。また、同じソースからのデータを結合することもできます。ジョイナトランスフォーメーションは、一致するカラムが少なくとも 1 つあるソースを結合します。ジョイナトランスフォーメーションでは、2 つのソース間の 1 つ以上のカラムのペアと一致する条件を使用します。ジョイナトランスフォーメーションはアクティブなトランスフォーメーションです。

2 つの入力パイプラインには、マスターパイプラインと明細パイプライン、またはマスターブランチと明細ブランチがあります。マスターパイプラインはジョイナトランスフォーメーションで終了しますが、明細パイプラインはターゲットまで継続します。

マッピング内の 3 つ以上のソースを結合するには、ジョイナトランスフォーメーションからの出力を他のソースパイプラインと結合します。すべてのソースパイプラインを結合するまで、マッピングにジョイナトランスフォーメーションを追加します。

ジョイナトランスフォーメーションは、ほとんどのトランスフォーメーションからの入力を受け付けます。ただし、ジョイナトランスフォーメーションに接続するパイプラインには以下の制限があることに注意してください。

- いずれかの入力パイプラインにアップデートストラテジトランスフォーメーションが含まれている場合、ジョイナトランスフォーメーションは使用できません。
- ジョイナトランスフォーメーションの前にシーケンスジェネレータトランスフォーメーションに直接接続した場合、ジョイナトランスフォーメーションは使用できません。

ジョイナトランスフォーメーションに関する作業

ジョイナトランスフォーメーションに関する作業を実行する場合、トランスフォーメーションのプロパティ、結合タイプ、および結合条件を設定する必要があります。ソート済み入力でジョイナトランスフォーメーションを設定し、Integration Service のパフォーマンスを向上させることができます。また、トランスフォーメーション範囲を設定して、Integration Service によるトランスフォーメーションロジックの適用方法を制御することもできます。ジョイナトランスフォーメーションに関する作業を実行するには、以下のタスクを実行します。

- **ジョイナトランスフォーメーションのプロパティを設定します。** ジョイナトランスフォーメーションのプロパティでは、キャッシュディレクトリの場所、Integration Service がトランスフォーメーションを処理する方法、および Integration Service がキャッシュを扱う方法を指定します。
- **ジョイン条件を設定します。** ジョイン条件は 2 つの入力ソースからのポートを含みます。Integration Service が 2 つの行を結合するためには、これらが一致する必要があります。選択した結合タイプに応じて、Integration Service は行を結果セットに追加するか、または行を無視します。
- **結合タイプを設定します。** 結合とは、異なるデータベースまたはフラットファイルの複数のテーブルのデータを 1 つの結果セットに連結する関係演算子です。ジョイナトランスフォーメーションで使用できる結合タイプには、ノーマル、マスターアウター、明細アウター、完全アウターがあります。
- **ソート済みまたは未ソートの入力セッションを設定します。** ジョイナトランスフォーメーションでソート済み入力を使用するように設定することで、セッションのパフォーマンスを向上させることができます。ソート済みデータを使用するようにマッピングを設定するには、Integration Service がジョイナトランスフォーメーションを処理する場合にソート済みデータを使用できるように、マッピング内でソート順を確立して維持します。
- **トランザクション範囲を設定します。** Integration Service ではジョイナトランスフォーメーション処理時に、トランザクション内のすべてのデータ、すべての入力データ、または 1 行のデータに対してトランスフォーメーションロジックを一括適用できます。

PowerCenter のパーティショニングオプションを利用すれば、パイプライン内のパーティション数を増やすことにより、セッションのパフォーマンスを向上させることができます。

ジョイナトランスフォーメーションのプロパティ

ジョイナトランスフォーメーションのプロパティでは、キャッシュディレクトリの場所、統合サービスがトランスフォーメーションを処理する方法、および統合サービスがキャッシュを扱う方法を指定します。また、統合サービスがテーブルおよびファイルを結合する方法も指定します。

マッピングを作成する場合、各ジョイナトランスフォーメーションのプロパティを指定します。セッションを作成する場合、各トランスフォーメーションのインデックスキャッシュサイズやデータキャッシュサイズなど、一部のプロパティをオーバーライドできます。

以下の表では、ジョイナトランスフォーメーションのプロパティについて説明します。

オプション	説明
キャッシュディレクトリ	マスターまたは明細行およびそのインデックスのキャッシュに使用するディレクトリを指定します。デフォルトでは、キャッシュファイルはプロセス変数\$PMCacheDirで指定されたディレクトリに作成されます。ディレクトリをオーバーライドする場合は、そのディレクトリが存在していて、かつキャッシュファイルを格納するための十分なディスク領域があることを確認します。マッピングされたドライブまたはマウントされたドライブを指定することができます。
結合タイプ	ジョイントタイプを指定します。[Normal]、[Master Outer]、[Detail Outer]、または [Full Outer] です。
Null Ordering in Master	この種類のトランスフォーメーションでは該当しません。
Null Ordering in Detail	この種類のトランスフォーメーションでは該当しません。
トレースレベル	トランスフォーメーションのセッションログに表示される情報の詳細度。このオプションには、[Terse]、[Normal]、[Verbose Data]、および [Verbose Initialization] があります。
ジョイナのデータキャッシュサイズ	トランスフォーメーションのデータキャッシュサイズ。デフォルトのキャッシュサイズは 2,000,000 バイトです。設定したキャッシュの合計サイズが 2GB 以上の場合、64 ビットの統合サービスでそのセッションを実行する必要があります。キャッシュの数値を使用できます。パラメータファイルのキャッシュ値を使用するか、統合サービスを設定し、自動設定を使用してキャッシュサイズを設定します。キャッシュサイズが決定されるように統合サービスを設定した場合、キャッシュに割り当てられる最大メモリ量も設定できます。
ジョイナのインデックスキャッシュサイズ	トランスフォーメーションのインデックスキャッシュサイズ。デフォルトのキャッシュサイズは 1,000,000 バイトです。設定したキャッシュの合計サイズが 2GB 以上の場合、64 ビットの統合サービスでそのセッションを実行する必要があります。キャッシュの数値を使用できます。パラメータファイルのキャッシュ値を使用するか、統合サービスを設定し、自動設定を使用してキャッシュサイズを設定します。キャッシュサイズが決定されるように統合サービスを設定した場合、キャッシュに割り当てられる最大メモリ量も設定できます。
ソート済み入力	データがソートされていることを指定します。ソートされたデータを結合するには、[Sorted Input] を選択します。ソート済み入力を使用するとパフォーマンスを向上させることができます。
マスタのソート順	マスターソースデータのソート順を指定します。マスターソースデータが昇順である場合は、昇順を選択します。昇順を選択した場合には、ソート済み入力も有効にします。デフォルトは [Auto] です。
トランスフォーメーション範囲	統合サービスが入力データにトランスフォーメーションロジックを適用する方法を指定します。[トランザクション]、[すべての入力]、または [行] を選択できます。

ジョイン条件の定義

ジョイン条件は2つの入力ソースからのポートを含みます。Integration Service が2つの行を結合するためには、これらが一致する必要があります。選択した結合タイプに応じて、Integration Service は行を結果セットに追加するか、または行を無視します。ジョイナトランスフォーメーションは、結合タイプ、ジョイン条件、および入力データのソースに基づいて結果セットを生成します。

ジョイン条件を定義する前に、マスターソースおよび明細ソースが最適なパフォーマンスに設定されていることを確認してください。Integration Service は、セッション中にマスターソースの各行を明細ソースと比較します。未ソートジョイナトランスフォーメーションのパフォーマンスを高めるには、行の比較的少ないソースをマスターソースとして使用します。ソート済みジョイナトランスフォーメーションのパフォーマンスを高めるには、重複キー値の少ないソースをマスターとして使用します。

デフォルトでは、ジョイナトランスフォーメーションにポートを追加すると、最初のソースパイプラインが明細ソースとして表示されます。2番目のソースパイプラインから追加されたポートは、マスターソースとして設定されます。この設定を変更するには、マスターソースに設定したいポートの「ポート」タブで「M」列をクリックします。これにより、このソースのポートがマスターポートとして設定され、他のソースのポートが明細ポートに設定されます。

指定したマスターおよび明細ソースが等しいかどうかに基づいて、1つまたは複数の条件を定義します。たとえばテーブルを含む2つのソース、EMPLOYEE_AGE と EMPLOYEE_POSITION があり、どちらも従業員 ID 番号を含んでいる場合、次の条件は両方のソースに含まれる従業員の行に一致します。

```
EMP_ID1 = EMP_ID2
```

ジョイン条件では、ジョイナトランスフォーメーションの入力ソースからの1つ以上のポートを使用します。ポートの数が増えると、2つのソースの結合に要する時間も長くなります。条件内でのポートの順番によって、ジョイナトランスフォーメーションのパフォーマンスに影響を与える場合があります。ジョイン条件で複数のポートを使用する場合、Integration Service は指定された順にポートを比較します。

Designer は条件内のデータタイプを検査します。条件の両方のポートは、同じデータタイプを持つ必要があります。データタイプが一致しない2つのポートを条件で使用しなければならない場合、データタイプが一致するように変換します。

Char データタイプと Varchar データタイプを結合する場合、Integration Service は Char 値に埋め込まれている空白の数を文字列の一部として数えます。

```
Char(40) = "abcd"  
Varchar(40) = "abcd"
```

この Char 値には "abcd" と 36 個の空白が埋め込まれています。Char フィールドは後ろに空白が含まれているため、Integration Service はこの2つのフィールドを結合しません。

注: ジョイナトランスフォーメーションは、NULL 値の一致は検出しません。例えば、EMP_ID1 と EMP_ID2 の両方に NULL 値を持つ行が含まれている場合、Integration Service はこれらを一致とは見なさないため、2つの行の結合は実行されません。NULL 値を持つ行を結合するには、NULL 入力をデフォルト値で置き換えてから、デフォルト値で結合します。

結合タイプの定義

SQL では、結合は、複数のテーブルからのデータを1つの結果セットに連結する関係演算子です。ジョイナトランスフォーメーションは、異なるタイプのソースからのデータから生成できる点を除き、SQL 結合に似ています。

結合タイプはトランスフォーメーションの「プロパティ」タブで定義することができます。ジョイナトランスフォーメーションは、以下のジョインタイプをサポートしています。

- ノーマル
- Master Outer（マスターアウター）
- Detail Outer（明細アウター）
- Full Outer（完全アウター）

注: Normal または Master Outer ジョインは、Full Outer や Detail Outer ジョインよりも高速に動作します。

どちらのソースのデータも含まないフィールドが結果セットにある場合、ジョイナトランスフォーメーションは空のフィールドを NULL 値で埋めます。フィールドが NULL を返すことが分かっていて、NULL をターゲットに挿入したくない場合は、対応するポートの「ポート」タブでデフォルト値を設定できます。

Normal ジョイン

Normal ジョインの場合、Integration Service は条件に基づいて、一致しないマスタソースおよび明細ソースのデータの行をすべて破棄します。

たとえば自動車部品に関する 2 つのソース、PARTS_SIZE および PARTS_COLOR に次のデータが入っているとします。

PARTS_SIZE (master source)

PART_ID1	DESCRIPTION	SIZE
1	Seat Cover	Large
2	Ash Tray	Small
3	Floor Mat	Medium

PARTS_COLOR (detail source)

PART_ID2	DESCRIPTION	COLOR
1	Seat Cover	Blue
3	Floor Mat	Black
4	Fuzzy Dice	Yellow

両方の PART_ID を照合して 2 つのテーブルを結合するには、次のように条件を設定します。

PART_ID1 = PART_ID2

これらのテーブルに対して Normal ジョインを実行すると、結果セットには以下のデータが含まれます。

PART_ID	DESCRIPTION	SIZE	COLOR
1	Seat Cover	Large	Blue
3	Floor Mat	Medium	Black

以下の例は、相当する SQL 文を示しています。

```
SELECT * FROM PARTS_SIZE, PARTS_COLOR WHERE PARTS_SIZE.PART_ID1 = PARTS_COLOR.PART_ID2
```

Master Outer ジョイン

Master Outer ジョインは、明細ソースのデータのすべての行およびマスターソースの一致する行をすべて保持します。一致しないマスターソースの行は無視されます。

サンプルのテーブルに対して同じ条件で Master Outer ジョインを実行すると、結果セットには以下のデータが含まれます。

PART_ID	DESCRIPTION	SIZE	COLOR
1	Seat Cover	Large	Blue
3	Floor Mat	Medium	Black
4	Fuzzy Dice	NULL	Yellow

Fuzzy Dice にはサイズが指定されていないため、Integration Service はフィールドに NULL を入力します。

以下の例は、相当する SQL 文を示しています。

```
SELECT * FROM PARTS_SIZE RIGHT OUTER JOIN PARTS_COLOR ON (PARTS_COLOR.PART_ID2 = PARTS_SIZE.PART_ID1)
```

Detail Outer ジョイン

Detail Outer ジョインは、マスターソースのデータのすべての行および明細ソースの一致する行をすべて保持します。一致しない明細ソースの行は無視されます。

サンプルのテーブルに対して同じ条件で Detail Outer ジョインを実行すると、結果セットには以下のデータが含まれます。

PART_ID	DESCRIPTION	SIZE	COLOR
1	Seat Cover	Large	Blue
2	Ash Tray	Small	NULL
3	Floor Mat	Medium	Black

Ash Tray で色が指定されていないため、Integration Service はフィールドに NULL を入力します。

以下の例は、相当する SQL 文を示しています。

```
SELECT * FROM PARTS_SIZE LEFT OUTER JOIN PARTS_COLOR ON (PARTS_SIZE.PART_ID1 = PARTS_COLOR.PART_ID2)
```

Full Outer ジョイン

Full Outer ジョインでは、マスターソースと明細ソースの両方のデータのすべての行が保持されます。

サンプルのテーブルに対して前と同じ条件で Full Outer ジョインを行うと、結果セットは次のようになります。

PART_ID	DESCRIPTION	SIZE	Color
1	Seat Cover	Large	Blue
2	Ash Tray	Small	NULL
3	Floor Mat	Medium	Black

PART_ID	DESCRIPTION	SIZE	Color
4	Fuzzy Dice	NULL	Yellow

Ash Tray では色が、Fuzzy Dice ではサイズが指定されていないため、Integration Service はフィールドに NULL を入力します。

以下の例は、相当する SQL 文を示しています。

```
SELECT * FROM PARTS_SIZE FULL OUTER JOIN PARTS_COLOR ON (PARTS_SIZE.PART_ID1 = PARTS_COLOR.PART_ID2)
```

ソート済み入力の使用

ジョイナトランスフォーメーションでソート済み入力を使用するように設定することで、セッションのパフォーマンスを向上させることができます。ジョイナトランスフォーメーションでソート済みデータを使うように設定すると、Integration Service はディスクの入出力を最小化してパフォーマンスを高めます。大量のデータセットを扱う場合に、パフォーマンスを最大限に向上させることができます。

ソート済みデータを使用するようにマッピングを設定するには、Integration Service がジョイナトランスフォーメーションを処理する場合にソート済みデータを使用できるように、マッピング内でソート順を確立して維持します。以下の作業を実行して、マッピングの設定を行います。

- **ソート順を設定する。** 結合するデータのソート順を設定します。ソート済みのフラットファイルを結合することができます。あるいは、ソース修飾子トランスフォーメーションを使ってリレーショナルデータをソートすることができます。ソータトランスフォーメーションを使用することもできます。
- **トランスフォーメーションを追加する。** ソート済みデータの順序を維持するトランスフォーメーションを使用します。
- **ジョイナトランスフォーメーションを設定する。** ソート済みデータを使用するようにジョイナトランスフォーメーションを設定し、ソートの基点ポートを使用するようにジョイン条件を設定します。ソートの基点は、ソート済みデータのソースを表します。

セッションでソート順を設定する場合、Integration Service のコードページに関連するソート順を選択できます。Integration Service を Unicode モードで実行する場合、選択されたセッションのソート順を用いて文字データをソートします。Integration Service を ASCII モードで実行する場合は、バイナリのソート順を用いてすべての文字データをソートします。確実にデータを Integration Service の要求どおりにソートするには、データベースのソート順とユーザー定義のセッションのソート順を一致させる必要があります。

パーティション化されたパイプラインから取得したソート済みデータを結合する場合は、ソート済みデータの順序を維持するようにパーティションを設定する必要があります。

ソート順の設定

ソート順は、Integration Service からソート済みデータがジョイナトランスフォーメーションに確実に渡されるように設定する必要があります。

以下のいずれかの方法でソート順を設定します。

- **ソート済みフラットファイルを使用する。** フラットファイルにソート済みデータが含まれている場合は、各ソースファイル内でソートカラムの順序が一致していることを検査します。
- **ソート済みリレーショナルデータを使用する。** ソース修飾子トランスフォーメーション内のソート済みポートを使用して、ソースデータベースのカラムをソートします。各ソース修飾子トランスフォーメーション内のソート済みポートの順序は同じ設定にします。

- **ソータトランスフォーメーションを使用する。** ソータトランスフォーメーションを使用して、リレーショナルデータまたはフラットファイルデータをソートします。ソータトランスフォーメーションをマスターおよび明細パイプラインに配置します。各ソータトランスフォーメーションで、同じ順のソートキーポートおよび同じソート順方向を使用するように設定します。

ソート済みデータを使用するように設定されたジョイナトランスフォーメーションに、未ソートのデータや正しくソートされていないデータを渡すと、セッションは失敗し、Integration Service はエラーをセッションログファイルに記録します。

マッピングへのトランスフォーメーションの追加

ソートの基点とジョイナトランスフォーメーションの間にトランスフォーメーションを追加する場合は、以下のガイドラインに従って、ソート済みデータを維持します。

- ソートの基点とジョイナトランスフォーメーションの間に、以下のトランスフォーメーションは配置しないでください。
 - カスタム
 - 未ソートアグリゲータ
 - ノーマライザ
 - ランク
 - 共有体トランスフォーメーション
 - XML パーサトランスフォーメーション
 - XML ジェネレータトランスフォーメーション
 - マプレット（上記のトランスフォーメーションのいずれかが含まれる場合）
- 以下のガイドラインに従うのであれば、ソートの基点とジョイナトランスフォーメーションの間にソート済みアグリゲータトランスフォーメーションを配置することができます。
 - アグリゲータトランスフォーメーションをソート済み入力に合わせて設定します。
 - アグリゲータトランスフォーメーション内のカラムのグループのポートは、ソートの基点でのポートと同じものを使用してください。
 - ポートのグループは、ソートの基点でのポートと同じ順序である必要があります。
- 別のパイプラインを持つジョイナトランスフォーメーションの結果セットを結合する場合は、最初のジョイナトランスフォーメーションからの出力されるデータがソート済みであることを確認してください。

ヒント: ジョイナトランスフォーメーションをソートの基点の後ろに直接置くことで、ソート済みデータを維持できます。

ジョイナトランスフォーメーションの設定

ジョイナトランスフォーメーションを設定するには、以下のタスクを実行します。

- [プロパティ] タブで [ソート済み入力] を有効化します。
- ジョイン条件を、ソートの基点と同じ順序でソート済みデータを受け取るように定義します。

ジョイン条件の定義

ジョイン条件を、ソートの基点で確立されたソート順を維持するように設定します。ソートの基点となるのは、ソート済みフラットファイル、ソース修飾子トランスフォーメーション、またはソータトランスフォーメーションです。ソートの基点とジョイナトランスフォーメーションの間にソート済みアグリゲータトランスフォーメーションを使用する場合は、そのソート済みアグリゲータトランスフォーメーションを、ジョイン条件を定

義するときのソートの基点として扱います。ジョイン条件を定義するときには以下のガイドラインに従ってください。

- ジョイン条件で使用するポートは、ソートの基点でのポートと一致している必要があります。
- 複数のジョイン条件を設定する場合、最初のジョイン条件内のポートは、ソートの基点での最初のポートと一致している必要があります。
- 複数の条件を設定する場合、条件の順序は、ソートの基点でのポートの順序と一致している必要があります、またどのポートもスキップしてはなりません。
- ソートの基点でのソート済みポートの数は、ジョイン条件でのポートの数以上とすることができます。

ジョイン条件の例

たとえば、以下のソート済みポートを含むマスターパイプラインと明細パイプラインでソータトランスフォーメーションを設定します。

1. ITEM_NO
2. ITEM_NAME
3. PRICE

ジョイン条件を設定するときには、以下のガイドラインを使ってソート順を維持します。

- 最初のジョイン条件で ITEM_NO を使用する必要があります。
- 2 番目のジョイン条件を追加する場合は、ITEM_NAME を使用する必要があります。
- ジョイン条件で PRICE を使用したい場合、2 番目のジョイン条件でも ITEM_NAME を使用する必要があります。

ITEM_NAME をスキップして ITEM_NO と PRICE を結合すると、ソート順が失われ、Integration Service はセッションに失敗します。

ジョイナトランスフォーメーションを使ってマスタパイプラインと明細パイプラインを結合する場合、次のジョイン条件のうちどれか 1 つを設定できます。

```
ITEM_NO = ITEM_NO
```

または

```
ITEM_NO = ITEM_NO1  
ITEM_NAME = ITEM_NAME1
```

または

```
ITEM_NO = ITEM_NO1  
ITEM_NAME = ITEM_NAME1  
PRICE = PRICE1
```

1 つのソースからのデータの結合

データの一部に対して計算を実行し、トランスフォーメーションが実行されたデータと元のデータを結合する場合、同じソースからのデータを結合できます。この方法を使ってデータを結合する場合は、同じ 1 つのマッピング内で元のデータを維持しながら元データの一部にトランスフォーメーションを行うことができます。次の方法で同じソースのデータを結合できます。

- 同じパイプラインの 2 つのブランチを結合します。
- 同じソースの 2 つのインスタンスを結合します。

同じパイプラインの2つのブランチの結合

同じソースからのデータを結合するときには、パイプラインの2つのブランチを作成します。パイプラインをブランチに分岐する場合は、パイプラインの少なくとも片方のブランチの、ソース修飾子とジョイナトランスフォーメーションの間にトランスフォーメーションを追加する必要があります。ソート済みデータを結合し、ジョイナトランスフォーメーションをソート済み入力用に設定する必要があります。

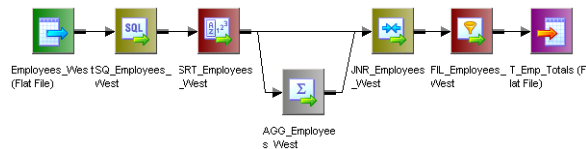
たとえば、以下のポートを含むソースがあるとします。

- Employee
- 部門
- Total Sales

ターゲットに、所属する部署の平均売上を超える売上を実現した従業員を表示するとします。そのためには、以下のトランスフォーメーションを含むマッピングを作成します。

- **ソートトランスフォーメーション。** データをソートします。
- **ソート済みアグリゲータトランスフォーメーション。** 売上データの平均を算出し、部署ごとにグループ化します。この集計を実行すると、個々の従業員のデータは失われます。従業員データを維持するには、パイプラインのブランチをアグリゲータトランスフォーメーションに渡し、同じデータのブランチをジョイナトランスフォーメーションに渡して元のデータを維持する必要があります。パイプラインの両方のブランチを結合すると、集計済みデータと元のデータを結合することになります。
- **ソート済みジョイナトランスフォーメーション。** ソート済みジョイナトランスフォーメーションを使用して、ソート済みの集計データと元のデータを結合します。
- **フィルタトランスフォーメーション。** 平均売上データと各従業員の売上データを比較し、平均売上に達していない従業員を除外します。

以下の図に、同じパイプラインの2つのブランチを結合するマッピングを示します。



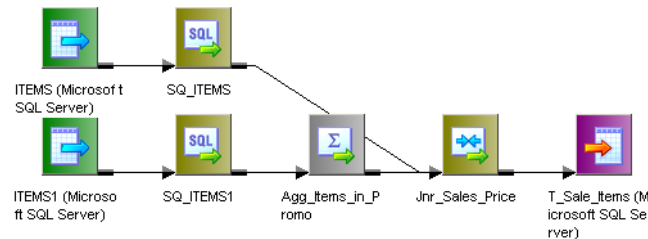
注: カスタムトランスフォーメーションやXMLソース修飾子トランスフォーメーションなどの同じトランスフォーメーションの出力グループからのデータを結合することもできます。各出力グループとジョイナトランスフォーメーションの間にソートトランスフォーメーションを配置して、ジョイナトランスフォーメーションがソート済みの入力を受け取れるように設定します。

ジョイナトランスフォーメーションが一方のブランチからのデータを受け取るのが、もう一方のブランチより大きく遅れる場合、2つのブランチの結合がパフォーマンスに影響を与えることがあります。ジョイナトランスフォーメーションは、最初のブランチからのすべてのデータをキャッシュに格納し、キャッシュがいっぱいになるとキャッシュをディスクに書き込みます。次にジョイナトランスフォーメーションが2番目のブランチからデータを受け取る際に、ディスクからデータを読み込む必要があります。この処理に時間がかかる場合があります。

同じソースの2つのインスタンスの結合

ソースの2つ目のインスタンスを作成することにより、同じソースデータを結合することもできます。2つ目のソースインスタンスを作成すると、2つのソースインスタンスからのパイプラインを結合できます。未ソートデータを結合したい場合は、同じソースのインスタンスを2つ作成して、パイプラインを結合する必要があります。

以下の図に、ジョイナトランスフォーメーションで結合される同じソースの2つのインスタンスを示します。



注: この方法を使用してデータを結合すると、Integration Service は各ソースインスタンスのソースデータを読み込みます。そのため、パイプラインの2つのブランチを結合する場合よりもパフォーマンスが低下する場合があります。

1つのソースからのデータの結合に関するガイドライン

パイプラインのブランチを結合するか、ソースの2つのインスタンスを結合するかの判断には、以下のガイドラインに従ってください。

- ソースが大きい場合、あるいはソースデータを1回しか読み込まない場合は、パイプラインの2つのブランチを結合します。たとえば、メッセージキューから1回しかソースデータを読み込まない場合が該当します。
- ソート済みデータを使う場合は、パイプラインの2つのブランチを結合します。ソースデータが未ソートデータで、ソータトランスフォーメーションを使ってデータをソートする場合には、データのソート後にパイプラインをブランチに分岐します。
- ソースとジョイナトランスフォーメーションの間のパイプラインにブロッキングトランスフォーメーションを追加する必要がある場合は、ソースの2つのインスタンスを結合します。
- 一方のパイプラインの処理がもう一方のパイプラインよりも遅い場合は、ソースの2つのインスタンスを結合します。
- 未ソートのデータを結合する必要がある場合、ソースの2つのインスタンスを結合します。

ソースパイプラインのブロック

ジョイナトランスフォーメーションを含むセッションを実行すると、Integration Service はマッピングの設定とジョイナトランスフォーメーションがソート済み入力用に設定されているかどうかによって、ソースデータのブロックとブロック解除を行います。

未ソートのジョイナトランスフォーメーション

Integration Service は未ソートのジョイナトランスフォーメーションを処理するとき、明細行を読み込む前にマスタ行をすべて読み込みます。明細行の前に確実にすべてのマスタ行を読み込むため、Integration Service はマスタソースから行をキャッシュに格納する間、明細ソースをブロックします。Integration Service はマスタ行をすべて読み込むと、明細ソースのブロックを解除して明細行を読み込みます。未ソートのジョイナトランスフォーメーションを含むマッピングは、データフローの検証に違反する場合があります。

ソート済みのジョイナトランスフォーメーション

Integration Service は、ソート済みのジョイナトランスフォーメーションを処理するとき、マッピングの設定に基づいてデータをブロックします。ジョイナトランスフォーメーションに対するマスタおよび明細入力異なるソースから生じている場合、ブロックロジックが可能になります。

ターゲットロード順グループのすべてのソースを同時にブロックせずにジョイナトランスフォーメーションを処理できる場合、Integration Service はブロックロジックを使用してジョイナトランスフォーメーションを処理します。それ以外の場合、ブロックロジックは使用しません。その代わりに、より多くの行をキャッシュに格納します。

ブロックロジックを使ってジョイナトランスフォーメーションを処理できる場合、Integration Service がキャッシュに格納する行は少なくなり、パフォーマンスが向上します。

マスタ行のキャッシュ

Integration Service は、ジョイナトランスフォーメーションを処理する場合、両方のソースから同時に行を読み込み、マスタ行に基づいてインデックスキャッシュおよびデータキャッシュを作成します。Integration Service は次に、明細ソースデータとキャッシュデータに基づいてジョインを実行します。統合サービスがキャッシュに格納する行数は、ジョイナトランスフォーメーションがソート済み入力用に設定されているかどうか、およびパーティションタイプとソースデータによって異なります。未ソートジョイナトランスフォーメーションのパフォーマンスを高めるには、行の比較的少ないソースをマスターソースとして使用します。ソート済みジョイナトランスフォーメーションのパフォーマンスを高めるには、重複キー値の少ないソースをマスターとして使用します。

トランザクションに関する作業

Integration Service ではジョイナトランスフォーメーション処理時に、トランザクション内のすべてのデータ、すべての入力データ、または1行のデータに対してトランスフォーメーションロジックを一括適用できます。Integration Service は、マッピングの設定およびトランスフォーメーション範囲に基づき、トランザクション境界を削除または保持できます。トランスフォーメーション範囲のプロパティを使用して、Integration Service によるトランスフォーメーションロジックの適用方法およびトランザクション境界の処理方法を設定します。

トランスフォーメーション範囲の値は、マッピングの設定と、トランザクション境界を削除するか保持するかに基づいて設定します。

以下のソースを結合する場合、トランザクション境界を保持できます。

- **同じソースパイプラインの2つのブランチを結合する場合。**トランザクション境界を保持するには、[トランザクション] トランスフォーメーション範囲を使用します。
- **2つのソースを結合し、明細ソースのトランザクション境界を保持する場合。**明細パイプラインでトランザクション境界を保持するには、[行] トランスフォーメーション範囲を使用します。

以下のソースを結合する場合、トランザクション境界を削除できます。

- **2つのソースまたは2つのブランチを結合し、トランザクション境界を削除する場合。**トランスフォーメーションロジックをすべての入力データに適用し、両方のパイプラインでトランザクション境界を削除する場合、[すべての入力] トランスフォーメーション範囲を使用します。

以下の表に、ジョイナトランスフォーメーションでのトランスフォーメーション範囲を使用したトランザクション境界の保持方法をまとめます。

トランスフォーメーション範囲	入力タイプ	Integration Service の動作
行	未ソート	明細パイプラインで、トランザクション境界を保持します。
行	並べ替え済み	セッションが失敗します。
トランザクション	並べ替え済み	マスターおよび明細の両方が同じトランザクションジェネレータから生成されているものである場合、トランザクション境界を保持します。マスターおよび明細の両方が同じトランザクションジェネレータから生成されているものでない場合、セッションが失敗します。
トランザクション	未ソート	セッションが失敗します。
すべての入力	ソート済み、未ソート	トランザクション境界を削除します。

注: トランスフォーメーション範囲が [すべての入力] または [トランザクション] の場合にリアルタイムのデータを使用すると、セッションは失敗します。

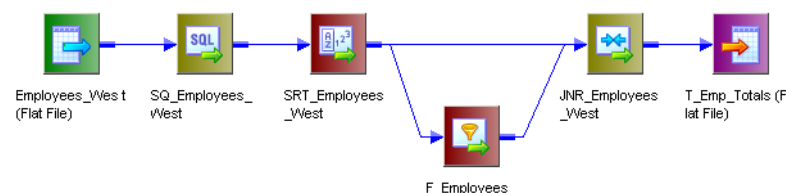
1つのパイプラインにおけるトランザクション境界の保持

同じソースからのデータを結合する場合、[トランザクション] トランスフォーメーション範囲を使用して、1つのパイプラインに対する入力トランザクション境界を保持します。ジョイナトランスフォーメーションが、同じソース、同じパイプラインの2つのブランチ、または1つのトランザクションジェネレータの2つの出力グループからのデータを結合する場合、[トランザクション] トランスフォーメーション範囲を使用します。このトランスフォーメーション範囲は、ソート済みデータおよび任意の結合タイプで使します。

[トランザクション] トランスフォーメーション範囲を使用する場合、マスターパイプラインおよび明細パイプラインが同じトランザクション制御点からのものであり、ソート済み入力を使用していることを確認してください。たとえば、[「1つのパイプラインにおけるトランザクション境界の保持」 \(ページ 289\)](#)ではソータトランスフォーメーションがトランザクション制御点です。ソータトランスフォーメーションとジョイナトランスフォーメーションの間に、別のトランザクション制御ポイントは設置できません。マッピングでは、マスタパイプラインブランチおよび明細パイプラインブランチは同じトランザクション制御ポイントから生成され、Integration Service はパイプラインブランチとジョイナトランスフォーメーションを結合し、トランザクション境界を保持します。

以下の図は、パイプラインの2つのブランチを結合し、トランザクション境界を保持するマッピングを示しています。

図 1.2つのパイプラインブランチを結合した場合のトランザクション境界の保持



明細パイプラインにおけるトランザクション境界の保持

明細パイプラインでトランザクション境界を保持する場合、[行] トランスフォーメーション範囲を選択します。[行] トランスフォーメーション範囲を使用すると、Integration Service はデータを 1 行ずつ処理できます。Integration Service は、マスターデータをキャッシュに格納し、そのマスターデータと明細データを照合します。

ソースデータが IBM MQ Series などのリアルタイムのソースからのものである場合、Integration Service はキャッシュに格納されたマスターデータと明細ソースから読み込まれる各メッセージとを照合します。

ノーマルまたは Master Outer の結合タイプで未ソートデータを使用する場合、[行] トランスフォーメーション範囲を使用します。

2 つのパイプラインのトランザクション境界の削除

2 つのソースまたは 2 つのブランチからのデータを結合する場合、トランザクション境界を保持する必要がなければ [すべての入力] トランスフォーメーション範囲を使用します。[すべての入力] を使用すると、Integration Service は両方のパイプラインについて入力トランザクション境界を削除し、トランスフォーメーションからのすべての行をオープントランザクションとして出力します。ジョイナトランスフォーメーションでは、ソート順の設定に応じて、マスターパイプラインからのデータを同時にキャッシュまたは結合できます。このトランスフォーメーション範囲は、ソート済みデータ、未ソートデータ、任意の結合タイプで使用します。

ジョイナトランスフォーメーションの作成

マッピングでジョイナトランスフォーメーションを使用するためには、ジョイナトランスフォーメーションをマッピングに追加し、入力ソースを設定し、トランスフォーメーションの条件と結合タイプ、およびソートタイプを設定します。

ジョイナトランスフォーメーションを作成するには：

1. Mapping Designer で、[トランスフォーメーション] - [作成] をクリックします。ジョイナトランスフォーメーションを選択します。名前を入力し、[OK] をクリックします。
ジョイナトランスフォーメーションの命名規則は「JNR_トランスフォーメーション名」です。トランスフォーメーションの説明を入力します。
Designer がジョイナトランスフォーメーションを作成します。
2. 1 番目のソースからジョイナトランスフォーメーションに、すべての入出力ポートをドラッグします。
デフォルトでは、Designer はジョイナトランスフォーメーション内のソースフィールドで入出力ポートを明細フィールドとして作成します。このプロパティはあとで編集することができます。
3. 2 番目のソースですべての入出力ポートを選択し、ジョイナトランスフォーメーションにドラッグします。
Designer はデフォルトで、ソースフィールドとマスターフィールドを 2 番目のセットとして設定します。
4. ジョイナトランスフォーメーションのタイトルバーをダブルクリックして、トランスフォーメーションを開きます。
5. このトランスフォーメーションには以下のポートがあります。
6. [M] カラムの任意のボックスをクリックしてソースのマスター/明細関係を切り替えます。
ヒント: 未ソートジョイナトランスフォーメーションのパフォーマンスを高めるには、行の比較の少ないソースをマスターソースとして使用します。ソート済みジョイナトランスフォーメーションのパフォーマンスを高めるには、重複キー値の少ないソースをマスターとして使用します。
7. 特定のポートのデフォルト値を追加します。

ソースのフィールドには空のものもあるため、一部のポートに NULL 値が含まれることもあります。ターゲットデータベースが NULL を扱わない場合、デフォルト値を指定できます。

8. [条件] タブをクリックして、ジョイン条件を設定します。
9. [追加] をクリックして条件を追加します。複数の条件を追加することができます。
マスターポートと明細ポートのデータタイプは一致しなければなりません。ジョイナトランスフォーメーションは等価 (=) 結合だけをサポートしています。
10. [プロパティ] タブをクリックし、トランスフォーメーションのプロパティを設定します。
注: ジョイン条件は、[条件] タブから編集できます。AND キーワードは複数の条件を区切ります。
11. [OK] をクリックします。
12. [メタデータエクステンション] タブを選択し、メタデータエクステンションを設定します。

ジョイナトランスフォーメーションに関するヒント

可能な場合は、データベース内で結合を実行します。

データベース内で結合を実行すると、セッション内で実行する場合よりも処理が高速になります。場合によっては、たとえば 2 つの異なるデータベースまたはフラットファイルシステムのテーブルの結合では、これが不可能なこともあります。データベース内で結合を実行したい場合は、次の選択肢があります。

- セッション実行前に起動されるストアドプロシージャを作成して、データベース内でテーブルを結合する。
- ソース修飾子トランスフォーメーションを使用して結合を実行する。

可能な場合は、ソート済みデータを結合します。

ジョイナトランスフォーメーションでソート済み入力を使用するように設定することで、セッションのパフォーマンスを向上させることができます。ジョイナトランスフォーメーションでソート済みデータを使うように設定すると、Integration Service はディスクの入出力を最小化してパフォーマンスを高めます。大量のデータセットを扱う場合に、パフォーマンスを最大限に向上させることができます。

未ソートジョイナトランスフォーメーションの場合、行の比較的少ないソースをマスターソースとして指定します。

パフォーマンスとディスク容量の最適化のために、行数が少ないソースをマスターソースとして指定します。セッション中に、ジョイナトランスフォーメーションはマスターソースの各行を明細ソースと比較します。マスター内の一意な行が少なければ、結合のための比較が繰り返される回数も少なくなり、その結果、結合プロセスが高速になります。

ソート済みジョイナトランスフォーメーションでは、重複するキー値の少ないソースをマスターソースとして指定します。

パフォーマンスとディスク容量の最適化のために、重複するキー値の少ないソースをマスターソースとして指定します。Integration Service は、ソート済みジョイナトランスフォーメーションを処理するときに、一度に 100 個のキーに対する行をキャッシュします。マスターソースに同じキー値を持つ多数の行が含まれる場合、Integration Service はより多くの行をキャッシュに格納する必要があり、パフォーマンスが低下することがあります。

第 17 章

ルックアップトランスフォーメーション

この章では、以下の項目について説明します。

- [ルックアップトランスフォーメーションの概要, 292 ページ](#)
- [ルックアップソースのタイプ, 293 ページ](#)
- [接続されたルックアップと接続されていないルックアップ, 296 ページ](#)
- [ルックアップのコンポーネント, 299 ページ](#)
- [ルックアッププロパティ, 301 ページ](#)
- [ルックアップクエリ, 308 ページ](#)
- [ルックアップ条件, 312 ページ](#)
- [ルックアップキャッシュ, 314 ページ](#)
- [戻り値としての複数の行, 315 ページ](#)
- [接続されていないルックアップトランスフォーメーションの設定, 316 ページ](#)
- [データベースデッドロックに対するレジリエンス, 318 ページ](#)
- [ルックアップトランスフォーメーションの作成, 318 ページ](#)
- [ルックアップトランスフォーメーションのヒント, 320 ページ](#)

ルックアップトランスフォーメーションの概要

マッピング内でルックアップトランスフォーメーションを使用して、フラットファイル、リレーショナルテーブル、ビュー、またはシノニムのデータをルックアップできます。PowerCenter クライアントおよび Integration Service の両方の接続先となる任意のフラットファイルまたはリレーショナルデータベースから、ルックアップ定義をインポートできます。また、ソース修飾子からルックアップ定義を作成することもできます。マッピングでは、複数のルックアップトランスフォーメーションを使用できます。ルックアップトランスフォーメーションは、アクティブまたはパッシブなトランスフォーメーションにすることができます。接続されたまたは接続されていないルックアップトランスフォーメーションを設定できます。

Integration Service は、トランスフォーメーションのルックアップポートおよびルックアップ条件に基づいて、ルックアップソースに対してクエリを実行します。ルックアップトランスフォーメーションは、ルックアップの結果をターゲットまたは別のトランスフォーメーションに返します。単一の行または複数の行を返すように、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ができます。

ルックアップトランスフォーメーションでは、以下のタスクを実行します。

- **関連する値を取得する。**ソースの値に基づいてルックアップテーブルから値を取得します。たとえば、ソースに従業員 ID があるものとします。ルックアップテーブルから従業員名を取得します。
- **複数の値を取得する。**ルックアップテーブルから複数の行を取得します。例えば、部門内のすべての従業員を返します。
- **計算を行う。**ルックアップテーブルから値を取得し、その値を計算に使用します。例えば、消費税率を取得し、税額を計算してターゲットに返します。
- **緩やかに変化する次元テーブルを更新する。**行がターゲットに存在するかどうかを判断します。

以下のタイプのルックアップを実行するようにルックアップトランスフォーメーションを設定します。

- **リレーショナルルックアップまたはフラットファイルルックアップ。**フラットファイルまたはリレーショナルテーブルに対してルックアップを実行します。リレーショナルテーブルをルックアップソースとして使用してルックアップトランスフォーメーションを作成すると、ODBC を使用してルックアップソースに接続し、そのテーブル定義をルックアップトランスフォーメーションの構造としてインポートできます。ルックアップソースとしてフラットファイルを使ったルックアップトランスフォーメーションを定義する（ファイル定義よりインポートする）際には、Designer 上はウィザードを起動して、フォーマット定義を支援いたします。
- **パイプラインルックアップ。**JMS や MSMQ などのアプリケーションソースに対してルックアップを実行します。ソースをマッピングにドラッグし、ルックアップトランスフォーメーションをソース修飾子に関連付けます。Integration Service がルックアップキャッシュのソースデータを取得するときにパフォーマンスが向上するようにパーティションを設定します。
- **接続されたルックアップまたは接続されていないルックアップ**接続されたルックアップトランスフォーメーションは、ソースデータを受け取り、ルックアップを実行し、パイプラインにデータを返します。コネクタされていないルックアップトランスフォーメーションは、ソースにもターゲットにも接続されません。パイプラインのトランスフォーメーションが、!LKP 式でルックアップトランスフォーメーションを呼び出します。コネクタされていないルックアップトランスフォーメーションは、呼び出し元のトランスフォーメーションにカラムを 1 つ返します。
- **キャッシュを使用するルックアップおよびキャッシュを使用しないルックアップ。**パフォーマンスを高めるには、ルックアップソースをキャッシュします。ルックアップソースをキャッシュする場合、動的キャッシュまたは静的キャッシュを使用できます。デフォルトでは、ルックアップキャッシュは静的であり、セッションの実行中に変化しません。動的キャッシュでは、Integration Service はキャッシュに行を挿入したり、キャッシュ内の行を更新したりします。ターゲットテーブルをルックアップソースとしてキャッシュすると、キャッシュ内の値をルックアップして値がターゲットに存在するかどうかを調べることができます。ルックアップトランスフォーメーションは、ターゲットを挿入または更新する行にマークを付けます。

ルックアップソースのタイプ

ルックアップトランスフォーメーションを作成するとき、ルックアップソースとしてリレーショナルテーブル、フラットファイル、またはソース修飾子を選択できます。

リレーショナルルックアップ

リレーショナルテーブルをルックアップソースとして使用してルックアップトランスフォーメーションを作成すると、ODBC を使用してルックアップソースに接続し、そのテーブル定義をルックアップトランスフォーメーションの構造としてインポートできます。

リレーショナルルックアップでは、以下のオプションを使用します。

- デフォルトの SQL 文を上書きして、WHERE 句の追加や複数のテーブルへのクエリを実行します。
- データベースサポートに基づいて、NULL データを昇順または降順にソートします。
- データベースサポートに基づいて、大文字と小文字を区別した比較を実行します。

フラットファイルルックアップ

フラットファイルをルックアップソースとして使用してルックアップトランスフォーメーションを作成するときは、リポジトリでフラットファイル定義を選択するか、またはトランスフォーメーション作成のソースをインポートします。フラットファイルルックアップソースをインポートする場合、Designer がフラットファイルウィザードを起動します。

フラットファイルルックアップでは、以下のオプションを使用します。

- ルックアップファイル名としてファイルリストを設定すると、間接ファイルをルックアップソースに使用できます。
- ルックアップでは、ソート済み入力を使用できます。
- NULL データを昇順または降順にソートします。
- フラットファイルルックアップでは、大文字と小文字を区別した文字列の比較を実行できます。

ソート済み入力の使用

ソート済み入力用にフラットファイルルックアップトランスフォーメーションを設定する場合、条件カラムをグループ化する必要があります。条件カラムがグループ化されていないと、ルックアップトランスフォーメーションは誤った結果を返します。最適なキャッシュパフォーマンスを得るには、条件カラムをソートします。

たとえば、ルックアップトランスフォーメーションが次の条件を持つとします。

```
OrderID = OrderID1  
CustID = CustID1
```

下のフラットファイルルックアップソースでは、キーはグループ化されていますが、ソートされていません。Integration Service はデータをキャッシュできますが、パフォーマンスが最適でない可能性があります。

OrderID	CustID	ItemNo.	ItemDesc	Comments
1001	CA502	F895S	Flashlight	Key data is grouped, but not sorted. CustID is out of order within OrderID.
1001	CA501	C530S	Compass	
1001	CA501	T552T	Tent	-
1005	OK503	S104E	Safety Knife	Key data is grouped, but not sorted. OrderID is out of order.
1003	CA500	F304T	First Aid Kit	
1003	TN601	R938M	Regulator System	-

以下のフラットファイルルックアップソースでは、キーがグループ化されていません。ルックアップトランスフォーメーションは誤った結果を返します。

OrderID	CustID	ItemNo.	ItemDesc	Comments
1001	CA501	T552T	Tent	-
1001	CA501	C530S	Compass	-
1005	OK503	S104E	Safety Knife	-
1003	TN601	R938M	Regulator System	-
1003	CA500	F304T	First Aid Kit	-
1001	CA502	F895S	Flashlight	

Key data for CustID is not grouped.

間接ファイルでソート済み入力を選択する場合は、データの範囲はファイル内で重ならないようにします。

パイプラインルックアップ

リレーショナルテーブルでもフラットファイルでもないアプリケーションソースに対してルックアップを実行するには、パイプラインルックアップトランスフォーメーションを作成します。パイプラインルックアップトランスフォーメーションでは、ソース修飾子がルックアップソースとなります。パイプラインルックアップは、アプリケーションマルチグループソース修飾子トランスフォーメーション以外のすべてのデータソースに対して実行できます。

パイプライン Lookup トランスフォーメーションを設定すると、ルックアップソースおよび Source Qualifier は Lookup トランスフォーメーションとは別のパイプラインに存在します。ソースおよび Source Qualifier は、ターゲットが含まれていない部分パイプラインにあります。Integration Service は、このパイプラインのソースデータを読み込んで、キャッシュを作成する Lookup トランスフォーメーションに渡します。パフォーマンスを高めるには、部分パイプラインに複数のパーティションを作成します。

リレーショナルルックアップソースまたはフラットファイルルックアップソースを処理するときにパフォーマンスを高めるには、リレーショナルやフラットファイルのルックアップトランスフォーメーションではなく、パイプラインルックアップトランスフォーメーションを作成します。ルックアップソースを処理してルックアップトランスフォーメーションに渡すパーティションを作成できます。

接続されたか、コネクタされていないパイプラインルックアップトランスフォーメーションを作成します。

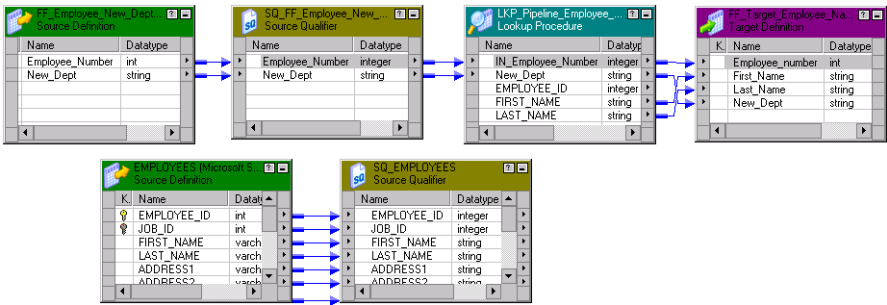
注: パイプラインルックアップに対するリアルタイムソースがあるセッションについては、HA リカバリを有効にしないようにします。予期しない結果になる可能性もあります。

マッピングでのパイプラインルックアップトランスフォーメーションの設定

パイプラインルックアップトランスフォーメーションが含まれているマッピングには、ルックアップソースおよびソース修飾子が含まれている部分パイプラインがあります。部分パイプラインにはターゲットがありません。Integration Service は、このパイプラインのルックアップソースデータを取得して、ルックアップキャッシュに渡します。

部分パイプラインは、セッションプロパティの独立したターゲットロード順グループにあります。パフォーマンスを高めるには、部分パイプラインに複数のパーティションを作成します。部分パイプラインでは、ターゲットロード順を設定できません。

以下のマッピングでは、パイプラインルックアップトランスフォーメーションのほか、ルックアップソースを処理する部分パイプラインも含まれているマッピングを示しています。



マッピングには次のオブジェクトが含まれます。

- ルックアップソース定義とソース修飾子は別のパイプラインにあります。Integration Service は、パイプラインのルックアップソースデータを処理した後、ルックアップキャッシュを作成します。
- フラットファイルソースには、従業員番号別に新しい部門名が記載されています。
- パイプラインルックアップトランスフォーメーションは、このソースファイルから Employee_Number および New_Dept を受け取ります。パイプラインルックアップは、ルックアップキャッシュの Employee_ID に対してルックアップを実行します。次に、ルックアップキャッシュから従業員の姓名を取得します。
- フラットファイルターゲットが、ルックアップトランスフォーメーションから Employee_ID、First_Name、Last_Name、および New_Dept を受け取ります。

接続されたルックアップと接続されていないルックアップ

接続されたまたは接続されていないルックアップトランスフォーメーションを設定できます。接続されたルックアップトランスフォーメーションは、マッピング内の他のトランスフォーメーションに接続する入出力ポートのあるトランスフォーメーションです。接続されていないルックアップトランスフォーメーションは、マッピングに含まれますが、他のトランスフォーメーションに接続されていません。

接続されていないルックアップトランスフォーメーションは、トランスフォーメーション（式トランスフォーメーションやアグリゲータトランスフォーメーションなど）の:LKP 式の結果から入力を受け取ります。:LKP 式は、ルックアップトランスフォーメーションに引数を渡し、ルックアップトランスフォーメーションから結果を受け取ります。:LKP 式は、ルックアップの結果を式トランスフォーメーションやアグリゲータトランスフォーメーションの別の式に渡して、結果をフィルタすることができます。

次の表に、接続されたルックアップと接続されていないルックアップの違いを示します。

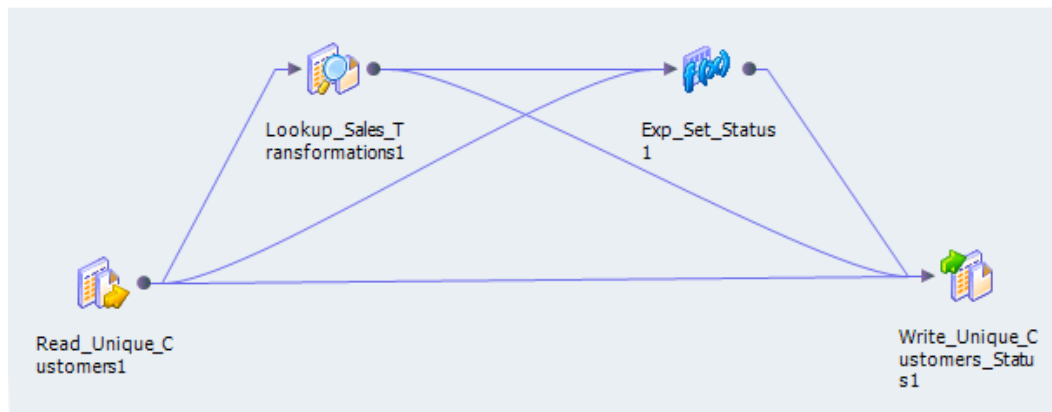
接続されたルックアップ	接続されていないルックアップ
入力値をパイプラインから直接受け取ります。	入力値を別のトランスフォーメーションの:LKP 式の結果から受け取ります。
動的キャッシュまたは静的キャッシュを使用します。	静的キャッシュを使用します。

接続されたルックアップ	接続されていないルックアップ
キャッシュには、ルックアップ条件のルックアップソースカラムおよび出力ポートとなるルックアップソースカラムが含まれています。	キャッシュには、ルックアップ条件のすべてのルックアップポート/出力ポートおよびルックアップポート/戻りポートが入っています。
同じ行から複数のカラムを返して、動的ルックアップのキャッシュに挿入します。	各行から戻りポートに 1 つのカラムを返します。
ルックアップ条件に一致するものがない場合、統合サービスはすべての出力ポートについてデフォルト値を返します。統合サービスで動的キャッシュが設定されている場合、キャッシュに行が挿入されるか、またはキャッシュは未変更のままとなります。	ルックアップ条件に一致するものがない場合、統合サービスは NULL を返します。
ルックアップ条件に一致するものがある場合、統合サービスはすべてのルックアップ/出力ポートについてルックアップ条件の結果を返します。統合サービスで動的キャッシュが設定されている場合、キャッシュ内の行が更新されるか、または未変更のままのどちらかとなります。	ルックアップ条件で一致があれば、データ統合サービスは戻りポートにルックアップ条件の結果を返します。
複数の出力値を別のトランスフォーメーションに渡します。ルックアップ/出力ポートを別のトランスフォーメーションにリンクします。	1 つの出力値を別のトランスフォーメーションに返します。ルックアップトランスフォーメーションの戻りポートは、他のトランスフォーメーションの:LKP 式が含まれるポートに値を渡します。
ユーザ定義デフォルト値をサポートします。	ユーザ定義デフォルト値をサポートしません。

接続されたルックアップ

接続されたルックアップトランスフォーメーションとは、マッピング内でソースまたはターゲットに接続されているルックアップトランスフォーメーションのことです。

次の図に、接続されたルックアップトランスフォーメーションとのマッピングを示します。



接続されたルックアップトランスフォーメーションが含まれているマッピングを実行すると、データ統合サービスは以下の手順を実行します。

1. データ統合サービスが値を別のトランスフォーメーションからルックアップトランスフォーメーションの入力ポートに渡します。

2. 各入力行に対して、データ統合サービスがトランスフォーメーションのルックアップポートおよびルックアップ条件に基づいて、ルックアップソースまたはキャッシュにクエリを実行します。
3. トランスフォーメーションがキャッシュを使用していない場合、または静的キャッシュを使用している場合、データ統合サービスはルックアップクエリから値を返します。

トランスフォーメーションが動的キャッシュを使用している場合、Integration Service は行を見つけられなかったキャッシュにその行を挿入します。Integration Service でキャッシュ内に行が検出された場合、キャッシュ内の行が更新されるかまたは未変更のままとなります。行に対して挿入、更新、または変更なしのフラグを設定します。

4. データ統合サービスがクエリからデータを返して、それをマッピング内の次のトランスフォーメーションに渡します。

トランスフォーメーションが動的キャッシュを使用している場合、行をフィルタまたはルータトランスフォーメーションに渡し、ターゲットへの新しい行をフィルタリングすることができます。

注: この章では、特に明示しない限り、接続されたルックアップトランスフォーメーションについて説明します。

接続されていないルックアップ

接続されていないルックアップトランスフォーメーションは、マッピング内のソースまたはターゲットに接続していないルックアップトランスフォーメーションです。式を使用できるトランスフォーメーションの:LKP 式を使用して、ルックアップを呼び出します。

コネクタされていないルックアップトランスフォーメーションは、一般に緩やかに変化する次元テーブルの更新に使用されます。緩やかに変化する次元テーブルの詳細については、Informatica ナレッジベース記事 (<http://mysupport.informatica.com>) を参照してください。

ルックアップ式の構文は、:LKP lookup_transformation_name(argument, argument, ...) です。

各引数を記述する順序は、ルックアップトランスフォーメーションのルックアップ条件の順序と一致しなければなりません。ルックアップトランスフォーメーションは、ルックアップトランスフォーメーションの戻りポートを介して、クエリの結果を返します。ルックアップを呼び出すトランスフォーメーションは、:LKP 式が含まれるポートでルックアップの結果値を受け取ります。ルックアップクエリが値を返せなかった場合、ポートは NULL 値を受け取ります。

接続されていないルックアップを実行するときは、マッピング内で同じルックアップを複数回実行できます。ルックアップの結果を別の式でテストし、その結果に基づいて行をフィルタすることができます。

接続されていないルックアップトランスフォーメーションを含むマッピングを実行すると、データ統合サービスは以下の手順を実行します。

1. 接続されていないルックアップトランスフォーメーションは、別のトランスフォーメーション（アグリゲータトランスフォーメーション、式トランスフォーメーション、アップデイトストラテジトランスフォーメーションなど）内の:LKP 式の結果から、入力値を受け取ります。
2. データ統合サービスは、ルックアップトランスフォーメーションのルックアップポートおよびルックアップ条件に基づき、ルックアップソースまたはキャッシュに対してクエリを実行します。
3. データ統合サービスはルックアップトランスフォーメーションの戻りポートを介して値を返します。
4. 統合サービスは、:LKP 式の入ったポートに戻り値を渡します。

ルックアップのコンポーネント

ルックアップトランスフォーメーションをマッピングに設定する場合、以下のコンポーネントを定義します。

- ルックアップソース
- ポート
- プロパティ
- 条件

ルックアップソース

ルックアップソースには、フラットファイル、リレーショナルテーブル、またはソース修飾子を使用します。ルックアップトランスフォーメーションを作成するとき、以下の場所からルックアップソースを作成できます。

- リポジトリ内のリレーショナルソース定義またはリレーショナルターゲット定義
- リポジトリ内のフラットファイルソース定義またはフラットファイルターゲット定義
- Integration Service および PowerCenter クライアントマシンから接続可能なテーブルまたはファイル
- マッピング内のソース修飾子定義

ルックアップテーブルを単一のテーブルにするか、またはルックアップ SQL オーバーライドを使用して同一のデータベースの複数のテーブルを結合することができます。Integration Service は、ルックアップテーブルまたはテーブルのインメモリキャッシュに対し、ルックアップトランスフォーメーションに入る行すべてについてクエリを実行します。

Integration Service は、ODBC ドライバまたはネイティブドライバを使用してルックアップテーブルに接続できます。最適なパフォーマンスを得るには、ネイティブドライバを設定します。

インデックスおよびルックアップテーブル

ルックアップテーブルを含むデータベースを変更する特権を持っている場合、ルックアップテーブルにインデックスを追加することで、ルックアップの初期化に要する時間を短縮することができます。サイズが非常に大きいルックアップテーブルでもパフォーマンスを高めることができます。Integration Service がルックアップカラムの値に対してクエリ、ソート、および比較を実行するため、インデックスにはルックアップ条件のどのカラムも含める必要があります。

以下の種類のルックアップにインデックスを付けることでパフォーマンスを向上させることができます。

- **キャッシュを使用するルックアップ。** ORDER BY ルックアップのカラムにインデックスを付けることで、パフォーマンスを向上させることができます。セッションログには ORDER BY 句が記録されます。
- **キャッシュを使用しないルックアップ。** Integration Service は、ルックアップトランスフォーメーションに渡される各行に対して SELECT 文を発行するため、ルックアップ条件のカラムにインデックスを付けることでパフォーマンスを向上させることが可能です。

ルックアップポート

[ポート] タブには、入力ポートおよび出力ポートが含まれています。この他、ルックアップソースから返されるデータのカラムとなるルックアップポートも含まれています。コネクタされていないルックアップトランスフォーメーションは、データのカラムを 1 つ、このポートの呼び出し元のトランスフォーメーションに返します。コネクタされていないルックアップトランスフォーメーションには、戻りポートが 1 つあります。

以下の表では、ルックアップトランスフォーメーションのポートタイプについて説明します。

ポート	ルックアップのタイプ	説明
I	接続された 接続されていない	入力ポート。ルックアップ条件に使用したいルックアップポートごとに入力ポートを作成します。それぞれのルックアップトランスフォーメーションにおいて少なくとも 1 つの入力または入出力ポートが必要です。
O	接続された 接続されていない	出力ポート。別のトランスフォーメーションにリンクしたいルックアップポートごとに出力ポートを作成します。入力ポートとルックアップポートの両方を出力ポートとして指定することができます。接続されたルックアップの場合、少なくとも 1 つの出力ポートが必要です。コネクต์されていないルックアップの場合、戻り値を渡す戻りポート (R) としてルックアップポートを選択します。
L	接続された 接続されていない	ルックアップポート。Designer は、ルックアップソース内の各カラムをルックアップポート (L) および出力ポート (O) として指定します。
R	接続されていない	戻りポート。コネクต์されていないルックアップトランスフォーメーションでのみ使用します。ルックアップ条件に基づいて返したいデータのカラムを指定します。1 つのルックアップポートを戻りポートとして指定できます。

ルックアップトランスフォーメーションは、動的キャッシュを使用する場合に設定する関連式プロパティを有効にすることもできます。関連式プロパティには、ルックアップキャッシュを更新するためのデータが含まれます。動的キャッシュを更新するための式を含めたり、入力ポート名を含めることもできます。

以下のガイドラインに従って、ルックアップポートを設定します。

- フラットファイルルックアップからルックアップポートを削除すると、セッションは失敗します。
- ルックアップポートはマッピングで使用されていない場合に限り、リレーショナルルックアップから削除できます。これにより、Integration Service がセッションを実行するために必要なメモリの量が削減されます。

ルックアップ プロパティ

[プロパティ] タブでは、リレーショナルルックアップの SQL 上書き、ルックアップソース名キャッシュ用プロパティなど、各種プロパティを設定します。

関連項目：

- [「ルックアッププロパティ」 \(ページ 301\)](#)

ルックアップ条件

[条件] タブでは、Integration Service がルックアップソースのデータを検索するのに使用する条件を入力します。

関連項目：

- [「ルックアップ条件」 \(ページ 312\)](#)

ルックアッププロパティ

[ルックアッププロパティ] タブでは、キャッシングや複数の一致などルックアッププロパティを設定します。ルックアップテーブルに対してクエリーを実行するには、ルックアップ条件または SQL 文を設定します。また、ルックアップテーブル名を変更することもできます。

マッピングを作成する場合は、ルックアップトランスフォーメーションごとにプロパティを設定します。セッションを作成する場合は、トランスフォーメーションごとにインデックスキャッシュサイズやデータキャッシュサイズなどのプロパティをオーバーライドできます。

以下の表では、ルックアップトランスフォーメーションのプロパティについて説明します。

オプション	ルックアップのタイプ	説明
Lookup SQL Override	Relational	ルックアップテーブルに問い合わせるデフォルト SQL 文を上書きします。 統合サービスがルックアップ値を問い合わせるのに使用する SQL 文を指定します。ルックアップキャッシュが有効な場合に使用します。
ルックアップテーブル名	パイプライン Relational	トランスフォーメーションがルックアップして値をキャッシュするテーブルまたはソース修飾子の名前。Lookup トランスフォーメーションを作成するときは、ルックアップソースとしてソース、ターゲット、または Source Qualifier を選択します。また、別のデータベースからテーブル、ビュー、またはシノニムをインポートすることもできます。 ルックアップ SQL 上書きを入力した場合は、ルックアップテーブル名を入力する必要がありません。
ルックアップソースフィルタ	Relational	ルックアップトランスフォーメーションのポートのデータの値に基づいて統合サービスが実行するルックアップを制限します。このオプションを指定する場合は、ルックアップキャッシュを有効にしてください。
ルックアップキャッシュを有効にする	フラットファイル パイプライン Relational	セッション中に統合サービスがルックアップ値をキャッシュに格納するかどうかを指定します。 ルックアップキャッシュを有効にすると、統合サービスはルックアップソースに対して一度クエリーを実行してから値をキャッシュに格納し、セッション中にキャッシュの値をルックアップします。ルックアップ値をキャッシュすると、セッションパフォーマンスを高めることができます。 キャッシュを無効にすると、行がトランスフォーメーションを通過するたびに、統合サービスはルックアップソースに Select 文を発行してルックアップ値を求めます。 注: 統合サービスは、フラットファイルルックアップとパイプラインルックアップを常にキャッシュします。

オプション	ルックアップのタイプ	説明
Lookup Policy on MultipleMatch	フラット ファイル パイプライン Relational	<p>ルックアップトランスフォーメーションがルックアップ条件に一致する複数の行を検出した場合に返す行を指定します。次のいずれかの値を選択します。</p> <ul style="list-style-type: none"> - [最初の値を使用]。ルックアップ条件に一致する最初の行を返します。 - [最後の値を使用]。ルックアップ条件に一致する最後の行を返します。 - [すべての値を使用]。一致するすべての行を返します。 - [任意の値を使用]。統合サービスは、ルックアップ条件に一致する最初の値を返します。すべてのルックアップトランスフォーメーションポートではなく、キーポートに基づいてインデックスを作成します。 - [エラーを報告]。統合サービスは、エラーを報告し、行を返しません。[更新時に古い値を出力] オプションを有効にしない場合、動的ルックアップの場合に [複数の一致に対するルックアップポリシー] オプションが [エラーを報告] に設定されます。
ルックアップ条件	フラット ファイル パイプライン Relational	[条件] タブで設定したルックアップ条件を表示します。
接続情報	Relational	<p>ルックアップテーブルを含むデータベースを指定します。データベースは、マッピング、セッション、またはパラメータファイルの中で定義できます。</p> <ul style="list-style-type: none"> - Mapping。接続オブジェクトを選択します。データベースの接続タイプを指定することもできます。リレーショナル接続の場合、接続名の先頭にリレーショナル:を入力します。アプリケーション接続の場合、接続名の先頭に Application:を入力します。 - セッション。接続変数\$Source または\$Target を使用します。これらの変数のいずれかを使用する場合、ルックアップテーブルはソースデータベースまたはターゲットデータベースに存在する必要があります。変数ごとにセッションプロパティにデータベース接続を指定します。 - パラメータファイル。セッションパラメータ\$DBConnection 名前または\$AppConnection 名前を使用し、これをパラメータファイル内で定義します。 <p>デフォルトでは、ルックアップトランスフォーメーションの作成時に、ソーステーブルを選択した場合は\$Source が、ターゲットテーブルを選択した場合は\$Target が、Designer によってそれぞれ指定されます。これらの値は、セッションプロパティで上書きできます。</p> <p>統合サービスがデータベース接続のタイプを決定できない場合、セッションは失敗します。</p>
ソースタイプ	フラット ファイル パイプライン Relational	Lookup トランスフォーメーションがリレーショナルテーブル、フラットファイル、または Source Qualifier から値を読み込むことを示します。

オプション	ルックアップのタイプ	説明
トレースレベル	フラット ファイル パイプライン Relational	セッションログに含める詳細の量を設定します。
Lookup キャッシュディレクトリ名	フラット ファイル パイプライン Relational	<p>ルックアップソースをキャッシュするようにルックアップトランスフォーメーションが設定されている場合に、ルックアップキャッシュファイルの作成先ディレクトリを指定します。永続ルックアップのオプションが選択されていると、永続ルックアップキャッシュファイルも保存されます。</p> <p>デフォルトでは、統合サービスは、統合サービスに対して設定されている\$PMCacheDir ディレクトリを使用します。</p>
Lookup CachePersistent	フラット ファイル パイプライン Relational	統合サービスが、2 つ以上のキャッシュファイルを含む永続ルックアップキャッシュを使用するかどうかを示します。ルックアップトランスフォーメーションが永続ルックアップキャッシュを使用するように設定されていても永続ルックアップキャッシュが存在しない場合、統合サービスはセッション実行時にファイルを作成します。ルックアップキャッシュが有効な場合に使用します。
Lookup のデータキャッシュサイズ Lookup のインデックスキャッシュサイズ	フラット ファイル パイプライン Relational	<p>デフォルトは [Auto] です。統合サービスがメモリ内のデータキャッシュおよびインデックスに割り当てるメモリサイズの最大値を示します。キャッシュの数値を使用できます。パラメータファイルのキャッシュ値を使用するか、統合サービスを設定し、自動設定を使用してキャッシュサイズを設定します。キャッシュサイズが決定されるように統合サービスを設定した場合、キャッシュに割り当てられる最大メモリ量も設定できます。</p> <p>セッションの初期化の際に、統合サービスが設定された量のメモリを割り当てることができない場合、セッションは失敗します。統合サービスはすべてのデータのキャッシュデータをメモリに格納できない場合、ディスクにページングを行います。</p> <p>ルックアップキャッシュが有効な場合に使用します。</p>
動的ルックアップキャッシュ	フラット ファイル パイプライン Relational	<p>動的ルックアップキャッシュを使用するように指定します。ルックアップキャッシュがターゲットテーブルに行を渡すときに、ルックアップキャッシュに新しい行を挿入またはキャッシュの中の行を更新します。</p> <p>ルックアップキャッシュが有効な場合に使用します。</p>
更新時に古い値を出力	フラット ファイル パイプライン Relational	<p>動的キャッシュが有効な場合に使用します。このプロパティを有効にすると、統合サービスはルックアップ/出力ポートから古い値を出力します。統合サービスがキャッシュ内の行を更新する場合、入力データに基づいて行を更新する前にルックアップキャッシュに存在していた値を出力します。統合サービスがキャッシュ内に行を挿入する場合、NULL 値を出力します。</p> <p>このプロパティを無効にすると、統合サービスはルックアップ/出力ポートおよび入出力ポートから同じ値を出力します。</p> <p>このプロパティはデフォルトで有効になっています。</p>

オプション	ルックアップのタイプ	説明
動的キャッシュの更新条件	フラットファイル パイプライン Relational	動的キャッシュを更新するかどうかを示す式。ルックアップポートまたは入力ポートを使用して式を作成します。式には、入力値またはルックアップキャッシュの値を含めることができます。統合サービスは、条件が True でデータがキャッシュに存在する場合にキャッシュを更新します。動的キャッシュが有効な場合に使用します。デフォルトは true です。
キャッシュファイル名のプレフィックス	フラットファイル パイプライン Relational	<p>永続ルックアップキャッシュで使います。永続ルックアップキャッシュファイルに使用するファイル名の接頭語を指定します。統合サービスは、ディスクに保存する永続キャッシュファイルの名前としてファイル名の接頭語を使用します。接頭語を入力します。 .idx または .dat と入力しないでください。</p> <p>ファイル名の接頭辞ごとにパラメータまたは変数を入力することができます。パラメータファイルで定義可能なパラメータまたは変数タイプを使用します。</p> <p>名前付き永続キャッシュファイルが存在する場合、統合サービスはそれらのファイルからメモリキャッシュを構築します。名前付き永続キャッシュファイルが存在しない場合、統合サービスは永続キャッシュファイルを再構築します。</p>
ルックアップソースからのキャッシュの再構築	フラットファイル パイプライン Relational	<p>ルックアップキャッシュが有効な場合に使用します。これを選択すると、統合サービスはルックアップトランスフォーメーションのインスタンスを最初に呼び出すときに、ルックアップソースからルックアップキャッシュを再構築します。</p> <p>永続ルックアップキャッシュを使用する場合、PowerCenter Server はキャッシュを使用する前に永続キャッシュファイルを再構築します。永続ルックアップキャッシュを使用しない場合、PowerCenter Server はキャッシュを使用する前にルックアップキャッシュをメモリに再構築します。</p>
挿入または更新	フラットファイル パイプライン Relational	動的キャッシュが有効な場合に使用します。行のタイプが「挿入」で、ルックアップトランスフォーメーションに入力される行に対して使用します。有効にすると、統合サービスは行をキャッシュに挿入し、既存の行を更新します。無効にすると、統合サービスは既存の行を更新しません。
更新(または挿入)	フラットファイル パイプライン Relational	<p>動的キャッシュが有効な場合に使用します。行のタイプが「更新」で、ルックアップトランスフォーメーションに入力される行に対して使用します。</p> <p>有効にすると、統合サービスは既存の行を更新し、行を挿入します。無効にすると、Integration Service は新しい行を挿入しません。</p>
Datetime フォーマット	フラットファイル	<p>日時フォーマットを選択するには、[開く] をクリックします。形式およびフィールド幅を定義します。ミリ秒、マイクロ秒、ナノ秒のフォーマットは、29 のフィールド幅になります。</p> <p>あるポートについて日時形式を選択しなかった場合、任意の日時形式を入力できます。デフォルトは、MM/DD/YYYY HH24:MI:SS です。日時形式によってポートのサイズが変わることはありません。</p>

オプション	ルックアップのタイプ	説明
桁区切り記号	フラットファイル	ポートの 1000 ごとの区切り文字を指定しない場合、統合サービスはここで定義されたプロパティを使用します。 区切り記号なし、カンマ、またはピリオドを選択できます。デフォルトは区切り記号なしです。
小数点記号	フラットファイル	ルックアップ定義または「ポート」タブで特定のフィールドに対して小数点記号を指定しない場合、統合サービスはここで定義されたプロパティを使用します。 小数点記号には、カンマまたはピリオドを選択できます。デフォルトはピリオドです。
Case-SensitiveString Comparison	フラットファイル パイプライン	統合サービスは、文字列のカラムに対してルックアップを実行する場合、大文字と小文字を区別した比較を使用します。 リレーショナルルックアップの場合、大文字と小文字を区別した比較はデータベースでサポートされているかどうかによって異なります。
Null Ordering	フラットファイル パイプライン	統合サービスによる NULL 値の順序付けの方法を決定します。 NULL 値を昇順にソートするか降順にソートするかを選択できます。デフォルトでは、統合サービスは NULL 値を昇順にソートします。これによって、統合サービスの設定が上書きされ、比較演算子において NULL 値の扱いを昇順、降順または NULL とします。 リレーショナルルックアップでは、NULL の順序はデータベースのデフォルト値によって決まります。
ソート済み入力	フラットファイル パイプライン	ルックアップファイルのデータがソートされているかどうかを示します。このオプションを選択すると、ファイルのルックアップパフォーマンスが向上します。ソート済み入力が無効化されていて条件カラムがグループ化されていない場合、統合サービスはセッションに失敗します。条件カラムがグループ化されていてソートされていない場合、統合サービスはソート済み入力が設定されていない場合と同様にルックアップを処理します。
ルックアップソースは静的です	フラットファイル パイプライン Relational	ルックアップソースは、セッション中に変化することがありません。

オプション	ルックアップのタイプ	説明
ルックアップキャッシュの事前作成	フラットファイル パイプライン Relational	<p>統合サービスは、ルックアップトランスフォーメーションがデータを受け取る前に、ルックアップキャッシュを作成できます。統合サービスは、複数のルックアップキャッシュファイルを同時に作成してパフォーマンスを向上させることができます。</p> <p>この設定は、マッピングまたはセッションで設定できます。ルックアップトランスフォーメーションオプションを「自動」に設定した場合、統合サービスはセッションレベルの設定を使用します。</p> <p>以下のいずれかのオプションを設定します。</p> <ul style="list-style-type: none"> - 自動。統合サービスは、このセッションに設定された値を使用します。 - 常に許可。統合サービスは、ルックアップトランスフォーメーションが最初のソース行を受け取る前に、ルックアップキャッシュを作成できます。統合サービスは追加のパイプラインを作成して、キャッシュを作成します。 - 常に不許可。統合サービスは、ルックアップトランスフォーメーションが最初の行を受け取るまではルックアップキャッシュを作成できません。 <p>統合サービスが同時に作成できるパイプラインの数を設定する必要があります。ルックアップキャッシュ作成セッションのプロパティ用に追加のコンカレントパイプラインを設定します。このプロパティがゼロより大きい場合、統合サービスはルックアップキャッシュを事前に作成できます。</p>
サブ秒の精度	Relational	<p>日時ポートのサブ秒の精度を指定します。</p> <p>リレーショナルルックアップの場合は、日付/時刻データの位取りを編集できるデータベースについては、精度を変更できます。サブ秒の精度を変更できるデータ型は、Oracle Timestamp、Informix Datetime、および Teradata Timestamp です。</p> <p>0-9 の正の整数値を入力してください。デフォルトは 6 マイクロ秒です。プッシュダウンの最適化を有効にすると、データベースはサブ秒の精度の設定に関係なく、完全な日時値を返します。</p>

セッションにおけるルックアッププロパティの設定

セッションを設定するときに、セッションに固有のルックアッププロパティを設定できます。

- **フラットファイルルックアップ。** ソースファイルディレクトリ、ファイル名、ファイルタイプなどルックアップの場所に関する情報を設定します。
- **リレーショナルルックアップ。** セッションプロパティで \$Source 変数および \$Target 変数を定義できます。また、接続情報を上書きして、セッションパラメータ \$DBConnection 名前または \$AppConnection 名前を使用することもできます。
- **パイプラインルックアップ。** ソースファイルディレクトリ、ファイル名、ファイルタイプなどルックアップソースファイルのプロパティを設定します。ソースがリレーショナルテーブルまたはアプリケーションソースである場合は、接続情報を設定します。

セッションにおけるフラットファイルルックアップの設定

セッションでフラットファイルルックアップを設定するときは、「マッピング」タブの「トランスフォーメーションビュー」でルックアップソースファイルの各種プロパティを設定します。ルックアップトランスフォーメ

ーションを選択し、そのトランスフォーメーション用のセッションプロパティでフラットファイルプロパティを設定します。

以下の表に、フラットファイルルックアップ用に設定するセッションプロパティを示します。

プロパティ	説明
ルックアップソースファイルディレクトリ	ディレクトリ名を入力します。デフォルトでは、Integration Service はプロセス変数ディレクトリ\$PMLookupFileDirを検索し、ルックアップファイルを探します。 ユーザーは完全パスとファイル名を入力することができます。[Lookup SourceFilename] フィールドにディレクトリとファイル名の両方を指定した場合は、[Lookup SourceFileDirectory] フィールドはクリアします。Integration Service はセッションの実行時に、このフィールドと [ルックアップソースファイル名] フィールドを連結します。 また、\$InputFile 名前セッションパラメータを入力してもファイル名を設定できます。
Lookup SourceFilename	ルックアップファイルの名前。間接ファイルを使用する場合は、Integration Service に読み込ませる間接ファイルの名前を入力します。 また、ルックアップファイルパラメータ\$LookupFile 名前を入力して、セッションで使用するルックアップファイルの名前を変更することもできます。 [ソースファイルのディレクトリ] フィールドにディレクトリとファイル名のどちらも設定する場合は、[ルックアップソースファイル名] をクリアします。Integration Service は、セッションでは [ルックアップソースファイル名] を [ルックアップソースファイルディレクトリ] フィールドと連結して使用します。例えば、[ルックアップソースファイルディレクトリ] フィールドに「C:\lookup_data\201d」が含まれ、[ルックアップソースファイル名] フィールドに「filename.tx」が含まれているとします。Integration Service はセッションの開始時に「C:\lookup_data\filename.txt」を検索します。
Lookup SourceFiletype	ルックアップソースファイルにソースデータまたは同じファイルプロパティを持つファイルのリストを含めるかどうかを示します。ルックアップソースファイルにソースデータを含める場合には、[Direct] を選択します。ルックアップソースファイルにファイルのリストを含める場合には、[Indirect] を選択します。 [間接] を選択した場合、Integration Service はすべてのファイルに対して1つのキャッシュを作成します。間接ファイルでソート済み入力を使用する場合には、ファイル内のデータ範囲に重複がないことを確認します。データの範囲が重なる場合、Integration Service はルックアップを未ソートのルックアップソースとして処理します。

セッションにおけるリレーショナルルックアップの設定

セッションでリレーショナルルックアップを設定するときは、[マッピング] タブの [トランスフォーメーションビュー] でルックアップデータベースの接続を設定します。ルックアップトランスフォーメーションを選択し、そのトランスフォーメーション用のセッションプロパティで接続を設定します。

以下の中から、リレーショナルルックアップトランスフォーメーションの接続を設定するためのオプションを選択します。

- リレーショナル接続またはアプリケーション接続を選択します。
- 接続変数\$Source または\$Target を使用してデータベース接続を設定します。
- セッションパラメータ\$DBConnection 名前または\$AppConnection 名前を設定し、パラメータファイルでセッションパラメータを定義します。

セッションにおけるパイプラインルックアップの設定

セッションでパイプラインルックアップを設定するときは、[マッピング] タブのソースノードでルックアップソースファイルの場所またはルックアップテーブルの接続を設定します。ルックアップソースを表すソース修飾子を選択します。

ルックアップクエリ

統合サービスは、ルックアップトランスフォーメーションで設定したポートおよびプロパティに基づいてルックアップクエリを生成します。統合サービスは、最初の行がルックアップトランスフォーメーションに入ると、デフォルトのルックアップクエリを実行します。

リレーショナルテーブルにリレーショナルルックアップまたはパイプラインルックアップを使用する場合は、ルックアップクエリをオーバーライドできます。リレーショナルテーブルにルックアップを使用する場合は、ルックアップクエリをオーバーライドできます。キャッシュされる前に、オーバーライドを使用して、ORDER BY 句の変更、WHERE 句の追加、またはルックアップデータの変換を実行できます。キャッシュされる前に、オーバーライドを使用して、WHERE 句の追加、またはルックアップデータの変換を行えます。

ルックアップクエリに SQL オーバーライドとフィルタを設定すると、統合サービスはフィルタを無視します。

デフォルトのルックアップクエリ

デフォルトルックアップクエリには、以下の文が含まれます。

SELECT

SELECT 文には、マッピングのすべてのルックアップポートが含まれます。ルックアップクエリの SELECT 文を表示するには、[ルックアップ SQL オーバーライド] プロパティを選択します。

SELECT

SELECT 文には、マッピングのすべてのルックアップポートが含まれます。ルックアップクエリの SELECT 文を表示するには、[カスタムクエリを使用] プロパティを選択します。

ORDER BY

ORDER BY 句はカラムを、そのカラムがルックアップトランスフォーメーションに出現する順序と同じ順序に並べかえます。統合サービスが ORDER BY 句を生成します。これは、デフォルトの SQL を生成するときには表示できません。

ルックアップクエリのオーバーライド

ルックアップ SQL オーバーライドは、SourceQualifier トランスフォーメーションのユーザー作成のクエリーの入力と似ています。リレーショナルルックアップの場合、ルックアップクエリをオーバーライドできます。オーバーライド全体を入力するか、デフォルトの SQL 文を生成して編集することもできます。Designer によって生成されたルックアップ SQL オーバーライドのデフォルト SQL 文には、ルックアップ条件のルックアップ/出力ポートおよびルックアップ/戻りポートが含まれます。

別のルックアップクエリを入力したり、ルックアップポート、出力ポート、戻りポートを含む既存のルックアップクエリを編集できます。

ORDER BY 句のオーバーライド

デフォルトでは、統合サービスはキャッシュされたルックアップに対して ORDER BY 句を生成します。ORDER BY 句にはすべてのルックアップ条件ポートが含まれます。パフォーマンスを向上させるには、デフォルトの ORDER BY 句を抑止し、カラム数の少ないオーバーライド ORDER BY を入力します。

注: オーバーライドする SQL は、ルックアップキーに格納されているデータを返す必要があります。トランスフォーメーションでルックアップのすべての行を取得したら、統合サービスはソートされた順序のキーでデータキャッシュを作成します。行がソートされていない場合、統合サービスでキャッシュからすべての行を取得することはできません。キーでデータがソートされていない場合、予期しない結果が生じることがあります。

プッシュダウンの最適化を使用する場合、ORDER BY 句をオーバーライドしたり、生成した ORDER BY 句にコメント表示して抑止したりすることはできません。

オーバーライドで ORDER BY 句を入力しても、統合サービスは常に ORDER BY 句を生成します。ORDER BY オーバーライドの後に 2 個のダッシュ (--) を挿入して、生成した ORDER BY 句を抑止します。たとえば、ルックアップ トランスフォーメーションが次のルックアップ条件を使用するとします。

```
ITEM_ID = IN_ITEM_ID
```

```
PRICE <= IN_PRICE
```

ルックアップトランスフォーメーションには、マッピングで使用される 2 つのルックアップ条件ポート (ITEM_ID、および PRICE) が含まれています。1 つ以上のカラムを持つ ORDER BY 句を入力する場合、ルックアップ条件にポートを入力したのと同じ順番でカラムを入力します。また、すべてのデータベース予約語は引用符で囲みます。ルックアップ SQL オーバーライドで次のルックアップクエリーを入力します。

```
SELECT ITEMS_DIM.ITEM_NAME AS ITEM_NAME, ITEMS_DIM.PRICE AS PRICE, ITEMS_DIM.ITEM_ID AS ITEM_ID FROM ITEMS_DIM  
ORDER BY ITEMS_DIM.ITEM_ID --
```

リレーショナルルックアップでデフォルトの ORDER BY 句をオーバーライドするには、以下の手順を実行します。

1. ルックアップトランスフォーメーションでルックアップクエリーを生成します。
2. ルックアップ条件で表示される順序どおりの条件ポートを含む ORDER BY 句を入力します。
3. ORDER BY 句の後にコメントを示す 2 個のダッシュ (--) を入れて、統合サービスが生成する ORDER BY 句を抑止します。

コメント指定を追加せずに ORDER BY 句を持つルックアップクエリーをオーバーライドすると、ルックアップは失敗します。

注: Sybase には 16 カラムの ORDER BY 制限があります。ルックアップトランスフォーメーションに 17 個以上のルックアップ/出力ポート (ルックアップ条件のポートを含む) がある場合は、ORDER BY 句をオーバーライドするか、複数のルックアップトランスフォーメーションを使用してルックアップテーブルに対してクエリーを実行します。

予約語

MONTH や YEAR などのデータベース予約語を含むルックアップ名またはカラム名がある場合、統合サービスがデータベースに対して SQL を実行すると、セッションが失敗し、データベースエラーが表示されます。予約語を格納するファイル (reswords.txt) は、統合サービスのインストールディレクトリに作成して管理できます。

予約語を格納するファイル (reswords.txt) は、統合サービスがインストールされているディレクトリで作成および管理することができます。統合サービスは、セッションを初期化するときに、reswords.txt ファイル内を検索し、予約語の前後に引用符を挿入してから、ソース、ターゲット、およびルックアップデータベースに対して SQL を実行します。

予約語を格納するファイル (reswords.txt) は、統合サービスがインストールされているディレクトリで作成および管理することができます。統合サービスは、マッピングを初期化するときに、reswords.txt ファイル内

を検索し、予約語の前後に引用符を挿入してから、ソース、ターゲット、およびルックアップデータベースに対して SQL を実行します。

引用符で括った識別子に SQL-92 標準を使用するには、場合によって Microsoft SQL Server や Sybase などいくつかのデータベースを有効にすることが必要です。接続環境 SQL を使用して、コマンドを発行することができます。たとえば Microsoft SQL Server では、次のコマンドを使用できます。

引用符で括った識別子に SQL-92 標準を使用するには、場合によって Microsoft SQL Server や Sybase などいくつかのデータベースを有効にすることが必要です。環境 SQL を使用してコマンドを発行します。たとえば Microsoft SQL Server では、次のコマンドを使用できます。

```
SET QUOTED_IDENTIFIER ON
```

ルックアップクエリのオーバーライドのガイドライン

ルックアップクエリを上書きするときに、一定のルールとガイドラインが適用されます。

ルックアップ SQL クエリを上書きする場合、以下のガイドラインを考慮してください。

- リレーショナルルックアップの場合は、ルックアップ SQL クエリを上書きできます。
- デフォルトクエリを生成してから、上書きを設定します。これにより、すべてのルックアップ/出力ポートを確実にクエリに含めることができます。SELECT 文にポートを追加したり、文からポートを削除したりすると、セッションは失敗します。
- ルックアップキャッシュに追加する行をフィルタリングするには、ルックアップソースフィルタを追加します。これにより、統合サービスは WHERE 句に一致する動的キャッシュとターゲットテーブルにのみ行を挿入します。
- 複数のルックアップトランスフォーメーションでルックアップキャッシュを共有している場合、各ルックアップトランスフォーメーションで同じルックアップ SQL オーバーライドを使用します。
- すべての行を返すルックアップトランスフォーメーションを設定する際に、統合サービスはソートされたキーを使用してルックアップキャッシュを構築します。トランスフォーメーションがルックアップのすべての行を取得する際に、統合サービスはソートされた順序でキーを使用してデータキャッシュを構築します。行がソートされていない場合は、統合サービスはキャッシュからすべての行を取得できません。データがキーに対してソートされていない場合は、予期しない結果になる可能性があります。
- ORDER BY 句には、ルックアップ条件で表示される順序どおりの条件ポートを含める必要があります。
- ORDER BY 句をオーバーライドする場合は、コメント表示を使用してルックアップトランスフォーメーションが生成する ORDER BY 句を抑制します。
- プッシュダウンの最適化を使用する場合は、ORDER BY 句をオーバーライドしたり、生成された ORDER BY 句をコメント表示して抑制したりすることはできません。
- ルックアップクエリ内のテーブル名やカラム名に予約語が含まれている場合、引用符で予約語を括ってください。
- キャッシュされていないルックアップにルックアップクエリを上書きするには、統合サービスで複数の一致が検出されたときに値を返すように指定します。
- デフォルトの SQL 文にはカラムの追加も削除もできません。
- Developer tool は、SQL クエリの構文を検証しません。接続されていないルックアップクエリの SQL オーバーライドが有効でない場合、マッピングは失敗します。

ルックアップクエリの上書きの手順

デフォルトのルックアップ SQL クエリを上書きする場合、以下の手順を使用してください。

1. [プロパティ] タブで、[ルックアップ SQL オーバーライド] フィールド内から SQL エディタを開きます。

2. [SQL 文の生成] をクリックし、デフォルトの SELECT 文を生成します。ルックアップ SQL 上書きを入力します。
3. データベースに接続し、[検証] をクリックしてルックアップ SQL オーバーライドをテストします。
4. [OK] をクリックして [プロパティ] タブに戻ります。

キャッシュを使用しないルックアップの SQL オーバーライド

キャッシュを使用しないルックアップに対する SQL オーバーライドを定義できます。統合サービスでは、キャッシュを使用しないルックアップに対するオーバーライド文からキャッシュが作成されることはありません。オーバーライド SELECT 文では SQL 関数を使用できます。WHERE 句と ORDER BY 句を含め、すべての SQL クエリをオーバーライドできます。

デフォルトの SELECT 文を生成すると、Designer では、ルックアップポートと出力ポート、およびルックアップ条件に応じた WHERE 句を含む SELECT 文が生成されます。ルックアップトランスフォーメーションが、接続されていないルックアップである場合、SELECT 文にはルックアップポートと戻りポートが含まれます。統合サービスでは、ルックアップトランスフォーメーションの [条件] タブで設定した条件から WHERE 句が生成されることはありません。

SELECT クエリ内の各カラムでは、エイリアスを使用して出力カラムが定義されます。SQL 文内ではこの構文を変更しないようにします。変更するとクエリは失敗します。WHERE 句内の入力ポートを参照するには、パラメータのバインドを設定します。以下の例には、Name ポートを参照する WHERE 文が含まれています。

```
SELECT EMPLOYEE.NAME as NAME, max(EMPLOYEE.ID) as ID from EMPLOYEE WHERE EMPLOYEE.NAME=?NAME1?
```

キャッシュを使用しないルックアップの SQL エディタでは、[ポート] タブに入力ポートとルックアップポートが表示されます。

SQL 文に関数を追加する場合は、戻りデータタイプが ALIAS カラムのデータタイプに一致している必要があります。例えば、ID のデータタイプは MAX 関数の戻り型に一致します。

```
SELECT EMPLOYEE.NAME as NAME, MAX(EMPLOYEE.ID) as ID FROM EMPLOYEE
```

注: キャッシュを使用しないルックアップに対して、SQL オーバーライド内のサブクエリを使用することはできません。

ルックアップソースフィルタ

キャッシュが有効になっているリレーショナルルックアップトランスフォーメーション用にルックアップソースフィルタを設定できます。ルックアップソースフィルタを追加することで、データ統合サービスがルックアップソーステーブルに対して実行するルックアップの数が制限できます。

ルックアップソースフィルタを設定すると、データ統合サービスはそのフィルタ文の結果に基づいてルックアップを実行します。たとえば、ID が 510 より大きい従業員の名字をすべて取得する必要があるとします。

EmployeeID カラムに対して、次のようにルックアップソースフィルタを設定します。

```
EmployeeID >= 510
```

EmployeeID は、ルックアップトランスフォーメーションの入力ポートです。データ統合サービスはソース行を読み取るときに、EmployeeID の値が 510 より大きいときにキャッシュに対してルックアップを実行します。EmployeeID が 510 以下の場合、ルックアップトランスフォーメーションは姓を取得しません。

プッシュダウン最適化用に設定されたマッピングでルックアップソースフィルタをルックアップクエリに追加すると、データ統合サービスは SQL オーバーライドを示すビューを作成します。次に、データ統合サービスは、このビューに対して SQL クエリを実行し、トランスフォーメーションロジックをデータベースにプッシュします。

プッシュダウン最適化用に設定されたセッションでルックアップソースフィルタをルックアップクエリに追加すると、データ統合サービスは SQL オーバーライドを示すビューを作成します。次に、データ統合サービス

は、このビューに対して SQL クエリを実行し、トランスフォーメーションロジックをデータベースにプッシュします。

ルックアップソース行のフィルタリング

ルックアップソースフィルタを追加することで、データ統合サービスがリレーショナルルックアップソースに対して実行するルックアップの数を制限できます。

1. Mapping Designer または Transformation Developer で、ルックアップトランスフォーメーションを開きます。
2. **【プロパティ】** タブを選択します。
3. キャッシュが有効化されていることを確認します。
4. **【ルックアップソースフィルタ】** フィールドで **【開く】** をクリックします。
5. SQL エディタで、入力ポートを選択するか、フィルタを適用するルックアップトランスフォーメーションポートを入力します。
6. フィルタ条件を入力します。
フィルタ条件にはキーワード WHERE を含めないでください。文字列マッピングパラメータおよび変数を文字列識別子で囲みます。
7. 条件を検証するには、クエリに含めたソースが格納されている ODBC データソースを選択します。
8. このデータベースに接続するために必要なユーザー名とパスワードを入力します。
9. **【検証】** をクリックします。
Designer がクエリを実行し、その構文が正しいかどうかを通知します。

ルックアップ条件

Integration Service は、ルックアップ条件でルックアップソースのデータを検索します。ルックアップ条件は、SQL クエリの WHERE 句に似ています。ルックアップトランスフォーメーションでルックアップ条件を設定すると、ソースデータの 1 つ以上のカラムの値がルックアップソースまたはキャッシュの値と比較されます。

たとえば、ソースデータに employee_number が含まれているとします。ルックアップソーステーブルには、employee_ID、first_name、および last_name が含まれています。以下のルックアップ条件を設定します。

employee_ID = employee_number

Integration Service は、employee_number ごとにルックアップソースの employee_ID、last_name、および first_name カラムを返します。

また、ルックアップソースの複数行を返すこともできます。以下のルックアップ条件を設定します。

employee_ID > employee_number

Integration Service は、employee_ID 番号がソースの従業員番号よりも大きい行をすべて返します。

ルックアップトランスフォーメーション設定の入力について、以下のガイドラインを使用してください。

- ルックアップ条件に含めるカラムのデータタイプは一致する必要があります。
- ルックアップ条件は、すべてのルックアップトランスフォーメーションに入力する必要があります。
- ルックアップ条件のルックアップポートごとに入力ポートを 1 つ使用します。同じ入力ポートをトランスフォーメーションの複数の条件で使用します。

- 複数の条件を入力した場合、Integration Service はそれぞれの条件を OR ではなく AND として評価します。Integration Service は、設定したすべての条件に一致する行を返します。
- 複数の条件を含める場合は、ルックアップのパフォーマンスを最適化するために以下の順序で条件を入力します。
 - 等しい (=)
 - より小さい (<)、より大きい (>)、より小さいまたは等しい (<=)、より大きいまたは等しい (>=)
 - 等しくない (!=)
- Integration Service は NULL 値を一致させます。例えば、入力ルックアップ条件カラムが NULL であった場合、Integration Service は NULL がルックアップの NULL と等しいとみなします。
- ソート済み入力用にフラットファイルルックアップを設定する場合、条件カラムがグループ化されていないと Integration Service はセッションに失敗します。カラムがグループ化されていてソートされていない場合、Integration Service はソート済み入力の設定されていない場合と同様にルックアップを処理します。

Integration Service では、トランスフォーメーションで動的キャッシュを使用する、キャッシュを使用しない、静的キャッシュを使用する、のどれを設定するかによって、ルックアップ一致の処理が異なります。

キャッシュを使用しない、または静的キャッシュ

静的なルックアップキャッシュまたはキャッシュを使用しないルックアップソースがあるルックアップトランスフォーメーションを設定する場合は、以下のガイドラインに従います。

- ルックアップ条件を作成する場合、下記の演算子を使用します。
 - =, >, <, >=, <=, !=
 複数のルックアップ条件を入れる場合は、ルックアップのパフォーマンスを最適化するために、次の順序で条件を指定します。
 - 等しい (=)
 - より小さい (<)、より大きい (>)、より小さいまたは等しい (<=)、より大きいまたは等しい (>=)
 - 等しくない (!=)
 例えば、下記のルックアップ条件を作成するとします。


```
ITEM_ID = IN_ITEM_ID
PRICE <= IN_PRICE
```
 - ルックアップによって値を返すためには、入力値がすべての条件を満足する必要があります。
- 値が等しいかどうかの条件を指定したり、しきい値としての条件を指定したりできます。たとえば、カリフォルニアに居住していない顧客、または給与が\$30,000 より高い従業員を検索すると想定します。この場合、ソースおよび条件の性質を反映して、ルックアップを実行すると複数の値が返される場合があります。

動的キャッシュ

動的キャッシュを使用するようにルックアップトランスフォーメーションを設定した場合、ルックアップ条件で使用できるのは等価演算子 (=) のみです。

複数の一致の処理

ルックアップトランスフォーメーションは、トランスフォーメーションで設定された条件に基づいて値を検索します。ルックアップ条件が一意的なキーに基づいていない場合やルックアップソースが非正規化されている場合、Integration Service はルックアップソースまたはルックアップキャッシュで複数の一致を検出することがあります。

ルックアップトランスフォーメーションは、以下の方法で複数の一致を処理するよう設定できます。

- **一致する最初の値を使用する、または一致する最後の値を使用する。** トランスフォーメーションを、一致する最初の値または一致する最後の値を返すように設定できます。最初の値および最後の値とは、ルックアップキャッシュ内で最初および最後に検出された、ルックアップ条件に一致する値です。ルックアップソースをキャッシュする場合、Integration Service はルックアップキャッシュ内の各カラムに対して ORDER BY 句を生成し、キャッシュの最初の行および最後の行を判断します。その後、Integration Service は各ルックアップソースカラムを昇順でソートします。

Integration Service は、数字のカラムを 0 から 10 のように昇順でソートします。日付/時刻カラムは 1 月から 12 月および月初から月末の順でソートします。Integration Service は、セッションに対して設定されたソート順に基づいて文字列カラムをソートします。

- **一致する値を使用する。** ルックアップトランスフォーメーションを、ルックアップ条件に一致する値を返すように設定できます。ルックアップトランスフォーメーションが一致する値を何か返すよう設定すると、トランスフォーメーションはルックアップ条件に一致する最初の値を返します。トランスフォーメーションは、すべてのルックアップトランスフォーメーションポートではなく、キーポートに基づいてインデックスを作成します。一致する値を使用すると、行のインデックス作成プロセスが単純化されるため、パフォーマンスを改善できます。
- **すべての値を使用する。** ルックアップトランスフォーメーションは、すべての一致する行を返します。このオプションを使用するには、ルックアップトランスフォーメーションを作成するときに、すべての一致を返す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必要があります。トランスフォーメーションはアクティブなトランスフォーメーションになります。トランスフォーメーションの作成後に、モードのパッシブとアクティブを変更することはできません。
- **エラーを返す。** ルックアップトランスフォーメーションが静的キャッシュを使用する、またはキャッシュを使用しない場合、Integration Service は行をエラーとしてマークします。ルックアップトランスフォーメーションは、デフォルトでその行をセッションに書き込み、エラーコードを 1 つ増やします。ルックアップトランスフォーメーションに動的キャッシュが含まれる場合、Integration Service は複数の一致を検出したときにセッションに失敗します。Integration Service がルックアップテーブルをキャッシュしている間、または重複するキー値をルックアップしている間、セッションは失敗します。また、更新時に古い値を出力するようにルックアップトランスフォーメーションを設定した場合、複数の一致が検出されるとルックアップトランスフォーメーションはエラーを返します。トランスフォーメーションは、すべてのルックアップトランスフォーメーションポートではなく、キーポートに基づいてインデックスを作成します。

関連項目：

- [「戻り値としての複数の行」 \(ページ 315\)](#)

ルックアップキャッシュ

ルックアップファイルまたはテーブルをキャッシュに格納するようにルックアップトランスフォーメーションを設定できます。Integration Service は、キャッシュを使用するルックアップトランスフォーメーションのデータの最初の行を処理するときにメモリにキャッシュを構築します。キャッシュのメモリは、トランスフォーメーションまたはセッションプロパティで設定した量に基づいて割り当てられます。Integration Service ではインデックスキャッシュに条件値が格納され、データキャッシュに出力値が格納されます。Integration Service は、トランスフォーメーションに入力される各行のキャッシュに対してクエリを実行します。

Integration Service は、デフォルトでキャッシュファイルを \$PMCacheDir に作成します。データがメモリキャッシュに入らない場合、Integration Service はオーバーフローした値をキャッシュファイルに格納します。セッションが完了すると、永続キャッシュを使用するようにルックアップトランスフォーメーションを設定していない限り、Integration Service はキャッシュメモリを解放し、キャッシュファイルを削除します。

ルックアップキャッシュを設定するときは、以下のオプションを設定できます。

- 永続キャッシュ
- ルックアップソースからのキャッシュの再構築
- 静的キャッシュ
- 動的キャッシュ
- 共有キャッシュ
- ルックアップキャッシュの事前作成

注: リレーショナルルックアップまたはフラットファイルルックアップでは、動的キャッシュを使用できます。

戻り値としての複数の行

一致するすべての行を返す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と、ルックアップトランスフォーメーションではルックアップ条件に一致するすべての行が返されます。トランスフォーメーションを作成する場合は、一致するすべての行を返すように設定する必要があります。ルックアップトランスフォーメーションは、アクティブなトランスフォーメーションになります。複数一致プロパティに対するルックアップポリシーは「すべての値を使用」です。プロパティは読み取り専用になります。トランスフォーメーションを作成した後は、プロパティを変更することはできません。

リレーショナルテーブルに顧客注文データがあるとします。テーブル内には、各顧客に対して複数の注文があります。ルックアップトランスフォーメーションを、ルックアップから顧客のすべての注文を返すように設定できます。ルックアップテーブルをキャッシュすることで、パフォーマンスを高めることができます。

キャッシュ用にルックアップトランスフォーメーションを設定すると、Integration Service はルックアップソースから読み込んだすべての行をキャッシュします。Integration Service は、ルックアップキーに対するすべての行をキーインデックスごとにキャッシュします。

戻り値としての複数の行に関するルールおよびガイドライン

複数の行が返される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場合は、以下のルールおよびガイドラインを使用します。

- 統合サービスでは、ルックアップソースから読み込んだすべての行が、キャッシュルックアップ用にキャッシュされます。
- 複数の行を返すキャッシュルックアップまたはキャッシュを使用しないルックアップに対して、SQL オーバーライドを設定できます。オーバーライドする SQL は、ルックアップキーに格納されているデータを返す必要があります。キーでデータがソートされていない場合、予想しない結果が生じることがあります。
- 複数の行を返すルックアップトランスフォーメーションに対して動的キャッシュを有効にすることはできません。
- 接続されていないルックアップトランスフォーメーションから複数の行を返すことはできません。
- ルックアップトランスフォーメーションに、複数の一致ポリシーに一致するキャッシュルックアップが含まれる場合には、複数のルックアップトランスフォーメーションが名前付きキャッシュを共有するように設定できます。
- 複数の行を返すルックアップトランスフォーメーションは、各入力行に対して一致する 1 行を返すルックアップトランスフォーメーションを持つキャッシュを共有できません。

関連項目：

- [「ルックアップトランスフォーメーションの作成」 \(ページ 318\)](#)

接続されていないルックアップトランスフォーメーションの設定

コネクタされていないルックアップトランスフォーメーションは、ソースまたはターゲットに接続していないルックアップトランスフォーメーションです。:LKP 式で別のトランスフォーメーションからルックアップを呼び出します。

式からルックアップを呼び出すときには、以下の作業を実行できます。

- 式内のルックアップの結果をテストする。
- ルックアップ結果に基づいて行をフィルタリングする。
- ルックアップの結果に基づいて更新対象行に印を付け、緩やかに変化する次元テーブルを更新する。
- 1 つのマッピング内で同じルックアップを複数回呼び出す。

接続されていないルックアップトランスフォーメーションを設定するには、入力ポートを追加し、ルックアップ条件を設定し、戻り値を指定して、別のトランスフォーメーションでルックアップ式を設定します。

手順 1. 入力ポートの追加

:LKP 式の引数ごとにルックアップトランスフォーメーションに入力ポートを作成します。作成したいルックアップ条件のそれぞれに対して、入力ポートをルックアップトランスフォーメーションに追加しなければなりません。各条件で異なるポートを作成できます。また、複数の条件で同じ入力ポートを使用することもできます。

たとえば、ある小売店のすべての部門が前月に商品の値段を上げたとします。経理部門では、値段を上げた商品の行だけをターゲットにロードしたいとします。これを行うために、以下の作業を実行します。

- ソースの ITEM_ID とターゲットの ITEM_ID を比較するルックアップ条件を作成します。
- ソース内の各項目の PRICE をターゲットテーブル内の値段と比較します。
 - その項目がターゲットテーブルにあり、ソース内の項目の値段がターゲットテーブル内の値段に等しいかそれ以下であれば、その行を削除します。
 - ソース内の値段がターゲットテーブル内の値段よりも高ければ、その行を更新します。
- データタイプが Decimal (37,0) の入力ポート (IN_ITEM_ID) を作成して ITEM_ID の照合対象とします。さらに、Decimal (10,2) の IN_PRICE 入力ポートを作成して PRICE ルックアップポートの照合対象とします。

手順 2. ルックアップ条件の追加

ポートを設定した後で、トランスフォーメーション入力値をルックアップソースの値またはキャッシュの値と比較するようにルックアップ条件を定義します。パフォーマンスを高めるためには、等号を含む条件を最初に追加します。

この例では、下記のルックアップ条件を追加します。

```
ITEM_ID = IN_ITEM_ID
```

```
PRICE <= IN_PRICE
```

項目がマッピングソースとルックアップソースに存在し、マッピングソースでの値段がルックアップでの値段以下であれば、条件は True となり、ルックアップは戻りポートで指定された値を返します。ルックアップ条件が False であれば、ルックアップは NULL を返します。UpdateStrategy 式を書く場合は、IIF 関数にネストした ISNULL を使用して NULL 値をテストします。

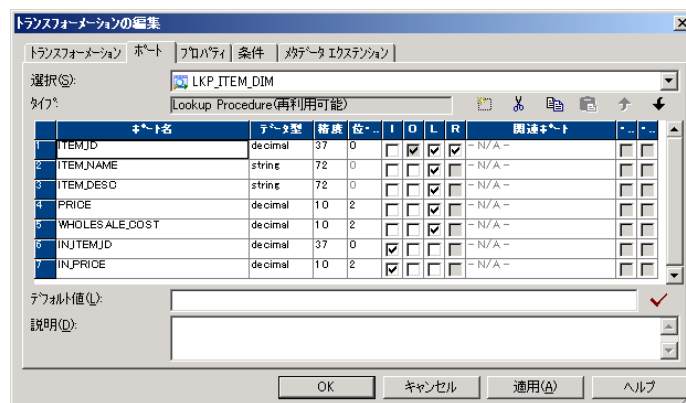
手順 3. 戻り値の指定

ルックアップトランスフォーメーションに複数の入力値を渡し、データの列を 1 つ返すことができます。接続されていないルックアップトランスフォーメーションには、複数の一致に対してすべての値を使用するルックアップポリシーは設定しないでください。1 つのルックアップ/出力ポートを戻りポートとして指定します。最初の値、最後の値、またはルックアップ条件に一致するいずれかの値を選択して使用します。

UpdateStrategy またはフィルタ式からコネクタされていないルックアップを呼び出す場合は、一般に NULL 値をチェックすることになります。この場合、戻りポートは何であってもかまいません。計算を行う式からルックアップを呼び出す場合、戻り値はその式に取りこみたい値でなければなりません。

アップデートストラテジの例を続行するには、ITEM_ID ポートを戻りポートとして定義します。アップデートストラテジの式は、NULL の戻り値をチェックします。ルックアップ条件が True であれば、Integration Service は ITEM_ID を返します。条件が False であれば、Integration Service は NULL を返します。

以下の図は、ルックアップトランスフォーメーションの戻りポートを示しています。



手順 4. 式からのルックアップの呼び出し

コネクタされていないルックアップトランスフォーメーションに、別のトランスフォーメーションの LKP 式から入力値を提供します。引数は、ルックアップ条件に使用されているルックアップトランスフォーメーションの入力ポートに一致するローカル入力ポートです。LKP 式には以下の構文を使用します。

:LKP.lookup_transformation_name(argument, argument, ...)

小売店の例に話を戻すと、UpdateStrategy 式を書く場合、式内のポートの順序がルックアップ条件内での順序と一致しなければなりません。この場合で言えば、ITEM_ID 条件は最初のルックアップ条件であるため、UpdateStrategy 式の最初の引数となります。

IIF(ISNULL(:LKP.lkpITEMS_DIM(ITEM_ID, PRICE)), DD_UPDATE, DD_REJECT)

コネクタされていないルックアップトランスフォーメーションを呼び出す式を記述するときは、以下のガイドラインに従ってください。

- 各引数を記述する順序は、ルックアップトランスフォーメーションのルックアップ条件の順序と一致しなければなりません。
- 式のポートのデータタイプは、ルックアップトランスフォーメーションの入力ポートのデータタイプと一致しなければなりません。データタイプが一致しない場合、Designer は式を有効とみなしません。

- ルックアップ条件の 1 つのポートがルックアップ/出力ポートではない場合、Designer は式を有効とみなしません。
 - 式の引数（ポート）は、ルックアップ条件の入力ポートと同じ順に並んでいなければなりません。
 - 使用した:LKP 構文が正しくない場合、Designer はマッピングを無効とします。
 - :LKP 式内で接続されたルックアップトランスフォーメーションを呼び出すと、Designer はマッピングを無効とします。
- ヒント:** 式を入力するときの構文エラーを防ぐために、ポイントアンドクリック機能を使用して関数およびポートを選択してください。

データベースデッドロックに対するレジリエンス

キャッシュを使用しないルックアップの場合、ルックアップトランスフォーメーションはデータベースデッドロックに対して復元性があります。データベースデッドロックエラーが発生した場合、セッションは失敗しません。Integration Service は、指定されたリトライ期間に、最後のステートメントの再実行を試みます。

Integration Service に対してデッドロックリトライ回数およびデッドロックスリープ間隔を設定できます。これらの値は、リレーショナルライタのデータベースデッドロックにも影響します。これらの値は、セッションレベルでカスタムプロパティとして上書きできます。

以下の Integration Service プロパティを設定します。

- **NumOfDeadlockRetries.** PowerCenter Integration Service がデータベースデッドロックのターゲット書き込みを再試行する回数。最小値は 0 です。デフォルトは 10 です。デッドロック時にセッションを失敗させる場合は、NumOfDeadlockRetries を 0 に設定します。
- **DeadlockSleep.** PowerCenter Integration Service がデータベースデッドロックのターゲット書き込みを再試行するまでの秒数。

デッドロックが発生した場合、Integration Service はステートメントを実行しようとします。Integration Service は、一定の遅延時間ごとに再試行します。すべての試行がデッドロックのために失敗した場合、セッションは失敗します。Integration Service は、ステートメントを再試行するたびに、メッセージをセッションログに書き込みます。

ルックアップトランスフォーメーションの作成

Transformation Developer で再利用可能なルックアップトランスフォーメーションを作成します。Mapping Designer で再利用不可能なルックアップトランスフォーメーションを作成します。

ルックアップトランスフォーメーションを作成するには：

1. 再利用可能なルックアップトランスフォーメーションを作成するには、Transformation Developer を開きます。
再利用不可能なルックアップトランスフォーメーションを作成するには、Mapping Designer でマッピングを開きます。パイプラインルックアップトランスフォーメーションを作成している場合は、ルックアップソースとして使用するソース定義をドラッグします。
2. [トランスフォーメーション] - [作成] をクリックします。ルックアップトランスフォーメーションを選択します。
3. トランスフォーメーションの名前を入力します。[作成] をクリックします。

ルックアップトランスフォーメーションの命名規則は、「LKP_トランスフォーメーション名」です。

4. トランスフォーメーションがアクティブかパッシブかを選択します。 [OK] をクリックします。このオプションは変更できません。
5. [ルックアップテーブルの選択] ダイアログボックスで、ルックアップ定義のインポート用に以下のいずれかのオプションを選択します。
 - リポジトリからのソース定義。
 - リポジトリからのターゲット定義。
 - マッピングからのソース修飾子。
 - リポジトリからのリレーショナルテーブルまたはファイルのインポート。

注: 定義をインポートすることなく、手動でルックアップポートを追加できます。ルックアップポートのうち、どのポートを出力ポートにするかを選択できます。

ルックアップソースを選択すると、Designer がそのソースにあるポートに基づいてトランスフォーメーションにポートを作成します。Designer は、各ポートをルックアップポートおよび出力ポートとして設定します。ルックアップポートは、ルックアップソースのカラムを表しています。ルックアップトランスフォーメーションは、各ルックアップポートでルックアップソースからデータを受け取り、ターゲットに渡します。

6. ルックアップトランスフォーメーションがすべての一致する行を返すようにする場合は、[複数一致の場合にすべての行を返す] を有効にします。 トランスフォーメーションを作成した後は、このオプションを変更できません。ルックアップトランスフォーメーションは、アクティブなトランスフォーメーションになります。
7. 定義をインポートせずに、手動でルックアップポートを追加する場合は、[OK] または [スキップ] をクリックします。ルックアップポートのうち、どのポートを出力ポートにするかを選択できます。
8. 接続されたルックアップトランスフォーメーションの場合、入力ポートおよび出力ポートを追加します。データをトランスフォーメーションに渡し、ルックアップテーブルからターゲットにデータを返すことができます。
9. コネクトされていないルックアップトランスフォーメーションの場合、ルックアップから返したい値の戻りポートを作成します。

ルックアップを呼び出したトランスフォーメーションにカラムを 1 つ返すことができます。

10. ルックアップトランスフォーメーションのプロパティを設定するため、[プロパティ] タブをクリックします。ルックアップキャッシングを設定します。

ルックアップキャッシングは、パイプラインおよびフラットファイルルックアップトランスフォーメーションではデフォルトで有効になっています。

11. 動的ルックアップキャッシュがあるルックアップトランスフォーメーションの場合、入力ポート、出力ポート、またはシーケンス ID を各ルックアップポートに関連付けます。

Integration Service は、各関連式からのデータを使用してルックアップキャッシュに対して行の挿入または更新を行います。シーケンス ID と関連付けた場合、Integration Service はルックアップキャッシュに挿入された行に対してプライマリキーを生成します。

12. [条件] タブでルックアップ条件を追加します。

ルックアップ条件は、ソースカラム値をルックアップソースの値と比較するものです。[条件] タブのトランスフォーメーションポートはソースカラム値です。ルックアップテーブルはルックアップソースです。

関連項目：

- [「再利用可能なパイプラインルックアップトランスフォーメーションの作成」 \(ページ 320\)](#)
- [「再利用不可能なパイプラインルックアップトランスフォーメーションの作成」 \(ページ 320\)](#)

再利用可能なパイプラインルックアップトランスフォーメーションの作成

Transformation Developer で再利用可能なパイプラインルックアップトランスフォーメーションを作成します。このトランスフォーメーションを作成するときは、ルックアップテーブル場所として [ソース] を選択します。Transformation Developer には、リポジトリからのソース定義のリストが表示されます。

リレーショナルテーブルソース定義またはフラットファイルソース定義を表すソース修飾子を選択すると、Designer がリレーショナルルックアップトランスフォーメーションまたはフラットファイルルックアップトランスフォーメーションを作成します。また、ソース修飾子がアプリケーションソースを表している場合には、パイプラインルックアップトランスフォーメーションを作成します。リレーショナルルックアップソースまたはフラットファイルルックアップソースのパイプラインルックアップトランスフォーメーションを作成するには、そのトランスフォーメーションの作成後にソースタイプをソース修飾子に変更します。[ルックアップテーブル] プロパティにソース定義の名前を入力します。

再利用可能なパイプラインルックアップトランスフォーメーションをマッピングにドラッグすると、Mapping Designer が [ルックアップテーブル] プロパティのソース定義をマッピングに追加します。また、ソース修飾子トランスフォーメーションも追加します。

ルックアップテーブル名をマッピングの別のソース修飾子に変更するには、[ルックアップテーブル名] プロパティで [開く] をクリックします。一覧からソース修飾子を選択します。

再利用不可能なパイプラインルックアップトランスフォーメーションの作成

Mapping Designer で再利用不可能なパイプラインルックアップトランスフォーメーションを作成します。ソース定義をマッピング内にドラッグします。Mapping Designer でトランスフォーメーションを作成するときは、ルックアップテーブル場所としてソース修飾子を選択します。マッピング内のソース修飾子のリストが表示されます。ソース修飾子を選択すると、そのソース修飾子からのポート名および属性がルックアップトランスフォーメーションに読み込まれます。

ルックアップトランスフォーメーションのヒント

ルックアップ条件に使用されるカラムにインデックスを追加します。

ルックアップテーブルを含むデータベースを変更する特権を持っている場合、キャッシュを使用するルックアップおよびキャッシュを使用しないルックアップのパフォーマンスを向上させることができます。これは大規模なルックアップテーブルの場合に重要です。Integration Service は、カラムの値のクエリ、ソート、および比較を行う必要があるため、インデックスにはルックアップ条件に使用されるすべてのカラムを含める必要があります。

等価演算子 (=) を持つ条件を先に配置してください。

複数のルックアップ条件を入れる場合は、ルックアップのパフォーマンスを最適化するために、次の順序で条件を指定します。

- 等しい (=)
- より小さい (<)、より大きい (>)、より小さいまたは等しい (<=)、より大きいまたは等しい (>=)
- 等しくない (!=)

小さなルックアップテーブルはキャッシュに格納してください。

小さなルックアップテーブルをキャッシュに格納することによって、セッションのパフォーマンスが向上します。ルックアップテーブルをキャッシュに格納するかどうかに関わらず、ルックアップクエリの結果および処理は同じです。

データベース内でテーブルを結合してください。

ルックアップテーブルがマッピングにおいてソーステーブルと同じデータベースにあり、キャッシュが使用できない場合は、ルックアップトランスフォーメーションを使用する代わりに、ソースデータベース内でテーブル同士を結合します。

静的ルックアップには永続ルックアップキャッシュを使用してください。

セッションとセッションの間にルックアップソースが変更されない場合は、永続ルックアップキャッシュを使用するようにルックアップトランスフォーメーションを設定します。次に、Integration Service は、セッション間でキャッシュファイルを保存して再利用することにより、ルックアップソースの読み込みにかかる時間を節約します。

:LKP 参照修飾子を使用する場合、コネクต์されていないルックアップトランスフォーメーションを呼び出してください。

:LKP 参照修飾子を使用した式を書いた場合、コネクต์されていないルックアップトランスフォーメーションだけを呼び出すことになります。接続されたルックアップトランスフォーメーションを呼び出そうとすると、Designer はエラーを表示し、マッピングを無効とします。

リレーショナルルックアップソースまたはフラットファイルルックアップソースを処理するときにパフォーマンスを高めるには、パイプラインルックアップトランスフォーメーションを設定します。

ルックアップソースをソース修飾子として定義したときには、リレーショナルルックアップソースまたはフラットファイルルックアップソースを処理するパーティションを作成できます。再利用不可能なパイプラインルックアップトランスフォーメーションを設定し、その部分パイプラインにルックアップソースを処理するパーティションを作成します。

第 18 章

ルックアップキャッシュ

この章では、以下の項目について説明します。

- [ルックアップキャッシュの概要, 322 ページ](#)
- [接続されたルックアップキャッシュの作成, 324 ページ](#)
- [パーシステントルックアップキャッシュの使用, 326 ページ](#)
- [キャッシュを使用しないルックアップまたは静的キャッシュに関する作業, 328 ページ](#)
- [ルックアップキャッシュの共有, 328 ページ](#)
- [ルックアップキャッシュに関するヒント, 335 ページ](#)

ルックアップキャッシュの概要

ルックアップソースをキャッシュしてルックアップパフォーマンスを向上させるように、ルックアップトランスフォーメーションを設定できます。ルックアップテーブルまたはファイルが大きい場合は、ルックアップキャッシュを有効にします。

統合サービスでは、キャッシュを使用するルックアップトランスフォーメーションの最初のデータ行を処理するときに、メモリにキャッシュを作成します。統合サービスでは、ルックアップトランスフォーメーションを入力するソース行としてキャッシュを作成します。トランスフォーメーションまたはセッションのプロパティで設定した容量に基づいて、キャッシュにメモリを割り当てます。統合サービスではインデックスキャッシュに条件値が格納され、データキャッシュに出力値が格納されます。統合サービスは、トランスフォーメーションに入力される各行のキャッシュに対してクエリを実行します。

データがメモリキャッシュに入らない場合、統合サービスはオーバーフローした値をキャッシュファイルに格納します。統合サービスでは、指定したキャッシュディレクトリにもキャッシュファイルを作成します。セッションマッピングが完了すると、永続キャッシュを使用するようにルックアップトランスフォーメーションを設定していない限り、統合サービスはキャッシュメモリを解放し、キャッシュファイルを削除します。

フラットファイルルックアップまたはパイプラインルックアップを使用する場合、統合サービスは常にルックアップソースをキャッシュします。ソート済み入力用にフラットファイルルックアップを設定する場合、条件カラムがグループ化されていなければ統合サービスはルックアップをキャッシュできません。カラムがグループ化されていてソートされていない場合、統合サービスはソート済み入力が設定されていない場合と同様にルックアップを処理します。

ルックアップキャッシュを設定する場合は、以下のキャッシュ設定を設定できます。

シーケンシャルキャッシュおよび同時キャッシュ

セッションを設定すると、キャッシュを順番に作成することも同時に作成することもできます。キャッシュを順番に作成する場合、ソース行が Lookup トランスフォーメーションに入力されると統合サービスがキャッシュを作成します。キャッシュを同時に作成するようにセッションを設定した場合、統合サービス

は最初の行が Lookup トランスフォーメーションに入力されるのを待たずにキャッシュを作成します。その代わりに、複数のキャッシュを同時に作成します。

永続キャッシュ

ルックアップキャッシュファイルを保存すると、キャッシュを使用するように設定されたルックアップトランスフォーメーションを統合サービスが次回処理する際に再利用できます。

ソースからのキャッシュの再構築

永続キャッシュがルックアップソースと同期化していない場合、ルックアップキャッシュを再構築する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ができます。

ルックアップキャッシュを再構築するようにルックアップトランスフォーメーションを設定できます。

静的キャッシュ

静的なキャッシュを、任意のルックアップソースに設定できます。デフォルトでは、統合サービスは静的キャッシュを作成します。ルックアップファイルまたはテーブルをキャッシュに格納し、トランスフォーメーションに入力される各行についてキャッシュにある値をルックアップします。ルックアップ条件が True の場合、統合サービスはルックアップキャッシュからの値を返します。統合サービスは、ルックアップトランスフォーメーションの処理中はキャッシュを更新しません。

動的キャッシュ

ルックアップソースをキャッシュし、そのキャッシュを更新するには、動的キャッシュでルックアップトランスフォーメーションを設定します。統合サービスはデータを動的にルックアップキャッシュに挿入または更新し、そのデータをターゲットに渡します。動的キャッシュはターゲットと同期化しています。

共有キャッシュ

複数のトランスフォーメーション間でルックアップキャッシュを共有できます。名前なしキャッシュを同じマッピング内のトランスフォーメーション間で共有することができます。名前付きキャッシュを同じマッピング内、または異なるマッピング内のトランスフォーメーション間で共有することができます。キャッシュ共有ルールが一致する場合、ルックアップトランスフォーメーションは、同じターゲットロード順グループ内の名前なし静的キャッシュを共有することができます。ルックアップトランスフォーメーションは、同じターゲットロード順グループ内の動的キャッシュを共有することはできません。

同じマッピングの複数のトランスフォーメーション間でルックアップキャッシュを共有できます。

ルックアップトランスフォーメーションにキャッシュを設定していない場合、統合サービスは各入力行のルックアップソースに対してクエリを実行します。ルックアップソースをキャッシュに格納するかどうかに関わらず、ルックアップクエリの結果および処理は同じです。ただし、ルックアップキャッシングを有効にすると、大きなルックアップソースでルックアップパフォーマンスを向上させることができます。

キャッシュの比較

統合サービスの実行は、設定するルックアップキャッシュのタイプに基づいてそれぞれ異なります。

次の表に、ルックアップトランスフォーメーションと、キャッシュを使用しないルックアップ、静的キャッシュおよび動的キャッシュとの比較を示します。

使用なし	静的キャッシュ	動的キャッシュ
統合サービスはキャッシュを挿入および更新しません。	統合サービスはキャッシュを挿入および更新しません。	統合サービスは、行をターゲットに渡すときに、キャッシュに行を挿入またはキャッシュ内の行を更新します。
リレーショナルルックアップを使用できます。	リレーショナルルックアップ、フラットファイルルックアップまたはパイプラインルックアップを使用できます。 リレーショナルルックアップまたはフラットファイルルックアップを使用できます。	リレーショナルルックアップ、フラットファイルルックアップ、またはソース修飾子ルックアップを使用できます。 リレーショナルルックアップまたはフラットファイルルックアップを使用できます。
条件が <code>True</code> の場合、統合サービスはルックアップテーブルまたはキャッシュから値を返します。 条件が <code>True</code> でない場合、統合サービスは接続されたトランスフォーメーションについてはデフォルト値を、接続されていないトランスフォーメーションについては <code>NULL</code> を返します。	条件が <code>True</code> の場合、統合サービスはルックアップテーブルまたはキャッシュから値を返します。 条件が <code>True</code> でない場合、統合サービスは接続されたトランスフォーメーションについてはデフォルト値を、接続されていないトランスフォーメーションについては <code>NULL</code> を返します。	条件が <code>True</code> の場合、統合サービスは行タイプに基づいて、キャッシュの行を更新するか、またはキャッシュを未変更のままにします。これは、キャッシュおよびターゲットテーブルに行があることを示しています。更新された行をターゲットテーブルに渡すことができます。 条件が <code>True</code> ではない場合、統合サービスは行タイプに基づいて、キャッシュに行を挿入するか、またはキャッシュを未変更のままにします。これは、キャッシュまたはターゲットに行がなかったことを示しています。挿入された行をターゲットテーブルに渡すことができます。

関連項目：

- [「キャッシュを使用しないルックアップまたは静的キャッシュに関する作業」](#) (ページ 328)
- [「動的ルックアップキャッシュの更新」](#) (ページ 344)

接続されたルックアップキャッシュの作成

Integration Service では、接続されたルックアップトランスフォーメーションのルックアップキャッシュを以下の方法で作成できます。

- **連続したキャッシュ。** Integration Service は、連続してルックアップキャッシュを作成します。Integration Service は、キャッシュを使用するルックアップトランスフォーメーションのデータの最初の行を処理するときに、メモリにキャッシュを作成します。
- **コンカレントキャッシュ。** Integration Service は、同時にルックアップキャッシュを作成します。データがルックアップトランスフォーメーションに達するのを待つ必要はありません。

注: キャッシュ作成の設定方法にかかわらず、Integration Service は未接続のルックアップトランスフォーメーションで順番にキャッシュを作成します。未接続のルックアップトランスフォーメーションでコンカレント

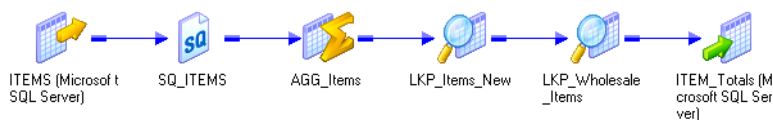
キャッシュを作成するようにセッションを設定している場合、Integration Service はこの設定を無視し、未接続のルックアップトランスフォーメーションキャッシュを順番に作成します。

連続したキャッシュ

デフォルトでは、キャッシュを使用するルックアップトランスフォーメーションのデータの最初の行を処理するときに、Integration Service がメモリにキャッシュを作成します。Integration Service は、パイプラインに各ルックアップキャッシュを順番に作成します。Integration Service は、アップストリームのアクティブなトランスフォーメーションが処理を完了するのを待ってから、ルックアップトランスフォーメーション内の行の処理を開始します。Integration Service は、アップストリームのルックアップトランスフォーメーションがキャッシュの作成を完了するまで、ダウストリームのルックアップトランスフォーメーションのキャッシュを作成しません。

例えば、以下のマッピングには、未ソートのアグリゲータトランスフォーメーションと、その後に2つのルックアップトランスフォーメーションが含まれています。

図 2. 連続したルックアップキャッシュの作成



Integration Service は、未ソートのアグリゲータトランスフォーメーションのすべての行を処理し、未ソートのアグリゲータトランスフォーメーションが完了してから最初のルックアップトランスフォーメーションの処理を開始します。最初の入力行を処理すると同時に、Integration Service は最初のルックアップキャッシュの作成を開始します。Integration Service が最初のルックアップキャッシュの作成を完了すると、ルックアップデータの処理を開始できます。データの最初の行がルックアップトランスフォーメーションに達すると、Integration Service は次のルックアップキャッシュの作成を開始できます。

ルックアップトランスフォーメーションは、行データを処理しない場合、ルックアップキャッシュを順番に処理します。条件に基づいて異なるパイプラインにデータをルーティングするようにトランスフォーメーションロジックが設定されている場合、ルックアップトランスフォーメーションは行データを処理しない可能性があります。連続したキャッシュの作成を設定した場合、不必要なルックアップキャッシュの作成を回避できます。たとえば、ルータトランスフォーメーションは、条件に対して True の場合は1つのパイプラインにデータをルーティングし、False の場合は別のパイプラインにデータをルーティングします。この場合、ルックアップトランスフォーメーションはデータをまったく受け取らない可能性があります。

コンカレントキャッシュ

同時にルックアップキャッシュを作成するように、Integration Service を設定できます。コンカレントキャッシュを使用してセッションのパフォーマンスを改善できます。パイプラインにルックアップトランスフォーメーションのアクティブなトランスフォーメーションアップストリームが含まれている場合、パフォーマンスの向上が特に期待できます。セッション内の各ルックアップトランスフォーメーション用にキャッシュを作成する必要があることが明らかであれば、コンカレントキャッシュを作成するようにセッションを設定するよう推奨します。

ルックアップトランスフォーメーションを設定してコンカレントキャッシュを作成する際、ルックアップキャッシュは、先行するトランスフォーメーションが完了するのを待たずに作成されます。また、他のルックアップキャッシュの構築を開始するまで、ルックアップキャッシュの構築を終える必要はありません。

以下の図に、同時に作成されたルックアップトランスフォーメーションのキャッシュを示します。



セッションを実行すると、Integration Service は同時にルックアップキャッシュを作成します。ここでは、アップストリームトランスフォーメーションの完了を待ちません。また、他のルックアップトランスフォーメーションによるキャッシュの作成の完了を待機することもしません。

注: 未接続のルックアップトランスフォーメーションのキャッシュは、同時に処理できません。

セッションを設定してコンカレントキャッシュを作成するには、セッションの設定属性である [ルックアップキャッシュ作成のための追加のコンカレントパイプライン] の値を設定します。

パースistentルックアップキャッシュの使用

非パースistentキャッシュまたはパースistentキャッシュを使用するように、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ができます。[ルックアップキャッシュパースistent] プロパティに基づくセッションが成功すると、Integration Service はルックアップキャッシュファイルを保存または削除します。

セッションとセッションの間にルックアップテーブルが変更されない場合は、パースistentルックアップキャッシュを使用するようにルックアップトランスフォーメーションを設定できます。Integration Service は、セッション間でキャッシュファイルを保存して再利用することにより、ルックアップテーブルの読み込みにかかる時間を節約します。

非永続キャッシュの使用

ルックアップトランスフォーメーションでキャッシュを有効にした場合、Integration Service ではデフォルトで非永続キャッシュが使用されます。Integration Service は、セッションの最後にキャッシュファイルを削除します。次にセッションを実行したときに、Integration Service はデータベースからメモリキャッシュを作成します。

永続キャッシュの使用

キャッシュファイルを保存して再使用する場合、永続キャッシュを使用できるようにトランスフォーメーションを設定します。永続キャッシュは、セッションとセッションの間にルックアップテーブルが変化しないことがわかっている場合に使用します。

Integration Service は、永続ルックアップキャッシュを使用して最初にセッションを実行するときに、キャッシュファイルを削除せずにディスクに保存します。次に Integration Service がセッションを実行したときは、メモリキャッシュをキャッシュファイルから作成します。ルックアップテーブルが変化することがある場合は、セッションプロパティを上書きして、データベースからルックアップを再構築することができます。

永続ルックアップキャッシュを使用する場合、キャッシュファイルの名前を指定できます。名前付きキャッシュを指定する場合、セッション間でルックアップキャッシュを共有できます。

ルックアップキャッシュの再構築

Integration Service が最後に永続キャッシュを作成してからルックアップソースが変更されていると思われる場合、ルックアップキャッシュの再構築を Integration Service に指示することができます。

キャッシュを再構築すると、Integration Service は新しいキャッシュファイルを作成し、既存の永続キャッシュファイルを上書きします。Integration Service は、キャッシュを再構築したときに、セッションログにメッセージを書き込みます。

マッピングに1つのルックアップトランスフォーメーションが含まれている場合、またはマッピングにキャッシュを共有する複数のターゲットロード順グループのルックアップトランスフォーメーションが含まれている場合に、キャッシュを再構築することができます。動的ルックアップが同じマッピングの静的ルックアップとキャッシュを共有する場合は、キャッシュを再構築する必要はありません。

キャッシュを再利用できない場合、Integration Service はマッピングおよびセッションのプロパティに応じてデータベースのルックアップキャッシュを再構築するか、またはセッションに失敗します。

以下の表に、Integration Service が名前付きキャッシュおよび名前なしキャッシュ用に永続キャッシュを処理する方法を示します。

セッション間のマッピングまたはセッションの変更	名前付きキャッシュ	名前なしキャッシュ
Integration Service がキャッシュファイルの場所を見つけられない。たとえば、ファイルが存在しないか、Integration Service がグリッドで実行されています。そうすると、キャッシュファイルはどのノードでも使用できません。	キャッシュの再構築	キャッシュの再構築
セッションプロパティの [Enable High Precision] オプションを有効または無効にした。	セッションが失敗	キャッシュの再構築
Mapping Designer、Mapplet Designer、または Reusable Transformation Developer でトランスフォーメーションを編集。 ¹	セッションが失敗	キャッシュの再構築
マッピングを編集（ルックアップトランスフォーメーションは除く）。	キャッシュの再利用	キャッシュの再構築
ルックアップトランスフォーメーションが含まれているパイプラインでパーティションの数を変更します。	セッションが失敗	キャッシュの再構築
ルックアップテーブルへのアクセスに使用するデータベース接続またはファイルの場所を変更。	セッションが失敗	キャッシュの再構築
Integration Service のデータ移動モードを変更。	セッションが失敗	キャッシュの再構築
Unicode モードのソート順を変更。	セッションが失敗	キャッシュの再構築
Integration Service のコードページを互換性のあるコードページに変更。	キャッシュの再利用	キャッシュの再利用
Integration Service のコードページを互換性のないコードページに変更。	セッションが失敗	キャッシュの再構築

¹. トランスフォーメーションの説明やポートの説明などのプロパティの編集は、永続キャッシュの処理に影響しません。

キャッシュを使用しないルックアップまたは静的キャッシュに関する作業

ルックアップトランスフォーメーションにキャッシュを設定した場合、Integration Service ではデフォルトで静的ルックアップキャッシュが作成されます。Integration Service は、最初のルックアップ要求を処理するときにキャッシュを構築します。トランスフォーメーションを通過する各行に対するルックアップ条件に基づいて、キャッシュに問い合わせます。Integration Service は、トランスフォーメーションの処理中にはキャッシュを更新しません。Integration Service は、キャッシュを使用しないルックアップを、キャッシュを使用するルックアップと同じ方法で処理します。ただし、キャッシュの作成およびキャッシュに対するクエリ実行の代わりに、ルックアップソースに対してクエリを実行します。

ルックアップ条件が True の場合、Integration Service はルックアップソースまたはキャッシュから値を返します。接続されたルックアップトランスフォーメーションの場合、Integration Service はルックアップ/出力ポートによって表される値を返します。未接続のルックアップトランスフォーメーションの場合、Integration Service は戻りポートによって表される値を返します。

条件が True でない場合、Integration Service は NULL またはデフォルト値を返します。条件が満たされない場合、接続されたルックアップトランスフォーメーションでは、Integration Service は出力ポートのデフォルト値を返します。条件が満たされない場合、未接続のルックアップトランスフォーメーションでは Integration Service は NULL を返します。

静的キャッシュを使用するパイプラインの中で複数のパーティションを作成すると、Integration Service は各パーティションに対してメモリキャッシュを 1 つずつ、各トランスフォーメーションに対してディスクキャッシュを 1 つずつ作成します。

ルックアップキャッシュの共有

マッピングの際、複数の Lookup トランスフォーメーションが 1 つのルックアップキャッシュを共有するように設定することができます。Integration Service では、最初のルックアップトランスフォーメーションが処理される際にキャッシュが作成されます。同じキャッシュを使用して、キャッシュを共有するそれ以降の Lookup トランスフォーメーションのルックアップを実行します。

名前なし、および名前付きキャッシュを共有することができます。

- **名前なしキャッシュ。**マッピング内のルックアップトランスフォーメーションに互換性のあるキャッシュ構造体がある場合、Integration Service ではデフォルトでキャッシュが共有されます。共有できるのは、静的名前なしキャッシュのみです。
- **名前付きキャッシュ。**マッピング全体でキャッシュファイルを共有する場合、または動的および静的キャッシュを共有する場合は、名前付き永続キャッシュを使用します。キャッシュ構造は、名前付きキャッシュと一致するか、または互換性がなければなりません。静的名前付きキャッシュおよび動的名前付きキャッシュを共有することができます。

Integration Service でルックアップキャッシュが共有される場合、セッションログにメッセージが書き込まれます。

名前なしルックアップキャッシュの共有

デフォルトでは、Integration Service によりマッピング内の互換性のあるキャッシュ構造体を持つルックアップトランスフォーメーションのキャッシュが共有されます。たとえば、1 つのマッピング内に、同じ再利用可能な Lookup トランスフォーメーションのインスタンスが 2 つあり、両方のインスタンスに同じ出力ポートを

使用した場合、この2つの Lookup トランスフォーメーションはデフォルトでルックアップキャッシュを共有します。

2つのルックアップトランスフォーメーションで名前なしキャッシュが共有される場合、Integration Service ではルックアップトランスフォーメーションのキャッシュが保存され、同じルックアップキャッシュ構造体を持つ後続のルックアップトランスフォーメーションで使用されます。

トランスフォーメーションプロパティまたはキャッシュ構造体により共有が許可されない場合、Integration Service で新しいキャッシュが作成されます。

名前なしルックアップキャッシュ共有のガイドライン

ルックアップトランスフォーメーションを設定して名前なしキャッシュを共有する場合は、以下のガイドラインに従います。

- 共有できるのは、静的名前なしキャッシュのみです。
- 共有トランスフォーメーションは、ルックアップ条件に同じポートを使用する必要があります。条件は異なる演算子を使用することができますが、ポートは同じでなければなりません。
- トランスフォーメーションプロパティをいくつか設定し、名前なしキャッシュの共有を有効にする必要があります。
- 共有トランスフォーメーションのキャッシュの構造は互換性がなければなりません。
 - 自動ハッシュキーパーティション化を使用する場合、各トランスフォーメーションのルックアップ/出力ポートは一致する必要があります。
 - 自動ハッシュキーパーティション化を使用しない場合、最初の共有トランスフォーメーションのルックアップ/出力ポートは、それ以降のトランスフォーメーションのルックアップ/出力ポートと一致するか、それらのスーパーセットでなければなりません。
- 自動ハッシュキーパーティション化を使用する複数のルックアップトランスフォーメーションが異なるターゲットロード順グループにある場合、各グループで同じ数のパーティションを設定する必要があります。自動ハッシュキーパーティション化を使用しない場合は、各ターゲットロード順グループで異なる数のパーティションを設定できます。

以下の表に、名前なし静的キャッシュおよび動的キャッシュを共有できる場合を示します。

共有キャッシュ	トランスフォーメーションの場所
静的と静的	マッピング内の任意の場所。
動的と動的	共有できません。
動的と静的	共有できません。

以下の表に、ルックアップトランスフォーメーションを名前なしキャッシュを共有するように設定する際のガイドラインを示します。

プロパティ	名前なし共有キャッシュの設定
Lookup SQL Override	[Lookup SQL Override] プロパティを使用する場合、すべての共有トランスフォーメーションで同じ上書きを使用する必要があります。
ルックアップテーブル名	一致する必要があります。
ルックアップキャッシュが有効	有効にする必要があります。

プロパティ	名前なし共有キャッシュの設定
Lookup Policy on MultipleMatch	-
ルックアップ条件	共有トランスフォーメーションは、ルックアップ条件に同じポートを使用する必要があります。条件は異なる演算子を使用することができますが、ポートは同じでなければなりません。
接続情報	接続は同じである必要があります。セッションを設定する際に、データベース接続を一致させる必要があります。
ソースタイプ	一致する必要があります。
トレースレベル	-
ルックアップキャッシュディレクトリ名	一致させる必要はありません。
ルックアップキャッシュパーシステント	オプション。永続および非永続を共有できます。
Lookup のデータキャッシュサイズ	Integration Service は各パイプラインステージの最初の共有トランスフォーメーションにメモリを割り当てます。同じパイプライン段階の後の共有トランスフォーメーションのメモリは割り当てません。
ルックアップインデックスキャッシュのサイズ	Integration Service は各パイプラインステージの最初の共有トランスフォーメーションにメモリを割り当てます。同じパイプライン段階の後の共有トランスフォーメーションのメモリは割り当てません。
動的ルックアップキャッシュ	名前なし動的キャッシュを共有することはできません。
更新時に古い値を出力	一致させる必要はありません。
キャッシュファイル名のプレフィックス	使用しません。名前付きキャッシュを名前なしキャッシュと共有することはできません。
RecacheFrom Lookup Source	<p>Lookup トランスフォーメーションにソースからのキャッシュの再構築を設定した場合、ターゲットロード順グループのそれ以降の Lookup トランスフォーメーションにソースからのキャッシュの再構築を設定したかどうかに関わらず、それらは既存のキャッシュを共有することができます。それ以降のルックアップトランスフォーメーションにソースからのキャッシュの再構築を設定した場合、Integration Service はそれらを処理するときにキャッシュを再構築せずに共有します。</p> <p>ターゲットロード順グループの最初のルックアップトランスフォーメーションにソースからのキャッシュの再構築を設定せずに、それ以降のルックアップトランスフォーメーションにソースからのキャッシュの再構築を設定した場合、トランスフォーメーションはキャッシュを共有できません。Integration Service は、各ルックアップトランスフォーメーションの処理時にキャッシュを再構築します。</p>
Lookup/Output Ports	2 番目のルックアップトランスフォーメーションのルックアップ/出力ポートは、Integration Service がキャッシュを作成するのに使用するトランスフォーメーションのポートと一致するか、またはそれらのサブセットである必要があります。ポートの順序を一致させる必要はありません。
挿入でなければ更新	-

プロパティ	名前なし共有キャッシュの設定
更新でなければ挿入	-
日時フォーマット	-
1000 ごとの区切り	-
小数点記号	-
大文字小文字を区別した文字列比較	一致する必要があります。
NULL の順序付け	一致する必要があります。
ソート済み入力	-

名前付きルックアップキャッシュの共有

永続ルックアップキャッシュを使用し、キャッシュファイルに名前を付けることによって、複数の Lookup トランスフォーメーション間でキャッシュを共有することもできます。同じマッピング内、またはマッピングをまたいで、複数の Lookup トランスフォーメーション間でキャッシュを共有することができます。

Integration Service は、以下の手順を用いて名前付きルックアップキャッシュを共有します。

- Integration Service は最初のルックアップトランスフォーメーションを処理する際に、キャッシュディレクトリからファイル名の接頭語が同じキャッシュファイルを検索します。
- Integration Service がキャッシュファイルを検出し、ソースからキャッシュを再構築するように設定されていなければ、Integration Service は保存されたキャッシュファイルを使用します。
- Integration Service でキャッシュファイルが検出されない場合、またはソースからキャッシュが再構築されるように設定されている場合、Integration Service ではデータベーステーブルを使用してルックアップキャッシュが構築されます。
- Integration Service は、各ターゲットロード順グループを処理した後で、キャッシュファイルをディスクに保存します。
- Integration Service は、以下の規則を用いて、キャッシュファイル名の接頭語が同じである 2 番目のルックアップトランスフォーメーションを処理します。
 - トランスフォーメーションが同じターゲットロード順グループにある場合、Integration Service はメモリキャッシュを使用します。
 - トランスフォーメーションが別のターゲットロード順グループにある場合、Integration Service は永続ファイルからメモリキャッシュを再構築します。
 - ソースからキャッシュを再構築するようにトランスフォーメーションが設定されており、最初のトランスフォーメーションが別のターゲットロード順グループにある場合、Integration Service はデータベースからキャッシュを再構築します。
 - ターゲットロード順グループの最初のルックアップトランスフォーメーションにソースからのキャッシュの再構築を設定せずに、それ以降のルックアップトランスフォーメーションにソースからのキャッシュの再構築を設定した場合、Integration Service はキャッシュを再構築しません。
 - キャッシュ構造どうしが一致しない場合、Integration Service はセッションに失敗します。

ルックアップキャッシュを共有する 2 つのセッションを同時に実行した場合、Integration Service は以下の規則を用いてキャッシュファイルを共有します。

- ルックアップトランスフォーメーションがキャッシュファイルの読み込みだけを必要とする場合、Integration Service は複数のセッションを同時に処理します。
- 1 つのセッションがキャッシュファイルを更新するときに、別のセッションがそのキャッシュファイルの読み込みまたは更新を試みると、Integration Service はセッションに失敗します。例えば、動的キャッシュの使用やソースからのキャッシュの再構築が設定されている場合、ルックアップトランスフォーメーションはキャッシュファイルを更新します。

名前付きルックアップキャッシュ共有のガイドライン

名前付きキャッシュを共有するように Lookup トランスフォーメーションを設定する場合、以下のガイドラインに従います。

- キャッシュが動的ルックアップキャッシュである場合、または、ソースから再キャッシュするようにトランスフォーメーションが設定されている場合は、セッション間でルックアップキャッシュを共有しません。
- キャッシュは動的および静的ルックアップトランスフォーメーション間で共有できますが、キャッシュの場所についてはガイドラインに従う必要があります。
- トランスフォーメーションプロパティをいくつか設定し、名前付きキャッシュの共有を有効にする必要があります。
- 名前付きキャッシュに重複する行がある場合、動的ルックアップはキャッシュを共有できません。
- [Lookup Policy of Error on Multiple Match] を指定して動的 Lookup トランスフォーメーションが作成した名前付きキャッシュは、任意のルックアップポリシーを指定した静的または動的 Lookup トランスフォーメーションで共有できます。
- 最初を使用する、最後を使用する、またはすべての値を使用するルックアップポリシーを持つ動的ルックアップトランスフォーメーションにより作成された名前付きキャッシュは、同じルックアップポリシーを持つルックアップトランスフォーメーションで共有できます。
- 共有トランスフォーメーションでは、マッピングの出力ポートと同じ出力ポートを使用しなければなりません。キャッシュの条件および結果カラムは、キャッシュファイルと一致する必要があります。
- 動的ルックアップトランスフォーメーションは、同じターゲットロード順グループ内の別のルックアップトランスフォーメーションとキャッシュを共有することはできません。ターゲットロード順グループで、キャッシュの処理が並行して行われ、ターゲットロード後に完了します。PowerCenter Integration Service はキャッシュ共有を実行できません。後続のルックアップトランスフォーメーションに対しては完全なキャッシュを実行できないからです。

Integration Service によりメモリキャッシュが使用されるか、またはファイルからメモリキャッシュが構築されるかは、ルックアップトランスフォーメーションのタイプと場所によって異なります。

以下の表に、静的ルックアップトランスフォーメーションと動的ルックアップトランスフォーメーション間で、いつ名前付きキャッシュを共有できるかを示します。

表 3. 名前付きキャッシュを共有するための場所

共有キャッシュ	トランスフォーメーションの場所	共有キャッシュ
静的と静的	<ul style="list-style-type: none"> - 同じターゲットロード順グループ。 - 別のターゲットロード順グループ。 - 別のマッピング。 	<ul style="list-style-type: none"> - Integration Service はメモリキャッシュを使用する。 - Integration Service はメモリキャッシュを使用する。 - Integration Service はファイルからメモリキャッシュを作成する。
動的と動的	<ul style="list-style-type: none"> - 別のターゲットロード順グループ。 - 別のマッピング。 	<ul style="list-style-type: none"> - Integration Service はメモリキャッシュを使用する。 - Integration Service はファイルからメモリキャッシュを作成する。
動的と静的	<ul style="list-style-type: none"> - 別のターゲットロード順グループ。 - 別のマッピング。 	<ul style="list-style-type: none"> - Integration Service はファイルからメモリキャッシュを作成する。 - Integration Service はファイルからメモリキャッシュを作成する。

以下の表に、ルックアップトランスフォーメーションに名前付きキャッシュの共有をいつ設定するかに従うガイドラインを示します。

表 4. 名前付きキャッシュを共有するためのプロパティ

プロパティ	名前付き共有キャッシュの設定
Lookup SQL Override	[Lookup SQL Override] プロパティを使用する場合、すべての共有トランスフォーメーションで同じ上書きを使用する必要があります。
ルックアップテーブル名	一致する必要があります。
ルックアップキャッシュが有効	有効にする必要があります。
Lookup Policy on MultipleMatch	<ul style="list-style-type: none"> - [Lookup Policy of Error on Multiple Match] を指定して動的 Lookup トランスフォーメーションが作成した名前付きキャッシュは、任意のルックアップポリシーを指定した静的または動的 Lookup トランスフォーメーションで共有できます。 - [Lookup Policy of Use First or Use Last] を指定して動的 Lookup トランスフォーメーションが作成した名前付きキャッシュは、同じルックアップポリシーを指定した Lookup トランスフォーメーションで共有できます。 - 名前付きキャッシュは、[ルックアップポリシーですべての値を使用] を指定した動的ルックアップトランスフォーメーションで共有できます（同じルックアップポリシーを指定した別のアクティブなルックアップトランスフォーメーションとキャッシュを共有する場合）。
ルックアップ条件	共有トランスフォーメーションは、ルックアップ条件に同じポートを使用する必要があります。条件は異なる演算子を使用することができますが、ポートは同じでなければなりません。
接続情報	接続は同じである必要があります。セッションを設定する際に、データベース接続を一致させる必要があります。

プロパティ	名前付き共有キャッシュの設定
ソースタイプ	一致する必要があります。
トレースレベル	-
ルックアップキャッシュディレクトリ名	一致する必要があります。
ルックアップキャッシュパーシステント	有効にする必要があります。
Lookup のデータキャッシュサイズ	同じマッピング内のトランスフォーメーションによりキャッシュが共有される場合、Integration Service により各パイプラインステージの最初の共有トランスフォーメーションにメモリが割り当てられます。同じパイプライン段階の後の共有トランスフォーメーションのメモリは割り当てません。
ルックアップインデックスキャッシュのサイズ	同じマッピング内のトランスフォーメーションによりキャッシュが共有される場合、Integration Service により各パイプラインステージの最初の共有トランスフォーメーションにメモリが割り当てられます。同じパイプライン段階の後の共有トランスフォーメーションのメモリは割り当てません。
動的ルックアップキャッシュ	静的キャッシュおよび動的キャッシュの共有の詳細については、「 名前付きルックアップキャッシュ共有のガイドライン 」(ページ 332)を参照してください。
更新時に古い値を出力	一致させる必要はありません。
動的キャッシュの更新	一致させる必要はありません。
キャッシュファイル名のプレフィックス	一致する必要があります。接頭語のみを入力してください。idx または.dat と入力しないでください。名前付きキャッシュを名前なしキャッシュと共有することはできません。
Recache from Lookup Source	Lookup トランスフォーメーションにソースからのキャッシュの再構築を設定した場合、ターゲットロード順グループのそれ以降の Lookup トランスフォーメーションにソースからのキャッシュの再構築を設定したかどうかに関わらず、それらは既存のキャッシュを共有することができます。それ以降のルックアップトランスフォーメーションにソースからのキャッシュの再構築を設定した場合、Integration Service はそれら进行处理するときにキャッシュを再構築せずに共有します。 ターゲットロード順グループの最初のルックアップトランスフォーメーションにソースからのキャッシュの再構築を設定せずに、それ以降のルックアップトランスフォーメーションにソースからのキャッシュの再構築を設定した場合、Integration Service はキャッシュを再構築しません。
Lookup/Output Ports	ルックアップ/出力ポートは同じでなければなりませんが、同じ順序である必要はありません。
挿入でなければ更新	-
更新でなければ挿入	-
1000 ごとの区切り	-
小数点記号	-
大文字小文字を区別した文字列比較	-

プロパティ	名前付き共有キャッシュの設定
NULL の順序付け	-
ソート済み入力	一致する必要があります。

注: 異なるオペレーティングシステムで作成されたルックアップキャッシュを共有することはできません。例えば、UNIX の Integration Service で作成されたルックアップキャッシュを読み込むことができるのは UNIX の Integration Service のみであり、Windows の Integration Service で作成されたルックアップキャッシュを読み込むことができるのは Windows の Integration Service のみです。

ルックアップキャッシュに関するヒント

小さなルックアップテーブルはキャッシュします。

小さなルックアップテーブルをキャッシュすることにより、パフォーマンスが向上します。

静的ルックアップテーブルに永続ルックアップキャッシュを使用してください。

セッションとセッションの間にルックアップテーブルが変更されない場合は、永続ルックアップキャッシュを使用するように Lookup トランスフォーメーションを設定します。統合サービスは、キャッシュファイルを保存して再利用することにより、ルックアップテーブルの読み取りにかかる時間を節約します。

マッピングの各実行の間にルックアップテーブルが変更されない場合は、永続ルックアップキャッシュを使用するようにルックアップトランスフォーメーションを設定します。統合サービスは、キャッシュファイルを保存して再利用することにより、ルックアップテーブルの読み取りにかかる時間を節約します。

第 19 章

動的ルックアップキャッシュ

この章では、以下の項目について説明します。

- [動的ルックアップキャッシュの概要, 336 ページ](#)
- [動的ルックアップのプロパティ, 337 ページ](#)
- [ルックアップトランスフォーメーションの値, 341 ページ](#)
- [動的ルックアップキャッシュの更新, 344 ページ](#)
- [動的ルックアップを使用したマッピング, 346 ページ](#)
- [条件付きの動的キャッシュの更新, 349 ページ](#)
- [式の結果を使用した動的キャッシュの更新, 350 ページ](#)
- [キャッシュとルックアップソースの同期, 351 ページ](#)
- [動的ルックアップキャッシュの例, 352 ページ](#)
- [動的ルックアップキャッシュのルールとガイドライン, 353 ページ](#)

動的ルックアップキャッシュの概要

動的ルックアップキャッシュを使用してキャッシュがターゲットに同期し続けるようにします。動的キャッシュは、リレーショナルルックアップ、フラットファイルルックアップ、またはパイプラインルックアップとともに使用できます。動的キャッシュは、リレーショナルルックアップまたはフラットファイルルックアップとともに使用できます。

統合サービスは、最初のルックアップ要求を処理する際に動的ルックアップキャッシュを構築します。トランスフォーメーションに渡す各行に対するルックアップ条件に基づいて、キャッシュに問い合わせます。統合サービスは、各行を処理するときにルックアップキャッシュを更新します。

ルックアップクエリの結果、行タイプ、ルックアップトランスフォーメーションのプロパティに基づいて、統合サービスは、ソースから行を読み取るときに、動的ルックアップキャッシュに以下のアクションのいずれかを実行します。

キャッシュに行を挿入

行がキャッシュになく、行をキャッシュに挿入するようにルックアップトランスフォーメーションを設定している場合、統合サービスは行をキャッシュに挿入します。入力ポートまたは生成されたシーケンス ID に基づいて行をキャッシュに挿入するように、トランスフォーメーションを設定することができます。統合サービスはその行に挿入のフラグを設定します。

キャッシュ内の行を更新

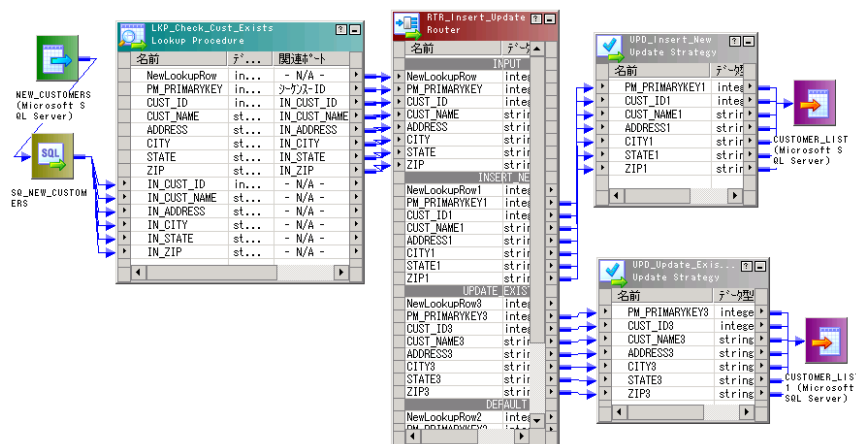
行がキャッシュにあり、キャッシュの行を更新するようにルックアップトランスフォーメーションを設定している場合、統合サービスは行を更新します。統合サービスは入力ポートに基づいてキャッシュ内の行を更新します。統合サービスはその行に更新行のフラグを設定します。

キャッシュに変更を加えない

行がキャッシュにあり、新しい行のみをキャッシュに挿入するようにルックアップトランスフォーメーションを設定している場合、統合サービスは何も変更しません。または、行がキャッシュになく、ユーザが既存の行のみを更新すると指定した場合です。もしくは、行がキャッシュにあっても、ルックアップ条件に基づき、何も変更されない場合です。統合サービスはその行に未変更のフラグを設定します。

NewLookupRow の値に基づいて、動的ルックアップトランスフォーメーションとともにルータトランスフォーメーションまたはフィルタトランスフォーメーションを設定して、挿入行または更新行をターゲットテーブルにルーティングできます。変更のない行を別のターゲットテーブルやフラットファイルにルーティングしたり、それらを削除したりできます。

以下の図に、動的ルックアップキャッシュを使用するルックアップトランスフォーメーションを含むマッピングを示します。



動的ルックアップのプロパティ

動的キャッシュを備えたルックアップトランスフォーメーションは以下のプロパティを持ちます。

- **NewLookupRow**。Designer は、動的キャッシュを使用するように設定されたルックアップトランスフォーメーションに、このポートを追加します。Integration Service がキャッシュに行を挿入するか、キャッシュ内の行を更新するか、またはキャッシュに何も変更を加えないかを数値で示します。ルックアップキャッシュとターゲットテーブルの同期を保持するには、NewLookupRow 値が 1 または 2 である場合に行をターゲットに渡します。
- **関連する式**。ルックアップポートまたは関連するポートを、式、入出力ポート、またはシーケンス ID に関連付けます。Integration Service は関連する式のデータを使用して、ルックアップキャッシュに行を挿入するか、キャッシュ内の行を更新します。シーケンス ID と関連付けた場合、Integration Service はルックアップキャッシュに挿入された行に対してプライマリキーを生成します。
- **更新の際に NULL 入力を無視する**。ユーザーが動的キャッシュを使用するようにルックアップトランスフォーメーションを設定したとき、Designer はルックアップ/出力ポートに対してこのポートプロパティを有効にします。このカラムのデータに NULL 値が含まれているときに Integration Service がキャッシュ内のカラムを更新しないようにする場合は、このプロパティを選択します。

- **比較において無視。** 動的キャッシュを使用するようにルックアップトランスフォーメーションを設定したとき、Designer はルックアップ条件に使用されていないルックアップ/出力ポートに対してこのポートプロパティを有効にします。Integration Service は、すべてのルックアップポートの値と、それらに関連付けられている入力ポートの値をデフォルトで比較します。行を更新する前に値を比較して、Integration Service にポートを無視させる場合はこのプロパティを選択します。
- **動的キャッシュの更新条件。** Integration Service で動的キャッシュを条件付きで更新できるようにします。入力行のキャッシュを更新するかどうかを決めるブール式を作成できます。または、Integration Service でキャッシュを入力行の式の結果で更新することができます。この式には、入力行の値やルックアップキャッシュの値を含めることができます。

NewLookupRows

動的キャッシュを使用する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と、Designer はそのトランスフォーメーションに NewLookupRow ポートを追加します。Integration Service は、ルックアップキャッシュに対して実行するアクションに応じて、ポートに値を割り当てます。

以下の表に、有効な NewLookupRow 値を示します。

NewLookupRow 値	説明
0	Integration Service はキャッシュ内の行の更新、またはキャッシュへの行の挿入を行わない。
1	Integration Service は行をキャッシュに挿入する。
2	Integration Service はキャッシュ内の行を更新する。

Integration Service は、ルックアップクエリの結果および定義したルックアップトランスフォーメーションのプロパティに応じて、行を読み込む際にルックアップキャッシュを変更します。キャッシュに行を挿入するときは、NewLookupRow ポートに値 0 を、キャッシュの行を更新するときは値 1 を、キャッシュを変更しないときは値 2 を割り当てます。

NewLookupRow 値は、Integration Service がルックアップキャッシュを変更する方法を示します。行の種類は変更しません。したがって、フィルタまたはルータトランスフォーメーションとアップデートストラテジトランスフォーメーションを使用して、ターゲットテーブルとルックアップキャッシュの同期を保持します。

フィルタトランスフォーメーションは、アップデートストラテジトランスフォーメーションに新規行および更新行を渡してから、キャッシュに格納されたターゲットに新規行および更新行を渡すように設定します。アップデートストラテジトランスフォーメーションを使って、NewLookupRow 値に応じて、挿入または更新する各行の行種類を変更します。

キャッシュで変更されない既存の行を削除するか、別のターゲットに渡すことができます。

NewLookupRow の値に基づいて、フィルタトランスフォーメーションにフィルタ条件を定義します。たとえば、下記の条件を使って挿入行と更新行の両方をキャッシュに格納されたターゲットへ渡します。

NewLookupRow != 0

関連項目：

- [「アップストリームのアップデートストラテジトランスフォーメーションの設定」 \(ページ 346\)](#)

関連する式

動的ルックアップキャッシュを設定するときは、各ルックアップポートを入力ポート、入出力ポート、シーケンス ID、または式に関連付ける必要があります。入出力ポートを選択すると、Designer はその入出力ポート

をルックアップ条件で使用されるルックアップ/出力ポートに関連付けます。式を設定すると、Integration Service は関連する式の結果でキャッシュを更新します。この式には、入力値やルックアップキャッシュの値を含めることができます。

ルックアップポートを入出力ポートに関連付けるには、ルックアップポートの [関連する式] カラムをクリックします。リストからポートを選択します。

式を作成するには、ルックアップポートの [関連] カラムをクリックします。リストから [関連する式] を選択します。式エディタが表示されます。

ターゲットテーブルのカラム用に生成キーを作成するには、[関連する式] カラムでシーケンス ID を選択します。ルックアップポートの入力ポートではなく生成キーを Bigint、Integer、Small Integer のいずれかのデータタイプに関連付けることができます。Bigint ルックアップポートの場合、生成キーの最大値は 9,223,372,036,854,775,807 です。Integer または Small Integer ルックアップポートの場合、生成キーの最大値は 2,147,483,647 です。

[関連する式] カラムでシーケンス ID を選択すると、Integration Service はルックアップキャッシュに行を挿入する際にキーを生成します。

Integration Service は、以下の手順に従ってシーケンス ID を生成します。

1. Integration Service が動的ルックアップキャッシュを作成するときは、動的ルックアップキャッシュ内にシーケンス ID を有する各ポートごとに値の範囲を追跡します。
2. Integration Service がデータの行をキャッシュに挿入するときは、最大のシーケンス ID の値に 1 を加えることにより、ポートのキーを生成します。
3. Integration Service は、生成されるシーケンス ID の最大値に達すると、1 から再び始めます。そして、Integration Service は既存の最小値から 1 を引いた値に達するまで、各シーケンス ID に 1 を加えます。Integration Service から一意のシーケンス ID 番号がなくなると、セッションは失敗します。

Integration Service は、キャッシュに挿入する行ごとに、シーケンス ID を生成します。

関連項目：

- [「式の結果を使用した動的キャッシュの更新」 \(ページ 350\)](#)

NULL 値

動的ルックアップキャッシュとターゲットテーブルを更新する際、ソースデータには NULL 値が含まれている可能性があります。Integration Service は以下の方法で NULL 値を処理できます。

- **NULL 値を挿入する。** Integration Service はソースの NULL 値を使用し、ソースの NULL 値をすべて使ってルックアップキャッシュとターゲットテーブルを更新します。
- **NULL 値を無視する。** Integration Service はソースの NULL 値を無視し、ソースの NULL 以外の値のみを使用してルックアップキャッシュとターゲットテーブルを更新します。

ソースデータに NULL 値が含まれていることがわかっており、Integration Service で NULL 値を使ってルックアップキャッシュやターゲットを更新しないようにする場合は、対応するルックアップ/出力ポートに対して [NULL を無視] プロパティを選択します。

たとえば、マスター顧客テーブルを更新したいとします。ソースには新規顧客と苗字に変更のあった既存顧客が含まれています。ソースにはカスタマ ID と名前に変更のあった顧客の名前が含まれていますが、住所カラムに NULL 値が含まれています。マスター顧客テーブルの既存の住所情報を保持しながら、新規顧客を挿入し、既存の顧客名を更新したいとします。

たとえば、マスター顧客テーブルに以下のデータが含まれるとします。

Primary Key	CUST_ID	CUST_NAME	ADDRESS	CITY	STATE	ZIP
100001	80001	Marion James	100 Main St.	Mt. View	CA	94040

Primary Key	CUST_ID	CUST_NAME	ADDRESS	CITY	STATE	ZIP
100002	80002	Laura Jones	510 Broadway Ave.	Raleigh	NC	27601
100003	80003	Shelley Lau	220 Burnside Ave.	Portland	OR	97210

ソースには下記のデータが含まれています。

CUST_ID	CUST_NAME	ADDRESS	CITY	STATE	ZIP
80001	Marion Atkins	NULL	NULL	NULL	NULL
80002	Laura Gomez	NULL	NULL	NULL	NULL
99001	Jon Freeman	555 6th Ave.	San Jose	CA	95051

マッピング内のルックアップトランスフォーメーションで [挿入でなければ更新] を選択します。ルックアップトランスフォーメーションの中のすべてのルックアップ/出力ポートについて [NULL を無視] オプションを選択します。セッションを実行すると、Integration Service がソースデータ内の NULL 値を無視し、NULL 以外の値を使用してルックアップキャッシュとターゲットテーブルを更新します。

PRIMARYKEY	CUST_ID	CUST_NAME	ADDRESS	CITY	STATE	ZIP
100001	80001	Marion Atkins	100 Main St.	Mt. View	CA	94040
100002	80002	Laura Gomez	510 Broadway Ave.	Raleigh	NC	27601
100003	80003	Shelley Lau	220 Burnside Ave.	Portland	OR	97210
100004	99001	Jon Freeman	555 6th Ave.	San Jose	CA	95051

注: NULL を無視するように選択した場合、Integration Service がルックアップキャッシュに書き込む値と同じ値をターゲットに出力するように確認する必要があります。NULL 値を無視するように選択した場合、ターゲットに NULL 入力値を渡すと、ルックアップキャッシュとターゲットテーブルが非同期になる可能性があります。Integration Service がキャッシュ内の行を更新するときに、ルックアップ/出力ポートから出力させる値に基づいて、マッピングを設定します。

- **新しい値。**ルックアップトランスフォーメーションからターゲットにルックアップ/出力ポートのみを接続します。
- **古い値。**ルックアップトランスフォーメーションの後、およびフィルタトランスフォーメーションまたはルータトランスフォーメーションの前に、式トランスフォーメーションを追加します。ターゲットテーブルの各ポートに式トランスフォーメーションの出力ポートを追加し、NULL 入力値をターゲットに出力しないように式を作成します。

比較においてポートを無視

動的ルックアップキャッシュを使用するセッションを実行するとき、Integration Service はすべてのルックアップポートの値と、それらに関連付けられている入力ポートの値をデフォルトで比較します。これらの値を比較することによって、ルックアップキャッシュ内の行を更新するかどうかを決定します。入力ポートの値がルックアップポートの値と異なる場合、Integration Service はキャッシュ内の行を更新する。

比較しないポートがある場合は、ポートを比較するときに Integration Service に無視させるポートを選択できます。Designer は、ルックアップ条件にルックアップ/出力ポートが使用されていないときにだけ、そのポートに対してこのプロパティを有効にします。比較中に一部のポートを無視することによって、パフォーマンスが向上します。

更新を必要とするデータが行に含まれているかどうかを示すカラムがソースデータにある場合に、これを使用します。キャッシュおよびテーブル内の行を更新するかどうかを示しているポート以外のすべてのルックアップポートについて [比較において無視] プロパティを選択します。

ルックアップトランスフォーメーションでは、少なくとも 1 つ以上のポートを比較する必要があります。すべてのポートを無視すると、Integration Service はセッションに失敗します。

[比較において無視] プロパティをクリアするには、ポートとルックアップポートを関連付けます。[比較において無視] プロパティをクリアすると、PowerCenter Integration Service によりキャッシュ内の行が更新されます。

SQL オーバーライド

動的キャッシュを使ってルックアップトランスフォーメーションに WHERE 句を追加する場合、フィルタトランスフォーメーションをルックアップトランスフォーメーションの前に接続して、キャッシュまたはターゲットテーブルに挿入しない行をフィルタリングします。フィルタトランスフォーメーションを含めない場合、キャッシュとターゲットテーブルの間で結果に矛盾が生じる可能性があります。

たとえば、ルックアップトランスフォーメーションを設定して従業員テーブルの EMP に対して動的ルックアップを行い、EMP_ID で一致する行を探すとします。下記のルックアップ SQL 上書きを定義します。

```
SELECT EMP_ID, EMP_STATUS FROM EMP ORDER BY EMP_ID, EMP_STATUS WHERE EMP_STATUS = 4
```

最初にセッションマッピングを実行する際、統合サービスはルックアップ SQL オーバーライドに基づいてターゲットテーブルからルックアップキャッシュを作成します。キャッシュ内のすべての行は WHERE 句の条件、EMP_STATUS = 4 に一致します。

例えば、統合サービスは指定したルックアップ条件に一致するソース行を読み取りますが、EMP_STATUS の値は 2 です。ターゲットに EMP_STATUS が 2 の行がある可能性があっても、統合サービスは SQL オーバーライドのため、キャッシュの行を見つけないことができません。統合サービスはキャッシュに行を挿入して、行をターゲットテーブルに渡します。行がすでに存在する場合、統合サービスがこの行をターゲットテーブルに挿入すると、結果に矛盾が生じる可能性があります。さらに、キャッシュ内のすべての行が SQL オーバーライドの WHERE 句の条件に一致するとは限りません。

WHERE 句に一致する行のみを確実にキャッシュに挿入するには、フィルタトランスフォーメーションをルックアップトランスフォーメーションの前に追加し、ルックアップ SQL 上書きで WHERE 句内の条件をフィルタ条件として定義します。

上記の例では、次のフィルタトランスフォーメーションのフィルタ条件と SQL オーバーライドの WHERE 句を入力します。

```
EMP_STATUS = 4
```

ルックアップトランスフォーメーションの値

ルックアップトランスフォーメーションには、入力ポート、ルックアップ、出力ポートの値が含まれます。動的ルックアップキャッシュを有効にする場合、出力ポートの値は動的ルックアップキャッシュの設定方法によって異なります。

ルックアップトランスフォーメーションには次のタイプの値が含まれます。

入力値

統合サービスがルックアップトランスフォーメーションに渡す値。

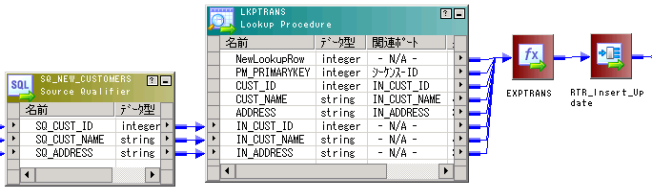
ルックアップ値

統合サービスがキャッシュに挿入する値。

出力値

統合サービスがルックアップトランスフォーメーションの出力ポートから渡す値。ルックアップ/出力ポートの出力値は、統合サービスが行を更新するときに古い値を出力するか、新しい値を出力するかのどちらかをユーザーが選択したかによって異なります。

例えば、動的ルックアップキャッシュを使用する以下のルックアップトランスフォーメーションがあるとし



例えば、次のオブジェクトを持つマッピングを作成します。

Source > Lookup transformation > Expression transformation> Router transformation > Target

ルックアップトランスフォーメーションに対して動的ルックアップキャッシングを有効にします。

以下のルックアップ条件を定義します。

IN_CUST_ID = CUST_ID

デフォルトでは、ルックアップトランスフォーメーションに入力されるすべての行の種類は「挿入」です。キャッシュおよびターゲットテーブルで挿入と更新の両方を実行するには、ルックアップトランスフォーメーションで【挿入でなければ更新】プロパティを選択します。

次にセッションを実行する際のキャッシュ内の行、入力行、ルックアップ行、および出力行の値を説明します。

次にマッピングを実行する際のキャッシュ内の行、入力行、ルックアップ行、および出力行の値を説明します。

初期キャッシュ値

セッションマッピングを実行する場合、統合サービスは次のデータを持つターゲットテーブルからルックアップキャッシュを構築します。

PK_PRIMARYKEY	CUST_ID	CUST_NAME	ADDRESS
100001	80001	Marion James	100 Main St.
100002	80002	Laura Jones	510 Broadway Ave.
100003	80003	Shelley Lau	220 Burnside Ave.

入力値

ソースには、ターゲットテーブルに存在する行と存在しない行が含まれます。ソース修飾子トランスフォーメーションからルックアップトランスフォーメーションに次の行が渡されます。

SQ_CUST_ID	SQ_CUST_NAME	SQ_ADDRESS
80001	Marion Atkins	100 Main St.
80002	Laura Gomez	510 Broadway Ave.
99001	Jon Freeman	555 6th Ave.

ルックアップ値

統合サービスは、ルックアップ条件に応じてキャッシュ内の値をルックアップします。既存のカスタマ ID80001 および 80002 についてはキャッシュ内の行を更新します。カスタマ ID99001 についてはキャッシュ内に行を挿入します。統合サービスは新規行に対して新規キー（PK_PRIMARYKEY）を生成します。

次の表に、ルックアップから返される行および値を示します。

PK_PRIMARYKEY	CUST_ID	CUST_NAME	ADDRESS
100001	80001	Marion Atkins	100 Main St.
100002	80002	Laura Gomez	510 Broadway Ave.
100004	99001	Jon Freeman	555 6th Ave.

出力値

統合サービスは、動的キャッシュに対して実行する挿入および更新に基づいて、ルックアップトランスフォーメーションの行にフラグを設定します。これらの行は式トランスフォーメーションを通じてルータトランスフォーメーションに渡され、ルータトランスフォーメーションは挿入行および更新行をフィルタリングし、アップデートストラテジトランスフォーメーションに渡します。アップデートストラテジトランスフォーメーションは、NewLookupRow ポートの値に基づいて行にフラグを設定します。

ルックアップ/出力ポートと入力/出力出力ポートの出力値は、統合サービスが行を更新するときに古い値を出力するか、新しい値を出力するかのどちらかをユーザーが選択したかによって異なります。ただし、NewLookupRow ポートと、シーケンス ID を使用するルックアップ/出力ポートの出力値は新しい行と更新された行で同じになります。

新しい値を出力するように選択した場合、ルックアップ/出力ポートは次の値を出力します。

NewLookupRow	PK_PRIMARYKEY	CUST_ID	CUST_NAME	ADDRESS
2	100001	80001	Marion Atkins	100 Main St.
2	100002	80002	Laura Gomez	510 Broadway Ave.
1	100004	99001	Jon Freeman	555 6th Ave.

古い値を出力するように選択した場合、ルックアップ/出力ポートは次の値を出力します。

NewLookupRow	PK_PRIMARYKEY	CUST_ID	CUST_NAME	ADDRESS
2	100001	80001	Marion James	100 Main St.
2	100002	80002	Laura Jones	510 Broadway Ave.
1	100004	99001	Jon Freeman	555 6th Ave.

統合サービスがルックアップキャッシュ内の行を更新するときは、キャッシュ内およびターゲットテーブル内の行のプライマリキー（PK_PRIMARYKEY）の値を使用します。

統合サービスはシーケンス ID を使って、キャッシュに存在しない顧客のプライマリキーを生成します。統合サービスはプライマリキーの値をルックアップキャッシュに挿入し、その値をルックアップ/出力ポートに返します。

統合サービスは、入力/出力出力ポートから入力値に一致する値を出力します。

注: 入力値が NULL で、ユーザーが関連入力ポートに対して [NULL を無視] プロパティを選択した場合、入力値はルックアップ値または入力/出力出力ポートからの値と等しくなりません。 [NULL を無視] プロパティを選択した場合、ターゲットに NULL 値を渡すと、ルックアップキャッシュとターゲットテーブルが非同期になる可能性があります。 NULL 値をターゲットに渡さないことを確認してください。

動的ルックアップキャッシュの更新

統合サービスでは、行タイプ、クエリ結果、ルックアップトランスフォーメーションのプロパティに基づいて、動的キャッシュを更新します。

動的ルックアップキャッシュを使用する場合、ルックアップトランスフォーメーションに入力される行の行タイプを「挿入」または「更新」として定義します。一部の行を「挿入」として一部を「更新」と定義したり、すべてを「挿入」または「更新」と定義することができます。デフォルトでは、ルックアップトランスフォーメーションに入力される全行の行種類は「挿入」です。アップデートストラテジトランスフォーメーションをルックアップトランスフォーメーションの前に追加して、行タイプを「更新」と定義することができます。

統合サービスはキャッシュに行を挿入または行を更新するか、キャッシュを変更しません。デフォルトでは、ルックアップトランスフォーメーションで、行タイプが「挿入」の場合はキャッシュに行が挿入され、行タイプが「更新」の場合は行が更新されます。

ただし、以下のルックアッププロパティを設定して、統合サービスでキャッシュへの挿入および更新を処理する方法を変更できます。

- **挿入でなければ更新。** 行タイプが「挿入」で、ルックアップトランスフォーメーションに入力される行に対して使用します。このオプションは、キャッシュ内にデータがすでに存在する場合にキャッシュの行を更新するのに使用します。このオプションを有効にしない場合、統合サービスは行の有無に関係なくキャッシュに行を挿入します。
- **更新でなければ挿入。** 行タイプが「更新」で、ルックアップトランスフォーメーションに入力される行に対して使用します。このオプションは、キャッシュ内にデータが存在しない場合に更新行を挿入するのに使用します。このオプションを有効にしない場合、統合サービスはキャッシュ内に行が存在しないと行を無視します。

注: [挿入でなければ更新] プロパティと [更新でなければ挿入] プロパティのどちらか、または両方を選択できます。また両方とも選択しないこともできます。

ルックアップトランスフォーメーションは、ルックアップソースと動的キャッシュを同期させるように設定できます。ルックアップソースとキャッシュを同期させる場合、ルックアップトランスフォーメーションは行をルックアップソースに直接挿入します。挿入行はターゲットに渡しません。この設定を使用して、同じターゲットに行を挿入する複数のセッションをルックアップトランスフォーメーションで実行します。

挿入でなければ更新

[更新でなければ挿入] プロパティを使用して、行タイプが「挿入」のときに動的ルックアップキャッシュの既存の行を更新します。

このプロパティは、行タイプが「挿入」で、ルックアップトランスフォーメーションに入力される行に対して使用します。更新行などの他のタイプの行がルックアップトランスフォーメーションに入力される場合、**[挿入でなければ更新]** プロパティは統合サービスの行の処理方法に影響しません。

[挿入でなければ更新] プロパティを選択し、ルックアップトランスフォーメーションに入力される行のタイプが「挿入」の場合、統合サービスは、行が新規行のときはキャッシュに挿入します。インデックスキャッシュに行は存在するがデータキャッシュが現在の行と異なる場合に、統合サービスはデータキャッシュ内の行を更新します。

【挿入でなければ更新】を選択せず、ルックアップトランスフォーメーションに入力される行のタイプが「挿入」の場合、統合サービスは、行が新規行のときはキャッシュに挿入し、すでに存在する行のときはキャッシュを変更しません。

以下の表に、ルックアップトランスフォーメーションに入力される行のタイプが「挿入」の場合に、統合サービスがルックアップキャッシュを変更する方法を示します。

【挿入でなければ更新】オプション	キャッシュ内で 行が見つかった か	データキャッ シュが異なる か	ルックアップキ ャッシュの結果	NewLookupRow 値
クリア - 挿入のみ	はい	-	変更なし	0
クリア - 挿入のみ	いいえ	-	挿入	1
選択済み	はい	はい	更新	2 ¹
選択済み	はい	いいえ	変更なし	0
選択済み	いいえ	-	挿入	1

¹ ルックアップ条件にないすべてのルックアップポートに対して【NULL を無視】を選択し、それらのポートすべてに NULL 値が含まれている場合、統合サービスはキャッシュを変更せず、NewLookupRow 値は 0 となります。

更新でなければ挿入

【更新でなければ挿入】プロパティを使用して、行タイプが「更新」のときに動的ルックアップキャッシュの新規の行を挿入します。

ルックアップトランスフォーメーションで【更新でなければ挿入】プロパティを選択できます。このプロパティは、行タイプが「更新」で、ルックアップトランスフォーメーションに入力される行に対してのみ使用します。他のタイプの行、例えば挿入行がルックアップトランスフォーメーションに入力される場合、このプロパティは統合サービスの行の処理方法に影響しません。

このプロパティを選択した場合で、ルックアップトランスフォーメーションに入力される行のタイプが「更新」の場合、その行がすでにインデックスキャッシュに存在し、キャッシュデータが既存の行と異なっていれば、統合サービスはその行を更新します。統合サービスは、新規行の場合にその行をキャッシュに挿入します。

このプロパティを選択していない場合、ルックアップトランスフォーメーションに入力される行のタイプが「更新」のときは、統合サービスは行がすでに存在する行ならばキャッシュ内で更新し、行が新規行ならばキャッシュを変更しません。

ルックアップ条件にないすべてのルックアップポートに対して【NULL を無視】を選択し、それらのポートすべてに NULL 値が含まれている場合、統合サービスはキャッシュを変更せず、NewLookupRow 値は 0 となります。

以下の表に、ルックアップトランスフォーメーションに入力される行のタイプが「更新」の場合に、統合サービスがルックアップキャッシュを変更する方法を示します。

【更新でなければ挿入】オプション	キャッシュ内で行が見つかったか	データキャッシュが異なるか	ルックアップキャッシュの結果	NewLookupRow 値
クリア（更新のみ）	はい	はい	更新	2
クリア（更新のみ）	はい	いいえ	変更なし	0
クリア（更新のみ）	いいえ	-	変更なし	0
選択済み	はい	はい	更新	2
選択済み	はい	いいえ	変更なし	0
選択済み	いいえ	-	挿入	1

動的ルックアップを使用したマッピング

動的なルックアップトランスフォーメーションを含むマッピングを設定するときは、ルックアップトランスフォーメーションのアップストリームトランスフォーメーションが、ルックアップトランスフォーメーションによる動的キャッシュの更新方法に影響を与えるように設定します。ルックアップトランスフォーメーションのダウストリームトランスフォーメーションは、Integration Service がターゲットで更新のマークが付いている行を更新し、挿入のマークが付いている行を挿入するように設定します。

アップストリームのアップデートストラテジトランスフォーメーションの設定

アップストリームのアップデートストラテジトランスフォーメーションを設定し、ルックアップトランスフォーメーションが受け取る行の行種類を変更できます。

動的ルックアップキャッシュを使用する場合、アップデートストラテジトランスフォーメーションを使って下記の行の行種類を定義します。

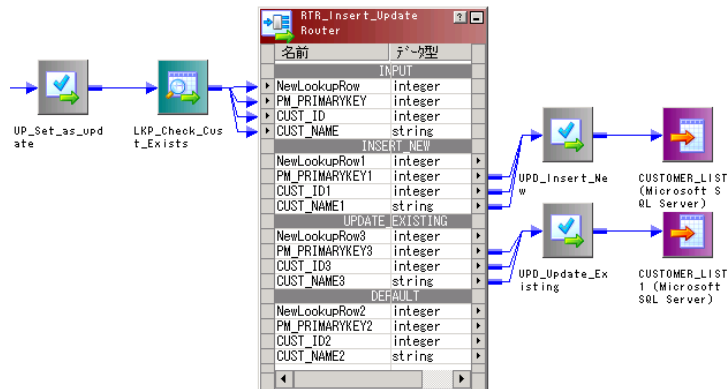
- ルックアップトランスフォーメーションに入力される行。**デフォルトでは、ルックアップトランスフォーメーションに入力される全行の行種類は「挿入」です。ただし、行を「更新」として定義する場合、または一部を「更新」、一部を「挿入」として定義する場合、ルックアップトランスフォーメーションの前にアップデートストラテジトランスフォーメーションを使用します。
- ルックアップトランスフォーメーションから出力される行。**NewLookupRow 値は、統合サービスによってルックアップキャッシュがどのように変更されたかを示しますが、行種類は変更しません。ルックアップトランスフォーメーションの後にフィルタまたはルータトランスフォーメーションを使って、NewLookupRow 値に基づいてルックアップトランスフォーメーションから出力される行を指定します。フィルタまたはルータトランスフォーメーションの後にアップデートストラテジトランスフォーメーションを使って、マッピング内でのターゲットの定義の前に挿入行または更新行をフラグ設定します。

注: 未変更の行を削除する場合、フィルタまたはルータトランスフォーメーションから NewLookupRow が 0 のターゲットに行を接続しないでください。

ルックアップトランスフォーメーションに入力される行の種類を「挿入」と定義する場合、ルックアップトランスフォーメーションの [挿入でなければ更新] プロパティを使用します。ルックアップトランスフォーメーション

ションに入力される行の種類を「更新」と定義する場合、ルックアップトランスフォーメーションの[更新でなければ挿入] プロパティを使用します。ルックアップトランスフォーメーションに入力される行の一部を「更新」、一部を「挿入」と定義する場合、[更新でなければ挿入] プロパティか [挿入でなければ更新] プロパティ、または両方のプロパティを使用します。

以下の図に、動的キャッシュを使用している複数のアップデートストラテジトランスフォーメーションと1つのルックアップトランスフォーメーションを含むマッピングを示します。



この例では、ルックアップトランスフォーメーションの前のアップデートストラテジトランスフォーメーションがすべての行に更新フラグを設定します。ルックアップトランスフォーメーションで[更新でなければ挿入] プロパティを選択します。ルータトランスフォーメーションは挿入行を Insert_New アップデートストラテジトランスフォーメーションに送り、更新行を Update_Existing アップデートストラテジトランスフォーメーションへ送ります。ターゲットに接続されていない出力行は削除されます。ルックアップトランスフォーメーションの右側の2つのアップデートストラテジトランスフォーメーションは、ターゲットに対して行に挿入フラグまたは更新フラグを設定します。

ダウンストリームトランスフォーメーションの設定

ダウンストリームトランスフォーメーションを設定して、動的ルックアップキャッシュとターゲットが同期するようにします。

動的ルックアップキャッシュを使用すると、統合サービスはルックアップキャッシュに書き込んでからターゲットテーブルに書き込みます。統合サービスがデータをターゲットに書き込まない場合、ルックアップキャッシュとターゲットテーブルは非同期になります。例えば、ターゲットデータベースはデータを拒否することがあります。

ルックアップキャッシュがルックアップテーブルに同期し続けるようにするには、次のガイドラインに従います。

- NewLookupRow 値が 1 または 2 の場合、ルータトランスフォーメーションを使用し、行をキャッシュに格納されたターゲットに渡します。
- NewLookupRow 値が 0 の場合、ルータトランスフォーメーションを使用し、行を削除します。あるいは、行を別のターゲットに出力します。
- アップデートストラテジトランスフォーメーションをルックアップトランスフォーメーションの後に使用して、行にターゲットへの挿入フラグまたは更新フラグを設定します。
- セッションを実行する場合、エラーしきい値を 1 に設定します。エラーしきい値を 1 に設定すると、セッションは最初のエラーに遭遇したときに失敗します。統合サービスは、新しいキャッシュファイルをディスクに書き込みません。代わりに、元のキャッシュファイルがあれば、それを回復します。また、セッション実行前のターゲットテーブルをターゲットデータベースに回復する必要もあります。

- 統合サービスがルックアップキャッシュに書き込む値と同じ値をルックアップトランスフォーメーションがターゲットに出力することを確認します。更新時に新しい値を出力するように選択するときは、ターゲットテーブルに入力/出力出力ポートの代わりにルックアップ/出力ポートを接続するだけです。更新時に古い値を出力すると選択した場合は、ルックアップトランスフォーメーションの後で、ルータトランスフォーメーションの前に式トランスフォーメーションを追加します。ターゲットテーブルの各ポートに式トランスフォーメーションの出力ポートを追加し、NULL 入力値をターゲットに出力しないように式を作成します。
- セッションプロパティで [Treat Source Rows As] プロパティを [Data Driven] に設定します。
- アップデートストラテジターゲットテーブルオプションを定義する場合は、[挿入と更新] を [更新] として選択します。これにより、統合サービスは「更新」と記された行を更新し、「挿入」と記された行を挿入します。セッションのプロパティの [マッピング] タブの [トランスフォーメーション] ビューでこれらのオプションを選択します。

ルックアップ条件カラムにおける NULL 値

統合サービスはルックアップ条件カラム内の NULL 値を含む行を、キャッシュに行が存在するかどうかに基づいて、異なる方法で処理します。

行がルックアップキャッシュに存在しない場合、統合サービスはキャッシュに行を挿入してターゲットテーブルに渡します。行がルックアップキャッシュ内に存在する場合、統合サービスはキャッシュまたはターゲットテーブル内の行を更新しません。

注: ソースデータのルックアップ条件カラムに NULL 値が含まれている場合、エラーしきい値を 1 に設定します。これにより、統合サービスがキャッシュに行を挿入し、データベースが Not Null 制約のために行を拒否した場合でも、確実にルックアップキャッシュとテーブルは同期化されます。

動的ルックアップキャッシュを持つセッションの設定

アップデートストラテジトランスフォーメーションと動的ルックアップキャッシュを使用してセッションを設定する場合は、以下のいずれかのアップデートストラテジテーブルオプションを設定する必要があります。

- [Insert] を選択。
- [更新として更新] を選択。
- [Delete] を非選択。

アップデートストラテジターゲットテーブルオプションにより、Integration Service は確実に「更新」と記された行を更新し、「挿入」と記された行を挿入します。

セッションプロパティの [プロパティ] タブの [全般オプション] 設定で、[ソース行の扱い] オプションを [データドリブン] に定義します。[データドリブン] を選択しない場合、Integration Service は [ソース行の扱い] オプションでユーザーが指定した行種類の行をすべてフラグ設定し、マッピング内のアップデートストラテジトランスフォーメーションを使って行をフラグ設定しません。Integration Service は、正しい行を挿入および更新しません。[更新として更新] を選択しない場合、Integration Service はターゲットテーブルで更新行としてフラグを設定された行を正しく更新しません。その結果、ルックアップキャッシュとターゲットテーブルが同期しなくなる可能性があります。

条件付きの動的キャッシュの更新

動的ルックアップキャッシュは、ブール式の結果に基づいて更新することができます。統合サービスは式が true のときにキャッシュを更新します。

例えば、ターゲットテーブルに製品番号、在庫数量、タイムスタンプの列があるとします。在庫数量は最新のソース値で更新する必要があります。ソースデータのタイムスタンプが動的キャッシュ内のタイムスタンプよりも大きい場合に、在庫数量を更新することができます。以下の式のようなルックアップトランスフォーメーションの式を作成します。

```
lookup_timestamp < input_timestamp
```

式にはルックアップポートや入力ポートを含めることができます。ビルトイン変数、マッピング変数、およびパラメータ変数にアクセスできます。ユーザー定義関数を含めたり、未接続のトランスフォーメーションを参照することもできます。

式は true、false、または NULL を返します。式の結果が NULL である場合、その式は false です。統合サービスはキャッシュを更新しません。式の結果が true に変更する必要がある場合は、式に NULL 値のチェックを追加します。式のデフォルト値は true です。

式は Transformation Developer で作成します。**【動的キャッシュの更新条件】**をセッションレベルでオーバーライドすることはできません。

セッション処理

条件付き動的キャッシュ更新を設定するときにデータが存在しない場合、統合サービスでは条件式が考慮されません。統合サービスでは「挿入でなければ更新」を有効にすると、データが存在しない場合に、キャッシュに行が挿入されます。データが存在する場合は、条件式が TRUE であればキャッシュが更新されます。「更新でなければ挿入」を有効にして、データがキャッシュ内に存在し、条件式が TRUE である場合は、統合サービスはキャッシュを更新します。キャッシュ内にデータが存在しない場合は、統合サービスはキャッシュに行を挿入します。

式が TRUE である場合、NewLookupRow の値は 1 で、統合サービスは行を更新します。式が FALSE または NULL の場合、NewLookupRow の値は 0 になります。統合サービスは行を更新しません。

動的キャッシュを同期させる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と、統合サービスはキャッシュに行を挿入すると同時にルックアップソースに行を挿入します。

条件付きの動的キャッシュルックアップの設定

条件式を作成するには、あらかじめルックアップトランスフォーメーションで条件付きの動的キャッシュルックアップを実行できるようにしておく必要があります。

条件付きの動的キャッシュルックアップを設定するには、次の手順を実行します。

1. Transformation Developer で動的なルックアップトランスフォーメーションを作成します。
2. 「プロパティ」タブをクリックします。
3. 「ルックアップキャッシュ」プロパティと「動的ルックアップキャッシュ」プロパティを有効にします。
4. 式エディタにアクセスするには、「動的キャッシュの更新条件」プロパティの「実行」ボタンをクリックします。
5. 式の条件を定義します。式に対して入力ポート、ルックアップポート、および関数を選択できます。
6. 「検証」をクリックして、式が有効であることを検査します。

式の検証を行わない場合には、式エディタを閉じた時点で、Designer がその検証を行います。式が有効でない場合、Designer は警告を表示します。

式の結果を使用した動的キャッシュの更新

動的ルックアップキャッシュの値は、式の結果で更新することができます。

例えば、製品テーブルのターゲットには注文数が含まれている数値列があります。ルックアップトランスフォーメーションは、製品の注文を受けるたびに動的キャッシュ `order_count` を次の式の結果で更新します。

```
order_count = order_count + 1
```

ルックアップトランスフォーメーションは `order_count` を返します。

式が NULL と評価した場合の統合サービスの処理方法を設定できます。

式の値が NULL

式のいずれかの値が null である場合、式は NULL を返します。ただし、式が NULL でない値を返すように設定できます。

式がルックアップポートを参照していても、ソースデータが新しい場合、ルックアップポートにはデフォルト値が格納されます。デフォルト値は NULL である場合があります。NULL 値をチェックするように `IsNull` 式を設定できます。

例えば、次の式は `lookup_column` が NULL であるかどうかをチェックします。

```
iif (isnull(lookup_column), input_port, user_expression)
```

そのカラムが NULL である場合は、`input_port` という値を返します。それ以外の場合は、式の値を返します。

セッション処理

統合サービスは、式の結果に基づいて動的ルックアップキャッシュに行を挿入および更新できます。式の結果は、ルックアップポートの値が NULL で式に含まれているかどうかに基づいて異なる可能性があります。

統合サービスでは「挿入でなければ更新」を有効にすると、データがキャッシュ内にない場合には、式の結果の行が挿入されます。キャッシュ内にデータが存在しない場合は、ルックアップポートの値が NULL になります。式がルックアップポートの値を参照している場合は、統合サービスは式のデフォルトのポートの値を置き換えます。「挿入でなければ更新」を有効にして、データがキャッシュ内に存在する場合は、統合サービスは式の結果によってキャッシュを更新します。

「更新でなければ挿入」を有効にして、データがキャッシュ内に存在する場合には、統合サービスは式の結果によってキャッシュを更新します。キャッシュ内にデータが存在しない場合は、統合サービスは式の結果を含む行を挿入します。式がルックアップポートの値を参照している場合は、統合サービスは式のデフォルトのポートの値を置き換えます。

動的キャッシュを同期させる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と、統合サービスはルックアップキャッシュに式の結果の行を挿入します。式の結果の行はルックアップソースに挿入されます。

動的キャッシュの更新のための式の設定

条件式を作成するには、あらかじめルックアップトランスフォーメーションで動的ルックアップを実行できるようにしておく必要があります。

動的キャッシュルックアップの式を設定するには、次の手順を実行します。

1. Transformation Developer で動的なルックアップトランスフォーメーションを作成します。
2. 「プロパティ」タブを開きます。
3. 「ルックアップキャッシュ」プロパティと「動的ルックアップキャッシュ」プロパティを有効にします。
4. 式を作成するには、更新するルックアップポートの「関連する式」リストをクリックします。

5. [関連する式] を選択します。

式エディタが表示されます。

6. 入力ポート、ルックアップポート、および関数を選択して、式を定義します。式の戻り値はルックアップポートのタイプと一致する必要があります。

7. [検証] をクリックして、式が有効であることを検査します。

式の検証を行わない場合には、式エディタを閉じた時点で、Designer がその検証を行います。式が有効でない場合、Designer は警告を表示します。

キャッシュとルックアップソースの同期

ルックアップトランスフォーメーションでは、ターゲットに渡した行を追跡するために、動的ルックアップキャッシュが保持されます。複数のセッションが同じターゲットを更新する場合は、各セッションのルックアップトランスフォーメーションを設定して、ターゲットではなく同じルックアップソースに動的ルックアップキャッシュを同期させることができます。

キャッシュがルックアップソースと同期する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と、ルックアップトランスフォーメーションはルックアップソース上でルックアップを実行します。ルックアップソース内にデータが存在しない場合、ルックアップトランスフォーメーションは動的ルックアップキャッシュを更新する前にルックアップソースに行を挿入します。

別のセッションで行が挿入された場合には、ルックアップソース内にデータが存在する可能性があります。ルックアップキャッシュをルックアップソースに同期させるために、Integration Service はルックアップソースから最新の値を取得します。ルックアップトランスフォーメーションはルックアップソースから取得した値を動的ルックアップキャッシュに挿入します。ルックアップソースはリレーショナルテーブルでなければなりません。

例えば、複数のセッションが同時に実行されているとします。各セッションは新しい製品名の製品番号を生成します。あるセッションが製品番号を生成した場合、他のセッションは同じ製品番号を使用してその製品を識別する必要があります。製品番号は1回生成され、ルックアップソースに挿入されます。その製品を含む行を別のセッションが処理する場合は、ルックアップソース内の製品番号を使用する必要があります。各セッションはルックアップソース上でルックアップを実行し、すでに生成されている製品番号を特定します。

Integration Service は挿入行に対して以下のタスクを実行します。

- Integration Service は動的ルックアップキャッシュ上でルックアップを実行します。動的ルックアップキャッシュ内にデータが存在しない場合、Integration Service はルックアップソース上でルックアップを実行します。
- データがルックアップソース内に存在する場合、Integration Service はルックアップソースからデータを取得します。Integration Service は、ルックアップソースのカラムを使用して、動的ルックアップキャッシュ内に行を挿入します。ソース行でキャッシュを更新することはありません。
- データがルックアップソース内に存在しない場合、Integration Service はデータをルックアップソースに挿入し、行をキャッシュに挿入します。

ルックアップソースには、ルックアップキャッシュと同じカラムがあります。カラムがルックアップトランスフォーメーションから射影されるか、カラムがルックアップ条件の一部でない限り、Integration Service がルックアップキャッシュにカラムを挿入することはありません。

NewLookupRow

動的キャッシュをルックアップソースと同期する場合、ルックアップトランスフォーメーションの動作は更新行に対しては変わりません。Integration Service は、更新行を受け取ると動的ルックアップキャッシュを更新

し、NewLookupRow の値を 2 に設定します。ルックアップソースは更新されません。更新行をアップデートストラテジトランスフォーメーションを通じてルーティングし、それらの行をターゲットに渡すことができます。

Integration Service は、行をルックアップソースに挿入するときに、NewLookupRow を 1 に設定します。Integration Service は動的ルックアップキャッシュを更新します。トランスフォーメーションが動的キャッシュを同期するように設定されていて、NewLookupRow が 1 の場合、挿入行をターゲットに渡す必要はありません。

ルックアップトランスフォーメーションで [挿入でなければ更新] プロパティを設定し、ソース行が挿入行の場合、Integration Service は行をキャッシュとルックアップソースに挿入するか、またはキャッシュを更新します。

ルックアップトランスフォーメーションで [更新でなければ挿入] プロパティを設定し、ソース行が更新行の場合、Integration Service はキャッシュを更新するか、または行をキャッシュとルックアップソースに挿入します。

注: 異なるセッションが同じテーブルの更新行を処理する場合、処理の順序はさまざまです。予期しない結果になる可能性もあります。

動的キャッシュ同期の設定

ルックアップトランスフォーメーションを作成する場合、動的キャッシュを挿入行のルックアップソースに同期するように設定することができます。

動的キャッシュ同期を設定する手順

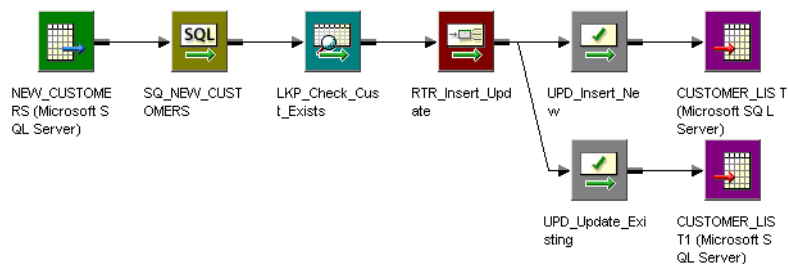
1. ルックアップトランスフォーメーションの [プロパティ] タブを開きます。
2. [動的ルックアップキャッシュ] を有効にします。
3. [動的キャッシュの同期] を有効にします。

動的ルックアップキャッシュの例

ターゲットで行の挿入および更新を実行する必要がある場合、動的ルックアップキャッシュを使用します。動的ルックアップキャッシュを使用すると、挿入または更新の目的でターゲットに渡したデータと同じデータをキャッシュに挿入または更新できます。

たとえば、顧客データが含まれているテーブルを更新する必要があるものとします。ソースデータには、ターゲットに挿入または更新する顧客データ行が含まれています。ターゲットとなる動的キャッシュを作成します。キャッシュ内の顧客をルックアップするようにルックアップトランスフォーメーションを設定します。

以下の図に、動的キャッシュがあるマッピングを示します。



ルックアップトランスフォーメーションは動的ルックアップキャッシュを使用します。セッションが開始すると、Integration Service は Customer_List テーブルからルックアップキャッシュを構築します。Customer_List テーブルは、マッピングのターゲットでもあります。Integration Service は、ルックアップキャッシュにない行を読み込むと、その行をキャッシュに挿入します。

ルックアップトランスフォーメーションは、ルータトランスフォーメーションにその行を返します。ルータトランスフォーメーションは、その行を UPD_Insert_New トランスフォーメーションまたは UPD_Update_Existing トランスフォーメーションに送ります。アップデートストラテジトランスフォーメーションは、この行を「挿入」または「更新」とマークし、ターゲットに渡します。

Customer_List テーブルは、セッションの実行に伴って変化します。Integration Service は、ルックアップキャッシュに新しい行を挿入し、既存の行を更新します。Integration Service は、ルックアップキャッシュと Customer_List テーブルを同期化します。

ターゲットのキーを生成するには、関連するポートのシーケンス ID を使います。シーケンス ID は、Integration Service がターゲットテーブルに挿入する新規行のプライマリキーを生成します。

動的ルックアップキャッシュを使用するとセッションパフォーマンスが向上します。データベースからキャッシュを構築するのは 1 回だけだからです。ターゲットテーブルのデータが変更されても、ルックアップキャッシュを使い続けることができます。

動的ルックアップキャッシュのルールとガイドライン

動的ルックアップキャッシュを使用する際、一定のルールとガイドラインが適用されます。

動的ルックアップキャッシュを使用するときは、次のガイドラインに従います。

- 動的ルックアップキャッシュを使用する場合は、**【複数の一致の検出時】** プロパティを **【エラーを報告】** に設定する必要があります。プロパティをリセットするには、動的ルックアップを静的ルックアップに変更し、プロパティを変更してから静的ルックアップを動的ルックアップに変更します。
- 同じターゲットロード順グループの動的ルックアップトランスフォーメーションと静的ルックアップトランスフォーメーション間で、キャッシュを共有することはできません。
- 動的ルックアップキャッシュは、リレーショナル、フラットファイル、またはソース修飾子トランスフォーメーションの各ルックアップに対して有効にできます。
- 動的ルックアップキャッシュは、リレーショナルルックアップまたはフラットファイルルックアップに対して有効にできます。
- ルックアップトランスフォーメーションは、接続されたトランスフォーメーションでなければなりません。
- 永続または非永続キャッシュのいずれかを使用できます。
- 動的キャッシュが永続でない場合、**【ルックアップソースからのキャッシュの再構築】** が有効になっていない場合でも、統合サービスは常にデータベースからキャッシュを再構築します。
- 動的キャッシュが永続でない場合、**【ルックアップソースからのキャッシュの再構築】** が有効になっていない場合でも、統合サービスは常にデータベースからキャッシュを再構築します。
- 動的キャッシュファイルとルックアップソーステーブルを同期させると、ルックアップトランスフォーメーションではルックアップソーステーブルと動的ルックアップキャッシュに行が挿入されます。ソース行が更新行である場合、ルックアップトランスフォーメーションは動的ルックアップキャッシュだけを更新します。
- 作成可能なのは、等価ルックアップ条件だけです。動的キャッシュでデータの範囲をルックアップすることはできません。

- ルックアップ条件にない各ルックアップポートを入力ポート、シーケンス ID、または関連する式に関連付ける必要があります。
- NewLookupRow 値が 1 または 2 の場合、ルータトランスフォーメーションを使用し、行をキャッシュに格納されたターゲットに渡します。
- NewLookupRow 値が 0 の場合、ルータトランスフォーメーションを使用し、行を削除します。あるいは、行を別のターゲットに出力できます。
- 統合サービスがルックアップキャッシュに書き込む値と同じ値がターゲットに出力されることを確認します。更新時に新しい値を出力するように選択するときは、ターゲットテーブルに入力/出力ポートの代わりにルックアップ/出力ポートを接続するだけです。更新時に古い値を出力すると選択した場合は、ルックアップトランスフォーメーションの後で、ルータトランスフォーメーションの前に式トランスフォーメーションを追加します。ターゲットテーブルの各ポートに式トランスフォーメーションの出力ポートを追加し、NULL 入力値をターゲットに出力しないように式を作成します。
- ルックアップ SQL オーバーライドを使用する場合は、カラムを適切なルックアップ用ターゲットに正しくマッピングするようにします。
- ルックアップ SQL オーバーライドに WHERE 句を追加する場合は、ルックアップトランスフォーメーションの前にフィルタトランスフォーメーションを使用します。これにより、統合サービスは WHERE 句に一致する動的キャッシュとターゲットテーブルにのみ行を挿入します。
- 動的キャッシュを使用するように再利用可能なルックアップトランスフォーメーションを設定する場合は、条件を編集したり、マッピングの動的ルックアップキャッシュプロパティを無効にすることはできません。
- アップデートストラテジトランスフォーメーションをルックアップトランスフォーメーションの後に使用して、ターゲットへの挿入または更新フラグを行に設定します。
- ルックアップトランスフォーメーションの [更新でなければ挿入] プロパティを使用したい場合は、アップデートストラテジトランスフォーメーションはルックアップトランスフォーメーションの前に使用して、一部またはすべての行を「更新」と定義します。
- セッションプロパティで行種別を [Data Driven] と設定します。
- セッションプロパティでターゲットテーブルオプションとして [挿入] と [更新として更新] を選択します。

第 20 章

ノーマライザトランスフォーメーション

この章では、以下の項目について説明します。

- [ノーマライザトランスフォーメーションの概要, 355 ページ](#)
- [ノーマライザトランスフォーメーションのコンポーネント, 356 ページ](#)
- [ノーマライザトランスフォーメーションの生成キー, 359 ページ](#)
- [VSAM ノーマライザトランスフォーメーション, 360 ページ](#)
- [パイプラインノーマライザトランスフォーメーション, 364 ページ](#)
- [マッピングにおけるノーマライザトランスフォーメーションの使用, 368 ページ](#)
- [ノーマライザトランスフォーメーションのトラブルシューティング, 372 ページ](#)

ノーマライザトランスフォーメーションの概要

ノーマライザトランスフォーメーションは、複数出現カラムが含まれている行を受け取り、複数出現データの出現ごとに行を返します。このトランスフォーメーションによって、各ソース行内の複数出現カラムまたは複数出現カラムグループが処理されます。ノーマライザトランスフォーメーションはアクティブなトランスフォーメーションです。

ノーマライザトランスフォーメーションでは、COBOL ソース、リレーショナルテーブル、または他のソースからの複数出現カラムが解析され、REDEFINES 句を含む COBOL ソースからの複数のレコードタイプを処理できます。

たとえば、リレーショナルテーブルに 4 四半期分の店舗別売上を格納することもできます。売上オカレンスごとに行を作成する必要があります。四半期ごとに別個の行を返すように、ノーマライザトランスフォーメーションを設定することができます。

4 四半期分の店舗別売上を格納するソース行は、次のとおりです。

```
Store1 100 300 500 700
Store2 250 450 650 850
```

ノーマライザによって店舗および売上ごとに行が返され、四半期番号を識別するインデックスも返されます。

```
Store1 100 1
Store1 300 2
Store1 500 3
Store1 700 4
Store2 250 1
Store2 450 2
```

Store2 650 3
Store2 850 4

ノーマライザトランスフォーメーションはソース行ごとにキーを生成します。生成キーシーケンス番号は、Integration Service でのソース行処理時に毎回増分されます。ソース行に複数出現カラムまたは複数出現カラムグループが含まれている場合、ノーマライザトランスフォーメーションはオカレンスごとに行を返します。各行に格納されている生成キー値はそれぞれ同じ値です。

また、ノーマライザによってソース行から複数の行が返されるときは、単独出現ソースカラムに対応した重複データが返されます。たとえば、Store1 および Store2 は売上インスタンスごとに繰り返します。

次のような VSAM ノーマライザトランスフォーメーションまたはパイプラインノーマライザトランスフォーメーションを作成することができます。

- **VSAM ノーマライザトランスフォーメーション。** COBOL ソース用ソース修飾子トランスフォーメーションである再利用不能トランスフォーメーション。Mapping Designer では、マッピング内の COBOL ソースから VSAM ノーマライザカラムが作成されます。カラム属性は読み取り専用です。VSAM ノーマライザは、1 つの入力ポートを介して複数出現ソースカラムを受け取ります。
- **パイプラインノーマライザトランスフォーメーション。** リレーショナルテーブルまたはフラットファイルの複数出現データを処理するトランスフォーメーションです。Transformation Developer または Mapping Designer で、カラムを手動で作成して編集します。パイプラインノーマライザトランスフォーメーションは、各ソースカラムオカレンス用の入力ポートを 1 つずつ備えた複数出現カラムを表します。

ノーマライザトランスフォーメーションのコンポーネント

ノーマライザトランスフォーメーションには、以下のタブが含まれています。

- **[トランスフォーメーション]。** トランスフォーメーションの名前および説明を入力します。ノーマライザトランスフォーメーションの命名規則は、「NRM_トランスフォーメーション名」です。また、パイプラインノーマライザトランスフォーメーションを再利用可能にすることもできます。
- **ポート。** トランスフォーメーションのポートおよび属性が表示されます。
- **プロパティ。** セッションログファイルに記録されたトランザクション詳細の量を決定するトレースレベルを設定します。発行されたキーシーケンスの値を次のセッションでリセットまたは再開する場合に選択してください。
- **ノーマライザ。** ソースデータの構造を定義します。[ノーマライザ] タブでは、ソースデータがカラムおよびカラムグループとして定義されます。
- **メタデータエクステンション。** エクステンション名、データタイプ、精度、および値を設定します。再利用不可能なメタデータエクステンションも、作成することができます。

[ポート] タブ

ノーマライザトランスフォーメーションを作成するときは、[ノーマライザ] タブでカラムを設定します。ポートは、Designer で作成されます。ノーマライザのポートおよび属性は、[ポート] タブで表示することができます。

パイプラインノーマライザトランスフォーメーションおよび VSAM ノーマライザトランスフォーメーションでは、複数出現ソースカラムがそれぞれ異なる形式で表示されます。VSAM ノーマライザトランスフォーメーションは、複数出現カラム用の出力ポートを 1 つ備えています。には、パイプラインノーマライザトランスフォーメーションには、複数出現カラム用に複数の入力ポートが用意されています。

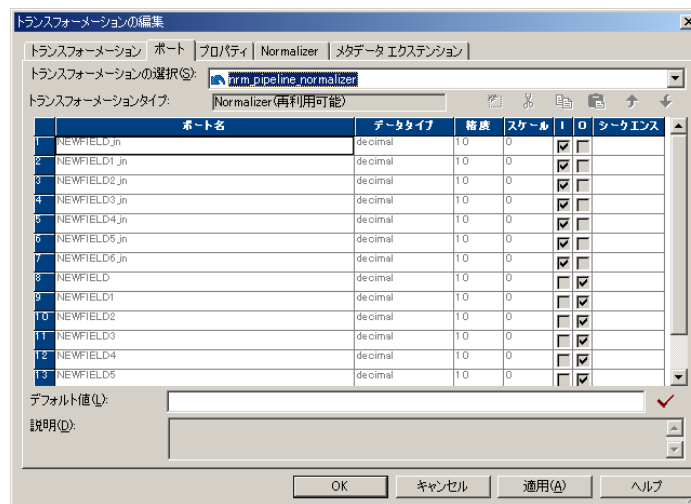
ノーマライザトランスフォーメーションは、各単独出現入力ポート用の出力ポートを1つずつ備えています。ソースカラムが複数出現カラムの場合、パイプラインノーマライザトランスフォーメーションおよび VSAM ノーマライザトランスフォーメーションは、カラム用の出力ポートを1つ持ちます。トランスフォーメーションによって、ソースカラムのオカレンスごとに1行が返されます。

ノーマライザトランスフォーメーションには、複数出現カラムごとに生成カラム ID（GCID）ポートがあります。生成カラム ID は、複数出現データのインスタンス用のインデックスです。たとえば、ソースレコード内にカラムが4回出現する場合、ノーマライザは行内に出現する複数出現データがどのインスタンスのものかに基づいて、1、2、3、または4の値を生成カラム ID の中に返します。

ノーマライザの生成カラム ID の命名規則は、「GCID_<発生するフィールド名>」です。

ノーマライザトランスフォーメーションは、生成キーポートを少なくとも1つは備えています。生成キーシーケンス番号は、Integration Service でのソース行処理時に毎回増分されます。

以下の図は、ノーマライザトランスフォーメーションの「ポート」タブを示しています。



この例では、Sales_By_Quarter はソース内で複数出現します。ノーマライザトランスフォーメーションは、Sales_By_Quarter 用の出力ポートを1つ備えています。ソース行ごとに4行が返されます。生成されたキーの開始値は1です。

パイプラインノーマライザトランスフォーメーション上のポートは、「ノーマライザ」タブ上のカラムを編集して変更することができます。VSAM ノーマライザトランスフォーメーションを変更するには、COBOL ソースを変更し、トランスフォーメーションを再作成する必要があります。

「プロパティ」タブ

「プロパティ」タブでは、ノーマライザトランスフォーメーションの全般的なプロパティが設定されます。

ノーマライザトランスフォーメーション次のプロパティは、ユーザー設定可能です。

プロパティ	必須/ オプション	説明
リセット	必須	セッションの終了時に、各生成キーの値シーケンスがセッション前の値にリセットされます。
リスタート	必須	生成キーのシーケンスは、1 から開始します。セッション実行時に毎回、キーシーケンス値が 1 から開始し、[ポート] タブ上のシーケンス値がオーバーライドされます。
トレースレベル	必須	このトランスフォーメーションを含むセッションを実行したときにセッションログに記録される情報の詳細度を設定します。

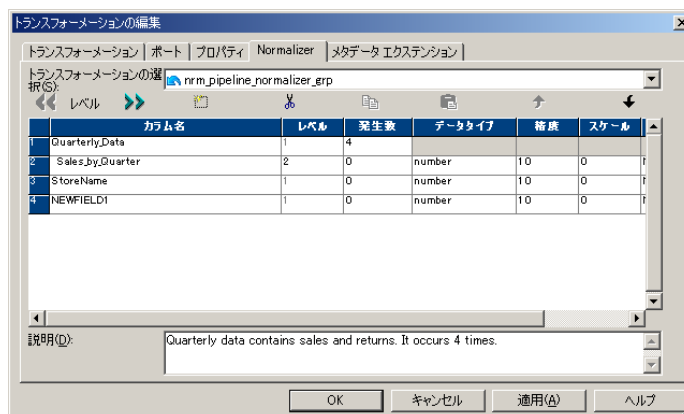
[ノーマライザ] タブ

[ノーマライザ] タブで、ソースデータの構造を定義できます。[ノーマライザ] タブでは、ソースデータがカラムおよびカラムグループとして定義されます。COBOL ソース内のレコードや、ソース内の複数出現フィールドのグループは、カラムのグループで定義できます。

データ内のカラムのグループは、カラムのレベル番号で識別されます。カラムのレベル番号によって、データ階層が定義されます。1 グループ内ではすべてのカラムに同一のレベル番号が付けられ、グループレベルのカラムの下部に順次に表示されます。グループレベルのカラムには下位レベル番号が付けられ、データは入りません。

以下の図に、パイプラインノーマライザトランスフォーメーションの [ノーマライザ] タブを示します。

図 3. [ノーマライザ] タブ



Quarterly_Data はグループレベルのカラムです。レベル 1 です。Quarterly_Data グループは、各行内に 4 回出現します。Sales_by_Quarter および Returns_by_Quarter はレベル 2 で、このグループに属しています。

各カラムには発生属性があります。OCCURS 属性によって、ソース行中に複数回出現するカラム、またはカラムのグループが識別されます。

パイプラインノーマライザトランスフォーメーションを作成すると、そのカラムの編集が可能になります。VSAM ノーマライザトランスフォーメーションを作成した場合、[ノーマライザ] タブは読み取り専用です。

以下の表では、VSAM ノーマライザおよびパイプラインノーマライザに共通する [ノーマライザ] タブ属性について説明します。

属性	説明
カラム名	ソースカラムの名前。
レベル	カラムがグループ化されます。同一グループ内では、下位レベル番号の付いたカラム下にカラムが発生します。各カラムが同一レベルの場合、トランスフォーメーションはカラムグループを含みません。
OCCURS	ソース行内のカラムまたはカラムグループのインスタンスの数。
データ型	トランスフォーメーションのカラムのデータタイプは、String、Nstring、または Number です。
精度	精度。カラムの長さ。
スケール	数値カラムの小数点以下の桁数。

VSAM ノーマライザトランスフォーメーションの [ノーマライザ] タブに含まれる属性には、パイプラインノーマライザトランスフォーメーションと同じものもありますが、COBOL ソース定義に固有の属性も含まれています。

ノーマライザトランスフォーメーションの生成キー

ノーマライザトランスフォーメーションは、出力行に生成キーカラムを少なくとも 1 つ返します。生成キーシーケンス番号は、Integration Service でのソース行処理時に毎回増分されます。Integration Service では、ノーマライザトランスフォーメーションの [ポート] タブ内の生成キー値から初期キー値が決定されます。ノーマライザトランスフォーメーションを作成すると、デフォルトで生成キー値が 1 になります。ノーマライザの生成キーの命名規則は、「GK_<再定義されるフィールド名>」です。

関連項目：

- [「キー値の生成」 \(ページ 370\)](#)

生成されたキー値の格納

現在の生成キー値は、ノーマライザトランスフォーメーションの [ポート] タブで表示することができます。Integration Service は各セッションの終了時に、ノーマライザトランスフォーメーション内の生成キー値を、セッションで最後に生成された値に 1 を加算した値に更新します。生成キーの最大値は 9,223,372,036,854,775,807 です。リポジトリ内にノーマライザトランスフォーメーションの複数インスタンスがある場合、Integration Service ではセッション実行時にすべてのバージョンにおける生成キー値が更新されます。

注: 現在の生成キー値をノーマライザトランスフォーメーションの [ポート] タブで変更することはできません。

生成キー値の変更

生成キー値を変更するには、以下の方法があります。

- **生成キーシーケンスのリセット。** ノーマライゼイトランスフォーメーションの「プロパティ」タブで、生成キーシーケンス値をリセットします。生成キーシーケンスをリセットすると、生成キーの開始値がセッション前の値にリセットされます。セッション実行のたびに同じ生成キー値を作成する場合は、生成キーシーケンスをリセットしてください。
- **生成キーシーケンスのリスタート。** ノーマライゼイトランスフォーメーションの「プロパティ」タブで、生成キーシーケンス値をリスタートします。生成キーシーケンスをリスタートすると、Integration Service では次のセッション実行時に生成キーシーケンスが 1 から開始されます。生成キー開始値は、生成キーシーケンスをリスタートしてもセッションが実行されるまでは、ノーマライゼイトランスフォーメーション内では変更されません。セッションを実行すると、「ポート」タブ上のシーケンス番号の値が Integration Service によってオーバライドされます。

生成キーシーケンスをリセットまたはリスタートした場合、生成キーシーケンスにリセットまたはリスタートの影響が及ぶのは、次のセッション実行時です。現在の生成キーシーケンス値は、ノーマライゼイトランスフォーメーション内では変更しないでください。生成キーシーケンスをリセットまたはリスタートした場合、オプションは無効にしない限り、すべてのセッションで有効になっています。

VSAM ノーマライゼイトランスフォーメーション

VSAM ノーマライゼイトランスフォーメーションは、COBOL ソース定義のソース修飾子です。COBOL ソースは、複数出現データおよび数種のレコードを同じファイル内に格納できるフラットファイルです。

VSAM (Virtual Storage Access Method) は、IBM メインフレームオペレーティングシステムのファイルアクセス方式です。VSAM ファイルでは、インデックス付きまたはシーケンシャルのフラットファイルでレコードが編成されます。ただし、VSAM ノーマライゼイトランスフォーメーションは、COBOL ソース定義で定義されるどのようなフラットソースファイルにも使用することができます。

COBOL ソース定義には、複数出現カラムを定義する OCCURS 文を含めることができます。また、COBOL のソース定義に REDEFINES 文を含めることにより、1 ファイル内に複数のレコードを定義することができます。

売上レコードを定義する COBOL コピーブックは、次のとおりです。

```
01 SALES_RECORD.  
  03 HDR_DATA.  
    05 HDR_REC_TYPE          PIC X.  
    05 HDR_STORE             PIC X(02).  
  03 STORE_DATA.  
    05 STORE_NAME            PIC X(30).  
    05 STORE_ADDR1           PIC X(30).  
    05 STORE_CITY            PIC X(30).  
  03 DETAIL_DATA REDEFINES STORE_DATA.  
    05 DETAIL_ITEM            PIC 9(9).  
    05 DETAIL_DESC            PIC X(30).  
    05 DETAIL_PRICE           PIC 9(4)V99.  
    05 DETAIL_QTY             PIC 9(5).  
    05 SUPPLIER_INFO OCCURS 4 TIMES.  
      10 SUPPLIER_CODE        PIC XX.  
      10 SUPPLIER_NAME        PIC X(8).
```

売上ファイルに格納できる売上レコードは、2 種類あります。店舗は Store_Data で定義され、店舗で販売されている商品は Detail_Data で定義されます。REDEFINES 句により、レコード内に Store_Data フィールドでなく Detail_Data フィールドが出現するよう指示されます。

各売上レコードの先頭 3 文字は、ヘッダです。ヘッダには、レコードタイプおよび店舗 ID が含まれています。残りのレコードの中に店舗情報または商品情報のどちらが格納されているかは、Hdr_Rec_Type の値で定義さ

れます。たとえば、Hdr_Rec_Type が"S"なら、レコードに格納されているのは店舗データです。
Hdr_Rec_Type が"D"なら、レコードに格納されているのは詳細データです。

詳細データがレコードに格納されているときは、Supplier_Info フィールドが含まれています。各 Detail_Data
レコード内の 4 サプライヤは、OCCURS 句で定義されます。

以下の図は、COBOL コピーブックから作成できる Sales_File COBOL ソース定義を示しています。

キ	名前	レベ	発	データ型	長
	SALES_REC	1	0	number	10
	HDR_DATA	1	0	number	0
	HDR_STORE	2	0	number	10
	HDR_REC_TY...	2	0	number	0
	STORE_DA...	3	0	number	10
	STORE_NA...	3	0	number	10
	NEWFIELD	3	0	number	10
	NEWFIELD1	3	0	number	10
	NEWFIELD2	3	0	number	10
	NEWFIELD3	3	0	number	10

Sales_Rec カラム、Hdr_Data カラム、Store_Data カラム、Detail_Data カラム、および Supplier_Info カラ
ムは、下位レベルデータのグループを識別するグループレベルのカラムです。グループレベルのカラムはデー
タを含まないため、長さがゼロです。これらのカラムはいずれも、ソース定義内の出力ポートには該当しませ
ん。

Supplier_Info グループには、Supplier_Code カラムおよび Supplier_Name カラムが含まれています。
Supplier_Info グループは、Detail_Data レコード内に 4 回出現します。

Mapping Designer で COBOL ソース定義から VSAM ノーマライゼーションを作成すると、
その COBOL ソース定義に応じた入出力ポートがノーマライゼーション内に作成されます。
ノーマライゼーションには、生成キーの出力ポートが少なくとも 1 つは含まれています。
COBOL ソースに複数出現カラムが含まれている場合、ノーマライゼーションは生成カラム
ID の出力ポートを持ちます。

以下の図は、Mapping Designer でソース定義から作成されたノーマライゼーションポート
を示しています。

キ	名前	レベ	0...	データ型
	WIC_HIST	1	0	
	HST_CSTLIST	3	0	
	HST_CUST	5	0	nstring
	HST_LIST	5	0	nstring
	HST_MTH	3	24	
	HST_A...	5	0	number
	HST_ACCR	5	0	number
	HST_A...	5	0	number
	HST_ADJ	5	0	number
	HST_REV	5	0	number
	HST_ACT	5	0	number
	HST_AMT	5	23	number
	HS...	6	0	number
	HST_ACCR...	3	0	number
	HST_ADJ...	3	0	number
	FILLER	3	0	string

名前	データ型
HST_CUST	nstring
HST_LIST	nstring
HST_ACCR_REM	decimal
HST_ACCR	decimal
HST_ADJ_REM	decimal
HST_ADJ	decimal
HST_REV	decimal
HST_ACT	decimal
HST_WD_AMT	decimal
HST_ACCR_DAYS	decimal
HST_ADJ_DAYS	decimal
FILLER	string
GK_FILE_ONE	integer
GK_HST_MTH	integer
GCID_HST_MTH	integer
GCID_HST_AMT	integer

Supplier_Info グループのカラムは、各 COBOL ソース行の中に 4 回出現します。

COBOL ソース行には、次のようなデータが含まれています。

Item1 ItemDesc 100 25 A Supplier1 B Supplier2 C Supplier3 D Supplier4

ノーマライゼーションは、Supplier_Code カラムおよび Supplier_Name カラムのオカレン
スごとに行を返します。各出力行に含まれる品目、説明、価格、および数量はそれぞれ同じ値です。

ノーマライザが COBOL ソース行から返す詳細データ行は、次のとおりです。

Item1 ItemDesc 100 25 A Supplier1 1 1
Item1 ItemDesc 100 25 B Supplier2 1 2

```
Item1 ItemDesc 100 25 C Supplier3 1 3
Item1 ItemDesc 100 25 D Supplier4 1 4
```

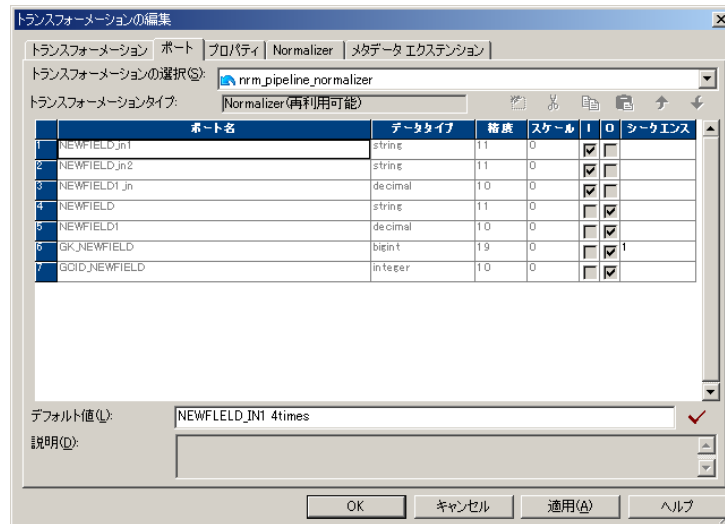
各出力行には、生成キーおよびカラム ID が含まれます。生成されたキー値は、新しいソース行の処理時に更新されます。詳細データ行の中の生成済みキー値は 1 です。

Supplier_Info カラムのオカレンス番号は、カラム ID で定義されます。カラム ID は、Supplier_Info カラムのオカレンス番号ごとに更新されます。詳細データ行の中のカラム ID 値は、1、2、3、4 です。

VSAM ノーマライザの [ポート] タブ

VSAM ノーマライザの [ポート] タブには、トランスフォーメーションの入力ポートおよび出力ポートが表示されます。そのトランスフォーメーションには、各 COBOL ソースカラム用に 1 つずつ、複数出現カラム用に 1 つの入出力ポートがあります。グループレベルカラム用の入出力ポートは、トランスフォーメーションに備わっていません。

以下の図は、VSAM ノーマライザの [ポート] タブを示しています。



この例では、Supplier_Code および Supplier_Name は、COBOL ソース内に 4 回出現します。[ポート] タブには、Supplier_Code ポートおよび Supplier_Name ポートが 1 つずつ表示されます。生成されるキーの開始値は 457 および 1087 です。

VSAM ノーマライザのタブ

Mapping Designer で VSAM ノーマライザトランスフォーメーションを作成すると、COBOL ソースからカラムが作成されます。[ノーマライザ] タブに表示される情報は、COBOL ソース定義の情報と同じです。VSAM ノーマライザのタブでカラムを編集することはできません。

以下の表では、VSAM ノーマライザのタブに表示される属性について説明します。

属性	説明
POffs	物理オフセット。ファイル内でのフィールドの位置。ファイル内の先頭バイトはゼロです。
PLen	物理長。フィールド内のバイトの数。

属性	説明
カラム名	ソースフィールドの名前。
レベル	カラムグループ階層が指定されます。レベル番号が大きいほど、階層構造内においてデータが下位に位置します。同一グループ内では、下位レベル番号の付いたカラム下にカラムが発生します。各カラムが同一レベルの場合、トランスフォーメーションはカラムグループを含みません。
OCCURS	ソース行内のカラムまたはカラムグループのインスタンスの数。
データ型	トランスフォーメーションのデータタイプは、String、Nstring、または Number です。
精度	精度。カラムの長さ。
スケール	数値カラムの小数点以下の桁数。
PICTURE	ソース内でのデータの格納形式または表示形式。数値フィールドと暗黙の小数点以下 2 桁は、Picture 99V99 と定義します。Picture X(10)は、10 桁の文字を示します。
使用方法	COBOL データの格納形式（例えば、COMP、BINARY、COMP-3）。使用方法が DISPLAY の場合、ソースデータを表示したときの書式設定を Picture 句で定義します。
Key Type	VSAM ファイルのフィールドに適用する、キー制約のタイプ。次のいずれかのキータイプを選択します。 <ul style="list-style-type: none"> - キーではありません。フィールドは VSAM ファイルのインデックスではありません。 - プライマリキー。フィールドは VSAM ファイルのプライマリインデックスです。フィールドには一意の値が含まれます。 - 代替キー。フィールドは VSAM ファイルのセカンダリインデックスです。フィールドには一意の値が含まれます。 - プライマリ重複キー。フィールドは VSAM ファイルのプライマリインデックスです。フィールドには、重複する値を含めることができます。 - 代替重複キー。フィールドは VSAM ファイルのセカンダリインデックスです。フィールドには、重複する値を含めることができます。
S（符号付き）	数値が符号付きであるかどうかを示します。
T（末尾の符号）	フィールドの最終桁に符号（+または-）が存在することを示します。有効にされていない場合、符号はフィールド内の先頭文字として表示されます。
I（含まれる符号）	フィールドに表示される値に符号を含めるかどうかを示します。
R（実数小数点）	小数点がピリオド（.）であるかそれとも数値フィールド内の V 文字であるかを示します。
Redefines	そのカラムによって別のカラムが再定義されることを示します。
ビジネス名	カラムに付けられた説明的な名前。

VSAM ノーマライザトランスフォーメーションの作成手順

Mapping Designer で VSAM ノーマライザトランスフォーメーションを作成するときに、COBOL ソースをマッピングにドラッグすると、ソースからトランスフォーメーションカラムが作成されます。ノーマライザトランスフォーメーションは、マッピング内の COBOL ソース用ソース修飾子です。

COBOL ソースをマッピングに追加すると、Mapping Designer はノーマライザトランスフォーメーションを作成して設定します。COBOL ソース内のネストされたレコードおよび複数出現フィールドは、Mapping Designer に識別されます。ソースカラムからノーマライザトランスフォーメーション内にカラムおよびポートが作成されます。

VSAM ノーマライザトランスフォーメーションを作成するには：

1. Mapping Designer で、新しいマッピングを作成するか、既存のマッピングを開きます。
2. COBOL ソース定義をマッピング内にドラッグします。

Designer はノーマライザトランスフォーメーションを追加し、COBOL ソース定義に接続します。ソース修飾子をデフォルトで作成するようにオプションが有効にされていない場合、[ノーマライザトランスフォーメーションの作成] ダイアログボックスが表示されます。

3. [ノーマライザトランスフォーメーションの作成] ダイアログボックスが表示されたら、以下のオプションのいずれかを選択することができます。

- **VSAM ソース。** マッピング内の COBOL ソース定義からトランスフォーメーションを作成します。
- **パイプライン。** トランスフォーメーションは作成されますが、COBOL ソースからカラムは定義されません。カラムは [ノーマライザ] タブで定義してください。マッピング内の別のトランスフォーメーションからの複数出現データを処理したい場合、このオプションを選択してください。

VSAM ノーマライザトランスフォーメーションを作成するには、VSAM ノーマライザトランスフォーメーションを選択します。ダイアログボックスに、マッピング内の COBOL ソース定義の名前が表示されます。COBOL ソース定義を選択して、[OK] をクリックします。

4. ノーマライザトランスフォーメーションを開きます。
5. [ポート] タブを選択し、ノーマライザトランスフォーメーション内のポートを表示します。

Designer ではデフォルトで、COBOL ソース定義からポートが作成されます。

6. [ノーマライザ] タブをクリックして、ソースカラムの編成を確認します。

[ノーマライザ] タブには、COBOL ソースの [カラム] タブと同じ情報が入ります。ただし、ノーマライザトランスフォーメーション内のカラム属性を修正することはできません。カラム属性を変更するには、COBOL コピーブックを変更し、COBOL ソースをインポートして、ノーマライザトランスフォーメーションを再作成します。

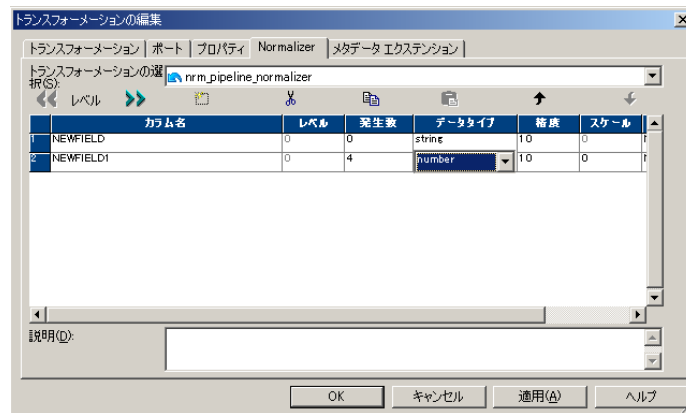
7. [プロパティ] タブを選択し、トレースレベルを設定します。

また、次のセッション開始時に生成キーシーケンス番号がリセットされるように、トランスフォーメーションを設定することもできます。

パイプラインノーマライザトランスフォーメーション

Transformation Developer でノーマライザトランスフォーメーションを作成する場合、マッピングパラメータまたはマッピング変数を使用します。パイプラインノーマライザトランスフォーメーションを作成するときは、そのトランスフォーメーションが別のタイプのトランスフォーメーション（たとえば、ソース修飾子トランスフォーメーション）から受け取るデータに応じたカラムを定義してください。定義されたカラムを基に、Designer で入力および出力のノーマライザトランスフォーメーションポートが作成されます。

以下の図は、各リレーショナルソース行内の売上カラム 4 つを受け取るトランスフォーメーションのためのノーマライザトランスフォーメーションカラムを示しています。



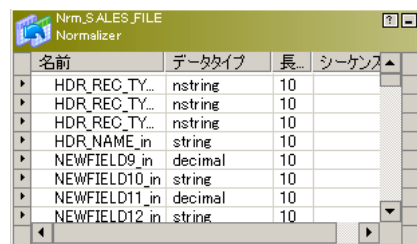
各ソース行には StoreName カラムと 4 つの Sales_By_Quarter インスタンスが含まれています。

ソース行に含まれるデータは、次のとおりです。

```
Dellmark 100 450 650 780
Tonys    666 333 444 555
```

パイプラインノーマライザトランスフォーメーションは、複数出現カラムの各インスタンス用の入力ポートを備えています。

以下の図は、Designer でノーマライザトランスフォーメーション内のカラムから作成されるポートを示しています。



ノーマライザトランスフォーメーションは、複数出現カラムのインスタンスごとに 1 行を返します。

```
Dellmark 100 1 1
Dellmark 450 1 2
Dellmark 650 1 3
Dellmark 780 1 4
Tonys    666 2 1
Tonys    333 2 2
Tonys    444 2 3
Tonys    555 2 4
```

生成キーシーケンス番号は、Integration Service でのソース行処理時に毎回増分されます。生成キーによって、各四半期の売上が同じ店舗にリンクされます。この例では、Dellmark 行の生成キーは 1 です。Tonys 店舗用の生成キーは 2 です。

トランスフォーメーションは、複数出現フィールドのインスタンスごとに生成カラム ID (GCID) を返します。この例の GCID_Sales_by_Quarter 値は常に 1、2、3、または 4 です。

パイプラインノーマライザの「ポート」タブ

パイプラインノーマライザの「ポート」タブには、トランスフォーメーション用の入力ポートおよび出力ポートが表示されます。トランスフォーメーション内で定義された単独出現カラム用の入出力ポートが1つ、複数出現カラムの各オカレンス用のポートが1つずつあります。グループレベルカラム用の入出力ポートは、トランスフォーメーションに備わっていません。

以下の図は、パイプラインノーマライザトランスフォーメーションの「ポート」タブを示しています。



Designer では入力ポートが複数出現カラムのオカレンスごとに作成されます。

パイプラインノーマライザトランスフォーメーション内のポートを変更するには、「ノーマライザ」タブでカラムを修正します。Designer でカラムオカレンスを追加すると、入力ポートが追加されます。Designer ではポートが最下位レベルのカラム用に作成されますが、グループレベルのカラム用には作成されません。

パイプラインノーマライザのタブ

パイプラインノーマライザトランスフォーメーションを作成する際には、「ノーマライザ」タブでカラムを定義します。「ノーマライザ」タブで入力されたカラムに基づいて、入力ポートおよび出力ポートが作成されます。

以下の表では、パイプラインノーマライザのタブ属性について説明します。

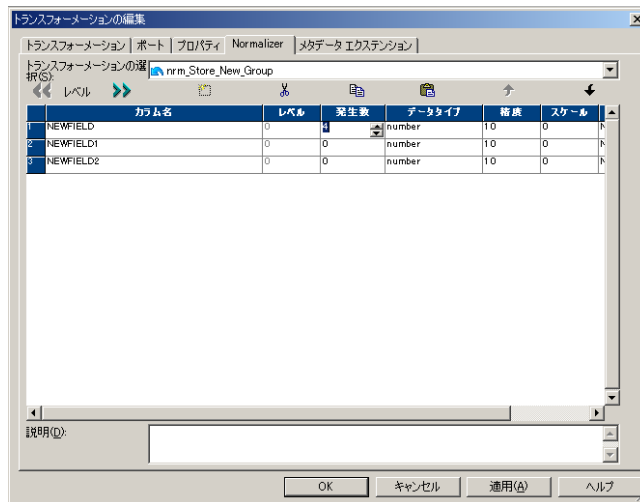
属性	説明
カラム名	カラムの名前。
レベル	カラムのグループが識別されます。同じグループ内のカラムには、同じレベル番号が付けられます。デフォルトはゼロです。各カラムが同一レベルの場合、トランスフォーメーションはカラムグループを含みません。
OCCURS	ソース行内のカラムまたはカラムグループのインスタンスの数。
データタイプ	カラムのデータタイプは、String、Nstring、または Number です。
精度	精度。カラムの長さ。
スケール	数値における小数点以下の桁数。

[ノーマライザ] タブのカラムグループ

ソース行に繰り返しカラムのグループが含まれている場合、[ノーマライザ] タブでカラムグループを定義することができます。ノーマライザトランスフォーメーションでは、カラムのオカレンス単位でなくカラムグループのオカレンス単位に行が返されます。

カラムの階層は、[ノーマライザ] タブ上のレベル番号で識別されます。カラムのグループは、グループレベルカラムで識別されます。グループレベルのカラムには、グループ内のカラムよりも下位のレベル番号が付けられます。同じグループ内ではカラムに同じレベル番号が付けられ、[ノーマライザ] タブ上のグループレベルカラムより下にシーケンシャルに表示されます。

以下の図に、[ノーマライザ] タブ内の複数出現カラムのグループを示します。



この例では、[NEWRECORD] カラムにはデータは含まれていません。そのカラムは、レベル 1 のグループカラムであるためです。そのグループは、各ソース行内に 4 回出現します。Store_Number および Store_Name は、レベル 2 のカラムであり、NEWRECORD グループに属しています。

パイプラインノーマライザトランスフォーメーションの作成手順

パイプラインノーマライザトランスフォーメーションを作成するには、[ノーマライザ] タブでカラムを定義します。

ノーマライザトランスフォーメーションは、Transformation Developer または Mapping Designer で作成できます。

ノーマライザトランスフォーメーションを作成するには：

1. Transformation Developer または Mapping Designer で、[トランスフォーメーション] - [作成] をクリックします。ノーマライザトランスフォーメーションを選択します。ノーマライザトランスフォーメーションの名前を入力します。

ノーマライザトランスフォーメーションの命名規則は「NRM_トランスフォーメーション名」です。

2. [作成] をクリックし、[完了] をクリックします。
3. ノーマライザトランスフォーメーションを開き、[ノーマライザ] タブをクリックします。
4. [追加] をクリックし、新規にカラムを追加します。

Designer で、デフォルト属性を備えたカラムが新規に作成されます。名前、データタイプ、精度、および位取りは、変更することができます。

5. 複数出現カラムを作成するには、[Occurs (出現回数)] カラムに出現回数を入力します。

6. 複数出現カラムグループを作成するには、[ノーマライザ] タブで少なくとも 1 つはカラムを入力します。カラムを選択します。[レベル] をクリックします。
Designer では、選択されたカラムより上に NEWRECORD グループレベルカラムが追加されます。NEWRECORD はレベル 1 になり、選択されたカラムはレベル 2 になります。NEWRECORD カラムは、名前の変更が可能です。
デフォルトでは、すべてのカラムが同じレベルです。グループ化されたカラムどうしは、レベルで定義されます。
7. 同じグループに他のカラムを追加する場合は、カラムのレベルを変更します。カラムを選択し、[レベル] をクリックして、上位のカラムと同じレベルに変更します。
同じグループ内のカラムは、[ノーマライザ] タブ内にシーケンシャルに出現しなければなりません。
8. グループレベルにあるオカレンスを、複数出現カラムグループになるように変更します。
9. [適用] をクリックし、カラムを保存して、入出力ポートを作成します。
Designer で、ノーマライザトランスフォーメーションの入力ポートおよび出力ポートが作成されます。また、生成キーカラムが作成されると共に、複数出現カラムまたはカラムグループごとのカラム ID も作成されます。
10. [プロパティ] タブを選択してトレースレベルを変更するか、または生成キーシーケンス番号を次のセッション後にリセットします。

マッピングにおけるノーマライザトランスフォーメーションの使用

ノーマライザトランスフォーメーションが COBOL ソースから受け取るデータのタイプが複数ある場合は、各行内のデータのタイプに応じた別々のターゲットにノーマライザ出力ポートを接続する必要があります。次の例に、ノーマライザトランスフォーメーションを介して Sales_File COBOL ソース定義をと複数のターゲットにマップする方法を示します。

Sales_File ソースレコードには、店舗情報または店舗で販売されている品目に関する情報のどちらかが含まれています。売上ファイルには、両方のタイプのレコードが含まれています。

以下の例に、2 つの売上ファイルレコードを挙げます。

Record Type	Data
Store Record	H01Software Suppliers Incorporated 1111 Battery Street San Francisco
Item Record	D01123456789USB Line - 10 Feet 001495000020 01Supp1 02Supp2 03Supp3 04Supp4

COBOL ソース定義およびノーマライザトランスフォーメーションには、両方のタイプのレコード内のフィールドを表すカラムがあります。品目行から店舗行をフィルタして、それらの行を種々のターゲットに渡す必要があります。

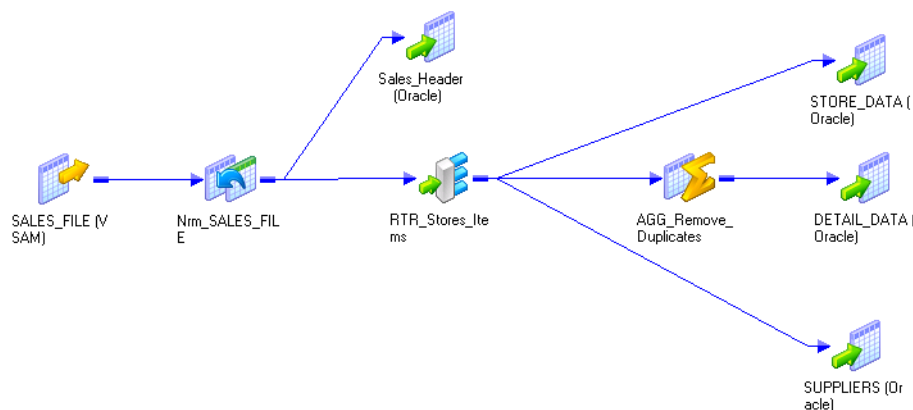
以下の図に、Sales_File COBOL ソースと、対応する Store_Data（値が"S"）および Detail_Data（値が"D"）の Sales_File COBOL ソースを示します。

キ	名前	レベ	発	データ型	長
	SALES_REC	1	0	number	10
	HDR_DATA	1	0	number	0
	HDR_STORE	2	0	number	10
	HDR_REC_TY...	2	0	number	0
	STORE_DA...	3	0	number	10
	STORE_NA...	3	0	number	10
	NEWFIELD	3	0	number	10
	NEWFIELD1	3	0	number	10
	NEWFIELD2	3	0	number	10
	NEWFIELD3	3	0	number	10

レコード内に店舗データまたは商品データのどちらが格納されているかは、Hdr_Rec_Type で定義されます。Hdr_Rec_Type 値が"S"ならレコードの内容は Store_Data で、Hdr_Rec_Type が"D"なら Detail_Data です。Detail_Data レコード内には常に、Supplier_Info フィールドのオカレンスが 4 つ含まれています。

データをフィルタするには、ノーマライザ出力行をルータ変換に接続し、店舗、品目、およびサプライヤデータを別々のターゲットにルーティングします。Hdr_Rec_Type の値に基づいて、ルータ変換内での行をフィルタすることができます。

以下の図は、Sales_File レコードを各種ターゲットにルーティングするマッピングを示しています。



マッピングによって、複数のレコードタイプが COBOL ソースからリレーショナルターゲットにフィルタされます。複数出現ソースカラムは、別個のリレーショナルテーブルにマップされます。各行は、ソース行内のオカレンスでインデックス付けされます。

マッピングには次の変換が含まれます。

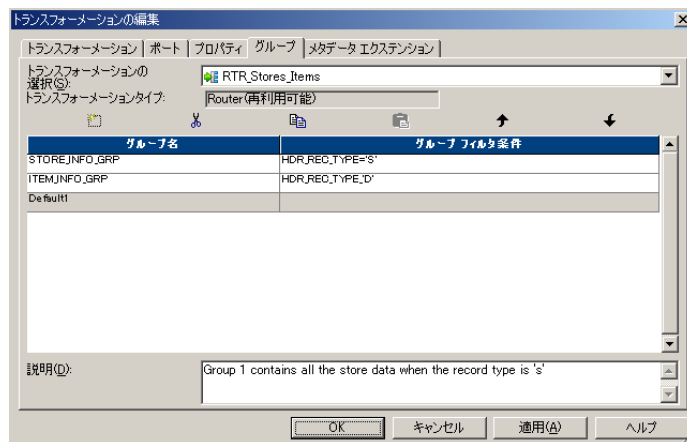
- **ノーマライザ変換。** ノーマライザ変換では、ソースに複数出現 Detail_Data が含まれている場合、複数行が返されます。また、同じソースからのいくつかのレコードタイプが処理されます。
- **ルータ変換。** ルータ変換は、Hdr_Rec_Type の値に応じてデータをターゲットにルーティングします。
- **アグリゲータ変換。** Aggregator 変換は、各 Supplier_Info オカレンスと共に出現した Detail_Data 行のうち重複している行を除去します。

マッピングの機能は次のとおりです。

1. ノーマライザ変換は、ヘッダレコードタイプおよびヘッダ店舗番号のカラムを Sales_Header ターゲットに渡します。各 Sales_Header レコードは、Sales_Header 行をターゲット行の Store_Data または Detail_Data にリンクするための生成キーを備えています。Hdr_Data および Store_Data は行ごとに 1 回返されます。

2. すべてのカラムがノーマライザトランスフォーメーションからルータトランスフォーメーションに渡されます。Detail_Data データは 1 行につき 4 回、Supplier_Info カラムのオカレンスごとに 1 回ずつ渡されます。Detail_Data カラム（ただし、Supplier_Info カラムを除く）には重複データが含まれます。
3. Hdr_Rec_Type が "S" の場合、ルータトランスフォーメーションで店舗名、所在地、市、および生成キーが Store_Data に渡されます。生成キーによって、StoreData 行が Sales_Header 行にリンクされます。
ルータトランスフォーメーションには、店舗データ用および商品用のユーザー定義グループが 1 つずつ含まれています。
4. ルータトランスフォーメーションは、品目、品目説明、価格、数量、および Detail_Data 生成キーをアグリゲータトランスフォーメーションに渡します。
5. Hdr_Rec_Type が "D" の場合、ルータトランスフォーメーションはサプライヤコード、名前、およびカラム ID を Suppliers ターゲットに渡します。このトランスフォーメーションによって渡された生成済みキーは、Suppliers 行を Detail_Data カラムにリンクするキーです。
6. 重複した Detail_Data カラムは、アグリゲータトランスフォーメーションによって削除されます。品目、説明、価格、数量、および生成キーの 1 インスタンスが、アグリゲータから Detail_Data に渡されます。Detail_Data の生成キーにより、Detail_Data 行が Suppliers 行にリンクされます。Detail_Data には、Detail_Data 行を Sales_Header 行にリンクするキーもあります。

以下の図は、ルータトランスフォーメーション内のユーザー定義グループおよびフィルタ条件を示しています。



ルータトランスフォーメーションで店舗データまたは品目データのどちらが渡されるかは、レコードタイプに基づきます。

キー値の生成

COBOL ソースに複数出現カラムグループが含まれている場合は、生成キーがノーマライザトランスフォーメーションにより作成されます。複数出現カラムグループの渡し先となるターゲットは、その行内の他のカラム以外でもかまいません。ターゲット間のプライマリキーと外部キーのリレーションは、生成キーを使用して作成することができます。

以下の図は、複数出現カラムグループを含む COBOL ソース定義を示しています。

Detail_Sales (VSAM)				
Name	Level	Occurs	Datatype	Length
DETAIL_RECORD	1	0		0
DETAIL_ITEM	3	0	number	9
DETAIL_DESC	3	0	string	30
DETAIL_PRICE	3	0	number	6
DETAIL_QTY	3	0	number	5
DETAIL_SUPPLIERS	3	4	number	0
SUPPLIER_CODE	10	0	string	2
SUPPLIER_NAME	10	0	string	8

この例では、カラムの Detail_Suppliers グループは、Detail_Record に 4 回出現します。

ノーマライザトランスフォーメーションはソース行ごとに GK_Detail_Sales キーを生成します。

GK_Detail_Sales キーは、1 つの Detail_Record ソース行を表します。

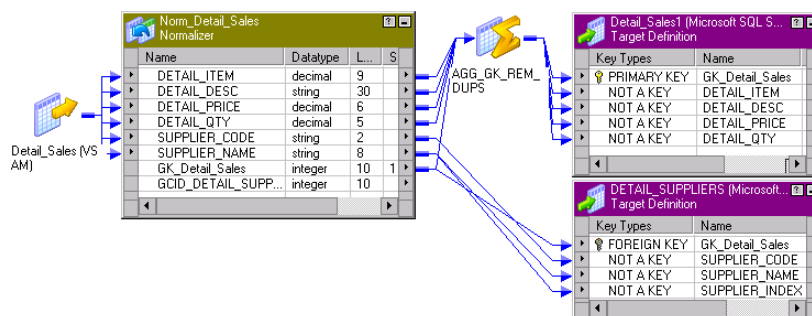
以下の図に、ターゲット間のプライマリキーと外部キーのキー関係を示します。

Detail_Sales (Microsoft SQL Server)			
名前	データタイプ	長さ/精度	
GK_Detail_Sales	int	10	
DETAIL_ITEM	numeric	9	
DETAIL_DESC	varchar	30	
DETAIL_PRICE	numeric	5	

DETAIL_SUPPLIERS (Microsoft SQL Server)			
名前	データタイプ	長さ/精度	
GK_DETAIL_SAL	int	10	
SUPPLIER_CODE	varchar	2	
SUPPLIER_NAME	varchar	8	

複数出現 Detail_Supplier 行はそれぞれ、自身と同じ Detail_Sales 行にリンクするための外部キーを備えています。Detail_Sales ターゲットは、Detail_Suppliers ターゲットと 1 対多の関係を持ちます。

以下の図に、ターゲット内のプライマリキーと外部キーに接続された GK_Detail_Sales 生成キーを示します。



GK_Detail_Sales を Detail_Sales のプライマリキーおよび Detail_Suppliers の外部キーに渡します。

ノーマライザの出力カラムを、次のオブジェクトにリンクします。

- **Detail_Sales_Target。** Detail_Item、Detail_Desc、Detail_Price、Detail_Qty の各カラムを Detail_Sales ターゲットに渡し、GK_Detail_Sales キーを Detail_Sales プライマリキーに渡します。
- **アグリゲータトランスフォーメーション。** アグリゲータトランスフォーメーションを介して各 Detail_Sales 行を渡し、重複する行を削除します。Detail_Suppliers のオカレンスごとに、重複している Detail_Sales カラムがノーマライザにより返されます。
- **Detail_Suppliers。** Detail_Suppliers カラムの各インスタンスを Detail_Suppliers ターゲットに渡し、GK_Detail_Sales キーを Detail_Suppliers 外部キーに渡します。Detail_Suppliers カラムの各インスタンスは、Detail_Suppliers 行を Detail_Sales 行に関連付ける外部キーを備えています。

ノーマライザトランスフォーメーションのトラブルシューティング

リレーショナルソースを使用しているときにノーマライザトランスフォーメーションのポートを編集できません。

ポートを手作業で作成する場合は、トランスフォーメーションの [ポート] タブではなく [ノーマライザ] タブで追加します。

COBOL ファイルのインポートは、多数のエラーが発生したため失敗しました。どうすれば良いですか？

COBOL プログラムがスペース、タブ、行末文字などの COBOL 規格に適合しているかどうかを調べます。COBOL ファイルヘッダは、次のようなテキスト形式になっている必要があります。

```
identification division.  
    program-id. mead.  
environment division.  
    select file-one assign to "fname".  
data division.  
file section.  
fd FILE-ONE.
```

COBOL プログラム内の隠し文字は、Designer に読み取られません。COBOL ファイルに変更を加えるときは、テキスト専用のエディタを使用してください。Word やワードパッドは使用厳禁です。余分なスペースを除去します。

バイナリデータを読み込むセッションが終了しましたが、ターゲットテーブル内の情報が正しくありません。

Workflow Manager でセッションを編集し、ソースファイル形式が正しく設定されていることを確認します。ソースファイル形式として良いのは、EBCDIC または ASCII です。また、[スキップするレコード間のバイト数] が 0 に設定されていなければなりません。

IBM COMP 以外の型を使用する COBOL フィールド記述があります。このソースをインポートするにはどうしたら良いですか？

ソース定義において [IBM COMP] オプションをオフにします。

自分のマッピングで、式トランスフォーメーションおよびルックアップトランスフォーメーションを 1 つずつ使用して、ノーマライザトランスフォーメーションからの出力ポート 2 つを修正します。これらの両トランスフォーメーションは、マッピングによって 1 つのトランスフォーメーションに連結されます。ポートはすべて同じレベルに属しています。ターゲットにロードされたデータをチェックしたところ、データが正しくありません。これはなぜですか？

ポートはレベル 1 でのみ連結することができます。連結を解除してください。

第 21 章

ランクトランスフォーメーション

この章では、以下の項目について説明します。

- [ランクトランスフォーメーションの概要, 373 ページ](#)
- [ランクトランスフォーメーションのポート, 374 ページ](#)
- [グループの定義, 375 ページ](#)
- [ランクトランスフォーメーションの作成, 376 ページ](#)

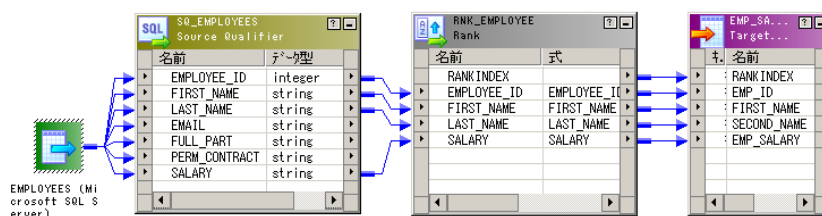
ランクトランスフォーメーションの概要

ランクトランスフォーメーションでは、最上位または最下位のランクのデータだけを選択できます。ランクトランスフォーメーションはアクティブなトランスフォーメーションです。ランクトランスフォーメーションを使用して、ポートまたはグループ内で最大または最小の数値を返すことができます。ランクトランスフォーメーションを使用して、セッションのソート順の最上位または最下位の文字列を返すこともできます。セッション実行中に、Integration Service はランク計算を実行できるまで入力データをキャッシュに格納します。

ランクトランスフォーメーションは、1 つの値だけではなく最上位または最下位の値のグループを選択できるという点で、トランスフォーメーション関数の MAX や MIN とは異なります。たとえば、ランクを使用して、指定された区域内での上位 10 人の販売員を選択できます。あるいは財務レポートを生成する場合も、ランクトランスフォーメーションを使用して、給与や経費の支出が最も少ない 3 つの部門を調べることができます。SQL 言語ではデータグループを取り扱う多くの関数が提供されていますが、標準 SQL 関数を使用して行セット内の最上位または最下位の層を特定することは不可能です。

トランスフォーメーションには、同じ行セットを表すすべてのポートを接続します。ランクトランスフォーメーションは、トランスフォーメーションを設定するときに指定した基準に基づいて、当該ランクに収まる行だけを通過させます。データを変換したり計算したりするための式を記述することもできます。

以下の図は、人的資源テーブルの従業員データをランクトランスフォーメーションに渡すマッピングを示しています。ランクトランスフォーメーションにより、給料が高い上位 10 人の従業員の行だけが次のトランスフォーメーションに渡されます。



ランクトランスフォーメーションはアクティブトランスフォーメーションであるため、通過する行の数を変更してしまう可能性があります。100 行をランクトランスフォーメーションへ渡し、その上位 10 行だけをランクトランスフォーメーションから別のトランスフォーメーションへ渡すようにすることができます。

ランクトランスフォーメーションへは、1 つのトランスフォーメーションからのポートしか接続できません。また、ローカル変数を作成して非集計式を書き込むこともできます。

文字列値のランク付け

ASCII データ移動モードで動作している Integration Service は、バイナリのソート順を使用してセッションデータをソートします。

Unicode データ移動モードで動作している Integration Service では、セッションに対して設定されたソート順が使用されます。セッションのソート順は、セッションプロパティで選択します。セッションプロパティには、Integration Service が使用するコードページに基づいて利用できるソート順がすべて一覧表示されます。

例えば、文字列ポートの最上位 3 つの値を返すようにランクトランスフォーメーションを設定したとします。ワークフローを設定する場合は、ワークフローを実行する Integration Service を選択します。セッションプロパティには、フランス語、ドイツ語、バイナリなど、選択した Integration Service のコードページに関連するソート順がすべて表示されます。バイナリのソート順を使用するようにセッションを設定すると、Integration Service は各文字列のバイナリ値を計算して、文字列のバイナリ値が最も大きい 3 行を返します。

ランクキャッシュ

Integration Service は、セッションの実行中に、入力行をデータキャッシュ内の行と比較します。キャッシュに格納されている行よりも入力行の方がランクが高い場合、Integration Service はキャッシュに格納されている行を入力行で置き換えます。複数のグループにまたがってランク付けを実行するようにランクトランスフォーメーションを設定した場合、Integration Service はランクの数字を増やす方法で、検出した各グループのランク付けを実行します。

Integration Service は、グループ情報をインデックスキャッシュに、行データをデータキャッシュに格納します。パイプライン内に複数のパーティションを作成した場合、Integration Service はパーティションごとの別々のキャッシュを作成します。

ランクトランスフォーメーションのプロパティ

ランクトランスフォーメーションを作成するときに、次のプロパティを設定することができます。

- キャッシュディレクトリを入力する。
- 最上位または最下位のランクを選択する。
- ランクを判定するための値を含む入出力ポートを選択する。ランクを定義するためのポートは 1 つだけ選択できます。
- ランクに収まる行の数を選択する。
- ランクのグループを定義する（たとえば各製造業者で最も安い 10 個の製品といったグループ）。

ランクトランスフォーメーションのポート

ランクトランスフォーメーションには、マッピング内のほかのトランスフォーメーションに接続されている入力ポートまたは入出力ポートが含まれます。またランクトランスフォーメーションには、変数ポートとランクポートも含まれます。ランクポートは、ランク付けの対象となるカラムを指定するために使用します。

以下の表に、ランクトランスフォーメーションのポートを示します。

ポート	必要な数	説明
I	1 つ以上	入力ポート。ほかのトランスフォーメーションからデータを受け取るための入力ポートを作成します。
O	1 つ以上	出力ポート。ほかのトランスフォーメーションにリンクしたいポートごとに出力ポートを作成します。入力ポートを出力ポートとして指定することができます。
V	なくてもよい	変数ポート。式で使用する値または計算を格納するために使用できます。変数ポートは入力ポートまたは出力ポートであってはなりません。変数ポートはトランスフォーメーション内でのみデータを渡します。
R	1 つのみ	ランクポート。値をランク付けする対象となるカラムを指定するために使用します。1 つのランクトランスフォーメーション内で指定できるランクポートは 1 つだけです。ランクポートは入出力ポートです。ランクポートはほかのトランスフォーメーションにリンクする必要があります。

ランクインデックス

Designer は、それぞれのランクトランスフォーメーションのそれぞれに対して RANKINDEX ポートを作成します。統合サービスは Rank Index ポートを使用して、グループ内における各行のランキング位置を格納します。

たとえば、四半期ごとに上位 5 人の販売員をランク付けする Rank トランスフォーメーションを作成した場合、ランクインデックスは販売員に 1 から 5 までの数字を付けます。

RANKINDEX	SALES_PERSON	SALES
1	Sam	10,000
2	Mary	9,000
3	Alice	8,000
4	Ron	7,000
5	Alex	6,000

RANKINDEX は出力ポートのみです。ランクインデックスは、マッピング内の別のトランスフォーメーションへ渡すか、直接ターゲットへ渡すことができます。

グループの定義

アグリゲータトランスフォーメーションの場合と同様に、ランクトランスフォーメーションでは情報をグループ化できます。たとえば製造業者別に最も高価な商品を 10 個選択したい場合は、まずそれぞれの製造業者についてグループを定義します。ランクトランスフォーメーションを設定するときに、そのいずれかの入出力ポートを Group By ポートとして設定することができます。グループポート内の一意の値それぞれに対して、トランスフォーメーションは、ランク定義（最上位または最下位、および各ランク内の特定の順位）に該当する行のグループを作成します。

したがって、ランクトランスフォーメーションは行の数を2つの方法で変更します。1つは、最上位または最下位のランクに収まる行を除いたすべての行をフィルタリングして除外することにより、トランスフォーメーションを通過する行の数を減らします。もう1つはグループを定義することにより、各グループでランク付けされた行のセットを1つ作成します。

たとえば、会社内で給料の高い50人の従業員を調べるために、ランクトランスフォーメーションを作成します。この場合、ランク付けの基準とする入出力ポートとしてSALARYカラムを指定し、上位50位以外のすべての行をフィルタで除外するようにトランスフォーメーションを設定します。

ランクトランスフォーメーションは、最上位または最下位のランクに属する行をすべて識別したあと、ランクインデックス値を割り当てます。給料を基準に上位50人の従業員を識別する場合、最高額の給料が支払われている従業員に対し、ランクインデックスとして1が与えられます。次に給料が多い従業員には、ランクインデックスとして2が与えられ、以下同じようにランクインデックスが割り当てられます。たとえば目録内で値段の安い10個の製品といった最下位ランクを求める場合、ランクトランスフォーメーションは最下位から最上位の順にランクインデックスを割り当てます。したがって、最も値段の安い商品のランクインデックスには1が与えられます。

2つのランク値が一致すると、それらは同じランクインデックス内で同じ値を受け取り、トランスフォーメーションはその次の値をスキップします。たとえば国内で最上位の5つの小売り店を調べたときに2つの小売り店が同じ売り上げである場合、返されるデータは次のようになります。

RANKINDEX	SALES	STORE
1	10000	Orange
1	10000	Brea
3	90000	Los Angeles
4	80000	Ventura

ランクトランスフォーメーションの作成

ランクトランスフォーメーションは、ソース修飾子以降のマッピング内の任意の場所に追加することができます。

ランクトランスフォーメーションを作成するには：

1. Mapping Designer で、[トランスフォーメーション] - [作成] をクリックします。Rank トランスフォーメーションを選択します。Rank トランスフォーメーションの名前を入力します。Rank トランスフォーメーションの命名規則は、「RNK_トランスフォーメーション名」です。
トランスフォーメーションの説明を入力します。この説明は、Repository Manager に表示されます。
2. [作成] をクリックし、次に [完了] をクリックします。
Designer がランクトランスフォーメーションを作成します。
3. 入力トランスフォーメーションからのカラムを Rank トランスフォーメーションへリンクします。
4. [ポート] タブをクリックし、ランクポートのランク (R) オプションを選択します。
ランク付けされた行のグループを作成したい場合は、グループを定義するポートに対して [Group By] を選択します。
5. [プロパティ] タブを選択し、最上位または最下位のどちらのランクを使用するかを選択します。
6. [Number of Ranks] オプションに対して、ランク用に選択する行の数を入力します。

7. 必要に応じて、他の Rank トランスフォーメーションプロパティを変更します。

以下の表では、ランクトランスフォーメーションのプロパティについて説明します。

設定	説明
キャッシュディレクトリ	統合サービスによるインデックスキャッシュファイルとデータキャッシュファイルの作成先となるローカルディレクトリ。統合サービスではデフォルトで、Workflow Manager でプロセス変数\$PMCacheDir に対して入力されたディレクトリが使用されます。新しいディレクトリを入力する場合は、そのディレクトリが存在していて、かつキャッシュファイルを格納するための十分なディスク領域があることを確認します。
上/下	カラムの最上位または最下位のどちらのランクを使用するかを指定します。
ランク数	ランク付けする行の数。
大文字と小文字を区別した文字列の比較	Unicode モードで動作している場合、統合サービスはセッションに対して選択されたソート順に基づいて文字列をランク付けします。セッションのソート順で大文字と小文字を区別する場合、このオプションを選択して大文字と小文字を区別する文字列の比較を有効にします。また、このオプションをクリアすると、統合サービスが大文字と小文字を区別しないように設定されます。ソート順で大文字と小文字が区別されない場合、統合サービスはこの設定を無視します。デフォルトでは、このオプションは選択されています。
トレースレベル	セッション中にこのトランスフォーメーションを通過するデータについて、統合サービスがセッションログに書き込む情報の量を指定します。
ランクのデータデータキャッシュサイズ	トランスフォーメーションのデータキャッシュサイズ。デフォルトは 2,000,000 バイトです。設定したセッションキャッシュの合計サイズが 2GB (2,147,483,648 バイト) 以上の場合は、そのセッションを 64 ビットの統合サービスで実行します。キャッシュの数値を使用できます。パラメータファイルのキャッシュ値を使用するか、統合サービスを設定し、自動設定を使用してキャッシュサイズを設定します。キャッシュサイズが決定されるように統合サービスを設定した場合、キャッシュに割り当てられる最大メモリ量も設定できます。
ランクのインデックスキャッシュサイズ	トランスフォーメーションのインデックスキャッシュサイズ。デフォルトは 1,000,000 バイトです。設定したセッションキャッシュの合計サイズが 2GB (2,147,483,648 バイト) 以上の場合は、そのセッションを 64 ビットの統合サービスで実行します。キャッシュの数値を使用できます。パラメータファイルのキャッシュ値を使用するか、統合サービスを設定し、自動設定を使用してキャッシュサイズを設定します。キャッシュサイズが決定されるように統合サービスを設定した場合、キャッシュに割り当てられる最大メモリ量も設定できます。
トランスフォーメーション範囲	<p>統合サービスが入力データにトランスフォーメーションロジックを適用する方法を指定します。</p> <ul style="list-style-type: none"> - Transaction。トランスフォーメーションロジックをトランザクションのすべての行に適用します。データの行が同一トランザクション内のすべての行に依存し、他のトランザクションの行には依存していない場合には、[Transaction] を選択します。 - すべての入力。トランスフォーメーションロジックをすべての入力データに適用します。[すべての入力] を選択すると、PowerCenter は入力トランザクションの境界を削除します。データの行がソース内のすべての行に依存している場合は、[すべての入力] を選択します。

8. [OK] をクリックします。

第 22 章

ルータトランスフォーメーション

この章では、以下の項目について説明します。

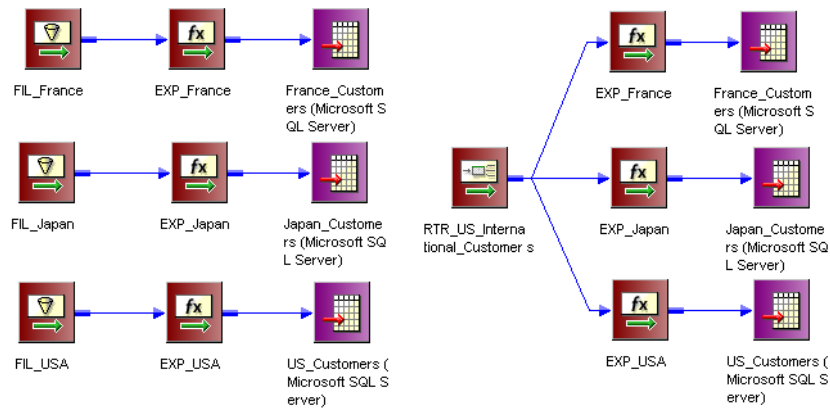
- [ルータトランスフォーメーションの概要, 378 ページ](#)
- [グループに関する作業, 380 ページ](#)
- [ポートに関する作業, 382 ページ](#)
- [マッピング内のルータトランスフォーメーションの接続, 383 ページ](#)
- [ルータトランスフォーメーションの作成, 383 ページ](#)

ルータトランスフォーメーションの概要

ルータトランスフォーメーションはフィルタトランスフォーメーションに似ており、両方とも条件を使用してデータをテストすることができます。ただし、フィルタトランスフォーメーションは 1 つの条件についてデータをテストし、条件を満たさない他のデータ行は削除します。ルータトランスフォーメーションは 1 つ以上の条件についてデータをテストし、どの条件も満足しないデータ行をデフォルト出力グループにルーティングするというオプションを提供します。ルータトランスフォーメーションはアクティブなトランスフォーメーションです。

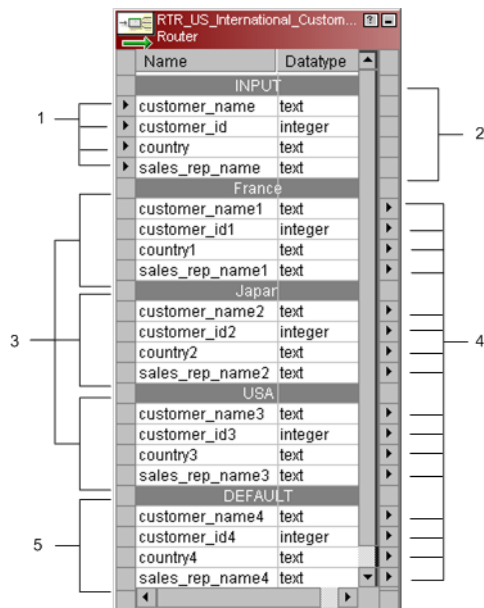
複数の条件に基づいて同じ入力データをテストする必要がある場合、同じ作業を実行する複数のフィルタトランスフォーメーションを作成する代わりに、マッピング内でルータトランスフォーメーションを使用します。ルータトランスフォーメーションの方が効率的です。たとえば、3 つの条件に基づいてデータをテストしたい場合、この作業を実行する 3 つのフィルタトランスフォーメーションの代わりに、ルータトランスフォーメーションを 1 つ作成するだけで済みます。同様に、マッピング内でルータトランスフォーメーションを使用すると、Integration Service は入力データを一度処理するだけで済みます。マッピング内で複数のフィルタトランスフォーメーションを使用すると、Integration Service は各トランスフォーメーションについて入力データを処理します。

以下の図は、同じタスクを実行する2つのマッピングを示しています。1つ目のマッピングがフィルタトランスフォーメーションを3つ使用しているのに対して、2つ目のマッピングはルータトランスフォーメーションを1つ使用して同じ結果を得ています。



ルータトランスフォーメーションは、入力および出力グループ、入力および出力ポート、グループフィルタ条件、Designerで設定されるプロパティによって構成されています。

以下の図はルータトランスフォーメーションの例を示しています。



1. 入力ポート。
2. 入力グループ。
3. ユーザー定義出力グループ
4. 出力ポート
5. デフォルト出力グループ

グループに関する作業

ルータトランスフォーメーションには、以下の種類のグループがあります。

- 入力
- 出力

入力グループ

Designer は入力グループの入力ポートからプロパティ情報をコピーし、各出力グループに対する一連の出力ポートを作成します。

出力グループ

次の 2 種類の出力グループがあります。

- ユーザー定義グループ
- デフォルトグループ

出力ポートやそのプロパティを変更または削除することはできません。

ユーザー定義グループ

ユーザー定義グループを作成して、到着するデータに応じて条件をテストします。ユーザー定義グループは、出力ポートおよびグループフィルタ条件で構成されています。Designer の [グループ] タブでは、ユーザー定義グループを作成および編集できます。指定したい各条件について、それぞれ 1 つのユーザー定義グループを作成します。

Integration Service は条件を使用して、入力データの各行を評価します。各ユーザー定義グループの条件をテストしてから、デフォルトグループを処理します。Integration Service は、接続された出力グループの順序に基づいて、各条件の評価順を決定します。Integration Service は、マッピング内のトランスフォーメーションまたはターゲットに接続されたユーザー定義グループを処理します。デフォルトグループがトランスフォーメーションまたはターゲットに接続されている場合、Integration Service はマッピング内の接続されていないユーザー定義グループだけを処理します。

行が複数のグループフィルタ条件を満たす場合、Integration Service はこの行を複数回渡します。

デフォルトグループ

Designer がデフォルトグループを作成するのは、ユーザーが新しいユーザー定義グループを作成した後です。Designer では、デフォルトグループを編集または削除することはできません。このグループに関連付けられたグループフィルタ条件はありません。すべての条件が FALSE と評価された場合、Integration Service は行をデフォルトグループに渡します。Integration Service でデフォルトグループの行をすべて削除する場合は、デフォルトグループをマッピング内のトランスフォーメーションまたはターゲットに接続しないようにします。

リストから最後のユーザー定義グループを削除すると、Designer はデフォルトグループを削除します。

グループフィルタ条件の使用

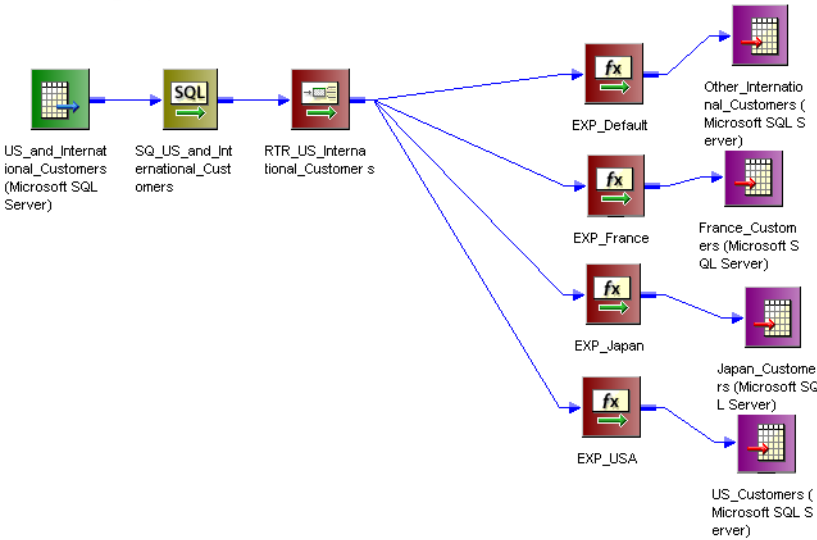
1 つ以上のグループフィルタ条件に基づいてデータをテストできます。式のエディタを使用して、[グループ] タブでグループフィルタ条件を作成します。単一の値を返す任意の式を入力することができます。条件に定数を指定することもできます。フィルタ条件は、指定された条件を行が満たすかどうかに基づいて、トランスフォーメーションを通過する行ごとに TRUE または FALSE を返します。FALSE に該当する値はゼロ (0) で、非

ゼロ値は TRUE に該当します。Integration Service は、TRUE と評価されたデータ行を、各ユーザー定義グループに関連付けられたそれぞれのトランスフォーメーションまたはターゲットに渡します。

たとえば、9 ヶ国の顧客について、その内の 3 ヶ国のデータに対してのみ種々の計算を行うとします。マッピング内でルータトランスフォーメーションを使用し、このデータをフィルタリングして 3 つの異なる式トランスフォーメーションに渡します。

デフォルトグループに関連付けられたグループフィルタ条件はありません。それでも、残りの 6 ヶ国のデータに基づいて計算を行う式トランスフォーメーションを作成することができます。

以下の図に、複数の条件でデータのフィルタリングを行うルータトランスフォーメーションによるマッピングを示します。



3 つの異なる国から得たデータに基づいて複数の計算を行うので、[グループ] タブでユーザー定義グループを 3 つ作成し、グループフィルタ条件を 3 つ指定します。

以下の表に、顧客データをフィルタリングするグループフィルタ条件を示します。

グループ名	グループフィルタ条件
フランス	customer_name='France'
Japan	customer_name='Japan'
USA	customer_name='USA'

Integration Service は、セッション実行中に、各ユーザー定義グループ（Japan、France、USA など）に関連付けられた各トランスフォーメーションまたはターゲットに、TRUE と評価されたデータ行を渡します。すべての条件が FALSE と評価された場合、Integration Service は行をデフォルトグループに渡します。その場合、Integration Service は他の 6 ヶ国のデータを、デフォルトグループに関連付けられたトランスフォーメーションまたはターゲットに渡します。Integration Service でデフォルトグループの行をすべて削除する場合は、デフォルトグループをマッピング内のトランスフォーメーションまたはターゲットに接続しないようにします。

ルータトランスフォーメーションは、条件を満たす各グループを通じてデータを渡します。したがって、データが 3 つの出力グループ条件を満たす場合、ルータトランスフォーメーションは 3 つの出力グループを通じてデータを渡します。

例えば、ルータトランスフォーメーションで以下のグループ条件を設定したとします。

グループ名	グループフィルタ条件
出力グループ 1	employee_salary > 1000
出力グループ 2	employee_salary > 2000

ルータトランスフォーメーションが入力行のデータを employee_salary=3000 で処理する場合、Output Group 1 および 2 を通じてデータがルーティングされます。

グループの追加

グループの追加は、他のトランスフォーメーションにおけるポートの追加に似ています。Designer は、入力ポートのプロパティ情報を出力ポートにコピーします。

ルータトランスフォーメーションにグループを追加するには：

1. [グループ] タブをクリックします。
2. [追加] ボタンをクリックします。
3. 新しいグループの名前を、[グループ名] セクションに入力します。
4. [グループフィルタ条件] フィールドをクリックして式のエディタを開きます。
5. グループフィルタ条件を入力します。
6. [検証] をクリックして条件の構文をチェックします。
7. [OK] をクリックします。

ポートに関する作業

ルータトランスフォーメーションには、入力ポートと出力ポートがあります。入力ポートは入力グループにあり、出力ポートは出力グループにあります。入力ポートは、別のトランスフォーメーションからコピーして作成するか、または [ポート] タブで手動で作成することができます。

Designer は、入力ポートから以下のプロパティをコピーすることによって出力ポートを作成します。

- ポート名
- データタイプ
- 精度
- スケール
- デフォルト値

入力ポートに変更を加えると、Designer は出力ポートを更新して変更を反映させます。出力ポートを編集または削除することはできません。出力ポートは、ルータトランスフォーメーションの通常ビューに表示されます。

Designer は、入力ポート名に基づいて出力ポート名を作成します。Designer は各入力ポートに対して、各出力グループ内に対応する出力ポートを作成します。

マッピング内のルータトランスフォーメーションの接続

トランスフォーメーションをマッピング内のルータトランスフォーメーションに接続する場合、以下の規則に留意してください。

- 1つのグループは、1つのトランスフォーメーションまたはターゲットに接続することができます。
- グループ内の1つの出力ポートは、複数のトランスフォーメーションまたはターゲットに接続することができます。
- 1つのグループ内の複数の出力ポートは、複数のトランスフォーメーションまたはターゲットに接続することができます。
- 複数のグループを1つのターゲットまたは1つの入力グループトランスフォーメーションに接続することはできません。
- 各出力グループを異なる入力グループに接続する場合は、ジョイナトランスフォーメーションを除いて、1つのグループを複数の入力グループトランスフォーメーションに接続することができます。

ルータトランスフォーメーションの作成

ルータトランスフォーメーションをマッピングに追加するには、以下の手順に従ってください。

1. Mapping Designer でマッピングを開きます。
2. [トランスフォーメーション] - [作成] をクリックします。
ルータトランスフォーメーションを選択し、新しいトランスフォーメーションの名前を入力します。ルータトランスフォーメーションの命名規則は、「RTR_トランスフォーメーション名」です。[作成] をクリックし、次に [完了] をクリックします。
3. ポートをすべてトランスフォーメーションから選択してドラッグし、ルータトランスフォーメーションに追加するか、または [ポート] タブで入力ポートを手動で作成することができます。
4. ルータトランスフォーメーションのタイトルバーをダブルクリックして、トランスフォーメーションプロパティを編集します。
5. [トランスフォーメーション] タブをクリックし、トランスフォーメーションプロパティを設定します。
6. [プロパティ] タブをクリックし、トレースレベルを設定します。
7. [グループ] タブをクリックし、[追加] をクリックしてユーザー定義グループを作成します。
最初のユーザー定義グループを作成すると、Designer はデフォルトグループを作成します。
8. [グループフィルタ条件] フィールドをクリックして式のエディタを開きます。
9. グループフィルタ条件を入力します。
10. [検証] をクリックして入力した条件の構文をチェックします。
11. [OK] をクリックします。
12. グループ出力ポートをトランスフォーメーションまたはターゲットに接続します。

第 23 章

シーケンスジェネレータトランスフォーマーション

この章では、以下の項目について説明します。

- [シーケンスジェネレータトランスフォーマーションの概要, 384 ページ](#)
- [シーケンスジェネレータのポート, 385 ページ](#)
- [シーケンスジェネレータトランスフォーマーションのプロパティ, 390 ページ](#)
- [シーケンスデータオブジェクト, 396 ページ](#)
- [シーケンスジェネレータトランスフォーマーションの作成, 398 ページ](#)
- [シーケンスジェネレータトランスフォーマーションの作成, 399 ページ](#)
- [FAQ \(よくある質問\) , 400 ページ](#)
- [シーケンスジェネレータトランスフォーマーション非ネイティブ環境で, 401 ページ](#)

シーケンスジェネレータトランスフォーマーションの概要

シーケンスジェネレータトランスフォーマーションは数値を生成するパッシブトランスフォーマーションです。一意なプライマリキー値の作成、欠落しているプライマリキーの置き換え、連続した数値のサイクル動作を実行するには、シーケンスジェネレータトランスフォーマーションを使用します。

シーケンスジェネレータトランスフォーマーションは、接続されたトランスフォーマーションです。出力ポートは 2 つあり、1 つまたは複数のトランスフォーマーションに接続することができます。統合サービスでは、接続されたトランスフォーマーションに複数行のブロックが入力されるたびに複数のシーケンス番号のブロックが生成されます。CURRVAL を接続すると、統合サービスは各ブロックごとに 1 行処理します。NEXTVAL が別のトランスフォーマーションの入力ポートに接続されている場合、統合サービスは連続した数値を生成します。CURRVAL が別のトランスフォーマーションの入力ポートに接続されている場合、統合サービスは NEXTVAL 値に増分値を加算した値を生成します。

シーケンスジェネレータには、パススルーポートと出力ポートがあります。NEXTVAL ポートを他のトランスフォーマーションの入力ポートに接続します。統合サービスはマッピングが実行されるときにシーケンスを増分します。

単一のマッピングで使用するシーケンスジェネレータトランスフォーマーションを作成したり、複数のマッピングで使用する再利用可能なシーケンスジェネレータトランスフォーマーションを作成したりできます。再利用可能なシーケンスジェネレータトランスフォーマーションは、シーケンスジェネレータトランスフォーマーションのインスタンスを使用する各マッピングでシーケンスの整合性を保持します。

シーケンスジェネレータトランスフォーメーションは、新しいシーケンスまたはシーケンスデータオブジェクトに基づいて作成できます。シーケンスデータオブジェクトは、値のシーケンスを作成して維持するオブジェクトです。

再利用可能なシーケンスジェネレータトランスフォーメーションを作成し、複数のマッピングで使用することができます。1つのターゲットに対して複数回のロードを実行するときには、シーケンスジェネレータトランスフォーメーションを再利用できます。

例えば、大きな入力ファイルを並列実行する3つのセッションに分ける場合、シーケンスジェネレータトランスフォーメーションを使用してプライマリーキー値を生成します。異なるシーケンスジェネレータトランスフォーメーションを使用すると、統合サービスにより重複するキー値が生成される可能性があります。その代わりに、再利用可能なシーケンスジェネレータトランスフォーメーションを3つのセッションすべてに対して使用して、それぞれのターゲット行に一意の値を提供します。

シーケンスジェネレータのポート

シーケンスジェネレータトランスフォーメーションは2つの出力ポートがあります。NEXTVAL と CURRVAL です。これらのポートは編集したり削除したりすることはできません。また、ほかのポートをこのトランスフォーメーションに追加することもできません。

シーケンスジェネレータトランスフォーメーションにはパススルーポートと1つの出力ポート（NEXTVAL）があります。出力ポートを編集または削除することはできません。

パススルーポート

シーケンスジェネレータトランスフォーメーションには、ポートをパススルーポートとして追加できます。パススルーポートは、入力データを受信し、同じデータを変更せずにマッピングに返す入出力ポートです。

NEXTVAL および CURRVAL 出力ポートをターゲットにリンクする前に、最低1つの入力ポートをトランスフォーメーションに追加してアップストリームのソースまたはトランスフォーメーションに接続する必要があります。パススルーポートをトランスフォーメーションに追加するには、ポートを、マッピングでアップストリームのソースまたはトランスフォーメーションからシーケンスジェネレータトランスフォーメーションにドラッグします。

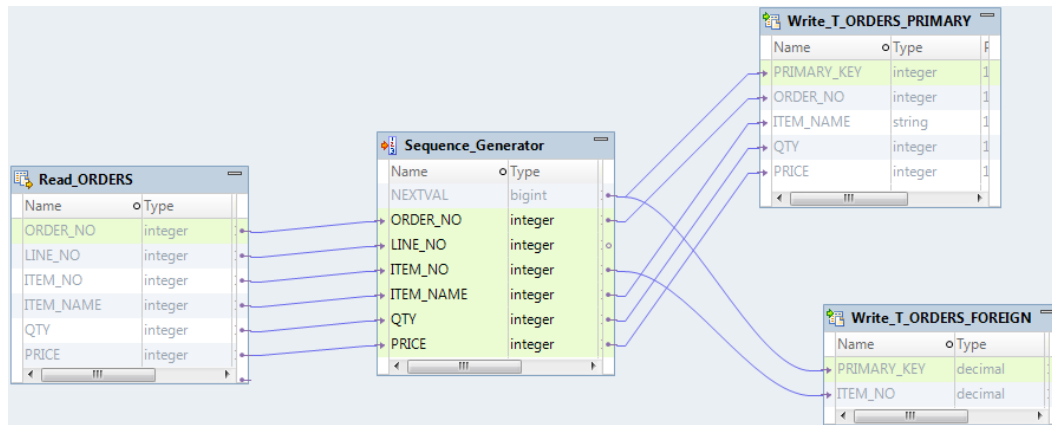
NEXTVAL 出力ポートをターゲットにリンクする前に、少なくとも1つの入力ポートをトランスフォーメーションに追加してアップストリームのソースまたはトランスフォーメーションに接続する必要があります。パススルーポートをトランスフォーメーションに追加するには、ポートを、マッピングでアップストリームのソースまたはトランスフォーメーションからシーケンスジェネレータトランスフォーメーションにドラッグします。

NEXTVAL ポート

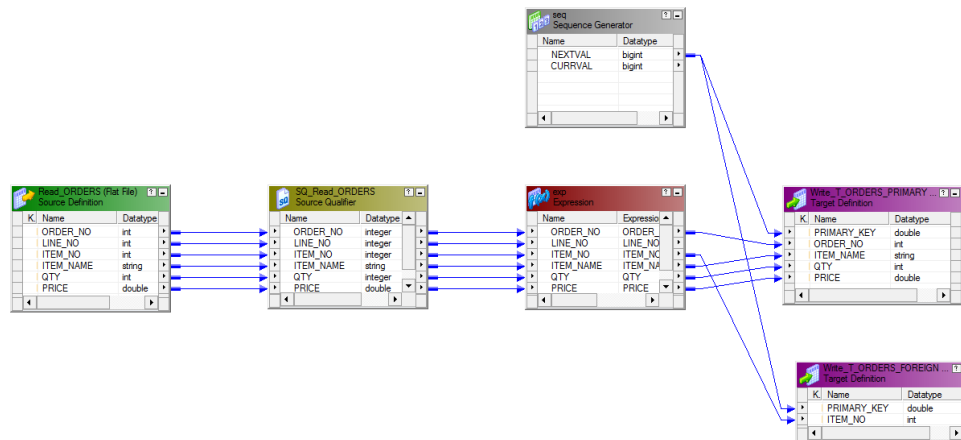
NEXTVAL をトランスフォーメーションに接続し、トランスフォーメーションの各行に対して一意の値を生成することができます。NEXTVAL ポートをダウンストリームのトランスフォーメーションまたはターゲットに接続し、番号のシーケンスを生成します。NEXTVAL を複数のトランスフォーメーションに接続した場合、統合サービスは各トランスフォーメーションに対して同じ番号のシーケンスを生成します。

NEXTVAL ポートを後続のトランスフォーメーションに接続し、[開始値] プロパティおよび [増分値] プロパティに基づいてシーケンスを生成します。シーケンスジェネレータがシーケンスでサイクル動作を実行するように設定されていない場合、NEXTVAL ポートは設定された終了値までキーシーケンス番号を生成します。

次の図に、ソースと2つのターゲットに接続され、プライマリーキー値と外部キー値を生成するシーケンスジェネレータトランスフォーメーションの NEXTVAL ポートを含むマッピングを示します。



次の図に、2つのターゲットに接続され、プライマリキー値と外部キー値を生成するシーケンスジェネレータートランスフォーメーションのNEXTVALポートを含むマッピングを示します。



シーケンスジェネレータートランスフォーメーションを開始値=1、増分値=1で設定すると、統合サービスはT_ORDERS_PRIMARYターゲットテーブルとT_ORDERS_FOREIGNターゲットテーブルに同じプライマリキー値を生成します。

NEXTVALを複数のトランスフォーメーションに接続して、各トランスフォーメーション内の各行に対して一意の値を生成します。NEXTVALポートを下流のトランスフォーメーションまたはターゲットに接続することによって、連続した数値を生成することができます。NEXTVALポートを後続のトランスフォーメーションに接続し、[Current Value]および[Increment By]プロパティに基づいてシーケンスを生成します。シーケンスジェネレーターがシーケンスでサイクル動作を実行するように設定されていない場合、NEXTVALポートは設定された終了値までキーシーケンス番号を生成します。

例えば、NEXTVALをマッピング内の2つのターゲットに接続して、一意なプライマリキー値を生成することができます。統合サービスは、各ターゲットテーブルに対して一意なプライマリキー値のカラムを作成します。一意なプライマリキー値のカラムはシーケンス番号のブロックとして1つのターゲットテーブルに送られます。2つ目のターゲットは、1つ目のターゲットテーブルがシーケンス番号のブロックを受け取った後、シーケンスジェネレータートランスフォーメーションからシーケンス番号のブロックを受け取ります。

例えば、シーケンスジェネレータトランスフォーメーションで、[現在の値] = 1、[増分] = 1 と設定します。統合サービスは、T_ORDERS_PRIMARY および T_ORDERS_FOREIGN ターゲットテーブル用に以下のプライマリキー値を生成します。

**T_ORDERS_PRIMARY TABLE:
PRIMARY KEY**

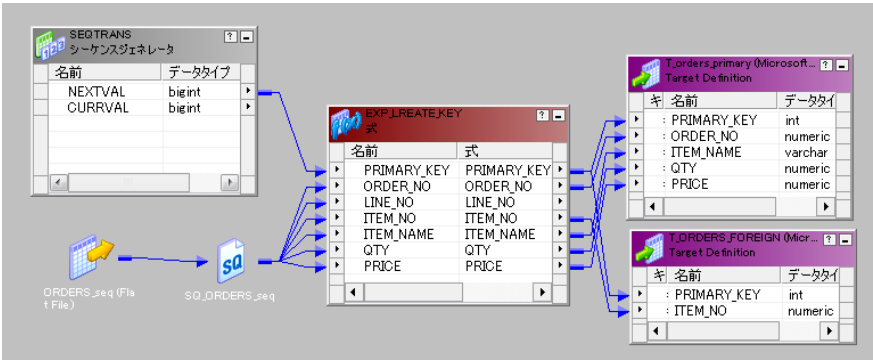
- 1
- 2
- 3
- 4
- 5

**T_ORDERS_FOREIGN TABLE:
PRIMARY KEY**

- 6
- 7
- 8
- 9
- 10

同じ値を 2 つ以上のターゲットに渡す場合、それらのターゲットがすべて 1 つのトランスフォーメーションからデータを受け取るのであれば、シーケンスジェネレータトランスフォーメーションをその先行するトランスフォーメーションに接続することができます。統合サービスは、この値をシーケンス番号のブロックに加工します。これにより、統合サービスは一意的な値をトランスフォーメーションに渡し、そのトランスフォーメーションからターゲットへ行をルーティングできます。

以下の図に、式トランスフォーメーションに一意的な値を渡すシーケンスジェネレータを使用したマッピングを示します。



式トランスフォーメーションは、同じプライマリキー値を両方のターゲットに渡します。

例えば、シーケンスジェネレータトランスフォーメーションで、[現在の値] = 1、[増分] = 1 と設定します。統合サービスは、T_ORDERS_PRIMARY および T_ORDERS_FOREIGN ターゲットテーブル用に以下のプライマリキー値を生成します。

**T_ORDERS_PRIMARY TABLE:
PRIMARY KEY**

- 1
- 2
- 3
- 4
- 5

**T_ORDERS_FOREIGN TABLE:
PRIMARY KEY**

- 1
- 2
- 3
- 4
- 5

注: グリッド上でパーティション化されたセッションを実行すると、各パーティションに含まれる行の数によっては、シーケンスジェネレータトランスフォーメーションは値をスキップすることがあります。

キーの作成

シーケンスジェネレータトランスフォーメーションでプライマリキーまたは外部キーの値を作成するには、NEXTVAL ポートをターゲットまたはダウンストリームのトランスフォーメーションに接続します。1～9,223,372,036,854,775,807 の範囲の値を、最小間隔 1 で使用することができます。

シーケンスジェネレータトランスフォーメーションでプライマリキーまたは外部キーの値を作成するには、NEXTVAL ポートをターゲットまたはダウンストリームのトランスフォーメーションに接続します。1～9,223,372,036,854,775,807 の範囲の値を、最小間隔 1 で使用することができます。複合キーを作成してテーブルの各行を識別することもできます。

プライマリキーまたは外部キーを作成する場合は、[サイクル] オプションを使用して、統合サービスが重複したプライマリキーを作成しないようにします。これは、セッションプロパティの [ターゲットテーブルの切り詰め] オプションを選択するか、複合キーを作成することによって行えます。

複合キーを作成するには、値の小さなセットでサイクル動作を実行するように統合サービスを設定します。例えば、3 つの店舗が注文番号を生成する場合は、シーケンスジェネレータトランスフォーメーションが 1 から 3 まで 1 ずつ増やしてサイクル動作するように設定します。ORDER_NO ポートをシーケンスジェネレータトランスフォーメーションに接続する場合は、生成される値によって一意の複合キーが作成されます。

次の例に、複合キーと注文番号を示します。

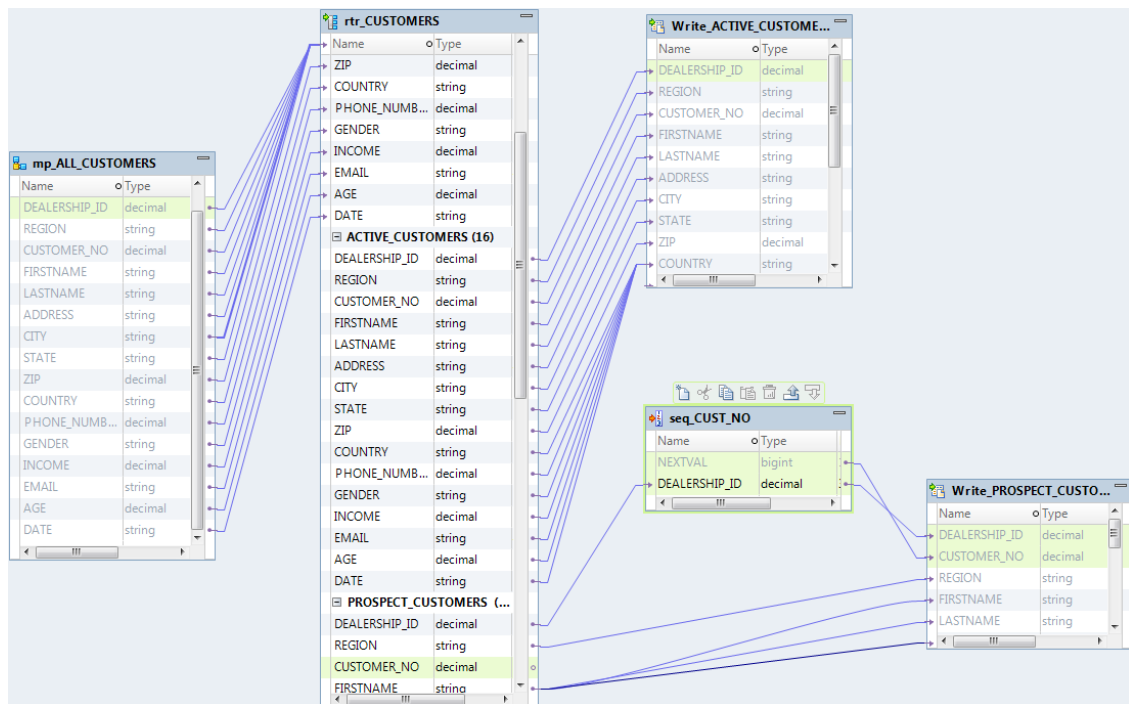
COMPOSITE_KEY	ORDER_NO
1	12345
2	12345
3	12345
1	12346
2	12346
3	12346

欠落値の置き換え

シーケンスジェネレータトランスフォーメーションを使用して欠落したキーを置き換える場合は、ルータトランスフォーメーションを使用して、値を割り当てたカラムから NULL 値のカラムをフィルタリングすることもできます。ルータトランスフォーメーションをシーケンスジェネレータトランスフォーメーションに接続し、NEXTVAL を使用して数値のシーケンスを生成して NULL 値を入力します。

例えば、CUSTOMER_NO カラムの NULL 値を置き換えるには、顧客データを含むソースを持つマッピングを作成します。ルータトランスフォーメーションを追加し、NULL 値の顧客から顧客番号が割り当てられている顧客をフィルタリングします。シーケンスジェネレータトランスフォーメーションを追加し、一意の CUSTOMER_NO 値を生成します。顧客ターゲットを追加し、データを書き込みます。

次の図に、CUSTOMER_NO カラムの NULL 値を置き換えるマッピングを示します。



シーケンスジェネレーター変換を使用し、NEXTVAL に IIF 関数と ISNULL 関数を使用することで、欠落したキーを置き換えます。

例えば、ORDER_NO カラム内の NULL 値を置き換えるには、プロパティを持つシーケンスジェネレーター変換を作成し、NEXTVAL ポートを式変換にドラッグします。式変換に、ORDER_NO ポート（およびその他の必要なポート）をドラッグします。そして出力ポート ALL_ORDERS を作成します。

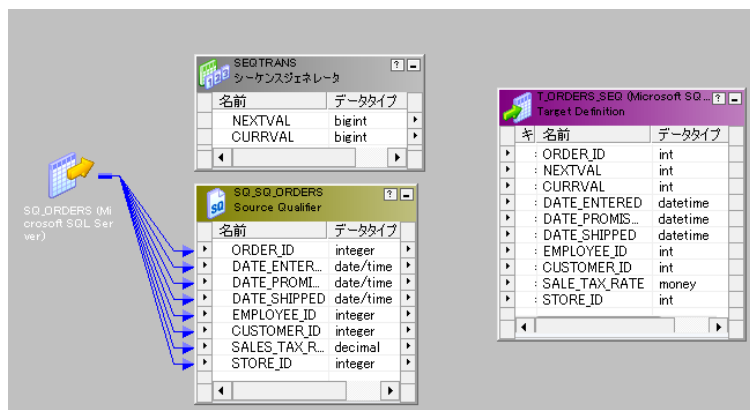
ALL_ORDERS で、次の式を入力して値が NULL の注文を置き換えます。

`IIF(ISNULL(ORDER_NO), NEXTVAL, ORDER_NO)`

CURRVAL

CURRVAL は、NEXTVAL 値に増分値を加えた値です。通常は、NEXTVAL ポートが既に後続の変換へ接続されているときにだけ CURRVAL ポートを接続します。CURRVAL ポートに接続された変換に行が入力されると、統合サービスは最後に作成された NEXTVAL 値に 1 を加算した値を渡します。

以下の図に、ターゲットへの CURRVAL ポートおよび NEXTVAL ポートの接続を示します。



例えば、シーケンスジェネレータトランスフォーメーションで、[現在の値] = 1、[増分] = 1 と設定します。統合サービスでは、NEXTVAL および CURRVAL 用に以下の値が生成されます。

NEXTVAL	CURRVAL
1	2
2	3
3	4
4	5
5	6

NEXTVAL ポートを接続せずに CURRVAL ポートを接続すると、統合サービスは各行の定数値を渡します。シーケンスジェネレータトランスフォーメーションで CURRVAL ポートを接続すると、統合サービスは各ブロックで 1 行ずつ処理します。マッピングで NEXTVAL ポートのみを接続すると、パフォーマンスを最適化できます。

注: グリッド上でパーティション化されたセッションを実行すると、各パーティションに含まれる行の数によっては、シーケンスジェネレータトランスフォーメーションは値をスキップすることがあります。

シーケンスジェネレータトランスフォーメーションのプロパティ

統合サービスが連続した値を生成するのに使用するトランスフォーメーションプロパティを設定します。
以下の表に、シーケンスデータオブジェクトと新しいシーケンスに対して構成するプロパティを示しています。

プロパティ	説明
開始値	[サイクル] オプションを選択する場合に、統合サービスが使用する生成シーケンスの開始値。 [サイクル] を選択すると、統合サービスのサイクルは終了値に到達したときにこの値に戻ります。 デフォルトは 0 です。 最大値は 9,223,372,036,854,775,806 です。
終了値	統合サービスが生成する最大値。統合サービスがセッション中にこの値に到達し、シーケンスのサイクル動作が設定されていない場合、セッションは失敗します。 最大値は 9,223,372,036,854,775,807 です。

プロパティ	説明
増分値	NEXTVAL ポートから出力される連続した 2 つの値の差。 デフォルトは 1 です。 正の整数にする必要があります。 最大値は 2,147,483,647 です。
サイクル	有効にすると、統合サービスはシーケンスの範囲でサイクル動作し、開始値を使用して再開します。 無効にすると、統合サービスは設定されている終了値でシーケンスを停止します。統合サービスが終了値に到達した時点で処理する必要がある行が残っていると、オーバーフローエラーによってセッションが失敗します。

次のリストでは、設定可能なシーケンスジェネレータトランスフォーメーションのプロパティについて説明します。

開始値

[サイクル] オプションを選択する場合に、統合サービスが使用する生成シーケンスの開始値。[サイクル] を選択すると、統合サービスのサイクルは終了値に到達したときにこの値に戻ります。

デフォルトは 0 です。

最大値は 9,223,372,036,854,775,806 です。

増分

NEXTVAL ポートから出力される連続した 2 つの値の差。

デフォルトは 1 です。

正の整数にする必要があります。

最大値は 2,147,483,647 です。

終了値

統合サービスが生成する最大値。統合サービスがセッション中にこの値に到達し、シーケンスのサイクル動作が設定されていない場合、セッションは失敗します。

最大値は 9,223,372,036,854,775,807 です。

NEXTVAL ポートをダウンストリームの整数ポートに接続する場合は、[終了値] を最大整数値ほどの大きさの値に設定します。NEXTVAL がそのダウンストリームポートのデータ型の最大値を超えると、セッションは失敗します。

現在の値

シーケンスのカレント値。Integration Service がシーケンスの最初の値として使用する値を入力します。一連の値でサイクル動作を実行するように設定する場合は、この値を開始値以上、終了値未満にする必要があります。

[キャッシュされる値の数] を 0 に設定した場合、統合サービスは現在の値を更新し、セッションで最後に生成された値に 1 を加算した値を反映し、更新された現在の値を次のセッション実行時の基準として使用します。ただし、[リセット] オプションを使用する場合、統合サービスは各セッションの終了後にこの値を元の値にリセットします。

注: この設定を編集すると、シーケンスは新しい設定にリセットされます。[現在の値] を 10 にリセットし、増分が 1 の場合、統合サービスは次のセッション実行時に最初の値として 10 を生成します。

最大値は 9,223,372,036,854,775,806 です。現在の値が最大値を上回っている場合、Integration Service は値を NULL に設定します。

サイクル

有効にすると、統合サービスはシーケンスの範囲でサイクル動作し、開始値を使用して再開します。

無効にすると、統合サービスは設定されている終了値でシーケンスを停止します。統合サービスが終了値に到達した時点で処理する必要がある行が残っていると、オーバーフローエラーによってセッションが失敗します。

キャッシュされる値の数

統合サービスが一度にキャッシュする連続値の数。このオプションは、複数のセッションが同じ再利用可能なシーケンスジェネレータを同時に使用する際に、各セッションが一意の値を受け取ることを保証するために使用します。統合サービスは、それぞれの値をキャッシュするたびにリポトリを更新します。0 に設定すると、統合サービスは値をキャッシュしません。

デフォルト値は 0 です。

再利用可能なシーケンスジェネレータのデフォルト値は 1,000 です。

最大値は 9,223,372,036,854,775,807 です。

このプロパティは、再利用可能なシーケンスジェネレータトランスフォーメーションのみに該当します。

リセット

有効にすると、統合サービスは各セッションの元の現在の値に基づいて値を生成します。無効にすると、統合サービスは現在の値を更新し、セッションで最後に生成された値に 1 を加算した値を反映します。更新後の現在の値は、次のセッション実行時の基準として使用します。

このプロパティは、再利用不可能なシーケンスジェネレータトランスフォーメーションのみに該当します。

トレースレベル

統合サービスがセッションログに書き込むトランスフォーメーションの詳細レベル。

開始値

「サイクル」を使用すると、月に対応する 1 から 12 の番号など、繰り返しのシーケンスを生成できます。

1. 統合サービスが開始値として使用するシーケンスの最小値を入力します。
2. 「End Value」として使用する最大値を入力します。
3. 「Cycle」を選択します。

サイクル動作の実行中にシーケンスで設定した終了値に到達すると、統合サービスは設定した開始値から再びサイクル動作を実行します。

増分

統合サービスは、シーケンスジェネレータトランスフォーメーションの「現在の値」および「増分」プロパティに基づいて、NEXTVAL ポートにシーケンスを生成します。

「現在の値」プロパティは、統合サービスが各セッション用シーケンスの作成を開始する際に使用する値です。「増分」は、統合サービスがシーケンスで新しい値を生成するために既存の値に追加する整数値です。デフォルトでは、「現在の値」と「増分」が共に 1 に設定されます。

たとえば [Current Value] を 1000 に、[Increment By] を 10 に設定したシーケンスジェネレータトランスフォーメーションを作成したとします。マッピングを介して 3 つの行を渡した場合は、統合サービスによって次の値のセットが生成されます。

```
1000
1010
1020
```

終了値

〔終了値〕は、統合サービスが生成する最大値です。統合サービスが終了値に達した場合、シーケンスジェネレータがシーケンスをサイクル動作するように設定されていないと、セッションはオーバーフローエラーを出力して失敗します。

終了値は、統合サービスが生成する最大値です。統合サービスが終了値に到達し、シーケンスジェネレータがシーケンスをサイクル動作するように設定されていない場合、マッピングの実行はオーバーフローエラーで失敗します。

End Value は、1~9,233,372,036,854,775,807 の任意の整数に設定できます。NEXTVAL ポートをダウンストリームの整数ポートに接続する場合は、〔終了値〕を最大整数値ほどの大きさの値に設定します。たとえば、NEXTVAL ポートを小整数ポートに接続する場合は、〔終了値〕を最大 32,767 に設定します。NEXTVAL がそのダウンストリームポートのデータタイプの最大値を超えると、セッションは失敗します。

End Value は、1~9,233,372,036,854,775,807 の任意の整数に設定できます。NEXTVAL ポートをダウンストリームの整数ポートに接続する場合は、〔終了値〕を最大整数値ほどの大きさの値に設定します。たとえば、NEXTVAL ポートを小整数ポートに接続する場合は、〔終了値〕を最大 32,767 に設定します。NEXTVAL がそのダウンストリームポートのデータタイプの最大値を超えると、マッピングは失敗します。

増分値

統合サービスは、シーケンスジェネレータトランスフォーメーションの〔現在の値〕および〔増分〕プロパティに基づいて、NEXTVAL ポートにシーケンスを生成します。

〔現在の値〕プロパティは、統合サービスが各セッション用シーケンスの作成を開始する際に使用する値です。〔増分〕は、統合サービスがシーケンスで新しい値を生成するために既存の値に追加する整数値です。デフォルトでは、〔現在の値〕と〔増分〕が共に 1 に設定されます。

たとえば [Current Value] を 1000 に、[Increment By] を 10 に設定したシーケンスジェネレータトランスフォーメーションを作成したとします。マッピングを介して 3 つの行を渡した場合は、統合サービスによって次の値のセットが生成されます。

```
1000
1010
1020
```

値の範囲内でのサイクル

シーケンスジェネレータトランスフォーメーションに値の範囲を設定できます。サイクルオプションを使用する場合、シーケンスジェネレータトランスフォーメーションは終了値に到達すると範囲を繰り返します。

例えば、シーケンス範囲を 10 で開始し 50 で終了するように設定し、増分値を 10 に設定すると、シーケンスジェネレータトランスフォーメーションは 10、20、30、40、50 の値を作成します。シーケンスは、10 から再び開始されます。

現在の値

統合サービスにおいて現在の値は、各セッションの生成値の基準として使用されます。統合サービスがシーケンスジェネレータトランスフォーメーションを使用する際に最初に使用する値を指定するためには、その値を

現在の値として入力する必要があります。シーケンスジェネレータトランスフォーメーションで一連の値をサイクルに従って使用する場合は、[現在の値] は、[開始値] 以上 [終了値] 未満である必要があります。

統合サービスにおいて現在の値は、各マッピング実行時の生成値の基準として使用されます。シーケンスジェネレータトランスフォーメーションにおいて開始値は、常に現在の値として使用されます。

シーケンスジェネレータの [キャッシュされる値の数] が 0 の場合、統合サービスでは各セッション終了時に現在の値が、セッションで最後に生成された値に 1 を加算した値に更新されます。例えば、統合サービスが生成値 101 でセッションを終了した場合、リポジトリ内のシーケンスジェネレータの現在の値が 102 に更新されます。統合サービスでは次のシーケンスジェネレータ使用時に、次の生成値の基準として 102 が使用されます。シーケンスジェネレータの [増分] が 1 の場合、統合サービスがシーケンスジェネレータを使用して別のセッションを開始すると、最初の生成値は 102 となります。

シーケンスジェネレータの [キャッシュされる値の数] が 0 の場合、統合サービスでは各セッション終了時に現在の値が、マッピングの実行で最後に生成された値に 1 を加算した値に更新されます。例えば、統合サービスが生成値 101 でセッションを終了した場合、リポジトリ内のシーケンスジェネレータの現在の値が 102 に更新されます。統合サービスでは次のシーケンスジェネレータ使用時に、次の生成値の基準として 102 が使用されます。シーケンスジェネレータの [増分] が 1 の場合、統合サービスがシーケンスジェネレータを使用して別のセッションを開始すると、最初の生成値は 102 となります。

複数のバージョンのシーケンスジェネレータトランスフォーメーションがある場合、統合サービスはセッション実行時にすべてのバージョンで現在の値を更新します。統合サービスでは、シーケンスジェネレータトランスフォーメーションまたは親マッピングがチェックアウト済みかどうかにかかわらず、すべてのバージョンで現在の値が更新されます。更新後の現在の値とシーケンスジェネレータトランスフォーメーションの編集後の現在の値が異なる場合、更新後の現在の値はシーケンスジェネレータトランスフォーメーションの編集後の現在の値を上書きします。

例えば、ユーザー 1 がシーケンスジェネレータトランスフォーメーションを作成してチェックインし、シーケンスジェネレータバージョン 1 に現在の値として 10 を保存したとします。次に、ユーザー 1 はシーケンスジェネレータトランスフォーメーションをチェックアウトし、シーケンスジェネレータバージョン 2 に新しい現在の値として 100 を入力します。ユーザー 1 は、シーケンスジェネレータトランスフォーメーションをチェックアウトしたままにします。その間、ユーザー 2 はシーケンスジェネレータトランスフォーメーションのバージョン 1 を使用するセッションを実行します。ユーザー 2 がセッションを実行するとき、統合サービスはチェックインされている値 (10) を現在の値として使用します。セッション完了時の現在の値は 150 です。この場合、ユーザー 1 がシーケンスジェネレータトランスフォーメーションをチェックアウトしていますが、統合サービスはシーケンスジェネレータトランスフォーメーションのバージョン 1 およびバージョン 2 の両方で現在の値を 150 に更新します。

セッションを実行したあとでマッピングを開くと、[現在の値] にはセッションで最後に生成された値に 1 を加えた値が表示されます。統合サービスでは各セッションの最初の値の判別に現在の値が使用されるため、シーケンスをリセットする場合にしか現在の値を編集しないようにする必要があります。

複数のバージョンのシーケンスジェネレータトランスフォーメーションがあり、シーケンスをリセットする場合は、[現在の値] を変更した後でマッピングまたは再利用可能なシーケンスジェネレータトランスフォーメーションをチェックインする必要があります。

注: シーケンスジェネレータで [リセット] に設定した場合、統合サービスは現在の値を各セッションの最初の生成値の基準として使用します。

キャッシュされる値の数

[キャッシュされる値の数] では、統合サービスが一度にキャッシュする値の数が決定されます。[キャッシュされる値の数] がゼロより大きい場合、統合サービスは設定された数の値をキャッシュに格納します。このとき、値をキャッシュするごとに現在の値も更新します。

複数のセッションが同じ再利用可能なシーケンスジェネレータトランスフォーメーションを同時に使用する際、シーケンスジェネレータトランスフォーメーションのインスタンスが複数存在する場合があります。各セッ

ョンで同じ値を生成しないように、[Number of Cached Values] を設定することによって、各セッションについて連続した値の範囲を予約します。

ヒント: グリッド上でセッションを実行する場合にパフォーマンスを向上させるには、シーケンスジェネレータトランスフォーメーションの [Number of Cached Values] を増やします。この結果、マスター DTM プロセスおよびワーカー DTM プロセスとリポジトリの間で必要な通信を削減できます。

再利用不可能なシーケンスジェネレータ

再利用不可能なシーケンスジェネレータトランスフォーメーションの場合、デフォルトでは [キャッシュされる値の数] はゼロに設定されるため、統合サービスではセッション中に値がキャッシュに格納されません。値をキャッシュに格納しない場合、統合サービスはセッション開始時にリポジトリにアクセスして現在の値を取得します。その後、統合サービスはシーケンスの値を生成します。統合サービスは、セッション終了時にリポジトリ内の現在の値を更新します。

[キャッシュされる値の数] にゼロより大きい値を設定すると、統合サービスではセッション中に値がキャッシュに格納されます。統合サービスは、セッションの開始時にリポジトリにアクセスして現在の値を取得し、設定された数の値をキャッシュに格納して、現在の値を更新します。キャッシュを使い果たした場合、統合サービスはリポジトリにアクセスして次の値のセットを取得し、現在の値を更新します。統合サービスは、セッション終了時にキャッシュ内に残っているすべての値を破棄します。

再利用不可能なシーケンスジェネレータトランスフォーメーションで [キャッシュされる値の数] をゼロより大きな値に設定すると、統合サービスがセッション実行中にリポジトリにアクセスする回数が増える可能性があります。また、キャッシュに格納されている未使用の値がセッションの最後で無視されるため、値がスキップされることがあります。

たとえば、シーケンスジェネレータトランスフォーメーションを、[Number of Cached Values] = 50、[Current Value] = 1、[Increment By] = 1 と設定します。統合サービスではセッション開始時にセッション用に 50 個の値がキャッシュされ、リポジトリ内の現在の値が 50 に更新されます。このとき、統合サービスは 1~39 の値をセッションで使用し、使用しなかった 40~49 の値は破棄します。統合サービスがセッションを再度実行するとき、リポジトリ内の現在の値（この場合は 50）を確認します。その後、次の 50 個の値をキャッシュに格納し、[Current Value] を 100 に更新します。ここでセッションの実行において PowerCenter Server が値 50~98 を使用したとします。2 つのセッションに対して生成された値は、1~39 と 50~98 です。

再利用可能なシーケンスジェネレータ

再利用可能なシーケンスジェネレータトランスフォーメーションを複数のセッションで使用していて、そのセッションを同時に実行する場合は、[Number of Cached Values] を使用して、各セッションが確実にシーケンス内の一意の値を受け取るようにします。再利用可能なシーケンスジェネレータの場合、[Number of Cached Values] はデフォルトで 1000 に設定されます。

複数のセッションが同じシーケンスジェネレータトランスフォーメーションを同時に使用すると、それぞれのセッションに対して同じ値を生成してしまう危険が伴います。これを回避するためには、[キャッシュされる値の数] を設定することにより、統合サービスで各セッション用に設定された数の値がキャッシュに格納されるようにします。

たとえば、再利用可能なシーケンスジェネレータトランスフォーメーションで、[Number of Cached Values] = 50、[Current Value] = 1、[Increment By] = 1 と設定したとします。2 つの異なるセッションはこのシーケンスジェネレータを使用し、それぞれが同時に実行されるようにスケジュール設定されたとします。統合サービスでは最初のセッション開始時にセッション用に 50 個の値がキャッシュされ、リポジトリ内の現在の値が 50 に更新されます。統合サービスは、セッションで 1~50 の値の使用を開始します。統合サービスが 2 番目のセッションを実行するとき、リポジトリ内の現在の値（この場合は 50）を確認します。その後、次の 50 個の値をキャッシュに格納し、[Current Value] を 100 に更新します。そして 2 番目のセッションで値 51~100 を使用します。どちらかのセッションがそのキャッシュに格納されている値をすべて使い果たすと、統合サービスは新しい値のセットをキャッシュに格納し、これらの値がシーケンスジェネレータに対して一意となるように現在の値を更新します。

再利用可能なシーケンスジェネレータトランスフォーメーションの場合、[キャッシュされる値の数] の値を小さくすることにより、無視される値を最小限にすることができます。ただしこの値は 1 より大きくなければなりません。[キャッシュされる値の数] の値を小さくすると、値をキャッシュに格納するために統合サービスがセッションの実行中にリポジトリにアクセスする回数が増える可能性があります。

リセット

再利用不可能なシーケンスジェネレータトランスフォーメーションで [リセット] を選択すると、統合サービスはセッションを開始するたびに元の開始値に基づいて値を生成します。[リセット] を選択しない場合、統合サービスは現在の値を更新して最後に生成された値に増分値を加算した値を反映し、次回シーケンスジェネレータトランスフォーメーションを使用する際に更新された値を使用します。

[リセット] プロパティを使用するように再利用不可能なシーケンスジェネレータトランスフォーメーションを設定すると、統合サービスはマッピングを実行するたびに元の開始値を使用します。使用しないように設定すると、統合サービスは現在の値をインクリメントし、次のマッピングの実行にその値を使用します。

例えば、1 から 1,000 までの値を増分値 1 で作成するようにシーケンスジェネレータトランスフォーメーションを設定します。リセットを選択して、開始値を 1 にリセットします。最初のセッションの実行中、統合サービスは 1~234 の値を生成します。統合サービスは、その後にマッピングを実行するたびに初期値 1 から始まる数字を再度生成します。

例えば、1 から 1,000 までの値を増分値 1、開始値 1 で作成するようにシーケンスジェネレータトランスフォーメーションを設定し、リセットを選択します。最初のマッピングの実行中、統合サービスは 1~234 の値を生成します。統合サービスは、その後にマッピングを実行するたびに開始値 1 から始まる数字を再度生成します。

リセットしない場合、統合サービスは最初の実行の終了時に現在の値を 235 に更新します。次回そのシーケンスジェネレータトランスフォーメーションを使用するとき、最初に生成される値は 235 になります。

注: [Reset] は、再利用可能なシーケンスジェネレータトランスフォーメーションに対しては無効になっています。

行順序の保持

トランスフォーメーションへの入力データの行順序を保持します。統合サービスが行順序を変更する可能性がある最適化を実行しないようにする場合に、このオプションを選択します。

統合サービスが最適化を実行すると、以前のマッピングで確立された順序が失われる場合があります。順序は、マッピングでソート済みのフラットファイルソース、ソート済みのリレーショナルソース、またはソートードトランスフォーメーションのいずれかを使用することで確立できます。行順序を保持するようにトランスフォーメーションを設定すると、統合サービスではマッピングの最適化が実行される際に、この設定が考慮されます。統合サービスは、行順序を保持できる場合には、トランスフォーメーションの最適化を実行します。最適化により行順序が変更される可能性がある場合には、統合サービスはトランスフォーメーションの最適化を実行しません。

シーケンスデータオブジェクト

シーケンスデータオブジェクトは、数値のシーケンスを作成して維持します。シーケンスジェネレータトランスフォーメーションは、シーケンスデータオブジェクトを使用してトランスフォーメーションの値を生成します。

再利用可能なシーケンスデータオブジェクトは、複数のシーケンスジェネレータトランスフォーメーションで使用できます。同じ統合サービスで実行される場合、同じシーケンスデータオブジェクトを使用するすべての

シーケンスジェネレータトランスフォーメーションは、同じ値シーケンスを使用します。再利用可能なシーケンスデータオブジェクトは、再利用不可能なシーケンスジェネレータトランスフォーメーションでも使用できます。再利用不可能なシーケンスデータオブジェクトは、再利用不可能なシーケンスジェネレータトランスフォーメーションで使用できます。

例えば、リレーショナルテーブルの同じプライマリキーフィールドに書き込む複数のマッピングを作成するとします。各マッピングは同じ再利用可能なシーケンスジェネレータトランスフォーメーションを使用し、そのトランスフォーメーションは同じ統合サービスで実行される同じ再利用可能なシーケンスデータオブジェクトを使用します。各マッピングは、プライマリキーフィールドに一意の値を書き込みます。

シーケンスデータオブジェクトの作成

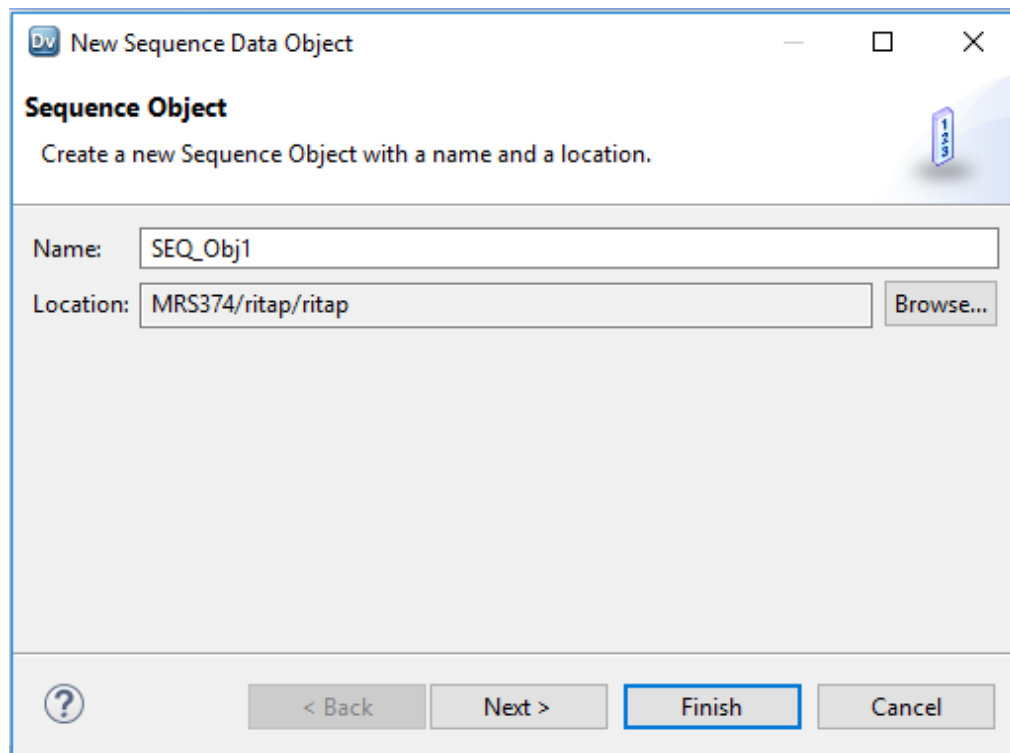
シーケンスデータオブジェクトを使用してシーケンスジェネレータトランスフォーメーションを作成するには、シーケンスデータオブジェクトを作成し、オブジェクトのプロパティを構成し、[シーケンスジェネレータトランスフォーメーション] ダイアログボックスでオブジェクトを選択します。

1. マッピングエディタで、マッピングパレットを下にスクロールして、シーケンスジェネレータトランスフォーメーションを見つけ、マッピングにドラッグします。

新しいトランスフォーメーションウィザードが開きます。

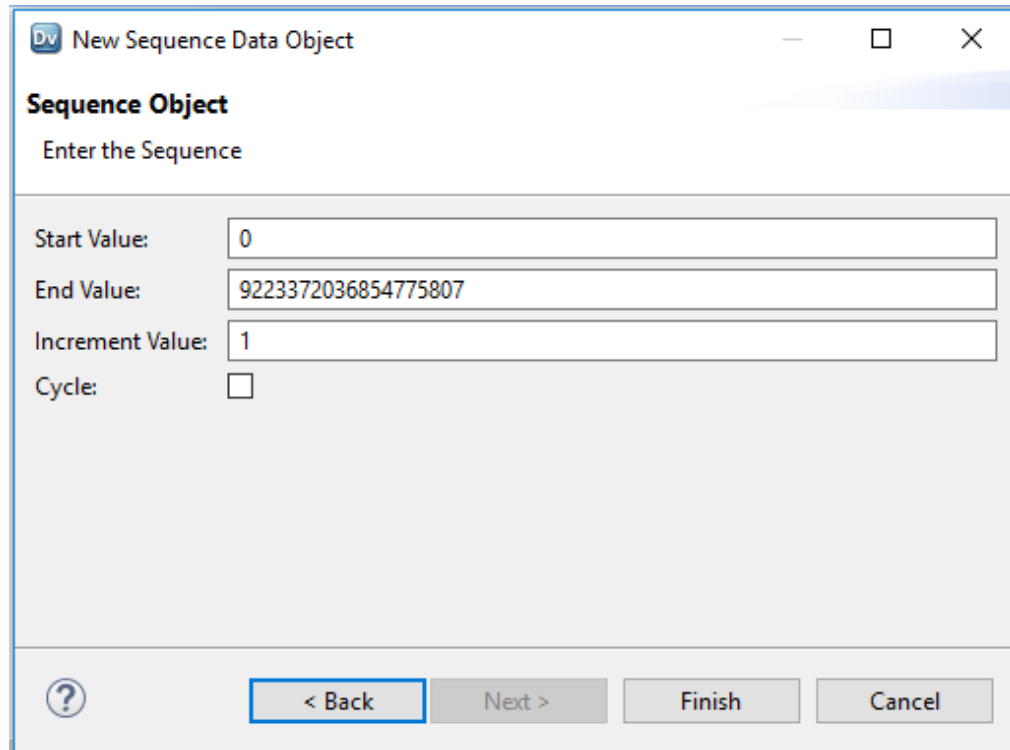
2. [新しいシーケンスデータオブジェクト] をクリックします。

新しいデータオブジェクトウィザードが開きます。



3. シーケンスデータオブジェクトの名前を入力します。
シーケンスデータオブジェクトの命名規則は、「SEQ_<データオブジェクト名>」です。
4. [次へ] をクリックして、シーケンスデータオブジェクトのプロパティを構成します。

このオブジェクトからシーケンスジェネレータ変換を作成する場合、変換ではデータオブジェクトに対して入力したプロパティを使用します。以下の図に、設定可能なプロパティを示します。



5. データオブジェクトのプロパティを構成した後、シーケンスデータオブジェクトを使用してシーケンスジェネレータ変換を作成できます。変換を作成するときは、シーケンスジェネレータ変換に名前を付け、**【既存のシーケンスオブジェクトを選択してください。】**を選択します。データオブジェクトに移動し、**【OK】** をクリックします。

シーケンスジェネレータ変換は、NEXTVAL 出力専用ポートを持つマッピングエディタに表示されます。NEXTVAL ポートをダウンストリームの変換またはターゲットに接続し、番号のシーケンスを生成できます。

シーケンスジェネレータ変換の作成

マッピングでシーケンスジェネレータ変換を使用するためには、まずそれをマッピングに追加します。次に、変換のプロパティを設定し、NEXTVAL または CURRVAL を 1 つまたは複数の変換に接続します。

シーケンスジェネレータ変換を作成するには：

1. Mapping Designer で、**【変換】** - **【作成】** をクリックします。Sequence Generator 変換を選択します。

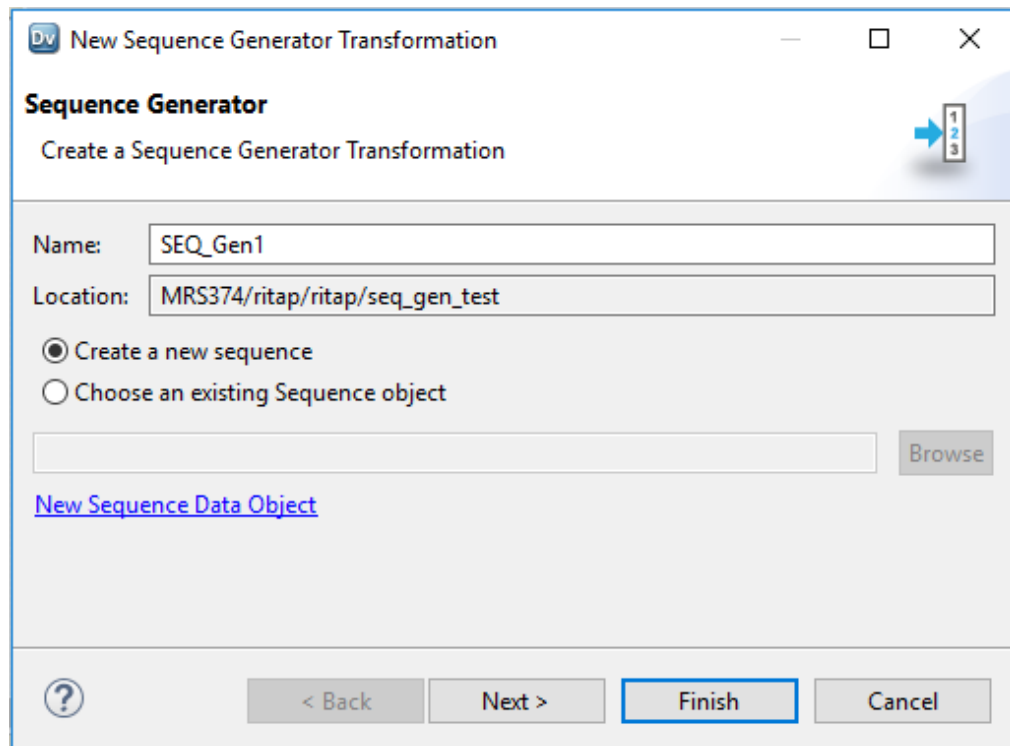
シーケンスジェネレータ変換の命名規則は、「SEQ_変換名」です。

2. Sequence Generator の名前を入力し、[作成] をクリックします。[完了] をクリックします。
Designer がシーケンスジェネレータトランスフォーメーションを作成します。
3. トランスフォーメーションのタイトルバーをダブルクリックします。
4. トランスフォーメーションの説明を入力します。
5. [プロパティ] タブを選択します。設定を入力します。
注: シーケンスジェネレータトランスフォーメーションのプロパティはセッションレベルで上書きすることができません。これにより、生成されるシーケンス値の整合性が保たれています。
6. [OK] をクリックします。
7. セッションの実行中に新しいシーケンスを生成するには、NEXTVAL ポートをマッピング内の 1 つ以上のトランスフォーメーションに接続します。
ほかのトランスフォーメーションの式の中で、NEXTVAL ポートまたは CURRVAL ポートを使用します。

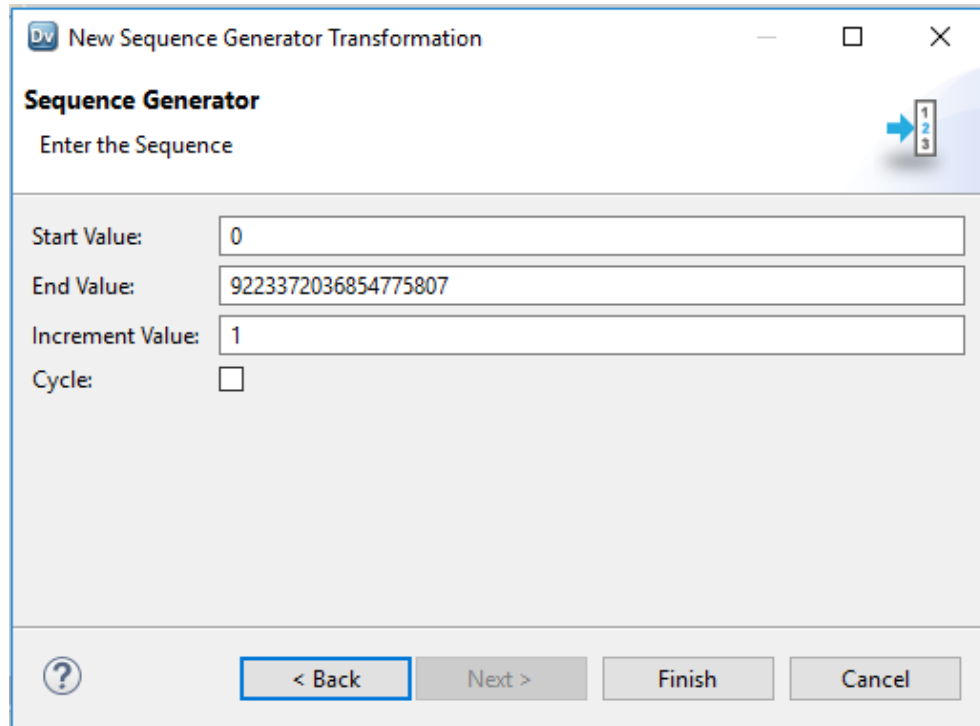
シーケンスジェネレータトランスフォーメーションの作成

マッピングでシーケンスジェネレータトランスフォーメーションを使用するためには、まずそれをマッピングに追加します。次に、トランスフォーメーションのプロパティを設定し、NEXTVAL を 1 つまたは複数のトランスフォーメーションに接続します。

1. マッピングエディタで、マッピングパレットを下にスクロールして、シーケンスジェネレータトランスフォーメーションを見つけ、マッピングにドラッグします。
新しいトランスフォーメーションウィザードが開きます。



2. シーケンスジェネレータトランスフォーメーションの名前を入力します。
シーケンスジェネレータトランスフォーメーションの命名規則は、「SEQ_<トランスフォーメーション名>」です。
3. 新しいシーケンスを作成するか、既存のシーケンスオブジェクトを使用するかを選択します。
 - 新しいシーケンスを作成するには、**【新しいシーケンスを作成】**を選択します。**【次へ】**をクリックして、シーケンスのプロパティを構成します。以下の図に、設定可能なプロパティを示します。



- 既存のシーケンスオブジェクトを使用するには、**【既存のシーケンスオブジェクトを選択してください。】**を選択します。シーケンスオブジェクトに移動し、**【OK】**をクリックします。

シーケンスジェネレータトランスフォーメーションは、NEXTVAL 出力専用ポートを持つマッピングエディタに表示されます。NEXTVAL ポートをダウンストリームのトランスフォーメーションまたはターゲットに接続し、番号のシーケンスを生成できます。

FAQ（よくある質問）

再利用不可能なシーケンスジェネレータトランスフォーメーションを再利用可能に変更できますか？

トランスフォーメーションを再利用可能にはできませんが、シーケンスデータオブジェクトを使用するようにトランスフォーメーションを変更することはできます。シーケンスデータオブジェクトは、シーケンスを使用するトランスフォーメーションの数に関係なくシーケンスの整合性を維持します。

再利用不可能なシーケンスジェネレータトランスフォーメーションをマップレットに配置できますか？

できません。マップレットは再利用可能なオブジェクトであるため、マップレット内のオブジェクトはすべて再利用可能である必要があります。代わりに再利用可能なシーケンスジェネレータトランスフォーメーションを使用します。

シーケンスジェネレータトランスフォーメーション 非ネイティブ環境で

非ネイティブ環境でのシーケンスジェネレータトランスフォーメーション処理は、そのトランスフォーメーションを実行するエンジンに依存します。

以下の非ネイティブランタイムエンジンでのサポートを考慮します。

- Blaze エンジン。制限付きのサポート。
- Spark エンジン。バッチマッピングで制限付きでサポートされます。ストリーミングマッピングではサポートされていません。
- Databricks Spark エンジン。サポートしません。

Blaze エンジンでのシーケンスジェネレータトランスフォーメーション

シーケンスジェネレータトランスフォーメーションを使用したマッピングでは、次の条件が当てはまる場合に大量のリソースが消費されます。

- トランスフォーメーションの **【行順序を保持】** プロパティを *true* に設定した。
- マッピングが単一パーティションで実行されている。

Spark エンジンでのシーケンスジェネレータトランスフォーメーション

シーケンスジェネレータトランスフォーメーションは、出力データの行順序を保持しません。トランスフォーメーションの **【行順序を保持】** プロパティを有効にした場合、データ統合サービスはこのプロパティを無視します。

第 24 章

ソータトランスフォーメーション

この章では、以下の項目について説明します。

- [ソータトランスフォーメーションの概要, 402 ページ](#)
- [データのソート, 402 ページ](#)
- [ソータトランスフォーメーションのプロパティ, 403 ページ](#)
- [ソータトランスフォーメーションの作成, 405 ページ](#)

ソータトランスフォーメーションの概要

ソータトランスフォーメーションを使用して、データをソートできます。指定されたソートキーに基づいて、昇順または降順にデータをソートできます。また、大文字小文字を区別したソートや、出力行を重複しないようにするかどうかもソータトランスフォーメーションで設定できます。ソータトランスフォーメーションはアクティブなトランスフォーメーションです。これはデータフローに接続する必要があります。

リレーショナルファイルソースまたはフラットファイルソースのデータをソートできます。また、ソート済み入力を使用するように設定されたアグリゲータトランスフォーメーションに渡されるデータをソートするようにソータトランスフォーメーションを設定することもできます。

マッピングでソータトランスフォーメーションを作成する場合、1 つまたは複数のポートをソートキーとして指定し、昇順または降順でソートを行うように各ソートキーポートを設定します。また、Integration Service がすべてのソートキーポートに適用するソート基準と、ソート操作を実行するために割り当てるシステムリソースを設定します。

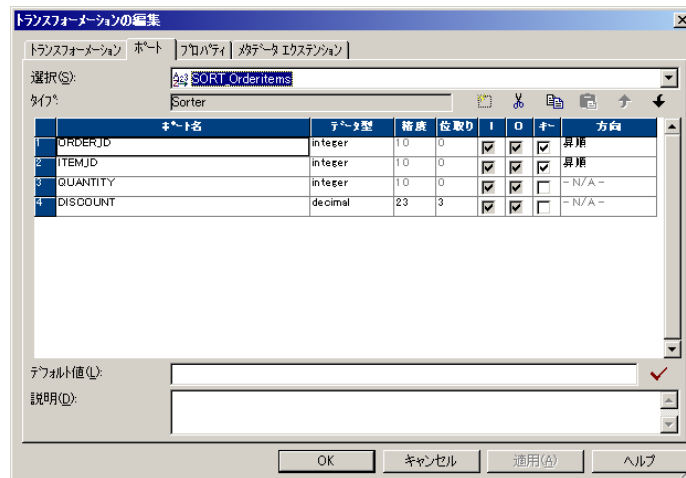
データのソート

ソータトランスフォーメーションに含まれるのは入出力ポートのみです。ソータトランスフォーメーションを通るデータはすべてソートキーに基づいてソートされます。ソートキーとは、ソート基準として使用する 1 つまたは複数のポートのことです。

ソートキーの一部として複数のポートを指定できます。ソートキーに対して複数のポートを指定した場合、Integration Service は各ポートを順番にソートします。ポートが [ポート] タブに表示される順番がソート操作の順番となります。ソータトランスフォーメーションでは、連続した各ソートキーポートを通るデータは前のポートで既に一度ソートされたデータとして扱います。

セッションの実行時、Integration Service はセッションプロパティで指定されたソート順でデータをソートします。ソート順は特殊文字や特殊記号のソート基準を決定します。

以下の図に、ソータランスフォーメーションで注文 ID および商品 ID に基づいて昇順でデータをソートする場合の「ポート」タブの設定を示します。



セッションの実行時、Integration Service は下記の行をソータランスフォーメーションに渡します。

ORDER_ID	ITEM_ID	QUANTITY	DISCOUNT
45	123456	3	3.04
45	456789	2	12.02
43	000246	6	34.55
41	000468	5	.56

データをソートした後で、Integration Service は下記の行をソータランスフォーメーションから渡します。

ORDER_ID	ITEM_ID	QUANTITY	DISCOUNT
41	000468	5	.56
43	000246	6	34.55
45	123456	3	3.04
45	456789	2	12.02

ソータランスフォーメーションのプロパティ

ソータランスフォーメーションには、その他のソート基準を指定するためのプロパティがいくつかあります。Integration Service は、これらの基準をすべてのソートキーポートに適用します。また、Integration Service でのデータのソート時に割り当てられるシステムリソースも、ソータランスフォーメーションのプロパティによって決定されます。

ソータキャッシュサイズ

統合サービスは [ソータキャッシュサイズ] プロパティを使用して、ソート操作を実行するために割り当てられるメモリの最大量を決定します。統合サービスは、ソート操作を実行する前に、すべての入力データをソータトランスフォーメーションに渡します。キャッシュの数値を使用できます。パラメータファイルのキャッシュ値を使用するか、統合サービスを設定し、自動設定を使用してキャッシュサイズを設定します。キャッシュサイズが決定されるように統合サービスを設定した場合、キャッシュに割り当てられる最大メモリ量も設定できます。設定したセッションキャッシュの合計サイズが 2GB (2,147,483,648 バイト) 以上の場合は、そのセッションを 64 ビットの統合サービスで実行します。

統合サービスは、ソート操作を開始する前に、ソータキャッシュサイズで設定されたメモリ量を割り当てます。統合サービスがパーティション化されたセッションを実行する場合、各パーティションに対して指定された量のソータキャッシュメモリを割り当てます。

十分なメモリ量を割り当てられない場合、統合サービスはセッションに失敗します。最高のパフォーマンスのためには、統合サービスマシンの利用可能な物理 RAM 量以下の値になるように、ソータキャッシュサイズを設定します。ソータトランスフォーメーションを使用してデータをソートするには、少なくとも 16MB (16,777,216 バイト) の物理メモリを割り当てます。デフォルトでは、ソータキャッシュサイズは 16,777,216 バイトに設定されています。

入力されるデータの量がソータキャッシュサイズよりも大きい場合、統合サービスはデータを一時的にソータトランスフォーメーションのワークディレクトリに保存します。統合サービスがデータをワークディレクトリに保存する場合、最低でも入力されるデータの量の 2 倍のディスク領域が必要です。入力されるデータの量がソータキャッシュサイズよりはるかに大きい場合、統合サービスはワークディレクトリで利用可能なディスク領域の 2 倍の量よりさらに多くの量を必要とする可能性があります。

ソータトランスフォーメーションのトレースレベルを [ノーマル] に設定すると、統合サービスは、ソータトランスフォーメーションが使用するメモリ量もセッションログに書き込みます。

大文字小文字の区別

[大文字小文字の区別] プロパティは、Integration Service がデータをソートする際に大文字と小文字を区別するかどうかを決定します。[大文字小文字の区別] プロパティを有効にすると、Integration Service はソートの際に大文字を小文字より優先させます。

作業ディレクトリ

Integration Service がデータをソートするときに一時ファイルの作成に使用するワークディレクトリを指定する必要があります。Integration Service はデータをソートした後で、一時ファイルを削除します。Integration Service マシンの任意のディレクトリをワークディレクトリとして指定できます。デフォルトでは、Integration Service は \$PMTempDir プロセス変数に指定されている値を使用します。

ソータトランスフォーメーションを使ってセッションをパーティション化する場合、パイプライン内のパーティションごとに異なるワークディレクトリを指定できます。セッションのパフォーマンスを向上させるためには、Integration Service システム上の物理的に異なるディスク上でワークディレクトリを指定します。

重複しない出力行

出力行を重複しないものとして扱うようにソータトランスフォーメーションを設定できます。出力行が重複しないようにソータトランスフォーメーションを設定した場合、Mapping Designer はすべてのポートをソートキーの一部として設定します。Integration Service は、ソート操作時に比較した重複行を破棄します。

トレースレベル

ソータトランスフォーメーションのトレースレベルを設定すると、Integration Service がセッションログに書き込むソータエラーおよびステータスメッセージの数とタイプを制御できます。 [ノーマル] トレースレベルでは、Integration Service はソータトランスフォーメーションに渡された行のサイズ、およびソータトランスフォーメーションがソート操作に割り当てたメモリ量を書き込みます。 また、Integration Service はソータトランスフォーメーションに最初と最後の入力行を渡した時間と日付も書き込みます。

ソータトランスフォーメーションのトレースレベルを [Verbose Data] に設定した場合、Integration Service は、ソータトランスフォーメーションがパイプラインの中の次のトランスフォーメーションにすべてのデータを渡し終えた時間を書き込みます。 また、Integration Service は、ソータトランスフォーメーションがメモリリソースを解放し、ワークディレクトリから一時ファイルを削除した時間もセッションログに書き込みます。

NULL を低位として処理

ソータトランスフォーメーションの NULL 値の扱い方を設定できます。Integration Service がソート操作を行う際に、NULL 値を他のすべての値より下位として処理するようにする場合は、このプロパティを有効にします。 Integration Service が NULL 値を他の値より上位として処理するようにする場合は、このオプションを無効にします。

トランスフォーメーション範囲

トランスフォーメーション範囲は、Integration Service がトランスフォーメーションロジックを入力データに適用する方法を示します。

- **トランザクション。** トランスフォーメーションロジックをトランザクションのすべての行に適用します。データの行が同一トランザクション内のすべての行に依存し、他のトランザクションの行には依存していない場合は、[トランザクション] を選択します。
- **すべての入力。** トランスフォーメーションロジックをすべての入力データに適用します。 [すべての入力] を選択すると、PowerCenter は入力トランザクションの境界を削除します。データの行がソース内のすべての行に依存している場合は、[すべての入力] を選択します。

ソータトランスフォーメーションの作成

ソータトランスフォーメーションをマッピングに追加するには、以下の手順に従ってください。

1. Mapping Designer で、[トランスフォーメーション] - [作成] をクリックします。ソータトランスフォーメーションを選択します。
ソータトランスフォーメーションの命名規則は、「SRT_トランスフォーメーション名」です。トランスフォーメーションの説明を入力します。この説明はトランスフォーメーションの処理内容がわかるようにするためのもので、Repository Manager 内に表示されます。
2. ソータの名前を入力して、[作成] をクリックします。
Designer がソータトランスフォーメーションを作成します。
3. [完了] をクリックします。
4. ソートしたいポートをソータトランスフォーメーションヘドラッグします。
Designer は、対象ポートのそれぞれについて入出力ポートを作成します。
5. トランスフォーメーションのタイトルバーをダブルクリックして [トランスフォーメーションの編集] ダイアログボックスを開きます。

6. [ポート] タブを選択します。
7. ソートキーとして使用するポートを選択します。
8. ソートキーの一部として選択したポートごとに、Integration Service でデータを昇順または降順のどちらでソートするのかを指定します。
9. [プロパティ] タブを選択します。ソータトランスフォーメーションのプロパティを変更します。
10. [メタデータエクステンション] タブを選択します。ソータトランスフォーメーションのメタデータエクステンションを作成または編集します。
11. [OK] をクリックします。

第 25 章

ソース修飾子トランスフォーメーション

この章では、以下の項目について説明します。

- [ソース修飾子トランスフォーメーションの概要, 407 ページ](#)
- [ソース修飾子トランスフォーメーションのプロパティ, 409 ページ](#)
- [デフォルトクエリ, 410 ページ](#)
- [ソースデータの結合, 412 ページ](#)
- [SQL クエリの追加, 414 ページ](#)
- [ユーザー定義ジョインの入力, 416 ページ](#)
- [アウタージョインのサポート, 416 ページ](#)
- [ソースフィルタの入力, 423 ページ](#)
- [ソート済みポートの使用, 424 ページ](#)
- [個別に選択, 425 ページ](#)
- [セッション実行前/実行後の SQL コマンドの追加, 426 ページ](#)
- [ソース修飾子トランスフォーメーションの作成, 427 ページ](#)
- [ソース修飾子トランスフォーメーションのトラブルシューティング, 428 ページ](#)

ソース修飾子トランスフォーメーションの概要

リレーショナルソース定義またはフラットファイルソース定義をマッピングに追加する場合は、追加するソース定義をソース修飾子トランスフォーメーションに接続する必要があります。ソース修飾子トランスフォーメーションは、Integration Service でのセッション実行時に読み込まれる行を表します。ソース修飾子トランスフォーメーションはアクティブなトランスフォーメーションです。

以下のタスクを完了するには、ソース修飾子トランスフォーメーションを使用します。

- **同じソースデータベースから取得したデータを結合する。**1つのソース修飾子トランスフォーメーションにソースをリンクすることで、プライマリキーと外部キーの関係を使用して複数のテーブルを結合できます。
- **Integration Service がソースデータを読み取るときに行をフィルタリングする。**フィルタ条件を含めると、Integration Service はデフォルトクエリに WHERE 句を追加します。
- **デフォルトのインナージョインの代わりにアウタージョインを指定する。**ユーザー定義ジョインを含めると、Integration Service は SQL クエリのメタデータで指定された結合情報を置き換えます。

- **ソート済みポートを指定する。**ソート済みポートに対して数値を指定すると、Integration Service は ORDER BY 句をデフォルトの SQL クエリに追加します。
- **ソースから重複しない値だけを選択する。**[個別に選択] を選択すると、Integration Service は SELECT DISTINCT 文をデフォルトの SQL クエリに追加します。
- **特殊な SELECT 文を発行するカスタムクエリを作成して、Integration Service にソースデータの読み込みを行わせる。**例えば、集計計算を行うカスタムクエリを使用することができます。

トランスフォーメーションデータタイプ

ソース修飾子トランスフォーメーションは、トランスフォーメーションのデータタイプを表示します。トランスフォーメーションのデータタイプは、Integration Service でデータを読み込むときにソースデータベースがデータをどのようにバインドしているかを示します。ソース修飾子トランスフォーメーションでデータタイプを変更してはいけません。ソース定義内のデータタイプとソース修飾子トランスフォーメーション内のデータタイプが一致しない場合、マッピングを保存するときに Designer はそのマッピングを無効とします。

ターゲットのロード順

ターゲットロード順は、マッピング内のソース修飾子トランスフォーメーションに基づいて指定します。複数のソース修飾子トランスフォーメーションを複数のターゲットに接続している場合、Integration Service がデータをターゲットにロードする順番を指定できます。

1つのソース修飾子トランスフォーメーションから複数のターゲットにデータを提供している場合は、セッション内で制約ベースのロードを有効にすることにより、Integration Service がターゲットテーブルのプライマリキーと外部キーの関係に基づいてデータをロードするように設定できます。

例えば、1つのマッピングに3つのパイプラインが含まれるとします。各パイプラインには、マッピング内で互いにリンクするソース修飾子、トランスフォーメーション、およびフラットファイルターゲットが含まれています。統合サービスで各ターゲットロード順序グループが順次処理されるように、マッピングのターゲットロード順序を構成できます。

日時の値

SQL クエリで日時の値または日時のパラメータや変数を使用するときは、日付の形式をソースで使用する形式に変更します。Integration Service は、日時の値を SQL クエリの文字列としてソースシステムに渡します。Integration Service は、ソースデータベースに応じて日時の値を文字列に変換します。

以下の表では、データベースタイプごとの日時形式について説明します。

ソース	日付の形式
DB2	YYYY-MM-DD-HH24:MI:SS
Informix	YYYY-MM-DD HH24:MI:SS
Microsoft SQL Server	YYYY/MM/DD HH24:MI:SS
ODBC	YYYY-MM-DD HH24:MI:SS
Oracle	YYYY/MM/DD HH24:MI:SS
Sybase	YYYY/MM/DD HH24:MI:SS
Teradata	YYYY-MM-DD HH24:MI:SS

データベースによっては、一重引用符やデータベース固有の関数など追加の文字によって日時値を識別しなければならない場合があります。たとえば、\$\$\$SessStartTime 値を Oracle ソース用に変換するには、次の Oracle 関数を SQL 上書きで使

to_date ('\$\$\$SessStartTime', 'mm/dd/yyyy hh24:mi:ss')

Informix の場合、次の Informix 関数を SQL オーバーライドで使用して、\$\$\$SessStartTime 値を変換します。

DATETIME (\$\$\$SessStartTime) YEAR TO SECOND

データベース固有の関数の詳細については、データベースのマニュアルを参照してください。

パラメータおよび変数

パラメータおよび変数は、SQL クエリ、ユーザー定義ジョイン、ソースフィルタ、およびソース修飾子トランスフォーメーションのセッション実行前/実行後 SQL コマンドで使用できます。パラメータファイルで定義可能なパラメータまたは変数タイプを使用します。パラメータまたは変数は、SQL 文の中に入力することも、あるいは SQL クエリとして使用することもできます。たとえば、セッションパラメータ\$ParamMyQuery は、SQL クエリとして使用することも、またパラメータファイル内の SQL 文に設定することもできます。

Integration Service は、まず SQL クエリを生成し、各パラメータまたは変数を展開します。各マッピングパラメータ、マッピング変数、およびワークフロー変数をそれぞれ開始値で置き換えます。次に、ソースデータベースのクエリーを実行します。

ソース修飾子トランスフォーメーションで文字列マッピングパラメータまたは変数を使用する場合、ソースシステムに適した文字列識別子を使用します。ほとんどのデータベースは、文字列識別子として一重引用符を使用します。たとえば、Microsoft SQL Server データベーステーブルのソースフィルタで文字列パラメータ\$IPAddress を使用するには、パラメータを一重引用符で囲んで'\$IPAddress'とします。

日時マッピングパラメータまたは変数を使用するとき、あるいはビルトイン変数\$\$\$SessStartTime を使用するときは、日付フォーマットをソースで使用されているフォーマットに変更します。Integration Service は、日時の値を SQL クエリの文字列としてソースシステムに渡します。

ヒント: ソースが使用する日時パラメータまたは変数の形式を確実に一致させるには、SQL クエリを検証します。

ソース修飾子トランスフォーメーションのプロパティ

以下のソース修飾子トランスフォーメーションのプロパティは、[プロパティ] タブで設定します。

オプション	説明
SQL クエリ	Integration Service がソース修飾子トランスフォーメーション内に表示されているソースからデータを読み込むために使用するデフォルトのクエリの代わりに、カスタムのクエリを定義します。ユーザー作成のクエリはユーザー作成のジョインやソースフィルタのエントリを上書きします。
User Defined Join	同じソース修飾子トランスフォーメーション内に表示されている複数のソースのデータを結合するための条件を指定します。
ソースフィルタ	行に対するクエリを実行するときに Integration Service が適用するフィルタ条件を指定します。

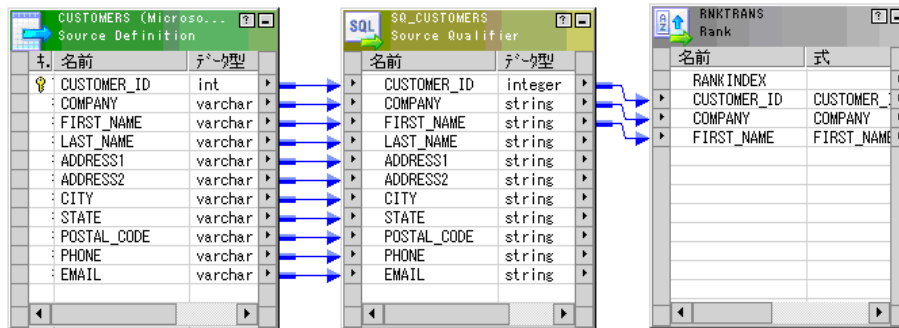
オプション	説明
ソートするポート数	リレーショナルソースから取り込む行をソートするときに使用するカラムの数を示します。このオプションを選択すると、Integration Service はソース行を読み込むときに ORDER BY をデフォルトクエリに追加します。ORDER BY には、指定された数のポートが、トランスフォーメーションの最初のポートから順に取り込まれます。選択した場合、データベースのソート順は、セッションのソート順に一致しなければなりません。
トレースレベル	このトランスフォーメーションを含むセッションを実行したときにセッションログに記録される情報の詳細度を設定します。
個別に選択	一意な行だけを選択したい場合に指定します。このオプションを選択すると、Integration Service に SELECT DISTINCT 文が含まれます。
実行前 SQL	Integration Service がソースを読み込む前にソースデータベースに対して実行されるセッション実行前 SQL コマンド。
実行後 SQL	Integration Service がターゲットに書き込んだ後にソースデータベースに対して実行されるセッション実行後 SQL コマンド。
出力が確定的かどうか	入力データがセッション間で一貫しているときに、セッション間で変わらないリレーショナルソースまたはトランスフォーメーション出力。このプロパティを設定するときに、パイプラインのトランスフォーメーションが常に繰り返し可能なデータを生成する場合は、Integration Service はリカバリのためにソースデータをステージングしません。
出力が再現可能	入力データの順序が一貫しているとき、セッション間で順序が変わらないリレーショナルソースまたはトランスフォーメーション出力。出力が決定的で繰り返し可能な場合、Integration Service はリカバリのためにソースデータをステージングしません。

警告: トランスフォーメーションを繰り返し可能で一意に定まるものとして設定する場合は、データが繰り返し可能で一意に定まることを保証する必要があります。セッションとリカバリで同じデータが生成されないトランスフォーメーションを使用してセッションをリカバリしようとする、リカバリプロセスを実行した結果、データが破損する可能性があります。

デフォルトクエリ

リレーショナルソースの場合、Integration Service は、セッション実行時にそれぞれのソース修飾子トランスフォーメーションに対してクエリを生成します。デフォルトのクエリは、マッピング内で使用されている各ソースカラムに対する SELECT 文です。つまり、Integration Service は他のトランスフォーメーションに接続されているカラムだけを読み込みます。

以下の図は、ソース修飾子トランスフォーメーションに接続されている単一のソース定義を示しています。多くのソース定義カラムがありますが、3つだけが最後のトランスフォーメーションに接続されています。



このソース定義内には多くのカラムがありますが、ここで他のトランスフォーメーションに接続されているカラムは3つだけです。この場合、Integration Service はこの3つのカラムだけを選択するデフォルトクエリを生成します。

```
SELECT CUSTOMERS.CUSTOMER_ID, CUSTOMERS.COMPANY, CUSTOMERS.FIRST_NAME
FROM CUSTOMERS
```

テーブル名またはカラム名にデータベース予約語が含まれる場合、予約語を含む reswords.txt ファイルを作成して管理できます。Integration Service はセッションを初期化する際に、Integration Service がインストールされたディレクトリ上の reswords.txt ファイルを探します。ファイルが存在する場合は、Integration Service はデータベースに対して SQL 文を実行する際に、一致した予約語を引用符で囲みます。SQL を上書きする場合は、すべての予約語は引用符で囲む必要があります。

デフォルトクエリを生成する時、Designer は下記の文字を含むテーブル名とフィールド名の場合は、テーブル名とフィールド名を二重引用符で区切ります。

```
/ + - = ~ ` ! % ^ & * ( ) [ ] { } ' ; ? , < > \ | <space>
```

デフォルトのクエリの表示

デフォルトクエリはソース修飾子トランスフォーメーションで表示することができます。

デフォルトクエリを表示するには：

1. 「プロパティ」タブで「SQL クエリ」を選択します。

SQL エディタに、Integration Service でソースデータの選択に使用されるデフォルトのクエリが表示されます。

2. 「SQL 文の生成」をクリックします。

3. 「キャンセル」をクリックして終了します。

SQL クエリをキャンセルしないと、Integration Service はカスタムの SQL クエリでデフォルトのクエリを上書きします。

ソースデータベースへ接続してはいけません。デフォルトクエリを上書きする SQL クエリを入力したときだけソースデータベースに接続します。

デフォルトクエリを生成する場合は、それに先立ってソース修飾子トランスフォーメーション内のカラムをほかのトランスフォーメーションまたはターゲットへ接続しなければなりません。

デフォルトのクエリの上書き

トランスフォーメーションプロパティのデフォルト設定を変更することによって、ソース修飾子トランスフォーメーションのデフォルトクエリを変更または上書きすることができます。選択されたポートのリストやク

エリーに表示される順序を変更しないでください。このリストは、接続されたトランスフォーメーションの出力ポートと一致する必要があります。

トランスフォーメーションプロパティを編集すると、ソース修飾子トランスフォーメーションはこれらの設定をデフォルトクエリーに含めます。ただし、SQL クエリーを入力した場合、Integration Service は定義済み SQL 文のみを使用します。SQL クエリーはソース修飾子トランスフォーメーションの [ユーザ定義ジョイン]、[ソースフィルタ]、[ソートするポート数] および [個別に選択] 設定を上書きします。

注: デフォルト SQL クエリーを上書きする場合、データベース予約語はすべて引用符で囲む必要があります。

ソースデータの結合

1 つのソース修飾子トランスフォーメーションを使用して、複数のリレーショナルテーブルからのデータを結合します。これらのテーブルは、同じインスタンスまたはデータベースサーバーからアクセスできなければなりません。

マッピングが関連するリレーショナルソースを使用する場合、両方のソースを 1 つのソース修飾子トランスフォーメーションで結合することができます。セッションの実行中に、ソースデータベースは結合を実行してから Integration Service にデータを渡します。これによって、ソーステーブルにインデックスが付けられたときにパフォーマンスが向上します。

ヒント: 異種ソースやフラットファイルを結合する場合は、ジョイナトランスフォーメーションを使用します。

デフォルトジョイン

1 つのソース修飾子トランスフォーメーション内で関連テーブルを結合する場合、Integration Service では各テーブル内の関連キーに基づいてテーブルが結合されます。

このデフォルトジョインは、WHERE 句に次の構文を使用するインナージョインと等価です。

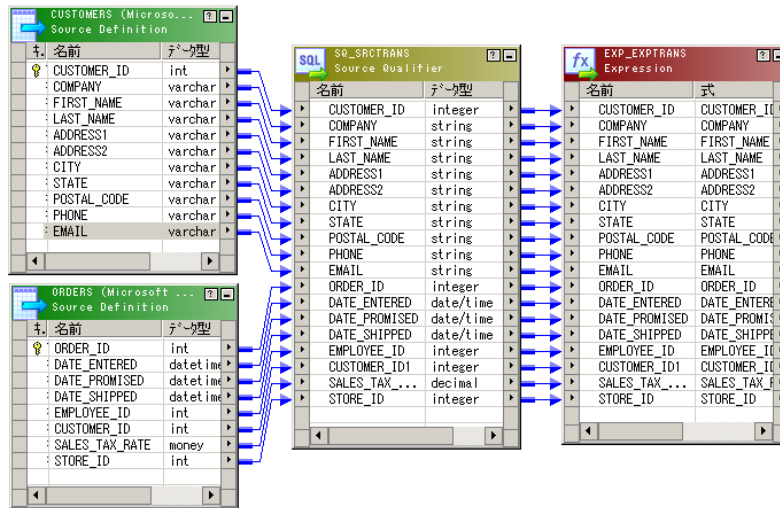
```
Source1.column_name = Source2.column_name
```

デフォルトジョインで使用するカラムに対しては次の要件があります。

- プライマリキーと外部キーの関係
- 一致するデータタイプ

たとえばある月におけるすべての注文をその注文番号、注文数量、顧客名を含めて調べたいとします。ここで、ORDERS テーブルには注文番号と注文数量のデータがありますが、顧客名のデータが入っていないとします。顧客名を取得するためには、ORDERS テーブルと CUSTOMERS テーブルを結合しなければなりません。両方のテーブルにはカスタマ ID が入っているため、この 2 つのテーブルを 1 つのソース修飾子トランスフォーメーションで結合することができます。

以下の図に、1つのソース修飾子トランスフォーメーションを使用して2つのテーブルを結合する例を示します。



複数のテーブルを取り込むと、Integration Service はマッピング内で使用されているすべてのカラムに対する SELECT 文を生成します。この場合の SELECT 文は次のようになります。

```
SELECT CUSTOMERS.CUSTOMER_ID, CUSTOMERS.COMPANY, CUSTOMERS.FIRST_NAME, CUSTOMERS.LAST_NAME,
CUSTOMERS.ADDRESS1, CUSTOMERS.ADDRESS2, CUSTOMERS.CITY, CUSTOMERS.STATE, CUSTOMERS.POSTAL_CODE,
CUSTOMERS.PHONE, CUSTOMERS.EMAIL, ORDERS.ORDER_ID, ORDERS.DATE_ENTERED, ORDERS.DATE_PROMISED,
ORDERS.DATE_SHIPPED, ORDERS.EMPLOYEE_ID, ORDERS.CUSTOMER_ID, ORDERS.SALES_TAX_RATE, ORDERS.STORE_ID
FROM CUSTOMERS, ORDERS
WHERE CUSTOMERS.CUSTOMER_ID=ORDERS.CUSTOMER_ID
```

この WHERE 句は、ORDERS テーブルと CUSTOMER テーブルの CUSTOMER_ID を取り込む等価結合です。

ユーザー作成のジョイン

デフォルトジョインを上書きする必要がある場合は、ユーザー作成のクエリ内で結合を指定する WHERE 句の内容を入力することができます。クエリがアウトジョインを実行すると、Integration Service ではデータベース構文に応じて WHERE 句または FROM 句のどちらかにジョイン構文が挿入されます。

次のような場合は、デフォルトジョインを上書きする必要があります。

- カラムがプライマリキーと外部キーの関係を持たない。
- 結合に使用するカラムのデータタイプが一致しない。
- アウタージョインのような、別の種類の結合を指定したい。

異種データのジョイン

異種データを結合する場合は、ジョイナトランスフォーメーションを使用します。下記の種類のソースを結合する必要がある場合にジョイナトランスフォーメーションを使用します。

- 異なるソースデータベースのデータの結合
- 異なるフラットファイルシステムのデータの結合
- リレーショナルソースとフラットファイルの結合

キー関係の作成

ソース修飾子トランスフォーメーションでは、テーブルがプライマリキーと外部キーの関係を持つ場合にのみ、テーブルを結合することができます。ただし異なるテーブルの対応するカラムをリンクすることによって、Source Analyzer でプライマリキーと外部キーの関係を作成することができます。これらのカラムはキーである必要はありませんが、それぞれのテーブルのインデックスに含まれている必要があります。

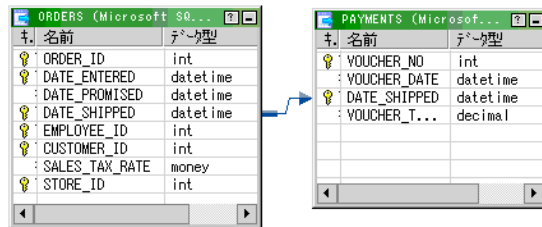
ヒント: ソーステーブルに 1,000 行を超える行が含まれている場合は、プライマリキーと外部キーにインデックスを付けることによってパフォーマンスを向上させることができます。ソーステーブルの行が 1,000 行に満たない場合は、プライマリキーと外部キーにインデックスを付けるとパフォーマンスが低下する可能性があります。

たとえば、小売り店チェーンの本部が、注文に基づいて領収済みの支払いを抽出したいとします。ORDERS テーブルと PAYMENTS テーブルは、プライマリキーと外部キーを共有していません。しかし、どちらのテーブルにも DATE_SHIPPED カラムがあります。この場合、Source Analyzer でプライマリキーと外部キーの関係をメタデータ内に作成することができます。

ここで、この 2 つのテーブルがリンクされていないことに注意してください。したがって、Designer は DATE_SHIPPED カラムについての関係を認識することができません。

そこで DATE_SHIPPED カラムをリンクすることによって、ORDERS テーブルと PAYMENTS テーブルの間に関係を作成します。Designer は、ORDERS テーブルと PAYMENTS テーブルの定義内の DATE_SHIPPED カラムに、プライマリキーと外部キーを追加します。

以下の図に、2 つのテーブル間の DATE_SHIPPED の関係を示します。



カラムを接続しないと、Designer は関係を認識しません。

プライマリキーと外部キーの関係はメタデータにのみ存在します。SQL を作成したりソーステーブルを変更したりする必要はありません。

キー関係が存在すれば、ソース修飾子トランスフォーメーションを使用して 2 つのテーブルを結合できます。デフォルトジョインは DATE_SHIPPED に基づいて行われます。

SQL クエリの追加

ソース修飾子トランスフォーメーションには、デフォルトクエリーをオーバーライドする [SQL クエリ] オプションがあります。ソースデータベースでサポートされている SQL 文を入力できます。クエリーを入力する前に、マッピングで使用したい入力ポートおよび出力ポートをすべて接続します。

SQL クエリーを編集する場合、デフォルトクエリーを生成して編集できます。Designer はデフォルトクエリーを生成すると、フィルタやソート済みポートの数など、設定されたほかのオプションをすべて組み込みます。編集したクエリーは、トランスフォーメーションでそれ以降設定するほかのオプションをすべてオーバーライドします。

パラメータまたは変数は、その一方を SQL クエリーとして使用することも、あるいはその両方をクエリー内に含めることもできます。文字列のマッピングパラメータまたは変数を含める場合は、ソースシステムに適した文

字列識別子を使用します。ほとんどのデータベースでは、文字列パラメータまたは変数の名前を一重引用符で囲む必要があります。

SQL クエリに日時の値または日時のマッピングパラメータや変数を含めるときは、日付フォーマットをソースで使用するフォーマットに変更します。統合サービスは、ソースシステムに応じて日時の値を文字列に変換します。

カスタム SQL クエリを入力する場合は、以下のルールとガイドラインに従ってください。

- SELECT 文はポート名をトランスフォーメーションに表示される順序でリストする必要があります。
- ソースが Microsoft SQL Server の場合、クエリ内の SELECT 文の列数がソース修飾子トランスフォーメーション内のポート数と一致しなければなりません。一致しない場合、次のエラーにより、セッションが失敗します: SQL Error [FnName: Fetch Optimize -- [Informatica][ODBC SQL Server Wire Protocol driver] Number of bound columns exceeds the number of result columns.]。

プッシュダウンの最適化が設定されたセッションに対するデフォルトの SQL クエリをオーバーライドすると、統合サービスは SQL オーバーライドを表すビューを作成します。次に、このビューに対して SQL クエリを実行し、トランスフォーメーションロジックをデータベースへプッシュします。

SQL クエリを編集する場合、データベース予約語はすべて引用符で囲む必要があります。

1. Source Qualifier トランスフォーメーションを開き、[プロパティ] タブをクリックします。
2. [SQL Query] フィールドで [式の編集] をクリックします。
[SQL エディタ] ダイアログボックスが表示されます。
3. [SQL 文の生成] をクリックします。

Designer は、ソース修飾子に取り込まれるすべてのソースの行に対して問い合わせを行う時に生成するデフォルトクエリーを表示します。

4. デフォルトクエリーが表示された部分にクエリーを入力します。

カラム名はすべて、それを含んでいるテーブル、ビュー、またはシノニムの名前で修飾されなければなりません。たとえば ORDERS テーブルから ORDER_ID カラムを取り込みたい場合は、「ORDERS.ORDER_ID」と入力します。[ポート] ウィンドウに表示されるカラム名をダブルクリックすれば、カラムの名前をタイプする手間が省けます。

パラメータまたは変数は、その一方をクエリーとして使用することも、あるいはその両方をクエリー内に含めることもできます。

文字列マッピングパラメータおよび変数を文字列識別子で囲みます。必要に応じて、日時マッピングパラメータおよび変数の日付フォーマットを変更します。

5. クエリーに含めたソースが格納されている ODBC データソースを選択します。
6. このデータベースに接続するために必要なユーザ名とパスワードを入力します。

[Kerberos 認証を使用] オプションは、接続内のデータベースが、Kerberos 認証を使用するネットワークで実行されることを示します。このオプションが選択されている場合、ユーザー名とパスワードは入力できません。接続では、Designer が実行されているマシンにログインしたユーザーアカウントの資格情報が使用されます。

7. [検査] をクリックします。

Designer がクエリーを実行し、その構文が正しいかどうかを通知します。

8. [OK] をクリックして [トランスフォーメーションの編集] ダイアログボックスに戻ります。もう一度 [OK] をクリックして Designer に戻ります。

ヒント: 式エディタのサイズは変更することができます。ダイアログボックスの境界線をドラッグすると、サイズが変更されます。Designer では、変更後のサイズをクライアント設定として保存します。

ユーザー定義ジョインの入力

ユーザー定義ジョイン（結合）を入力する操作は、ユーザー作成の SQL クエリの入力に似ています。この場合、クエリ全体ではなく、WHERE 句の内容だけを入力します。アウタージョインを実行する場合、Integration Service はデータベース構文に基づき、クエリの WHERE 句または FROM 句にジョイン構文を挿入する場合があります。

ユーザー定義ジョインを追加すると、ソース修飾子トランスフォーメーションはデフォルトの SQL クエリの設定を取り込みます。ただし、ユーザー定義ジョインを追加した後でデフォルトのクエリを変更すると、Integration Service はソース修飾子トランスフォーメーションの [SQL クエリ] プロパティに定義されたクエリだけを使用します。

パラメータまたは変数は、その一方をユーザー定義ジョインとして使用することも、あるいはその両方をジョインに含めることもできます。文字列のマッピングパラメータまたは変数を含める場合は、ソースシステムに適した文字列識別子を使用します。ほとんどのデータベースでは、文字列パラメータまたは変数の名前を二重引用符で囲む必要があります。

日時パラメータまたは変数を含める場合、日付フォーマットをソースで使用されているフォーマットに変更しなければならない場合があります。日時パラメータおよび変数は、Integration Service によってソースシステムに応じた文字列に変換されます。

ユーザー定義ジョインを作成するには：

1. 複数のソースまたは関連ソースのデータを含むソース修飾子トランスフォーメーションを作成します。
2. ソース修飾子トランスフォーメーションを開き、[プロパティ] タブをクリックします。
3. [User Defined Join] フィールドで [式の編集] をクリックします。

[SQL エディタ] ダイアログボックスが表示されます。

4. ジョイン構文を入力します。

キーワード WHERE は、ジョインの先頭に入力しないでください。このキーワードは、Integration Service によって行にクエリを実行する際に追加されます。

文字列マッピングパラメータおよび変数を文字列識別子で囲みます。必要に応じて、日時マッピングパラメータおよび変数の日付フォーマットを変更します。

5. [OK] をクリックして [トランスフォーメーションの編集] ダイアログボックスに戻り、再び [OK] をクリックして Designer に戻ります。

アウタージョインのサポート

ソース修飾子トランスフォーメーションおよびアプリケーションソース修飾子トランスフォーメーションを使用して、同じデータベース内の 2 つのソースのアウタージョインを実行できます。Integration Service ではアウタージョイン実行時に、1 つのソーステーブルからすべての行が返され、2 番目のソーステーブルからジョイン条件に一致する行が返されます。

2 つのテーブルを結合し、一方のテーブルからすべての行を返したい場合に、アウタージョインを使用します。たとえば、登録済み顧客テーブルを月ごとの購入テーブルと結合して登録済み顧客の行動を調べたい場合に、アウタージョインを使用することができます。アウタージョインを使用することによって、登録済み顧客テーブルを月ごとの購入テーブルと結合し、前月に何も購入しなかった顧客を含む登録済み顧客テーブルのすべての行を返すことができます。Normal ジョインを実行した場合、Integration Service はその月に何か購入した登録済み顧客のみ、および登録済み顧客による購入のみを返します。

アウタージョインの場合、ジョイナトランスフォーメーションの Master Outer（マスターアウター）ジョインまたは Detail Outer（明細アウター）ジョインとして同じ結果を生成することができます。しかし、アウタージョインを使用した場合、データフロー内の行数が削減されます。これによってパフォーマンスが向上します。

Integration Service は、2 種類のアウタージョインをサポートしています。

- **レフト**。Integration Service は、ジョイン構文の左辺のテーブルからすべての行を、両方のテーブルからジョイン条件に一致する行を返します。
- **ライト**。Integration Service は、ジョイン構文の右辺のテーブルからすべての行を、両方のテーブルからジョイン条件に一致する行を返します。

注: デフォルトクエリを上書きする場合、ネストされたクエリ文でアウタージョインを使用します。

Informatica ジョイン構文

ジョイン構文を入力する場合、Informatica ジョイン構文またはデータベース固有のジョイン構文を使用します。Informatica ジョイン構文を使用すると、Integration Service はセッション実行中に構文を変換してソースデータベースに渡します。

注: ジョイン条件については、必ずデータベース固有の構文を使用してください。

Informatica ジョイン構文を使用する場合、結合文全体を「{Informatica 構文}」のようにブレースで囲みます。データベース構文を使用する場合、ソースデータベースがサポートしている構文をブレースで囲まずに使用します。

Informatica ジョイン構文を使用する場合、テーブル名をカラム名の先頭に付けます。たとえば、REG_CUSTOMER テーブルの FIRST_NAME という名前のカラムの場合、ジョイン構文には「REG_CUSTOMER.FIRST_NAME」と入力します。また、テーブル名にエイリアスを使用している場合、Integration Service がエイリアスを確実に認識するように、Informatica ジョイン構文内でエイリアスを使用します。

以下の表に、アウタージョインを作成する場合に、各ソース修飾子トランスフォーメーションのそれぞれの場所で入力可能なジョイン構文を示します。

トランスフォーメーション	トランスフォーメーションの設定	説明
Source Qualifier トランスフォーメーション	User Defined Join	ジョイン上書きを作成します。Integration Service は、デフォルトクエリーの WHERE 句または FROM 句にジョイン上書きを追加します。
ソース修飾子トランスフォーメーション	SQL クエリ	ジョイン構文をデフォルトクエリーの WHERE の直後に入力します。
Application Source Qualifier トランスフォーメーション	結合のオーバーライド	ジョイン上書きを作成します。Integration Service は、デフォルトクエリーの WHERE 句にジョインオーバーライドを追加します。
アプリケーションソース修飾子トランスフォーメーション	抽出オーバーライド	ジョイン構文をデフォルトクエリーの WHERE の直後に入力します。

1 つのソース修飾子でレフトアウタージョインおよびライトアウタージョインを Normal ジョインと統合することができます。複数の Normal ジョイン、および複数のレフトアウタージョインを使用します。

ジョインを統合する場合、次の順で入力します。

1. ノーマル

2. レフトアウター
3. ライトアウター

注: 一部のデータベースでは、ライトアウタージョインの使用に制限があります。

Normal ジョイン構文

ソース修飾子のジョイン条件を使用して Normal ジョインを作成することができます。ただし、アウタージョインを作成する場合は、アウタージョインを実行するためにデフォルトジョインを上書きする必要があります。したがって、Normal ジョインをジョイン上書きに含める必要があります。Normal ジョインをジョイン上書きに組み込む場合、アウタージョインの前に Normal ジョインを記述します。ジョイン上書きには複数の Normal ジョインを入力することができます。

Normal ジョインを作成するには、以下の構文を使用します。

```
{ source1 INNER JOIN source2 on join_condition }
```

以下の表に、ジョイン上書き内の Normal ジョインの構文を示します。

構文	説明
<i>source1</i>	ソーステーブル名。Integration Service は、このテーブルからジョイン条件に一致する行を返します。
<i>source2</i>	ソーステーブル名。Integration Service は、このテーブルからジョイン条件に一致する行を返します。
<i>join_condition</i>	ジョイン条件。ソースデータベースがサポートしている構文を使用してください。複数のジョイン条件を AND 演算子によって組み合わせることができます。

たとえば、登録済み顧客のデータを含む REG_CUSTOMER テーブルがあるとします。

CUST_ID	FIRST_NAME	LAST_NAME
00001	Marvin	Chi
00002	Dinah	Jones
00003	John	Bowden
00004	J.	Marks

毎月更新される PURCHASES テーブルに以下のデータが含まれているとします。

TRANSACTION_NO	CUST_ID	DATE	AMOUNT
06-2000-0001	00002	6/3/2000	55.79
06-2000-0002	00002	6/10/2000	104.45
06-2000-0003	00001	6/10/2000	255.56
06-2000-0004	00004	6/15/2000	534.95
06-2000-0005	00002	6/21/2000	98.65
06-2000-0006	NULL	6/23/2000	155.65

TRANSACTION_NO	CUST_ID	DATE	AMOUNT
06-2000-0007	NULL	6/24/2000	325.45

6月の各取り引きの顧客名を表示する行を返すには、次の構文を使用します。

```
{ REG_CUSTOMER INNER JOIN PURCHASES on REG_CUSTOMER.CUST_ID = PURCHASES.CUST_ID }
```

Integration Service は以下のデータを返します。

CUST_ID	DATE	AMOUNT	FIRST_NAME	LAST_NAME
00002	6/3/2000	55.79	Dinah	Jones
00002	6/10/2000	104.45	Dinah	Jones
00001	6/10/2000	255.56	Marvin	Chi
00004	6/15/2000	534.95	J.	Marks
00002	6/21/2000	98.65	Dinah	Jones

Integration Service は、顧客 ID が一致する行を返します。6月に何も購入していない顧客は含まれません。非登録顧客による購入も含まれません。

レフトアウトージョイン構文

ジョイン上書きでレフトアウトージョインを作成することができます。1つのジョイン上書きに複数のレフトアウトージョインを入力することができます。他のジョインと共にレフトアウトージョインを使用する場合、文においては、レフトアウトージョインをすべてまとめて Normal ジョインの後に一覧表示します。

レフトアウトージョインを作成するには、以下の構文を使用します。

```
{ source1 LEFT OUTER JOIN source2 on join_condition }
```

以下の表に、ジョイン上書き内のレフトアウトージョインの構文を示します。

構文	説明
<i>source1</i>	ソーステーブル名。レフトアウトージョインの場合、Integration Service はこのテーブルのすべての行を返します。
<i>source2</i>	ソーステーブル名。Integration Service は、このテーブルからジョイン条件に一致する行を返します。
<i>join_condition</i>	ジョイン条件。ソースデータベースがサポートしている構文を使用してください。複数のジョイン条件を AND 演算子によって組み合わせることができます。

たとえば、「[Normal ジョイン構文](#)」(ページ 418)と同じ REG_CUSTOMER および PURCHASES テーブルを使用した場合、以下のジョイン上書きによって、6月に何回購入した顧客の数を求めることができます。

```
{ REG_CUSTOMER LEFT OUTER JOIN PURCHASES on REG_CUSTOMER.CUST_ID = PURCHASES.CUST_ID }
```

Integration Service は以下のデータを返します。

CUST_ID	FIRST_NAME	LAST_NAME	DATE	AMOUNT
00001	Marvin	Chi	6/10/2000	255.56

CUST_ID	FIRST_NAME	LAST_NAME	DATE	AMOUNT
00002	Dinah	Jones	6/3/2000	55.79
00003	John	Bowden	NULL	NULL
00004	J.	Marks	6/15/2000	534.95
00002	Dinah	Jones	6/10/2000	104.45
00002	Dinah	Jones	6/21/2000	98.65

Integration Service は、6 月に何も購入しなかった顧客には NULL 値を使用し、REG_CUSTOMERS テーブル内の登録済み顧客をすべて返します。非登録顧客による購入は含まれません。

複数のジョイン条件を使用して、6 月において 1 回の購入金額が\$100.00 を超える登録済み顧客の人数を求めることができます。

```
{REG_CUSTOMER LEFT OUTER JOIN PURCHASES on (REG_CUSTOMER.CUST_ID = PURCHASES.CUST_ID AND PURCHASES.AMOUNT > 100.00) }
```

Integration Service は以下のデータを返します。

CUST_ID	FIRST_NAME	LAST_NAME	DATE	AMOUNT
00001	Marvin	Chi	6/10/2000	255.56
00002	Dinah	Jones	6/10/2000	104.45
00003	John	Bowden	NULL	NULL
00004	J.	Marks	6/15/2000	534.95

同じ期間の返品に関する情報を組み込みたい場合、複数のレフトアウタージョインを使用することができます。たとえば、RETURNS テーブルには以下のデータが含まれているとします。

CUST_ID	CUST_ID	RETURN
00002	6/10/2000	55.79
00002	6/21/2000	104.45

6 月に何か購入および返品した顧客の人数を求めるために、2 つのレフトアウタージョインを使用します。

```
{ REG_CUSTOMER LEFT OUTER JOIN PURCHASES on REG_CUSTOMER.CUST_ID = PURCHASES.CUST_ID LEFT OUTER JOIN RETURNS on REG_CUSTOMER.CUST_ID = PURCHASES.CUST_ID }
```

Integration Service は以下のデータを返します。

CUST_ID	FIRST_NAME	LAST_NAME	DATE	AMOUNT	RET_DATE	RETURN
00001	Marvin	Chi	6/10/2000	255.56	NULL	NULL
00002	Dinah	Jones	6/3/2000	55.79	NULL	NULL
00003	John	Bowden	NULL	NULL	NULL	NULL
00004	J.	Marks	6/15/2000	534.95	NULL	NULL
00002	Dinah	Jones	6/10/2000	104.45	NULL	NULL

CUST_ID	FIRST_NAME	LAST_NAME	DATE	AMOUNT	RET_DATE	RETURN
00002	Dinah	Jones	6/21/2000	98.65	NULL	NULL
00002	Dinah	Jones	NULL	NULL	6/10/2000	55.79
00002	Dinah	Jones	NULL	NULL	6/21/2000	104.45

Integration Service は、欠落値には NULL を使用します。

ライトアウトージョイン構文

ジョイン上書きでライトアウトージョインを作成することができます。ライトアウトージョインは、ジョイン構文でテーブルの順序を逆にした場合、レフトアウトージョインと同じ結果を返します。ジョイン上書きには、ライトアウトージョインを 1 つだけ使用します。複数のライトアウトージョインを作成する場合、ソーステーブルの順序を逆にして、結合タイプをレフトアウトージョインに変更してみてください。

他のジョインと共にライトアウトージョインを使用する場合、ジョイン上書きの最後にライトアウトージョインを入力します。

ライトアウトージョインを作成するには、以下の構文を使用します。

```
{ source1 RIGHT OUTER JOIN source2 on join_condition }
```

以下の表に、ジョイン上書きでのライトアウトージョインの構文を示します。

構文	説明
<i>source1</i>	ソーステーブル名。Integration Service は、このテーブルからジョイン条件に一致する行を返します。
<i>source2</i>	ソーステーブル名。ライトアウトージョインの場合、Integration Service はこのテーブルのすべての行を返します。
<i>join_condition</i>	ジョイン条件。ソースデータベースがサポートしている構文を使用してください。複数のジョイン条件を AND 演算子によって組み合わせることができます。

ライトアウトージョインをレフトアウトージョインと一緒に使用することによって、両方のテーブルのすべてのデータを結合して返し、Full Outer（フルアウト）ジョインのシミュレートを行うことができます。たとえば、次のジョイン上書きによって、登録済み顧客および 6 月の購入をすべて抽出することができます。

```
{REG_CUSTOMER LEFT OUTER JOIN PURCHASES on REG_CUSTOMER.CUST_ID = PURCHASES.CUST_ID RIGHT OUTER JOIN PURCHASES on REG_CUSTOMER.CUST_ID = PURCHASES.CUST_ID }
```

Integration Service は以下のデータを返します。

CUST_ID	FIRST_NAME	LAST_NAME	TRANSACTION_NO	DATE	AMOUNT
00001	Marvin	Chi	06-2000-0003	6/10/2000	255.56
00002	Dinah	Jones	06-2000-0001	6/3/2000	55.79
00003	John	Bowden	NULL	NULL	NULL
00004	J.	Marks	06-2000-0004	6/15/2000	534.95
00002	Dinah	Jones	06-2000-0002	6/10/2000	104.45
00002	Dinah	Jones	06-2000-0005	6/21/2000	98.65

CUST_ID	FIRST_NAME	LAST_NAME	TRANSACTION_NO	DATE	AMOUNT
NULL	NULL	NULL	06-2000-0006	6/23/2000	155.65
NULL	NULL	NULL	06-2000-0007	6/24/2000	325.45

アウタージョインの作成

アウタージョインはジョイン上書きとして、またはデフォルトクエリー上書きの一部として入力することができます。

ジョイン上書きを作成すると、Designer はジョイン上書きをデフォルトクエリーの WHERE 句に追加します。セッションの実行中、Integration Service は Informatica ジョイン構文を変換してから、ソースデータの抽出に使用するデフォルトクエリーに含めます。可能な限り、デフォルトクエリーを上書きする代わりに、ジョイン上書きを入力してください。

デフォルトクエリーを上書きする場合は、デフォルトクエリーの WHERE 句にジョイン構文を入力します。セッションの実行中、Integration Service は Informatica ジョイン構文を変換してから、クエリーを使用してソースデータを抽出します。上書き作成後にトランスフォーメーションを変更した場合、Integration Service はその変更を無視します。したがって、可能な限り、アウタージョイン構文をジョイン上書きとして入力します。

ジョイン上書きとしてのアウタージョインの作成

アウタージョインをジョイン上書きとして作成するには：

1. ソース修飾子トランスフォーメーションを開き、[プロパティ] タブをクリックします。
2. ソース修飾子トランスフォーメーションで、[User Defined Join] フィールドのボタンをクリックします。
アプリケーションソース修飾子トランスフォーメーションの場合は、[Join Override] フィールドのボタンをクリックします。
3. ジョイン構文を入力します。

結合の最初に WHERE を入力しないでください。Integration Service で行にクエリーを実行する際に追加されます。

Informatica ジョイン構文をブレース ({}) で囲みます。

テーブルのエイリアスと Informatica ジョイン構文を使用する場合、Informatica ジョイン構文内でエイリアスを使用します。

テーブル名をカラム名の先頭に付けて、「テーブル名.カラム名」のようにします。

ソースデータベースがサポートしているジョイン条件を使用してください。

複数のジョインを入力する場合、ジョインを種別によってグループ化してから、Normal、レフトアウター、ライトアウターの順で記述します。ネストクエリー 1 つにつき、ライトアウタージョインは、1 つだけ含めます。

正確さを保証するために、[ポート] タブからポート名を選択します。

4. [OK] をクリックします。

アウタージョインを Extract Override として作成

アウタージョインを Extract Override として作成する手順

1. アプリケーションソース修飾子トランスフォーメーションの入力および出力ポートを接続したら、トランスフォーメーションのタイトルバーをダブルクリックし、[プロパティ] タブを選択します。
2. アプリケーションソース修飾子トランスフォーメーションで、[Extract Override] フィールドのボタンをクリックします。

3. [SQL 文の生成] をクリックします。
4. WHERE の直後の WHERE 句にジョイン構文を入力します。
Informatica ジョイン構文をブレース ({}) で囲みます。
テーブルのエイリアスと Informatica ジョイン構文を使用する場合、Informatica ジョイン構文内でエイリアスを使用します。
テーブル名をカラム名の先頭に付けて、「テーブル名.カラム名」のようにします。
ソースデータベースがサポートしているジョイン条件を使用してください。
複数のジョインを入力する場合、ジョインを種別によってグループ化してから、Normal、レフトアウター、ライトアウターの順で記述します。ネストクエリー 1 つにつき、ライトアウタージョインは、1 つだけ含めます。
正確さを保証するために、[ポート] タブからポート名を選択します。
5. [OK] をクリックします。

一般的なデータベース構文の制約

データベースによって、アウタージョイン構文の制約は異なります。アウタージョインを作成する際は下記の制約に注意してください。

- アウタージョイン構文の ON 句で OR 演算子によってジョイン条件を統合してはなりません。
- アウタージョイン構文の ON 句で IN 演算子を使用してカラムを比較してはなりません。
- アウタージョイン構文の ON 句で、カラムを副クエリーと比較してはなりません。
- 2 つ以上のアウタージョインを統合する場合、同じテーブルを複数のアウタージョインのインナーテーブルとして使用してはなりません。たとえば、次のアウタージョインはいずれも使用できません。

```
{ TABLE1 LEFT OUTER JOIN TABLE2 ON TABLE1.COLUMNA = TABLE2.COLUMNA TABLE3 LEFT OUTER JOIN TABLE2 ON  
TABLE3.COLUMNB = TABLE2.COLUMNB }  
{ TABLE1 LEFT OUTER JOIN TABLE2 ON TABLE1.COLUMNA = TABLE2.COLUMNA TABLE2 RIGHT OUTER JOIN TABLE3 ON  
TABLE2.COLUMNB = TABLE3.COLUMNB }
```
- 通常のジョイン条件では、アウタージョインの両方のテーブルを使用しないでください。たとえば、次のようなジョイン条件は使用しないでください。

```
{ TABLE1 LEFT OUTER JOIN TABLE2 ON TABLE1.COLUMNA = TABLE2.COLUMNA WHERE TABLE1.COLUMNB = TABLE2.COLUMNC }
```


ただし、次のようなフィルタ条件では両方のテーブルを使用します。

```
{ TABLE1 LEFT OUTER JOIN TABLE2 ON TABLE1.COLUMNA = TABLE2.COLUMNA WHERE TABLE1.COLUMNB = 32 AND  
TABLE2.COLUMNC > 0 }
```


注: ON 句に条件を入力した場合、同じ条件を WHERE 句に入力した場合とは異なる結果が返されることがあります。
- テーブルのエイリアスを使用する場合、エイリアスをテーブルのカラムの先頭に付加します。たとえば、REG_CUSTOMER テーブルを C としている場合、FIRST_NAME カラムを参照するときは「C.FIRST_NAME」を使用します。

ソースフィルタの入力

ソースフィルタを入力すると、Integration Service がクエリの対象とする行の数を減らすことができます。ソースフィルタに文字列「WHERE」または Large objects が含まれる場合、Integration Service はセッションに失敗します。

マッピングのソース修飾子トランスフォーメーションにソースフィルタを追加すると、デフォルトの SQL クエリにそのフィルタ条件が含まれます。ただし、ソースフィルタを追加した後でデフォルトクエリを変更した場合、Integration Service はソース修飾子トランスフォーメーションの SQL クエリ部分で定義されたクエリのみを使用します。

パラメータまたは変数は、その一方をソースフィルタとして使用することも、あるいはその両方をソースフィルタ内に含めることもできます。文字列のマッピングパラメータまたは変数を含める場合は、ソースシステムに適した文字列識別子を使用します。ほとんどのデータベースでは、文字列パラメータまたは変数の名前を二重引用符で囲む必要があります。

日時パラメータまたは変数を含める場合、日付フォーマットをソースで使用されているフォーマットに変更しなければならない場合があります。日時パラメータおよび変数は、Integration Service によってソースシステムに応じた文字列に変換されます。

注: セッションのプロパティに SQL クエリを入力すると、そのマッピングレベルにおけるフィルタ条件と SQL クエリが上書きされます。

ソースフィルタを入力するには：

1. Mapping Designer でソース修飾子トランスフォーメーションを開きます。
[トランスフォーメーションの編集] ダイアログボックスが表示されます。
2. [プロパティ] タブを選択します。
3. [ソースフィルタ] フィールドの右端のボタンをクリックします。
4. [SQL エディタ] ダイアログボックスで、フィルタを入力します。
テーブル名とポート名を含めます。フィルタにはキーワード WHERE を入れないでください。
文字列マッピングパラメータおよび変数を文字列識別子で囲みます。必要に応じて、日時マッピングパラメータおよび変数の日付フォーマットを変更します。
5. [OK] をクリックします。

ソート済みポートの使用

ソート済みポートを使用する場合、Integration Service はそのポートをデフォルトクエリの ORDER BY 句に追加します。Integration Service は、設定された数のポートをソース修飾子トランスフォーメーションの最初のポートから順に追加していきます。ポートのサブセットがダウンストリームに接続されている場合、デフォルトのクエリにはポートのサブセットのみが含まれます。ソート済みポートはソース修飾子トランスフォーメーションのトップで開始されるポートではなく、接続済みのポートに適用されます。

マッピングに以下のトランスフォーメーションが含まれる場合、ソート済みポートを使用してパフォーマンスを高めることができます。

- **アグリゲータ。**アグリゲータトランスフォーメーションをソート済み入力用に設定すると、ソート済みポートを使ってソート済みデータを送ることができます。アグリゲータトランスフォーメーションでの Group By ポートは、ソース修飾子トランスフォーメーション内のソート済みポートの順序と一致しなければなりません。
- **ジョイナ。**ジョイナトランスフォーメーションをソート済み入力用に設定すると、ソート済みポートを使ってソート済みデータを送ることができます。各ソース修飾子トランスフォーメーション内のソート済みポートの順序は同一に設定してください。

注: また、アグリゲータおよびジョイナトランスフォーメーションの前にソータトランスフォーメーションを使用して、リレーショナルデータおよびフラットファイルデータをソートすることもできます。

ソート済みポートはリレーショナルソースのみに使います。ソート済みポートを使用する場合、ソースデータベースのソート順はセッションに対して設定されたソート順と一致しなければなりません。Integration Service は、ソート済みポートの ORDER BY 句を含め、ソースデータの抽出に使用する SQL クエリを作成します。データベースサーバーはクエリを実行して、結果データを Integration Service に渡します。データを確実に Integration Service の要求どおりにソートするには、データベースのソート順とユーザー定義のセッションのソート順を一致させる必要があります。

Integration Service でデータコードページの検証機能を設定して、Unicode データ動作モードでワークフローを実行する場合、Integration Service は選択されたソート順を使用して文字データをソートします。

Integration Service でのデータコードページの検証基準を緩和した場合、Integration Service は選択されたソート順を使用して、選択したソート順の言語範囲内の文字データをすべてソートします。Integration Service は、選択したソート順の言語範囲外の文字データはすべて、標準の Unicode ソート順どおりにソートします。

Integration Service が ASCII モードで動作している場合、この設定は無視され、すべての文字データがバイナリソート順でソートされます。デフォルトのソート順は、Integration Service のコードページに基づいています。

ソース修飾子トランスフォーメーションでは、デフォルトの SQL クエリにソート済みポートがいくつか含まれます。ただし [ソート対象ポートの数] を選択した後でデフォルトのクエリを変更すると、Integration Service は [SQL クエリ] プロパティで定義されたクエリだけを使用します。

ソート済みポートを使用するには：

1. Mapping Designer で、ソース修飾子トランスフォーメーションを開き、[プロパティ] タブをクリックします。
2. [ソートするポート数] をクリックして、ソートしたいポート数を入力します。

Integration Service は、ソース修飾子トランスフォーメーションの先頭から数えて、設定された数のカラムを ORDER BY 句に追加します。

ソースデータベースのソート順は、セッションのソート順に対応していなければなりません。

ヒント: Sybase は ORDER BY 句内で最大 16 カラムをサポートします。ソースが Sybase の場合、16 カラムを超えるソートは実行しないでください。

3. [OK] をクリックします。

個別に選択

Integration Service でソースから一意の値が選択されるようにする場合は、[個別に選択] オプションを使用します。この機能は、たとえば売り上げ合計を記録しているテーブルから一意なカスタマ ID を抽出する場合などに使用します。[個別に選択] を使用すると、データフローの初期段階で不必要なデータを除外することができるため、パフォーマンスが向上します。

デフォルトでは、Designer は SELECT 文を生成します。[個別に選択] を選択すると、ソース修飾子トランスフォーメーションはこの設定をデフォルトの SQL クエリに取り込みます。

たとえば、「[ソースデータの結合](#)」(ページ 412)のソース修飾子トランスフォーメーションで、[個別に選択] オプションを有効にします。Designer は、次のように SELECT DISTINCT をデフォルトクエリに追加します。

```
SELECT DISTINCT CUSTOMERS.CUSTOMER_ID, CUSTOMERS.COMPANY, CUSTOMERS.FIRST_NAME, CUSTOMERS.LAST_NAME,
CUSTOMERS.ADDRESS1, CUSTOMERS.ADDRESS2, CUSTOMERS.CITY, CUSTOMERS.STATE, CUSTOMERS.POSTAL_CODE,
CUSTOMERS.EMAIL, ORDERS.ORDER_ID, ORDERS.DATE_ENTERED, ORDERS.DATE_PROMISED, ORDERS.DATE_SHIPPED,
ORDERS.EMPLOYEE_ID, ORDERS.CUSTOMER_ID, ORDERS.SALES_TAX_RATE, ORDERS.STORE_ID
FROM
CUSTOMERS, ORDERS
WHERE
CUSTOMERS.CUSTOMER_ID=ORDERS.CUSTOMER_ID
```

ただし「個別に選択」を選択したあとでデフォルトのクエリを変更すると、Integration Service は「SQL クエリ」プロパティで定義されたクエリだけを使用します。すなわち、その SQL クエリによって「個別に選択」の設定が上書きされます。

「個別に選択」を使用するには：

1. マッピングでソース修飾子トランスフォーメーションを開き、「プロパティ」タブをクリックします。
2. 「個別に選択」にチェックマークを付け、「OK」をクリックします。

セッションでの「個別に選択」の上書き

「個別に選択」に対するトランスフォーメーションレベルのオプションは、Workflow Manager でセッションを設定するときに上書きすることができます。

「個別に選択」オプションを上書きするには：

1. Workflow Manager で「セッション」タスクを開き、「マッピング」タブをクリックします。
2. 「トランスフォーメーション」ビューをクリックし、「ソース」ノードの下のソース修飾子トランスフォーメーションをクリックします。
3. 「プロパティ」設定で「個別に選択」を有効にして、「OK」をクリックします。

セッション実行前/実行後の SQL コマンドの追加

ソース修飾子トランスフォーメーションの「プロパティ」タブで、セッション実行前および実行後の SQL コマンドを追加できます。セッション実行前の SQL を使えば、セッションを開始する際にソーステーブルヘタイムスタンプ行を記入できます。

Integration Service は、ソースデータベースに対してセッション実行前の SQL コマンドを実行した後、ソースを読み込みます。また、ターゲットへ書き込みをした後にソースデータベースに対してセッション実行後 SQL コマンドを実行します。

SQL コマンドは、セッションプロパティの「マッピング」タブの「トランスフォーメーション」ビューで上書きできます。Integration Service では、セッション実行前/実行後の SQL コマンドの実行時にエラーが発生した場合に、処理を停止するか続行するかを設定することもできます。

ソース修飾子トランスフォーメーションにセッション実行前および実行後 SQL コマンドを入力する場合には、下記のガイドラインに従ってください。

- そのデータベースタイプで有効な任意のコマンドを使用します。ただし Integration Service では、データベースで許可されている場合でも、ネストされたコメントを使用することはできません。
- パラメータおよび変数は、その両方をソースのセッション実行前/実行後 SQL コマンド内に使用することも、あるいはその一方をコマンドとして使用することもできます。パラメータファイルで定義可能なパラメータまたは変数タイプを使用します。
- 複数の文を区切るにはセミコロン (;) を使用します。Integration Service では各ステートメントの後にコミットが発行されます。
- Integration Service では、/*...*/内のセミコロンは無視されます。
- コメントの外部でセミコロンを使用する必要がある場合は、バックスラッシュ (\) でセミコロンをエスケープします。セミコロンをエスケープすると、Integration Service はバックスラッシュを無視し、セミコロンを文の区切り文字として使用しません。
- Designer では、SQL は検証されません。

注: マッピングのターゲットインスタンスの [プロパティ] タブでもセッション実行前および実行後 SQL コマンドを入力できます。

ソース修飾子トランスフォーメーションの作成

Designer を設定して、ソースをマッピングにドラッグするとデフォルトでソース修飾子トランスフォーメーションが作成されるようにすることができます。また、ソース修飾子トランスフォーメーションを手動で作成することもできます。

ソース修飾子トランスフォーメーションの手動での作成

Mapping Designer で、ソース修飾子トランスフォーメーションを手動で作成することができます。

ソース修飾子トランスフォーメーションを手動で作成するには：

1. Mapping Designer で、[トランスフォーメーション] - [作成] をクリックします。
2. トランスフォーメーションの名前を入力して、[作成] をクリックします。
3. ソースを選択して、[OK] をクリックします。
4. [完了] をクリックします。

ソース修飾子トランスフォーメーションオプションの設定

ソース修飾子トランスフォーメーションを作成したら、いくつかのオプションを設定することができます。

ソース修飾子トランスフォーメーションの設定を行うには：

1. Designer でマッピングを開きます。
2. ソース修飾子トランスフォーメーションのタイトルバーをダブルクリックします。
3. [トランスフォーメーションの編集] ダイアログボックスで [名前の変更] をクリックし、トランスフォーメーションのわかりやすい名前を入力してから、[OK] をクリックします。
ソース修飾子トランスフォーメーションの命名規則は、「SQ_トランスフォーメーション名」（たとえば SQ_AllSources）です。
4. [プロパティ] タブをクリックします。
5. ソース修飾子トランスフォーメーションのプロパティを入力します。
6. [ソース] タブをクリックし、このトランスフォーメーションに対して定義したい関連ソース定義を指定します。
関連ソースは、複数のデータベースまたはフラットファイルシステムのデータを結合する場合にのみ指定します。
7. [OK] をクリックします。

ソース修飾子トランスフォーメーションのトラブルシューティング

ポートの接続などでドラッグアンドドロップ操作ができません。

ステータスバーに表示されるエラーメッセージを参照してください。

ソース定義をターゲット定義に接続できません。

ソースをターゲットに直接接続することはできません。リレーショナルソースやフラットファイルソースについては、ソース修飾子トランスフォーメーションを通して接続し、また COBOL ソースについてはノーマライザトランスフォーメーションを通して接続する必要があります。

1 つのターゲットに複数のソースを接続できません。

Designer は 1 つのターゲットに複数のソース修飾子トランスフォーメーションを接続することを禁止します。対策としては 2 つあります。

- **ターゲットを再利用する。**ターゲット定義は再利用可能であるため、同一のターゲットを複数回マッピングに追加することができます。次に、それぞれのソース修飾子トランスフォーメーションをそれぞれのターゲットに接続します。
- **ソース修飾子トランスフォーメーションで複数のソースを結合します。**その後、SQL クエリから WHERE 句を削除します。

ソースのカラムの中には（数値ではない）QNaN 値が設定されているものがありますが、ターゲットでは 1.#QNaN と表示されます。

NaN の文字列表示は、オペレーティングシステムによって異なります。Integration Service は、Win64 EMT プラットフォームでは QNaN 値を 1.#QNaN に変換します。1.#QNaN は QNaN の有効な表示です。

ユーザー作成のクエリを入力しましたが、セッションを含むワークフローを実行してもこのユーザー作成のクエリが動作しません。

必ずソース修飾子トランスフォーメーションに対してこの設定をテストしてから、ワークフローを実行してください。ソース修飾子トランスフォーメーションに戻り、ユーザー作成のクエリを入力したダイアログを再度開いてください。データベースに接続し、[検証] をクリックすれば、SQL の確認ができます。エラーがあれば Designer に表示されます。詳細については、セッションログファイルを確認してください。

セッションが失敗する最も一般的な原因に、セッションとソース修飾子トランスフォーメーションでのデータベースログインがテーブルのオーナー以外によって行われていることがあります。セッション内において、およびソース修飾子トランスフォーメーションで SQL クエリを生成する際には、テーブルオーナーを指定する必要があります。

SQL クエリをカットアンドペーストでデータベースクライアントツール（Oracle Net など）に貼り付け、それがエラーを返すかどうかを確認することにより、SQL クエリをテストできます。

ソースフィルタでマッピング変数を使用したら、セッションが失敗しました。

ソース修飾子トランスフォーメーションで SQL を生成および検証して、クエリをテストしてみてください。変数またはパラメータが文字列である場合、それを一重引用符で囲む必要があるかもしれません。日時変数またはパラメータの場合は、フォーマットをソースシステムで使用されているフォーマットに変更しなければならない場合があります。

第 26 章

SQL トランスフォーメーション

この章では、以下の項目について説明します。

- [SQL トランスフォーメーションの概要, 429](#) ページ
- [スクリプトモード, 430](#) ページ
- [クエリモード, 432](#) ページ
- [データベースへの接続, 437](#) ページ
- [セッション処理, 440](#) ページ
- [入力行と出力行のカーディナリティ, 444](#) ページ
- [SQL トランスフォーメーションプロパティ, 448](#) ページ
- [SQL 文, 451](#) ページ
- [SQL トランスフォーメーションの作成, 452](#) ページ

SQL トランスフォーメーションの概要

SQL トランスフォーメーションによって、パイプライン内の SQL クエリの間中ストリームが処理されます。SQL トランスフォーメーションは、アクティブまたはパッシブなトランスフォーメーションにすることができます。データベースに行を挿入したり、データベースの行を削除、更新、および取得したりできます。実行時に、データベース接続情報を SQL トランスフォーメーションに入力データとして渡すことができます。トランスフォーメーションでは、外部 SQL スクリプトまたは SQL エディタで作成した SQL クエリが処理されます。SQL トランスフォーメーションでクエリが処理され、行およびデータベースエラーが返されます。

たとえば、新しいトランザクションを追加する前にデータベーステーブルを作成する必要がある場合があります。SQL トランスフォーメーションを作成して、ワークフロー内でテーブルを作成できます。SQL トランスフォーメーションによって、データベースエラーが出力ポートに返されます。SQL トランスフォーメーションによってエラーが返されない場合に別のワークフローが実行されるように設定できます。

SQL トランスフォーメーションを作成するとき、以下のオプションを設定します。

- **モード。** SQL トランスフォーメーションは、以下のいずれかのモードで実行されます。
 - **スクリプトモード。** SQL トランスフォーメーションによって、外部に配置された ANSI SQL スクリプトが実行されます。スクリプト名を各入力行と共にトランスフォーメーションに渡します。SQL トランスフォーメーションによって、入力行ごとに 1 行出力されます。
 - **クエリモード。** SQL トランスフォーメーションによって、クエリエディタで定義したクエリが実行されます。文字列またはパラメータをクエリに渡し、動的クエリを定義するか、または選択パラメータを変更できます。クエリに SELECT 文がある場合、複数の行を出力できます。

- **パッシブまたはアクティブなトランスフォーメーション。** デフォルトでは、SQL トランスフォーメーションはアクティブなトランスフォーメーションです。このトランスフォーメーションは、作成時にパッシブなトランスフォーメーションとして設定できます。
- **データベースタイプ。** SQL トランスフォーメーションが接続するデータベースのタイプです。
- **接続タイプ。** データベース接続情報を SQL トランスフォーメーションに渡すか、または接続オブジェクトを使用します。

スクリプトモード

スクリプトモードで実行される SQL トランスフォーメーションでは、テキストファイルから SQL スクリプトが実行されます。ソースから SQL トランスフォーメーションの ScriptName ポートに、各スクリプトファイルの名前を渡します。スクリプトファイル名には、スクリプトファイルの完全パスが含まれます。

スクリプトモードで実行されるようにトランスフォーメーションを設定する場合、パッシブなトランスフォーメーションを作成します。トランスフォーメーションによって、入力行ごとに 1 行が返されます。出力行には、クエリの結果とデータベースエラーが含まれます。

SQL トランスフォーメーションがスクリプトモードで実行される場合、クエリ文とクエリデータは変更されません。異なるクエリをスクリプトモードで実行する必要がある場合、ソースデータでスクリプトを渡します。スクリプトモードを使用して、テーブルの作成や削除などのデータ定義クエリを実行します。

スクリプトモードで実行されるように SQL トランスフォーメーションを設定すると、Designer によって ScriptName 入力ポートがトランスフォーメーションに追加されます。マッピングを作成するときに、各行に対して実行するスクリプトの名前が含まれるポートに ScriptName ポートを接続します。入力行ごとに異なる SQL スクリプトを実行できます。Designer によって、クエリ結果に関する情報を返すデフォルトのポートが作成されます。

スクリプトモード用に設定された SQL トランスフォーメーションには、以下のデフォルトのポートがあります。

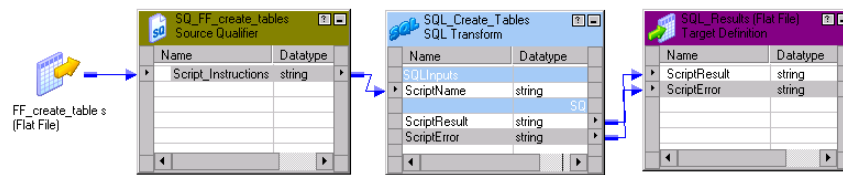
ポート	タイプ	説明
ScriptName	入力	現在の行に対して実行するスクリプトの名前を受け取ります。
ScriptResult	出力	行に対してスクリプト実行が成功した場合、PASSED を返します。それ以外の場合は、FAILED が含まれます。
ScriptError	出力	行に対してスクリプトが失敗した場合に発生するエラーを返します。

例

テーブルに新しいデータを追加する前に、注文テーブルおよび在庫テーブルを作成する必要があります。テーブルを作成する SQL スクリプトを作成し、スクリプトを実行する SQL トランスフォーメーションを設定できます。

テーブルを作成する SQL 文が含まれる create_order_inventory.txt という名前のファイルを作成します。

以下のマッピングは、スクリプト名を SQL トランスフォーメーションに渡す方法を示します。



Integration Service によって、ソースから行が読み取られます。ソース行には、以下のように SQL スクリプトファイルの名前とパスが含まれます。

C:\81\server\shared\SrcFiles\create_order_inventory.txt

トランスフォーメーションによって、ScriptName ポートでファイル名が受け取られます。Integration Service によってスクリプトファイルが検出され、スクリプトが解析されます。SQL プロシージャが作成され、処理のためにデータベースに送信されます。データベースによって SQL が検証され、クエリーが実行されます。

SQL トランスフォーメーションによって、ScriptResults および ScriptError が返されます。スクリプトが正常に実行されると、ScriptResult 出力ポートによって PASSED が返されます。これ以外の場合は、ScriptResult 出力ポートによって FAILED が返されます。ScriptResult が FAILED の場合、SQL トランスフォーメーションによって ScriptError ポートでエラーメッセージが返されます。SQL トランスフォーメーションによって、受け取られた入力行ごとに 1 行が返されます。

スクリプトモードのルールとガイドライン

スクリプトモードで実行される SQL トランスフォーメーションには、以下の規則およびガイドラインを使用します。

- 静的または動的データベース接続をスクリプトモードで使用できます。
- 複数のクエリ文をスクリプトに含めるには、セミコロンで区切ります。
- スクリプトファイル名でマッピング変数またはパラメータを使用できます。
- スクリプトコードページは、デフォルトでオペレーティングシステムのロケールに設定されます。スクリプトのロケールは変更できます。
- スクリプトファイルは、Integration Service によってアクセス可能である必要があります。Integration Service には、スクリプトが含まれるディレクトリの読み取り権限が必要です。Integration Service にオペレーティングシステムプロファイルが使用されている場合、そのオペレーティングシステムプロファイルのオペレーティングシステムユーザーは、スクリプトが格納されているディレクトリの読み取り権限を持つ必要があります。
- SQL スクリプトに含まれる SELECT 文の出力は、Integration Service では無視されます。スクリプトモードの SQL トランスフォーメーションでは、入力行ごとに 1 行を超えるデータは出力されません。
- Oracle PL/SQL や Microsoft/Sybase T-SQL などのスクリプト言語をスクリプト内で使用することはできません。
- SQL スクリプトが別の SQL スクリプトを呼び出すネストされたスクリプトは使用できません。
- スクリプトは実行時引数を受け入れません。

クエリモード

SQL トランスフォーメーションがクエリモードで実行される場合、トランスフォーメーションに定義した SQL クエリが実行されます。トランスフォーメーションの入力ポートから文字列またはパラメータをクエリに渡し、クエリ文またはクエリデータを変更します。

クエリモードで実行されるように SQL トランスフォーメーションを設定する場合、アクティブなトランスフォーメーションを作成します。入力行ごとに複数の行をトランスフォーメーションで返すことができます。

SQL トランスフォーメーションの SQL エディタでクエリを作成します。クエリを作成するには、SQL エディタのメインウィンドウでクエリ文を入力します。SQL エディタには、クエリで参照できるトランスフォーメーションポートのリストが表示されます。ポート名をダブルクリックして、クエリパラメータとして追加できます。

SQL クエリを作成すると、クエリ内のポート名が SQL エディタによって検証されます。また、文字列の置換に使用するポートが文字列データタイプであるのかも確認されます。SQL エディタでは、SQL クエリの構文は検証されません。

以下のタイプの SQL クエリを SQL トランスフォーメーションで作成できます。

- **静的 SQL クエリ。** クエリ文は変更されませんが、クエリパラメータを使用してデータを変更できます。Integration Service によってクエリが一度準備され、そのクエリがすべての入力行に対して使用されます。
- **動的 SQL クエリ。** クエリ文およびクエリデータを変更できます。Integration Service によって、入力行ごとにクエリが準備されます。

静的クエリを作成すると、Integration Service によって SQL プロシージャが一度準備され、各行に対して使用されます。動的クエリを作成すると、Integration Service によって入力行ごとに SQL が準備されます。静的クエリを作成することによって、パフォーマンスを最適化できます。

静的 SQL クエリの使用

各入力行に対して同じクエリ文を実行する必要があるが、クエリ内のデータを入力行ごとに変更する場合、静的 SQL クエリを作成します。静的 SQL クエリを作成する場合、SQL エディタでパラメータのバインドを使用して、クエリデータのパラメータを定義します。

クエリ内のデータを変更するには、クエリパラメータを設定し、これらのパラメータをトランスフォーメーションの入力ポートにバインドします。パラメータを入力ポートにバインドする場合、クエリにポート名を指定します。SQL エディタでは、名前は疑問符 (?) で囲まれます。クエリデータは、入力ポートのデータの値に基づいて変更されます。

SQL トランスフォーメーションの入力ポートは、クエリ内のデータ値のデータ、またはクエリの WHERE 句の値を受け取ります。

以下の静的クエリでは、パラメータのバインドが使用されています。

```
DELETE FROM Employee WHERE Dept = ?Dept?
INSERT INTO Employee(Employee_ID, Dept) VALUES (?Employee_ID?, ?Dept?)
UPDATE Employee SET Dept = ?Dept? WHERE Employee_ID > 100
```

以下の静的 SQL クエリには、SQL トランスフォーメーションの Employee_ID および Dept 入力ポートにバインドするクエリパラメータがあります。

```
SELECT Name, Address FROM Employees WHERE Employee_Num =?Employee_ID? and Dept = ?Dept?
```

ソースに以下の行があるとします。

Employee_ID	Dept
100	Products

Employee_ID	Dept
123	HR
130	Accounting

Integration Service によって、行から以下のクエリ文が生成されます。

```
SELECT Name, Address FROM Employees WHERE Employee_ID = '100' and DEPT = 'Products'
SELECT Name, Address FROM Employees WHERE Employee_ID = '123' and DEPT = 'HR'
SELECT Name, Address FROM Employees WHERE Employee_ID = '130' and DEPT = 'Accounting'
```

複数のデータベース行の選択

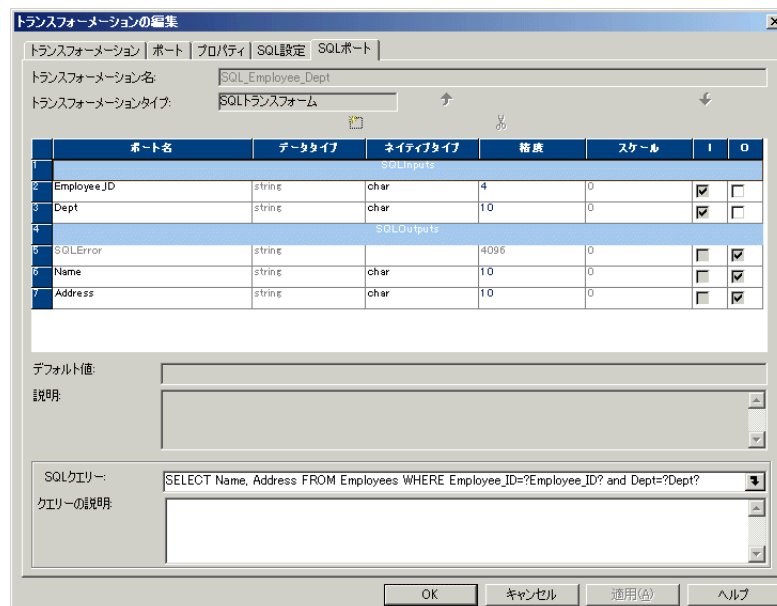
SQL クエリに SELECT 文が含まれる場合、トランスフォーメーションによって、取得したデータベース行ごとに 1 行が返されます。SELECT 文のカラムごとに出力ポートを設定する必要があります。出力ポートは、SELECT 文のカラムと同じ順序である必要があります。

データベースカラムの出力ポートを設定する場合、選択した各データベースカラムのデータタイプを設定する必要があります。ネイティブデータタイプをリストから選択します。ネイティブデータタイプを選択すると、Designer によってトランスフォーメーションのデータタイプが設定されます。

トランスフォーメーションのネイティブデータタイプは、データベースカラムのデータタイプと一致する必要があります。実行時に Integration Service によって、データベース内のカラムのデータタイプがトランスフォーメーションのネイティブデータタイプと照合されます。データタイプが一致しない場合、Integration Service によって行エラーが生成されます。

注: Teradata データベースでは Bigint カラムが許可されていますが、Transformation Developer では SQL トランスフォーメーションに使用できるネイティブデータタイプに Bigint データタイプが含まれていません。

以下の図は、クエリモードで実行されるように設定されたトランスフォーメーションのポートを示しています。



入力ポートは、WHERE 句のデータを受け取ります。出力ポートは、SELECT 文のカラムを返します。SQL クエリによって、従業員テーブルから名前および住所が選択されます。SQL トランスフォーメーションによって、取得したデータベース行ごとに行がターゲットに書き込まれます。

動的 SQL クエリの使用

動的 SQL クエリでは、入力行ごとに異なるクエリ文を実行できます。動的 SQL クエリを作成する場合、文字列の置換を使用してクエリ内に文字列パラメータを定義し、これらのパラメータをトランスフォーメーションの入力ポートにリンクします。

クエリ文を変更するには、変更するクエリ部分の文字列変数をクエリ内に設定します。文字列変数を設定するには、クエリに入力ポートを名前指定し、名前をティルダ（~）で囲みます。クエリは、ポートのデータの値に基づいて変更されます。クエリパラメータが含まれるトランスフォーメーションの入力ポートは、文字列データタイプである必要があります。文字列の置換を使用して、クエリ文とクエリデータを変更できます。

動的 SQL クエリを作成すると、Integration Service によって入力行ごとにクエリが準備されます。完全なクエリまたはクエリの一部を入力ポートで渡すことができます。

- **完全なクエリ。** SQL クエリ全体をソースデータのクエリ文で置換できます。
- **部分クエリ。** クエリ文の一部（テーブル名など）を置換できます。

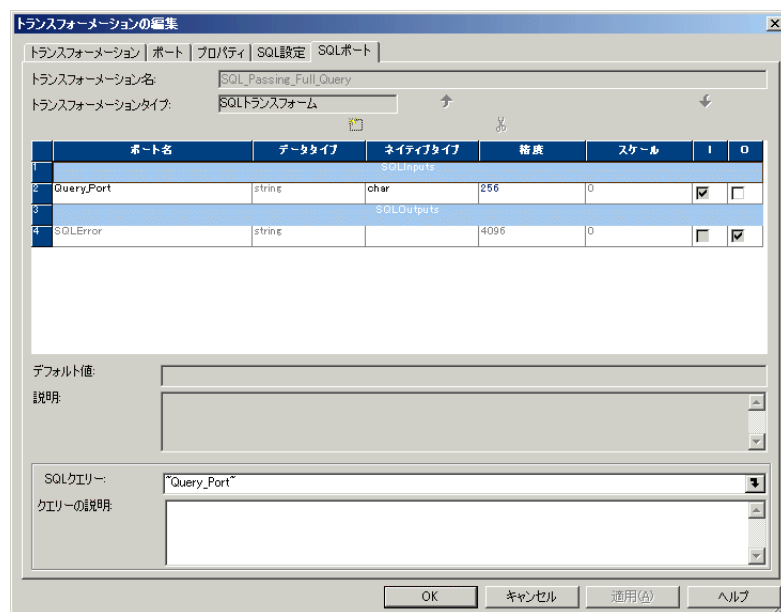
完全なクエリの受け渡し

トランスフォーメーションの入力ポートを介して完全な SQL クエリを渡すことができます。完全なクエリを渡すには、以下のように完全なクエリを表す 1 つの文字列変数で構成されたクエリを SQL エディタで作成します。

~Query_Port~

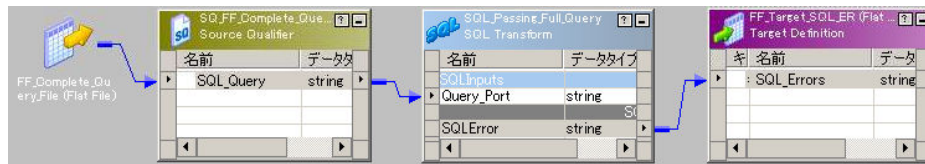
トランスフォーメーションは、Query_Port 入力ポートでクエリを受け取ります。

以下の図に、SQL トランスフォーメーションのポートを示します。



Integration Service によって、動的クエリ内の~Query_Port~変数は、ソースの SQL 文で置換されます。クエリが準備され、処理のためにデータベースに送信されます。データベースによってクエリが実行されます。SQL トランスフォーメーションによって、データベースエラーが SQLError ポートに返されます。

以下のマッピングは、クエリ名を SQL トランスフォーメーションに渡す方法を示します。



完全なクエリを渡すときに、入力行ごとに複数のクエリ文を渡すことができます。たとえば、ソースに以下の行が含まれている場合があります。

```
DELETE FROM Person WHERE LastName = 'Jones'; INSERT INTO Person (LastName, Address) VALUES ('Smith', '38 Summit Drive')
DELETE FROM Person WHERE LastName = 'Jones'; INSERT INTO Person (LastName, Address) VALUES ('Smith', '38 Summit Drive')
DELETE FROM Person WHERE LastName = 'Russell';
```

任意のタイプのクエリをソースデータで渡すことができます。クエリに SELECT 文を設定する場合、データベースから取得するデータベースカラムの出力ポートを設定する必要があります。SELECT 文と他のタイプのクエリを混在させると、データベースカラムが取得されない場合、データベースカラムを表す出力ポートには NULL 値が入ります。

文字列内のテーブル名の置換

クエリ内のテーブル名を置換できます。テーブル名を置換するには、各入力行からテーブル名を受け取るように入力ポートを設定します。クエリに入力ポートを名前で指定し、名前をティルダ (~) で囲みます。

以下の動的クエリには、文字列変数 ~Table_Port~ が含まれています。

```
SELECT Emp_ID, Address from ~Table_Port~ where Dept = 'HR'
```

ソースによって、以下の値が Table_Port カラムに渡される場合があります。

Table_Port

Employees_USA

Employees_England

Employees_Australia

Integration Service によって、~Table_Port~ 変数は入力ポートのテーブル名で置換されます。

```
SELECT Emp_ID, Address from Employees_USA where Dept = 'HR'
SELECT Emp_ID, Address from Employees_England where Dept = 'HR'
SELECT Emp_ID, Address from Employees_Australia where Dept = 'HR'
```

関連項目：

- [「動的更新の例」 \(ページ 454\)](#)

パススルーポートの設定

SQL トランスフォーメーションにはパススルーポートを追加できます。パススルーポートは、トランスフォーメーションを経由してデータを受け渡しする入力/出力ポートです。SQL トランスフォーメーションは、SQL クエリによって行が返されるか否かにかかわらず、パススルーポートからデータを返します。

ソース行に SELECT クエリー文が含まれている場合、SQL トランスフォーメーションは、データベースから返される行ごとに、パススルーポートでデータを返します。クエリー結果に複数の行が含まれている場合、SQL トランスフォーメーションは、各行についてパススルーデータの生成を繰り返します。

クエリーによって行が返されなかった場合、SQL トランスフォーメーションは、出力カラムに、NULL 値を含むパススルーカラムデータを返します。たとえば、INSERT、UPDATE、および DELETE の各文を含むクエリーでは、行が返されません。クエリーでエラーが発生した場合、SQL トランスフォーメーションは、パススルーカラムデータ、SQLException メッセージ、および NULL 値を出力ポートに返します。

パススルーポートを作成するには：

- 入力ポートを作成し、出力用に有効にします。Designer によって出力ポートが作成され、「_output」接尾語がポート名に追加されます。
- ソース修飾子トランスフォーメーションから SQL トランスフォーメーションにポートをドラッグします。Designer がパススルーポートを作成します。

パッシブモードの設定

SQL トランスフォーメーションを作成する場合には、SQL トランスフォーメーションがアクティブモードではなくパッシブモードで実行されるように設定できます。パッシブトランスフォーメーションでは、通過する行の数は変更されません。トランザクション境界と行タイプが維持されます。

トランスフォーメーションがクエリモードで実行されるように設定する場合は、トランスフォーメーションの作成時にパッシブモードを設定できます。トランスフォーメーションの作成後にモードを変更することはできません。

パッシブモードのルールとガイドライン

パッシブモードで実行されるように SQL トランスフォーメーションを設定する場合は、以下のルールおよびガイドラインを使用します。

- SELECT クエリによって複数の行が返される場合、Integration Service は最初の行とエラーを SQLException ポートに返します。このエラーは、SQL トランスフォーメーションが複数行を生成したことを示します。
- SQL クエリに複数の SQL 文がある場合、Integration Service はすべての文を実行します。Integration Service は最初の SQL 文に対してのみデータを返します。SQL トランスフォーメーションでは 1 行が返されます。SQLException ポートには、すべての SQL 文のエラーが含まれています。複数のエラーが発生した場合は、SQLException ポート内でセミコロンで区切られます。
- SQL クエリに複数の SQL 文があり、統計ポートが有効になっている場合は、Integration Service は最初の SQL 文のデータと統計情報を返します。SQLException ポートには、すべての SQL 文のエラーが含まれています。

クエリモードのルールとガイドライン

クエリモードで実行されるように SQL トランスフォーメーションを設定する場合は、以下の規則およびガイドラインを使用します。

- 出力ポートの数および順序は、クエリの SELECT 句内のフィールドの数および順序と一致する必要があります。
- トランスフォーメーション内の出力ポートのネイティブデータタイプは、データベース内の対応するカラムのデータタイプと一致する必要があります。データタイプが一致しない場合、Integration Service によって行エラーが生成されます。
- SQL クエリに INSERT、UPDATE、または DELETE 句が含まれる場合、トランスフォーメーションによってデータが SQLException ポート、パススルーポート、および NumRowsAffected ポート（有効な場合）に返されます。出力ポートを追加した場合、ポートは NULL データ値を受け取ります。
- SQL クエリに SELECT 文が含まれており、トランスフォーメーションにパススルーポートが設定されている場合、トランスフォーメーションはクエリからデータベースのデータが返されるかどうかにかかわらず、

パススルーポートにデータを返します。SQL トランスフォーメーションは、NULL データを含む行を出力ポートに返します。

- 作成した出力ポートの名前に「_output」接尾語を追加することはできません。
- SELECT クエリからデータを返すためにパススルーポートを使用することはできません。
- 出力ポートの数が SELECT 句内のカラム数よりも多い場合、余分のポートは NULL 値を受け取ります。
- 出力ポートの数が SELECT 句内の列数よりも少ない場合、Integration Service によって行エラーが生成されます。
- クエリ内のパラメータのバインドの代わりに文字列の置換を使用できます。ただし、入力ポートは文字列データタイプである必要があります。

データベースへの接続

静的データベース接続を使用するか、または実行時にデータベース接続情報を SQL トランスフォーメーションに渡すことができます。

SQL トランスフォーメーションと共に、接続環境の SQL 文またはトランザクションの SQL 文を使用できます。SQL 文はリレーショナル接続オブジェクトで設定します。Integration Service は、データベースに接続するときに、接続環境 SQL を実行します。これにより、各トランザクションの開始前にトランザクションの SQL 文が実行されます。

以下のいずれかのタイプの接続を使用して、SQL トランスフォーメーションをデータベースに接続します。

- **静的接続。**セッション内に接続オブジェクトを設定します。最初に Workflow Manager で接続オブジェクトを作成する必要があります。
- **論理接続。**実行時に接続名を入力データとして SQL トランスフォーメーションに渡します。最初に Workflow Manager で接続オブジェクトを作成する必要があります。
- **完全なデータベース接続。**実行時に、接続文字列、ユーザー名、パスワードなどの接続情報を SQL トランスフォーメーションの入力ポートに渡します。

注: セッションで複数のパーティションが使用される場合、SQL トランスフォーメーションによって各パーティションごとに個別のデータベース接続が作成されます。

静的データベース接続の使用

静的接続を使用してデータベースに接続するように SQL トランスフォーメーションを設定できます。静的データベース接続は、Workflow Manager で定義されるデータベース接続です。

静的接続を使用するには、セッションを設定するときにリレーショナル接続オブジェクトを選択します。データタイプ変換エラーを回避するには、トランスフォーメーションで設定される同じデータベースタイプに対して 1 つのリレーショナル接続を使用します。

論理データベース接続の受け渡し

論理データベース接続を使用してデータベースに接続するように SQL トランスフォーメーションを設定できます。論理データベース接続は、実行時にトランスフォーメーションに渡す接続オブジェクト名です。リレーショナル接続オブジェクトを Workflow Manager で定義します。論理データベース接続を使用するようにトランスフォーメーションを設定すると、Designer によって LogicalConnectionObject 入力ポートが作成されます。

入力行ごとに論理接続を渡すことができます。接続オブジェクト名を LogicalConnectionObject ポートに渡すマッピングを設定します。データタイプ変換エラーを回避するには、トランスフォーメーションで設定される同じデータベースタイプに対して 1 つのリレーショナル接続を使用します。

フル接続情報を渡す

すべてのデータベース接続情報を入力ポートデータとして SQL トランスフォーメーションに渡すことができます。完全な接続を使用してデータベースに接続するように SQL トランスフォーメーションを設定すると、Designer によって接続コンポーネント用の入力ポートが作成されます。デフォルトで、データベースタイプはトランスフォーメーションに対して設定したデータベースタイプになります。

以下の表に、完全な接続を使用してデータベースに接続するよう SQL トランスフォーメーションを設定した場合に Designer によって作成されるポートの説明を示します。

ポート	必須/ オプション	説明
ConnectionString	必須	データベース名およびデータベースサーバー名が含まれます。
DBUser	必須	データベースに対する読み取りおよび書き込み権限を持つユーザーの名前です。
DBPasswd	必須	DBUser のパスワードです。
CodePage	オプション	Integration Service がデータベースからの読み取りおよびデータベースへの書き込みに使用するコードページです。ISO コードページ名 (ISO-8859-6 など) を使用します。コードページ名では大文字と小文字が区別されません。
AdvancedOptions	オプション	接続属性です。属性を名前-値のペアとして渡します。各属性はセミコロンで区切ります。属性名では大文字と小文字が区別されません。

接続文字列の受け渡し

ネイティブ接続文字列には、データベース名およびデータベースサーバー名が含まれます。接続文字列によって、PowerCenter およびデータベースクライアントは呼び出しを適切なデータベースに送信できます。

以下の表に、各データベースのネイティブ接続文字列の構文の一覧を示します。

データベース	接続文字列の構文	例
IBM DB2	<i>dbname</i>	mydatabase
Microsoft SQL Server	<i>servername@dbname</i>	sqlserver@mydatabase
Oracle	<i>dbname.world</i> (TNSNAMES エントリと同じ)	oracle.world
Sybase ASE ¹	<i>servername@dbname</i>	sambrown@mydatabase

データベース	接続文字列の構文	例
Teradata ²	<i>ODBC_data_source_name</i> または <i>ODBC_data_source_name@db_name</i> または <i>ODBC_data_source_name@db_user_name</i>	TeradataODBC TeradataODBC@mydatabase TeradataODBC@sambrown
Vertica	<i>ODBC_data_source_name</i>	VerticaDSN

- ¹. Sybase ASE サーバー名は、インタフェースファイルからの Adaptive Server の名前です。
². Teradata ODBC ドライバを使用して、ソースおよびターゲットデータベースに接続します。

詳細オプションの受け渡し

オプションの接続属性を設定できます。属性を設定するには、属性を名前-値のペアとして渡します。属性はセミコロンで区切ります。属性名では大文字と小文字が区別されません。

たとえば、以下の文字列を渡して接続オプションを設定する場合があります。

Use Trusted Connection = 1; Connection Retry Period = 5

以下の表に、設定可能な詳細オプションを示します。

属性	データベースタイプ	説明
接続リトライ期限	すべて	整数。 データベースへの接続が失敗した場合に Integration Service が再接続を試行する秒数です。
データソース名	Teradata	文字列。 Teradata ODBC データソースの名前です。
データベース名	Sybase ASE Microsoft SQL Server Teradata	文字列。 ODBC 内のデフォルトのデータベース名を上書きします。データベース名を入力しない場合、接続に関連するメッセージにデータベース名は表示されません。
ドメイン名	Microsoft SQL Server	文字列。 Microsoft SQL Server が実行されているドメインの名前です。
パラレルモードを有効にする	Oracle	整数。 一括モードでデータをロードする場合に、並列処理を有効化します。 0 は有効になっていないことを示します。 1 は有効。 デフォルトでは有効になっています。
オーナー名	すべて	文字列。 テーブルオーナー名です。
パケットサイズ	Sybase ASE Microsoft SQL Server	整数。 Sybase ASE および Microsoft SQL Server への ODBC 接続を最適化します。

属性	データベースタイプ	説明
サーバ名	Sybase ASE Microsoft SQL Server	文字列。 データベースサーバの名前です。
信頼接続関係を用いる	Microsoft SQL Server	整数。 有効になっている場合、Integration Service は、Windows 認証を使用して Microsoft SQL Server データベースにアクセスします。統合サービスを起動するユーザ名は、MS SQL Server データベースへのアクセス権限を持つ、有効な Windows ユーザでなければなりません。 0 は有効になっていないことを示します。 1 は有効。

データベース接続のルールおよびガイドライン

SQL トランスフォーメーションのデータベース接続を設定する場合は、以下の規則およびガイドラインを使用します。

- 異なるデータベースタイプを接続するには、PowerCenter ライセンスキーが必要です。データベースに接続するように PowerCenter がライセンスされていない場合、セッションは失敗します。
- パフォーマンスを向上させるには、静的データベース接続を使用します。動的接続を設定すると、Integration Service によって入力行ごとに新しい接続が確立されます。
- セッションで使用する接続数が制限されている場合、複数の SQL トランスフォーメーションを設定できます。異なる静的接続を使用するように各 SQL トランスフォーメーションを設定します。ルータトランスフォーメーションを使用し、行の接続性情報に基づいて行を SQL トランスフォーメーションにルーティングします。
- 完全な接続データを使用するように SQL トランスフォーメーションを設定する場合、データベースパスワードはプレーンテキストです。セッションで使用する必要がある接続数が制限されている場合、論理接続を渡すことができます。論理接続には完全な接続と同じ機能があり、データベースパスワードはセキュリティ保護されます。
- 論理データベース接続を SQL トランスフォーメーションに渡す場合、Integration Service はリポジトリにアクセスし、入力行ごとに接続情報を取得します。処理する行が多い場合、論理データベース接続を渡すとパフォーマンスに影響を与えることがあります。
- SQL トランスフォーメーションはデフォルトでネイティブデータベースタイプを使用します。ODBC を使用してセッションを実行する場合、ODBC データベースタイプでトランスフォーメーションを設定します。トランスフォーメーションに datetime または datetime2 ポートが含まれる場合、対応するネイティブタイプをタイムスタンプとして設定します。

セッション処理

Integration Service によって SQL トランスフォーメーションが処理されると、パイプライン内の SQL クエリミッドストリームが処理されます。SELECT クエリでデータベース行が取得されると、SQL トランスフォーメーションによってデータベースカラムが出力ポートに返されます。その他のタイプのクエリの場合、SQL トランスフォーメーションによってクエリ結果、パススルーデータ、またはデータベースエラーが出力ポートに返されます。

スクリプトモードで実行されるように設定された SQL トランスフォーメーションでは、常に、入力行ごとに 1 行が返されます。クエリモードで実行される SQL トランスフォーメーションでは、入力行ごとに異なる数の行が返されます。SQL トランスフォーメーションによって返される行の数は、実行されるクエリのタイプおよびクエリの成功に応じて異なります。

Integration Service がデータベースに渡して処理する SQL クエリのログを表示できます。ロギングを Verbose に設定すると、Integration Service は各 SQL クエリをセッションログに書き込みます。SQL トランスフォーメーションによってセッションのデバッグを行う必要がある場合は、ロギングを Verbose に設定します。

静的データベース接続を使用するようにトランスフォーメーションを設定する場合、SQL トランスフォーメーションでトランザクション制御を使用できます。また、クエリでコミット文およびロールバック文を発行することもできます。

SQL トランスフォーメーションには、データベース接続回復機能があります。それによって、データベースのデッドロックに対するレジリエンスが得られます。

関連項目：

- [「入力行と出力行のカーディナリティ」 \(ページ 444\)](#)
- [「高可用性」 \(ページ 442\)](#)

トランザクション制御

スクリプトモードで実行される SQL トランスフォーメーションでは、先行するソースまたはトランザクションジェネレータから入力されるトランザクション境界は削除されます。Integration Service によって、SQL トランスフォーメーションの各入力行のスクリプトの実行後にコミットが発行されます。トランザクションには、スクリプトの影響を受けた行のセットが含まれます。

クエリモードで実行される SQL トランスフォーメーションでは、データベースの接続タイプに基づいて、異なるポイントでトランザクションがコミットされます。

- **動的データベース接続。** Integration Service によって、各入力行の SQL の実行後に、コミットが発行されます。トランザクションは、スクリプトの影響を受けた行のセットです。クエリモードの動的接続では、トランザクション制御トランスフォーメーションは使用できません。
- **静的接続。** Integration Service によって、すべての入力行の処理後にコミットが発行されます。トランザクションには、更新するすべてのデータベース行が含まれます。トランザクション制御トランスフォーメーションを使用してトランザクションを制御するか、または SQL クエリでコミット文およびロールバック文を使用することによって、デフォルトの動作をオーバーライドできます。

行をコミットまたはロールバックするように SQL 文を設定する場合、トランザクションの生成トランスフォーメーションのプロパティを使用してトランザクションを生成するように、SQL トランスフォーメーションを設定してください。ユーザー定義のコミット用のセッションを設定します。

SQL トランスフォーメーションでは、以下のトランザクション制御 SQL 文は無効です。

- **SAVEPOINT。** トランザクション内のロールバックポイントを指定します。
- **SET TRANSACTION。** トランザクションオプションを変更します。

関連項目：

- [「動的接続の例」 \(ページ 459\)](#)

自動コミット

SQL トランスフォーメーションのデータベース接続を、自動コミットモードで動作するように設定することができます。自動コミットモードを設定すると、SQL 文が完了したときにコミットが発生します。

自動コミットを有効にするには、SQL トランスフォーメーションの [SQL 設定] タブで [AutoCommit] を選択します。

HA リカバリが自動コミットと共に有効になっている場合にクエリに挿入文、更新文、または削除文が含まれていると、データの整合性を保証できないという警告メッセージがセッションログに記録されます。

高可用性

高可用性オプションがある場合、SQL トランスフォーメーションによって、データベース接続回復機能が静的および動的接続に提供されます。Integration Service がデータベースに接続できない場合、接続が再試行されます。接続の接続リトライ期限を設定できます。設定した時間内に Integration Service がデータベースに接続できない場合、動的接続では行エラーが生成され、静的接続ではセッションが失敗します。

注: SQL トランスフォーメーションデータベース接続には、Informix 接続または ODBC 接続に対する回復機能はありません。

リアルタイムセッションで正確に 1 度だけ処理

Integration Service では、SQL トランスフォーメーションによってリアルタイムソースのメッセージを正確に 1 度だけ配信します。リアルタイムメッセージは SQL トランスフォーメーションに 1 度配信される必要があります。処理の中断が発生した場合、Integration Service はメッセージの再送信を要求することなくリカバリできます。メッセージが再送信されても、Integration Service はメッセージの DML 文を 2 度実行しません。Integration Service はその他の SQL 文 (SELECT または SET など) を再び実行する場合があります。

「正確に 1 度だけ処理」を実行する際に、Integration Service はチェックポイントの操作の状態を PM_REC_STATE テーブルに格納します。各 SQL トランスフォーメーションには個別の操作の状態があります。各 SQL トランスフォーメーションは一貫した状態を維持し、接続を共有しません。「正確に 1 度だけ処理」を行うには高可用性が必要です。

SQL トランスフォーメーションによってリアルタイムセッションをリカバリする際には、次のルールおよびガイドラインに従います。

- SQL トランスフォーメーションを含むセッションのリカバリを有効にするには、トランスフォーメーション範囲を「トランザクション」に設定する必要があります。
- 自動コミット、コミット文、または DDL 文を SQL クエリに含めることはできません。
- SQL トランスフォーメーションがパッシブなトランスフォーメーションである場合や、動的接続またはスク립トモードに対して設定されている場合、SQL トランスフォーメーションの HA リカバリを有効にすることはできません。
- SQL トランスフォーメーションのリカバリを有効にした場合、ワークフローの同時実行が有効になっていたり、セッションが複数のパーティションで実行されていると、セッションが失敗する可能性があります。PM_REC_STATE テーブルでデータベースのデッドロックが発生する可能性があります。

クエリモードレジリエンス

Integration Service が入力行に対して SQL クエリを実行しているときに、データベース接続エラーが発生した場合、Integration Service はデータベースへの再接続を試みます。

クエリモードで回復を図る場合には、以下の規則およびガイドラインに従ってください。

- Integration Service がリストの最初のクエリを実行しているときに、データベース接続障害が発生した場合、Integration Service はその最初のクエリを行に対して再実行します。次に、その同じ行に対して残りのクエリを実行します。
- 最初のクエリ文が SELECT 文で、パイプラインのダウンストリームで行がいくつかフラッシュされた後に通信障害が発生した場合、セッションは失敗します。

- Integration Service がリストの先頭以外のクエリを実行しているときに、接続障害が発生した場合、Integration Service はデータベースに再接続し、現在の行に対する残りのクエリをスキップします。現在よりも前の行に対するクエリ結果は失われる可能性があります。
- Integration Service がデータベースに再接続できない場合、SQL トランスフォーメーションで静的接続を使用していると、セッションは失敗します。SQL トランスフォーメーションが動的接続を使用している場合は、セッションが続行します。
- リストの SQL クエリに明示的なコミット文またはロールバック文がある場合、コミットポイント後に作成されたクエリ結果があっても、セッションが続行すると失われます。

スクリプトモード回復機能

Integration Service が入力行に対してスクリプトを実行しているときに、通信障害が発生した場合、Integration Service はデータベースへの再接続を試みます。

スクリプトモードで回復を図る場合には、以下の規則およびガイドラインに従ってください。

- Integration Service がデータベースへの接続に失敗し、接続が静的である場合、セッションは失敗します。SQL トランスフォーメーションが動的接続を使用している場合は、セッションが続行します。
- データベースに再接続した場合、Integration Service は現在の行の処理をスキップし、次の行に進みます。

データベースデッドロック回復機能

[デッドロック] セッションプロパティで [セッションリトライ] を有効にした場合、SQL トランスフォーメーションはデータベースデッドロックエラーに対する復元性があります。SQL トランスフォーメーションは、クエリモードではデータベースデッドロックエラーに対する復元性がありますが、スクリプトモードではデッドロックエラーに対する復元性がありません。デッドロックがクエリモードで発生した場合、Integration Service は設定したデッドロックリトライ回数だけデータベースへの再接続を試みます。デッドロックリトライ回数およびデッドロックスリープタイム周期を指定するように、Integration Service を設定します。

デッドロックが発生すると、Integration Service は現在の行に DML 文がない場合は現在の行の SQL 文を再試行します。行に INSERT、UPDATE、DELETE などの DML 文が含まれている場合、Integration Service は現在の行を再び処理しません。

動的接続の場合に再試行が失敗すると、Integration Service は SQL Error ポートでエラーを返します。Integration Service は、[行内の SQL エラー時でも処理を継続する] プロパティに基づいて次の文を処理します。このプロパティが無効になっている場合、Integration Service は現在の行をスキップします。現在の行に INSERT、UPDATE、DELETE などの DML 文が含まれている場合、Integration Service はエラーカウントを 1 つ増やします。

静的接続の場合に再試行が失敗すると、Integration Service は SQL Error ポートでエラーを返します。現在の行に DML 文が含まれている場合、Integration Service はそのセッションに失敗します。Integration Service は、[行内の SQL エラー時でも処理を継続する] プロパティに基づいて次の文を処理します。このプロパティが無効になっている場合、Integration Service は現在の行をスキップします。

SQL クエリログ

Integration Service がデータベースに渡して処理する SQL クエリのログを表示できます。ロギングを Verbose に設定すると、Integration Service は各 SQL クエリをセッションログに書き込みます。SQL トランスフォーメーションによってセッションのデバッグを行う必要がある場合は、ロギングを Verbose に設定します。

クエリに CLOB または BLOB データが含まれる場合、セッションログにクエリは含まれません。セッションログには、CLOB データなどのデータを示すメッセージが含まれています。

入力行と出力行のカーディナリティ

Integration Service によって SELECT クエリーが実行されると、SQL トランスフォーメーションによって、取得した行ごとに 1 行が返されます。クエリーでデータが取得されない場合、SQL トランスフォーメーションによって、入力行ごとにゼロまたは 1 行が返されます。

SQL トランスフォーメーションが返す出力行の数は、以下の要因によって異なります。

- **クエリ文の処理。**クエリに SELECT 文が含まれる場合、Integration Service は複数の出力行を取得できます。SELECT クエリーが成功した場合、SQL トランスフォーメーションによって複数の行が取得されることがあります。クエリーに他の文が含まれる場合、Integration Service によって、SQL エラーまたは影響を受けた行の数が含まれる行が生成されることがあります。
- **ポート設定。**NumRowsAffected 出力ポートには、1 つの入力行の更新、挿入、または削除によって影響を受けた行数の合計が含まれます。SQL トランスフォーメーションにパススルーポートが設定されている場合、トランスフォーメーションは各ソース行について最低 1 回カラムデータを返します。
- **最大行数の設定。**「最大出力行数」によって、SQL トランスフォーメーションで SELECT クエリから返される行数が制限されます。
- **エラー行。**Integration Service によって接続エラーまたは構文エラーが検出されると、行エラーが返されます。SQL トランスフォーメーションは、クエリーモードで動作している場合、SQLException ポートにエラーを返します。SQL トランスフォーメーションは、スクリプトモードで動作している場合、ScriptError ポートにエラーを返します。
- **SQL エラー時の継続。**SQL 文にエラーがあっても処理を継続するように SQL トランスフォーメーションを設定できます。SQL トランスフォーメーションは、行エラーを生成しません。

クエリ文の処理

クエリのタイプによって、SQL トランスフォーメーションで返される行数が決定されます。クエリーモードで実行される SQL トランスフォーメーションでは、ゼロ、1、または複数の行が返されることがあります。クエリに SELECT 文が含まれる場合、SQL トランスフォーメーションによってデータベースの各列が出力ポートに返されます。該当するすべての行がトランスフォーメーションによって返されます。

以下の表に、クエリーモードでエラーが発生しない場合に、さまざまなタイプのクエリ文について SQL トランスフォーメーションによって生成される出力行を示します。

クエリ文	出力行
UPDATE、INSERT、DELETE のみ	ゼロ行。
1 つ以上の SELECT 文	取得されたデータベース行の合計数。
CREATE、DROP、TRUNCATE などの DDL クエリ	ゼロ行。

影響を受けた行数

NumRowsAffected 出力ポートを有効にすると、各入力行に指定された INSERT、UPDATE、または DELETE クエリー文の影響を受けた行数を返すことができます。Integration Service は、クエリーの文ごとに NumRowsAffected を返します。NumRowsAffected はデフォルトでは無効になっています。

クエリーモードで NumRowsAffected を有効にし、SQL クエリに INSERT、UPDATE、または DELETE 文が含まれない場合、各出力行の NumRowsAffected はゼロになります。

注: NumRowsAffected を有効にし、トランスフォーメーションがスクリプトモードで実行されるように設定されている場合、NumRowsAffected は常に NULL です。

以下の表に、クエリモードで NumRowsAffected を有効にした場合に、SQL トランスフォーメーションによって生成される出力行を示します。

クエリ文	出力行
UPDATE、INSERT、DELETE のみ	文ごとにその文の NumRowsAffected が含まれる 1 行。
1 つ以上の SELECT 文	取得されたデータベース行の合計数。 各行の NumRowsAffected はゼロです。
CREATE、DROP、TRUNCATE などの DDL クエリー	ゼロの NumRowsAffected が含まれる 1 行。

SQL トランスフォーメーションがクエリーモードで動作し、クエリーに複数の文が含まれている場合、Integration Service は文ごとに NumRowsAffected を返します。NumRowsAffected には、入力行に指定された INSERT、UPDATE、DELETE の各文の影響を受ける行の合計が含まれます。

たとえば、クエリーに以下の文が含まれています。

```
DELETE from Employees WHERE Employee_ID = '101';  
SELECT Employee_ID, LastName from Employees WHERE Employee_ID = '103';  
INSERT into Employees (Employee_ID, LastName, Address)VALUES ('102', 'Gein', '38 Beach Rd')
```

DELETE 文は、1 行に影響を与えます。SELECT 文は、どの行にも影響を与えません。INSERT 文は、1 行に影響を与えます。

Integration Service は、DELETE 文から 1 行返します。NumRowsAffected は 1 になります。SELECT 文から 1 行返しますが、NumRowsAffected はゼロです。INSERT 文から 1 行返し、NumRowsAffected は 1 になります。

以下の条件がすべて真の場合、NumRowsAffected ポートによってゼロが返されます。

- データベースが Informix です。
- トランスフォーメーションがクエリーモードで実行されています。
- クエリーにパラメータが含まれません。

最大出力行数

SELECT クエリーで SQL トランスフォーメーションによって返される行数を制限できます。[最大出力行数] プロパティを設定して、行数を制限します。クエリーに複数の SELECT 文が含まれる場合、SQL トランスフォーメーションによって、すべての SELECT 文からの合計の行数が制限されます。

たとえば、[最大出力行数] を 100 に設定します。クエリーに以下の 2 つの SELECT 文が含まれるとします。

```
SELECT * FROM table1; SELECT * FROM table2;
```

最初の SELECT 文によって 200 行が返され、2 番目の SELECT 文によって 50 行が返される場合、SQL トランスフォーメーションによって最初の SELECT 文から 100 行が返されます。2 番目の SELECT 文からは行は返されません。

出力行が制限されないように設定するには、[最大出力行数] をゼロに設定します。

エラー行について

Integration Service によって接続エラーまたは構文エラーが検出されると、行エラーが返されます。SQL トランスフォーメーションには、エラーテキストを出力する以下のデフォルトのポートがあります。

- **SQLException**。SQL トランスフォーメーションがクエリモードで実行される場合に、データベースエラーを返します。
- **ScriptError**。SQL トランスフォーメーションがスクリプトモードで実行される場合に、データベースエラーを返します。

SQL クエリに構文エラーがある場合、エラーポートにはデータベースからのエラーテキストが含まれます。たとえば、以下の SQL クエリでは、Oracle データベースからの行エラーが生成されます。

SELECT Product_ID FROM Employees

Employees テーブルには Product_ID はありません。Integration Service によって 1 行が生成されます。SQLException ポートに、以下のエラーテキストが 1 行で含まれます。

ORA-0094: "Product_ID": invalid identifier Database driver error... Function Name: Execute SQL Stmt: SELECT Product_ID from Employees Oracle Fatal Error

クエリに複数の文が含まれ、SQL エラー時でも処理を継続するように SQL トランスフォーメーションを設定した場合、SQL トランスフォーメーションでは、1 つのクエリ文に対してデータベースから行が返されますが、別のクエリ文に対してデータベースエラーが返されることがあります。SQL トランスフォーメーションは、データベースエラーを別の行に返します。

パススルーポートまたは NumRowsAffected ポートを設定した場合、SQL トランスフォーメーションは各ソース行について最低 1 行のデータを返します。クエリから行が返されない場合、SQL トランスフォーメーションはパススルーデータと NumRowsAffected 値を返しますが、出力カラムには NULL 値を返します。NULL 値の行を削除するには、フィルタトランスフォーメーション経由で出力行を渡します。

以降の表では、SQL トランスフォーメーションによって返される出力行について、クエリ文の種類ごとに説明します。

以下の表では、UPDATE、INSERT、または DELETE の各クエリ文について、SQL トランスフォーメーションが生成する行を示します。

設定されている NumRowsAffected ポートまたは パススルーポート	SQLException	出力される行
いずれのポートも設定されていません。	いいえ	ゼロ行。
いずれのポートも設定されていません	はい	SQLException ポートにエラーを含む 1 行
どちらかが設定済み	いいえ	クエリ文ごとに、NumRowsAffected またはパススルーのカラムデータを含む 1 行。
どちらかが設定済み	はい	SQLException ポート、NumRowsAffected ポート、またはパススルーポートのデータにエラーを含む 1 行。

以下の表では、SELECT 文について SQL トランスフォーメーションが生成する出力行の数を示します。

設定されている NumRowsAffected ポートまたは パススルーポート	SQLError	出力される行
いずれのポートも設定されていません	いいえ	ゼロ以上の行。各 SELECT 文で返された行によって異なります。
いずれのポートも設定されていません	はい	正常に終了した文の出力行の合計よりも大きい 1 行。最後の行には、SQLError ポートにエラーが含まれます。
どちらかが設定済み	いいえ	1 つ以上の行。各 SELECT 文で返された行によって異なります。 <ul style="list-style-type: none"> - NumRowsAffected が有効な場合、各行には値 0 の NumRowsAffected カラムが含まれます。 - パススルーポートが設定されている場合、各行にはパススルーカラムデータが含まれます。クエリによって複数の行が返された場合、パススルーカラムデータは各行について重複して生成されます。
どちらかが設定済み	はい	1 つ以上の行。各 SELECT 文で返された行によって異なります。最後の行には、SQLError ポートのエラーが含まれます。 <ul style="list-style-type: none"> - NumRowsAffected が有効な場合、各行には値 0 の NumRowsAffected カラムが含まれます。 - パススルーポートが設定されている場合、各行にはパススルーカラムデータが含まれます。クエリによって複数の行が返された場合、パススルーカラムデータは各行について重複して生成されます。

以下の表では、CREATE、DROP、TRUNCATE などの DDL クエリについて、SQL トランスフォーメーションが生成する出力行の数を示します。

設定されている NumRowsAffected ポートまたは パススルーポート	SQLError	出力される行
いずれのポートも設定されていません	いいえ	- ゼロ行。
いずれのポートも設定されていません	はい	- SQLError ポートのエラーを含む 1 行。
どちらかが設定済み	いいえ	- 値 0 の NumRowsAffected カラムおよびパススルーカラムデータを含む 1 行
どちらかが設定済み	はい	- SQLError ポートのエラー、値 0 の NumRowsAffected カラム、およびパススルーカラムデータを含む 1 行

SQL エラー時の処理の継続

[行内の SQL エラー時でも処理を継続する] オプションを有効にすることによって、文の SQL エラーを無視できます。Integration Service によって、行の残りの SQL 文の実行が継続されます。Integration Service によって行エラーは生成されません。ただし、SQLError ポートには、失敗した SQL 文とエラーメッセージが含まれます。行のエラー数がセッションエラーのしきい値を超えた場合、セッションは失敗します。

たとえば、クエリーに以下の文がある場合があります。

```
DELETE FROM Persons WHERE FirstName = 'Ed';
```

```
INSERT INTO Persons (LastName, Address)VALUES ('Gein', '38 Beach Rd')
```

DELETE 文が失敗した場合、SQL トランスフォーメーションによってデータベースからエラーメッセージが返されます。Integration Service によって INSERT 文の処理は継続されます。

ヒント: データベースエラーをデバッグするには、[行内の SQL エラー時でも処理を継続する] オプションを無効にします。このオプションを無効にしない場合、エラーと発生元のクエリー文を関連付けることができないことがあります。

SQL トランスフォーメーションプロパティ

SQL トランスフォーメーションの作成後、以下のトランスフォーメーションのタブで、ポートを定義し、属性を設定できます。

- **Ports.** [SQL ポート] タブで作成したトランスフォーメーションのポートおよび属性が表示されます。
- **プロパティ.** SQL トランスフォーメーションの全般的なプロパティです。
- **SQL 設定.** SQL トランスフォーメーションに固有の属性です。
- **SQL ポート.** SQL トランスフォーメーションのポートおよび属性です。

注: [ポート] タブでカラムを更新することはできません。 [SQL ポート] タブでポートを定義すると、[ポート] タブにポートが表示されます。

[プロパティ] タブ

[プロパティ] タブでは、SQL トランスフォーメーションの全般的なプロパティを設定します。トランスフォーメーションの一部のプロパティは、SQL トランスフォーメーションには適用されないか、または設定できません。

以下の表では、SQL トランスフォーメーションのプロパティについて説明します。

プロパティ	説明
実行時位置	DLL または共有ライブラリの格納場所。 SQL トランスフォーメーションセッションを実行する Integration Service ノードへの相対パスを入力します。 このプロパティが空白の場合、Integration Service は、Integration Service ノードで定義されている環境変数を使用して DLL または共有ライブラリの位置を探します。 Integration Service ノードで定義されている実行時位置または環境変数に、すべての DLL または共有ライブラリをコピーする必要があります。DLL、共有ライブラリ、または参照されるファイルが見つからない場合、Integration Service は手続きのロードに失敗します。
トレースレベル	このトランスフォーメーションを含むセッションを実行したときにセッションログに記録される情報の詳細度を設定します。SQL トランスフォーメーションのトレースレベルを [Verbose Data] に設定した場合、Integration Service によって準備された各 SQL クエリがセッションログに書き込まれます。

プロパティ	説明
IsPartitionable	<p>パイプラインに複数のパーティションがあると、このトランスフォーメーションを使用できます。次のオプションを使用します。</p> <ul style="list-style-type: none"> - いいえ。トランスフォーメーションはパーティション化できません。同一パイプライン内のこのトランスフォーメーションおよびその他のトランスフォーメーションは、1つのパーティションに含まれる必要があります。データクレンジングなど、トランスフォーメーションによりすべての入力データが一度に処理される場合は、[いいえ]を選択する場合があります。 - ローカルで。トランスフォーメーションをパーティション化することはできますが、同じノード上のパイプラインですべてのパーティションが実行される必要があります。トランスフォーメーションの異なるパーティションがメモリ内でオブジェクトを共有する必要がある場合、[ローカルで]を選択します。 - グリッドをまたがる。トランスフォーメーションをパーティション化することができ、各パーティションは異なるノードに配分されます。 <p>デフォルトは [No]。</p>
アップデートストラテジトランスフォーメーション	<p>このトランスフォーメーションは、出力行の更新方式を定義します。このプロパティは、クエリモードの SQL トランスフォーメーションで有効にできます。デフォルトでは無効になっています。</p>
トランスフォーメーション範囲	<p>Integration Service が入力データに対してトランスフォーメーションロジックを適用する方法です。次のオプションを使用します。</p> <ul style="list-style-type: none"> - 行 - トランザクション - すべての入力 <p>静的クエリモードでトランザクション制御を使用する場合、トランザクション範囲をトランザクションに設定します。</p> <p>スクリプトモードのトランスフォーメーションの場合、デフォルトは [Row] です。クエリモードのトランスフォーメーションの場合、デフォルトは [すべての入力] です。</p>
出力が再現可能	<p>出力データの順序をセッションの実行ごとに一致させるかどうかを指定します。</p> <ul style="list-style-type: none"> - Never。出力データの順序はセッションの実行ごとに異なります。 - 入力順による。入力データの順序がセッションの実行ごとに一致している場合、出力順序をセッションの実行ごとに一致させます。 - Always。入力データの順序がセッションの実行ごとに異なる場合でも、出力データの順序は常に同じです。 <p>デフォルトは [Never] です。</p>
トランザクションの生成	<p>このトランスフォーメーションは、トランザクション行を生成します。SQL クエリでデータをコミットするクエリモードの SQL トランスフォーメーションの場合、このプロパティを有効にします。デフォルトでは無効になっています。</p>
パーティションごとに1つのスレッドを要求します	<p>Integration Service によってプロシージャの各パーティションが1つのスレッドで処理される場合に指定します。</p>
出力が確定的かどうか	<p>トランスフォーメーションは、セッションを実行するたびに一致する出力データを生成します。このトランスフォーメーションを使用するセッションでリカバリを実行するには、このプロパティを有効にします。デフォルトでは有効になっています。</p>

警告: トランスフォーメーションを繰り返し可能で一意に定まるものとして設定する場合は、データが繰り返し可能で一意に定まることを保証する必要があります。セッションとリカバリで同じデータが生成されないト

ランスフォーメーションを使用してセッションをリカバリしようとする、リカバリプロセスを実行した結果、データが破損する可能性があります。

[SQL 設定] タブ

[SQL 設定] タブでは、SQL トランスフォーメーションの属性を設定します。SQL 属性は、SQL トランスフォーメーションに固有です。

以下の表に、[SQL 設定] タブで設定できる属性を示します。

オプション	説明
行内の SQL エラー時でも処理を継続する	SQL エラーの発生後、クエリー内の残りの SQL 文の処理を継続します。
統計出力ポートの追加	NumRowsAffected 出力ポートを追加します。ポートは、入力行に指定された INSERT、DELETE、UPDATE の各クエリー文の影響を受けたデータベース行の総数を返します。
AutoCommit	各データベース接続に対する自動コミットを有効にします。クエリ内の各 SQL 文によってトランザクションが定義されます。コミットは、SQL 文が終了するかまたは次の文が実行されるか、いずれか早い方で実行されます。
最大出力行数	SQL トランスフォーメーションで SELECT クエリーから出力できる最大行数を定義します。行が制限されないように設定するには、[最大出力行数] をゼロに設定します。
スクリプトロケール	SQL スクリプトのコードページを指定します。リストからコードページを選択します。デフォルトは、オペレーティングシステムのロケールです。
接続プールの使用	データベース接続用に接続プールを保持します。接続プールは、動的接続のときのみ使用できます。
プール内の最大接続数	接続プールでの最大接続可能数。最小接続数は 1。最大値は 20 です。デフォルトは 10 です。

[SQL ポート] タブ

SQL トランスフォーメーションを作成すると、トランスフォーメーションの設定に応じて、Designer によってデフォルトのポートが追加されます。トランスフォーメーションを作成した後、[SQL ポート] タブでポートをトランスフォーメーションに追加できます。

以下の表に、SQL トランスフォーメーションのポートを示します。

ポート	タイプ	モード	説明
ScriptName	入力	スクリプト	各行に対して実行するスクリプトの名前。
ScriptError	出力	スクリプト	スクリプトが失敗したときにデータベースがトランスフォーメーションに返す SQL エラー。
ScriptResult	出力	スクリプト	クエリ実行の結果。クエリが正常に実行された場合は、PASSED です。クエリが失敗した場合は、FAILED です。

ポート	タイプ	モード	説明
SQLException	出力	クエリ	データベースがトランスフォーメーションに返す SQL エラー。
LogicalConnectionObject	入力	クエリ	動的接続。Workflow Manager 接続で定義される接続の名前。
接続文字列	入力	クエリスクリプト	接続オブジェクトのみ。データベース名およびデータベースサーバー名。
DBUser	入力	クエリスクリプト	フル接続のみ。データベースに対する読み取りおよび書き込み権限を持つユーザーの名前。
DBPasswd	入力	クエリスクリプト	フル接続のみ。データベースパスワード。
CodePage	入力	クエリスクリプト	フル接続のみ。Integration Service がデータベースからの読み取りおよびデータベースへの書き込みに使用するコードページ。
詳細オプション	入力	クエリスクリプト	フル接続のみ。オプションの接続属性。パケットサイズ、接続リトライ期限、パラレルモードを有効にするなど。
NumRowsAffected	出力	クエリスクリプト	入力行の INSERT、DELETE、および UPDATE クエリ文の影響を受けたデータベース行の合計数。

SQL 文

以下の表に、SQL トランスフォーメーションで利用できる文を示します。

文のタイプ	文	説明
データ定義	ALTER	データベースの構造を変更します。
データ定義	COMMENT	データディクショナリにコメントを追加します。
データ定義	CREATE	データベース、テーブル、またはインデックスを作成します。
データ定義	DROP	インデックス、テーブル、またはデータベースを削除します。
データ定義	RENAME	データベースオブジェクトの名前を変更します。
データ定義	TRUNCATE	テーブルからすべての行を削除します。

文のタイプ	文	説明
データ操作	CALL	PL/SQL または Java サブプログラムを呼び出します。
データ操作	DELETE	テーブルから行を削除します。
データ操作	EXPLAIN PLAN	データベースの Explain テーブルに文のアクセスプランを書き込みます。
データ操作	INSERT	行をテーブルに挿入します。
データ操作	LOCK TABLE	アプリケーションプロセスが同時にテーブルを使用または変更することを防止します。
データ操作	MERGE	ソースデータを使用してテーブルを更新します。
データ操作	SELECT	データベースからデータを取得します。
データ操作	UPDATE	テーブルの行の値を更新します。
データ制御言語	GRANT	データベースユーザに特権を付与します。
データ制御言語	REVOKE	データベースユーザのアクセス特権を削除します。
トランザクションコントロール	コミット	作業ユニットを保存し、その作業ユニットのデータベース変更を実行します。
トランザクションコントロール	ROLLBACK	最後の COMMIT 以降のデータベースへの変更を取り消します。

SQL トランスフォーメーションの作成

SQL トランスフォーメーションは、Transformation Developer または Mapping Designer で作成できます。

SQL トランスフォーメーションを作成するには：

1. [トランスフォーメーション] - [作成] をクリックします。
2. [SQL トランスフォーメーション] を選択します。
3. トランスフォーメーションの名前を入力します。
SQL トランスフォーメーションの命名規則は、「SQL_トランスフォーメーション名」です。
4. トランスフォーメーションの説明を入力して、[作成] をクリックします。
5. SQL トランスフォーメーションのモードを設定します。
 - **クエリモード**。動的 SQL クエリを実行するアクティブなトランスフォーメーションを設定します。
 - **スクリプトモード**。外部 SQL スクリプトを実行するパッシブなトランスフォーメーションを設定します。
6. SQL トランスフォーメーションの接続先のデータベースタイプを設定します。データベースタイプをリストから選択します。

7. SQL トランスフォーメーションの接続オプションを設定します。

オプション	説明
静的接続	セッション用に設定した接続オブジェクトを使用します。SQL トランスフォーメーションは、セッション中に一度だけデータベースに接続します。
動的接続	マッピングでトランスフォーメーションに渡す接続情報に応じてデータベースに接続します。動的接続を設定する場合、トランスフォーメーションが接続オブジェクト名を必要とするか、またはトランスフォーメーションがすべての接続情報を必要とするかを選択します。デフォルトは接続オブジェクトです。
接続オブジェクト	動的接続のみ。LogicalConnectionObject ポートを使用して接続オブジェクト名を受け取ります。接続オブジェクトを Workflow Manager の接続で定義します。
フル接続情報	動的接続のみ。入力ポートを使用して、すべての接続コンポーネントを受け取ります。

8. SQL トランスフォーメーションをパッシブモードで実行するように設定するには、[SQL トランスフォーメーションをパッシブモードで実行] を選択します。
9. [OK] をクリックしてトランスフォーメーションを設定します。
- 選択したオプションに基づいて、Designer によりトランスフォーメーションにデフォルトのポートが作成されます。データベースタイプ以外の設定を変更することはできません。
10. トランスフォーメーションにポートを追加するには、[ポート] タブをクリックします。パススルーポートは、データベースポートの後に追加します。

第 27 章

マッピングにおける SQL トランスフォーメーションの使用

この章では、以下の項目について説明します。

- [SQL トランスフォーメーションの例の概要, 454 ページ](#)
- [動的更新の例, 454 ページ](#)
- [動的接続の例, 459 ページ](#)

SQL トランスフォーメーションの例の概要

SQL トランスフォーメーションによって、パイプライン内の SQL クエリの間 streams が処理されます。トランスフォーメーションでは、外部 SQL スクリプトまたは SQL エディタで作成した SQL クエリが処理されます。実行時に、データベース接続情報を SQL トランスフォーメーションに入力データとして渡すことができます。

この章では、SQL トランスフォーメーションの機能を、2 つの例を挙げて説明します。この章に挙げられている例を使用して、動的 SQL クエリを作成、実行し、動的にデータベースに接続しましょう。この章には、マッピングに含めることの可能なトランスフォーメーションのサンプルデータおよび説明が掲載されています。

この章で挙げられている例は、次の 2 つです。

- **データベースを更新する動的 SQL クエリの作成。** 動的なクエリ更新の例は、ソースファイルから受け取った価格コードに基づいてテーブル内の製品価格を更新する方法を示しています。
- **動的データベース接続の設定。** 動的接続の例を挙げて、ソース行内の顧客サイトの値に応じて異なるデータベースに接続する方法を示します。

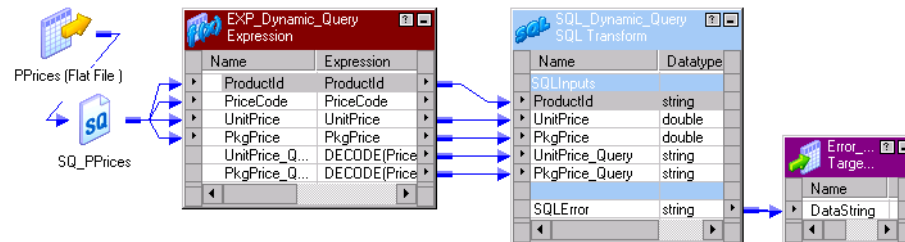
動的更新の例

ここでは、式トランスフォーメーションおよび SQL トランスフォーメーションを、ソースファイル内のカラムの値に基づいて SQL クエリを生成するように設定する方法を例示します。

この例では、製品価格を格納するデータベーステーブルがあるとします。トランザクションファイルから価格を更新する必要があります。各トランザクション行では、データベース内の卸売価格、小売価格、または製造価格が、価格コードカラムに基づいて更新されます。

ソースファイルはフラットファイルです。各ソース行内の価格コードカラムの値に基づいて更新されるカラム名を返すように、式トランスフォーメーションを設定することができます。そのカラム名は、式トランスフォーメーションから SQL トランスフォーメーションに渡されます。SQL トランスフォーメーションは、受け取られたカラム名に基づいて Prod_Cost テーブル内のカラムを更新する動的 SQL クエリを実行し、データベースエラーを Error_File ターゲットに返します。

以下の図に、式トランスフォーメーションが SQL トランスフォーメーションにカラム名をどのように渡すかを示します。



マッピングは、以下のようなコンポーネントから構成されています。

- **PPrices ソース定義。** PPrices フラットファイルには、製品 ID、パッケージ価格、単価、および価格コードが含まれています。パッケージ価格および単価が卸売価格、小売価格、または製造価格のどれに該当するかは、価格コードで定義されます。
- **Error_File フラットファイルターゲット定義。** SQL トランスフォーメーションからのデータベースエラーを受け取る Datastring フィールドが、ターゲットに含まれています。
- **Exp_Dynamic_Expression トランスフォーメーション。** 式トランスフォーメーションは、PriceCode カラムの値に基づいて更新対象の Prod_Cost カラム名を定義し、そのカラム名を UnitPrice_Query ポートおよび PkgPrice_Query ポート内に返します。
- **SQL_Dynamic_Query トランスフォーメーション。** SQL トランスフォーメーションは、Prod_Cost テーブル内の UnitPrice カラムおよび PkgPrice カラムを更新する動的 SQL クエリであり、UnitPrice_Query カラムおよび PkgPrice_Query カラムで名前が指定されたカラムを更新します。

注: マッピングには、Prod_Cost テーブル用のリレーショナルテーブル定義は含まれません。SQL トランスフォーメーションは、Prod_Cost テーブルが格納されているデータベースに静的に接続します。テーブル内の単価およびパッケージ価格を更新する SQL 文を生成します。

ソースファイルの定義

トランザクションファイルは、データベース内で更新される価格が格納されているフラットファイルソースです。各行には、価格が卸売価格、小売価格、または製造価格のどれに該当するかを定義するコードが含まれます。ソースファイル名は PPrices.dat です。

次の行を含む PPrices.dat ファイルを、Srcfiles 内に作成することができます。

```
100,M,100,110
100,W,120,200
100,R,130,300
200,M,210,400
200,W,220,500
200,R,230,600
300,M,310,666
300,W,320,680
300,R,330,700
```

PPrices.dat ファイルをインポートして、PPrices ソースファイルをリポジトリ内に作成することができます。

PPrices ファイル内には、次のようなカラムがあります。

カラム	データタイプ	精度	説明
ProductID	String	10	更新される製品を識別する一意の番号。
PriceCode	String	2	M、W、または R。価格を製造価格、卸売価格、または小売価格のいずれかとして定義します。
UnitPrice	番号	10	製品の各ユニットの価格。
PkgPrice	番号	10	製品のパッケージ価格。

ターゲット定義の作成

Error_File は、SQL トランスフォーメーションからデータベースエラーメッセージを受け取るフラットファイルターゲット定義です。

以下は、Error_File 定義に含まれる入力ポートです。

ポート名	データ型	精度	スケール
DataStream	String	4000	0

Target Designer s で、ターゲット定義を作成します。

データベーステーブルの作成

SQL トランスフォーメーションによって、製品価格が Prod_Cost リレーショナルテーブルに書き込まれます。Prod_Cost テーブルには、各製品の単価やパッケージ価格が含まれ、ユニット別/パッケージ別に卸売価格、小売価格、および製造価格が格納されます。

Prod_Cost テーブル用のリレーショナルターゲット定義は、リポジトリ内に作成しないでください。データベース内のテーブルが更新する SQL 文は、SQL トランスフォーメーションで生成されます。

Prod_Cost テーブルは、次のようなカラムから構成されています。

コストのタイプ	データタイプ	説明
ProductID	varchar	更新される製品を識別する一意の番号。
WUnitPrice	番号	卸売単価。
WPkgPrice	番号	卸売パッケージ価格。
RUnitPrice	番号	小売単価。
RPkgPrice	番号	小売パッケージ価格。
MUnitPrice	番号	製造単価。
MPkgPrice	番号	製造パッケージ価格。

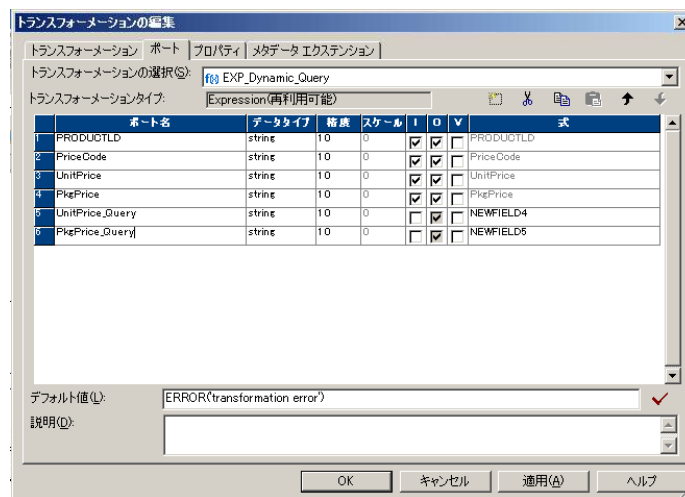
次の SQL 文により、Prod_Cost テーブルおよび 3 つの製品行が Oracle データベースに作成されます。

```
Create table Prod_Cost (ProductId varchar (10), WUnitPrice number, WPkgPrice number, RUnitPrice number,
RpkgPrice number,MUnitPrice number, MPkgPrice number );
insert into Prod_Cost values('100',0,0,0,0,0,0);
insert into Prod_Cost values('200',0,0,0,0,0,0);
insert into Prod_Cost values('300',0,0,0,0,0,0);
commit;
```

式トランスフォーメーションの設定

式トランスフォーメーションには、各ソースカラムに対応した入出力ポートが備わっています。PriceCode カラムの値に応じたカラム名を SQL トランスフォーメーションに渡します。

以下の図に、カラム名を返す式トランスフォーメーション内のポートを示します。



SQL トランスフォーメーションには、式の結果を格納する以下のようなカラムがあります。

- **UnitPrice_Query**。価格コードが"M"、"R"または"W"のどれに該当するかに基づいて、カラム名 ("MUnitPrice"、"RUnitPrice"、または"WUnitPrice") が返されます。
DECODE(PriceCode,'M','MUnitPrice','R','RUnitPrice','W','WUnitPrice')
- **PkgPrice_Query**。価格コードが"M"、"R"または"W"のどれに該当するかに基づいて、カラム名 ("MPkgPrice"、"RPkgPrice"、または"WPkgPrice") が返されます。
DECODE(PriceCode,'M','MPkgPrice','R','RPkgPrice','W','WPkgPrice')

SQL トランスフォーメーションの定義

単価およびパッケージ価格データを Prod_Cost テーブルに挿入する動的 SQL クエリは、SQL トランスフォーメーションによって実行されます。SQL トランスフォーメーションの UnitPrice_Query ポートおよび PkgPrice_Query ポート内には、更新対象のカラム名が受け取られます。

SQL トランスフォーメーションの作成時には、トランスフォーメーションモード、データベースタイプ、および接続タイプを定義します。モードまたは接続タイプは、SQL トランスフォーメーションの作成後に変更不能になります。

SQL トランスフォーメーションの作成には、次のプロパティが使用されます。

- **クエリモード**。SQL トランスフォーメーションによって、動的 SQL クエリが実行されます。
- **静的接続**。SQL トランスフォーメーションは、Workflow Manager 内で定義されている接続オブジェクトを使用して、データベースに 1 回接続します。

以下の図は、SQL トランスフォーメーションの「ポート」タブを示しています。[SQL クエリ] と [クエリの説明] があります。

ポート名	データタイプ	ネイティブタイプ	精度	スケール	I	O
SQLInputs						
ProductId	string	varchar	10	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
UnitPrice	double	float	15	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
PkgPrice	double	float	15	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
UnitPrice_Query	string	varchar	10	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
PkgPrice_Query	string	varchar	10	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
SQLOutputs						
SQL_Error	string		4096	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>

デフォルト値:
 説明:
 SQLクエリ: `Update Prod_Cost set ~UnitPrice_Query= ?UnitPrice?, ~PkgPrice_Query= ?PkgPrice? where ProductId = ?`
 クエリ説明: `UnitPrice_Column_Type is either WUnitPrice, RUnitPrice, or MUnitPrice
 PkgPrice_Column_Type is either WPkgPrice, RPkgPrice, MPkgPrice
 UnitPrice contains the UnitPrice value
 PkgPrice contains the PkgPrice value`

SQL トランスフォーメーションは、UnitPrice_Query ポートおよび PkgPrice_Query ポート内に受け取られたカラム名に基づいて Prod_Cost テーブル内の PkgPrice カラム 1 つと UnitPrice カラム 1 つを更新する、動的 SQL クエリを備えています。

SQL トランスフォーメーションでのクエリーを次に挙げます。

```
Update Prod_Cost set ~UnitPrice_Query= ?UnitPrice?, ~PkgPrice_Query= ?PkgPrice? where ProductId = ? ProductId?;
```

SQL トランスフォーメーションによって、UnitPrice_Query および PkgPrice_Query 文字列変数が更新対象カラム名に置換され、

クエリー内の ProductId、UnitPrice、および PkgPrice パラメータが、対応するポート内に受け取られるデータにバインドされます。

例として、製品 100 の単価およびパッケージ価格を含むソース行を次に挙げます。

```
100,M,100,110
```

PriceCode が "M" なら、価格は製造価格です。更新対象の MUnitprice および MPkgPrice カラム名は、式トランスフォーメーションから SQL トランスフォーメーションに渡されます。

SQL トランスフォーメーションによって、次のクエリーが実行されます。

```
Update Prod_Cost set MUnitprice = 100, MPkgPrice = 110 where ProductId = '100';
```

製品 100 の卸売価格を含むソース行は、次のとおりです。

```
100,W,120,200
```

WUnitprice および WPkgPrice のカラム名は、式トランスフォーメーションから SQL トランスフォーメーションに渡されます。SQL トランスフォーメーションによって、次のクエリーが実行されます。

```
Update Prod_Cost set WUnitprice = 120, WPkgPrice = 200 where ProductId = '100';
```

セッション属性の設定

セッションの設定時には、ソースファイル、ターゲットファイル、およびデータベース接続を設定します。

属性	値
ソースファイル名	PPrices.dat
Output File Name	ErrorFile.dat
SQL_Dynamic_Query リレーショナル接続	Prod_Cost テーブルが格納されているテストデータベースへの接続。

ターゲットデータの結果

サンプルデータを使ってセッションを実行すると、次のデータが Prod_Cost ターゲットテーブルに移入されます。

ProductID	WUnitPrice	WPkgPrice	RUnitPrice	RPkgPrice	MUnitPrice	MPkgPrice
100	120	200	130	300	100	110
200	220	500	230	600	210	400
300	320	680	330	700	310	666

何らかのエラーがデータベースから SQL トランスフォーメーションに返された場合、エラーテキストが Error_File ターゲットに格納されます。

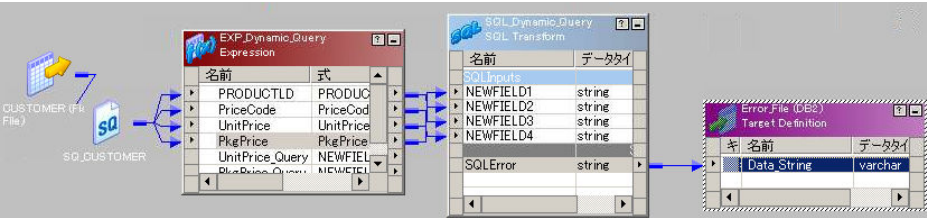
動的接続の例

動的接続 SQL トランスフォーメーションの例は、ソースファイルデータに基づいてデータベースに動的に接続する方法を示しています。

この例では、米国、英国、およびカナダ用の顧客データベースがあります。顧客所在地に基づいて、トランザクションファイルからデータベースに顧客データを挿入する必要があります。

式トランスフォーメーションは、[場所] カラムの値に応じたデータベース接続オブジェクト名を返します。接続オブジェクト名は、式トランスフォーメーションから SQL トランスフォーメーションの LogicalConnectionObject ポートに渡されます。SQL トランスフォーメーションは、LogicalConnectionObject カラムの値に基づいてデータベースに接続します。

以下の図に、マッピングにおける式トランスフォーメーションおよび SQL トランスフォーメーションを示します。



マッピングは、以下のようなコンポーネントから構成されています。

- **Customer ソース定義。**顧客情報が含まれているフラットファイルの定義です。SQL トランスフォーメーションが顧客データ挿入時に接続するデータベースは、顧客サイトに応じて決定されます。
- **Error_File ターゲット定義。**SQL トランスフォーメーションからのデータベースエラーを受け取る Datastring フィールドが、ターゲットに含まれています。
- **Exp_Dynamic_Connection トランスフォーメーション。**式トランスフォーメーションは、[場所] カラムの値に基づいて接続先のデータベースを定義します。式トランスフォーメーションは、接続オブジェクト名を接続ポート内に返します。接続オブジェクトは、Workflow Manager で定義されるデータベース接続です。
- **SQL_Dynamic_Connection トランスフォーメーション。**SQL トランスフォーメーションは、接続オブジェクト名を LogicalConnectionPort 内に受け取り、データベースに接続し、データベース内に顧客データを挿入します。

ソースファイルの定義

トランザクションファイルは、データベースに追加される顧客を含むフラットファイルソースです。各行には、顧客の居住国を定義する場所が格納されています。ソースファイル名は Customer.dat です。

顧客ファイルレコードは、次のようなフィールドから構成されます。

ポート名	データタイプ	精度	スケール	説明
CustomerID	番号	10	0	顧客を一意に識別する顧客番号。
CustomerName	String	25	0	顧客の名前。
PhoneNumber	String	15	0	電話番号には、市外局番および 7 桁の数字を空白文字を含めずに入力します。
電子メール	String	25	0	顧客の電子メールアドレス。
場所	String	10	0	顧客サイト。値は US、UK、CAN です。

次の行を含む Customer.dat ファイルを、Srcfile 内に作成することができます。

```
1,John Smith,6502345677,jsmith@catgary.com,US
2,Nigel Whitman,5123456754,nwhitman@nwhitman.com,UK
3,Girish Goyal,5674325321,ggoyal@netcompany.com,CAN
4,Robert Gregson,5423123453,rgregson@networkmail.com,US
```

Customer.dat ファイルをインポートして、リポジトリ内に Customer ソース定義を作成することができます。

ターゲット定義の作成

Error_File は、SQL トランスフォーメーションからデータベースエラーメッセージを受け取るフラットファイルターゲット定義です。

Error_File 定義には、次の入力ポートが含まれています。

ポート名	データ型	精度	スケール
DataStream	String	4000	0

Target Designer s で、ターゲット定義を作成します。

データベーステーブルの作成

この例では、3 つの Oracle データベース（United States データベース、United Kingdom データベース、および Canadian データベース）を作成する必要があります。各データベースの Cust テーブルには、その地域の顧客データが格納されています。

次の SQL 文により、Cust テーブルが Oracle データベース上に作成されます。

```
Create table Cust (CustomerId number, CustomerName varchar2(25), PhoneNumber varchar2(15), Email varchar2(25));
```

注: この例では、3 つのデータベースを挙げて、動的データベース接続を説明します。テーブルがそれぞれ同じデータベースに含まれている場合は、静的データベース接続を使用してパフォーマンスを高めることができます。

データベース接続の作成

US、UK、および CAN データベースへのデータベース接続を作成するには、Workflow Manager を使用します。

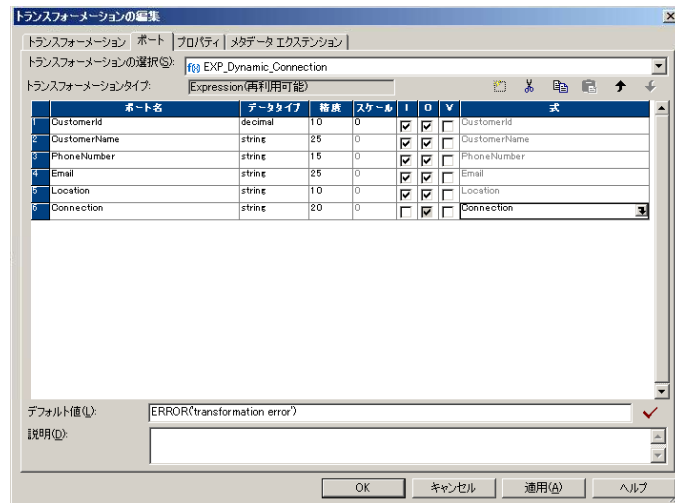
この例では、SQL トランスフォーメーションからデータベースへの接続に、次の接続名を使用します。

データベース	接続オブジェクト名
US	DBORA_US
UK	DBORA_UK
CAN	DBORA_CAN

式トランスフォーメーションの設定

式トランスフォーメーションには、各ソースカラムに対応した入出力ポートが備わっています。式の結果に基づいて接続オブジェクト名が、トランスフォーメーションから接続ポート内に返されます。

以下の図は、式トランスフォーメーションの「ポート」タブ、および各ソースカラムの入力/出力ポートカラムを示しています。



以下は、接続ポートに用いられる式です。

```
Decode(Location, 'US', 'DBORA_US', 'UK', 'DBORA_UK', 'CAN', 'DBORA_CAN', 'DBORA_US')
```

この式では、所在地が US、UK、または CAN のどれに該当するかに応じて接続オブジェクト名が返されます。所在地が式の中の所在地と一致しない場合、接続オブジェクト名が"DBORA_US"にデフォルト設定されます。

例として、次のソース行を挙げます。この顧客情報は、米国からの顧客に関するものです。

```
1, John Smith, 6502345677, jsmith@catgary.com, US
```

顧客サイトが"US"なら、式トランスフォーメーションが返す接続オブジェクト名は"DBORA_US"です。"DBORA_US"は、式トランスフォーメーションから SQL トランスフォーメーションの LogicalConnectionObject ポートに渡されます。

SQL トランスフォーメーションの定義

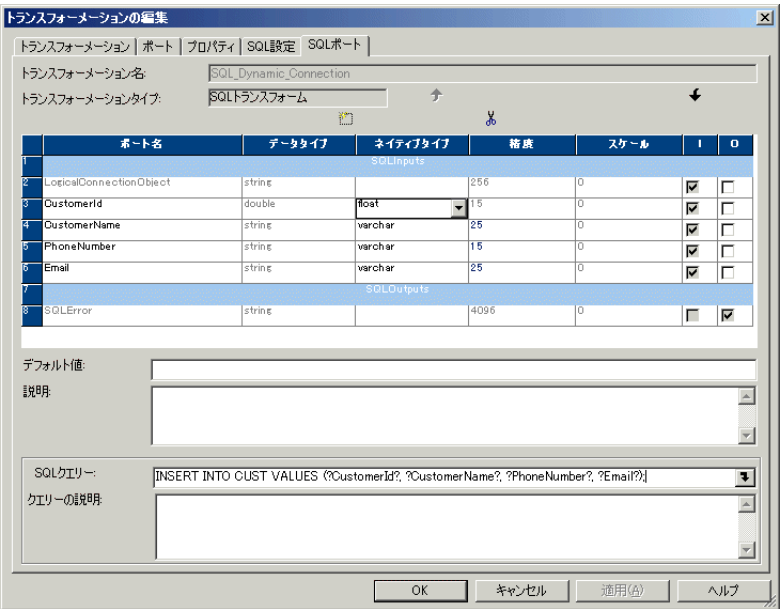
SQL トランスフォーメーションはデータベースに接続し、顧客データを CUST テーブルに挿入する動的 SQL クエリを実行します。

SQL トランスフォーメーションの作成時には、トランスフォーメーションモード、データベースタイプ、および接続タイプを定義します。モードまたは接続タイプは、SQL トランスフォーメーションの作成後に変更不能になります。

SQL トランスフォーメーションの作成には、次のプロパティが使用されます。

- **クエリモード。** SQL トランスフォーメーションによって、動的 SQL クエリが実行されます。
- **動的接続。** SQL トランスフォーメーションは、マッピングでトランスフォーメーションに渡す接続情報に応じて、データベースに接続します。
- **接続オブジェクト。** SQL トランスフォーメーションは、接続オブジェクト名を受け取る LogicalConnectionObject ポートを備えています。接続オブジェクトは、Workflow Manager 接続で定義する必要があります。

以下の図に、SQL トランスフォーメーションのポートを示します。



SQL トランスフォーメーションは、LogicalConnectionObject ポート内に接続オブジェクト名を受け取り、行の処理時に毎回その接続オブジェクト名を使用してデータベースに接続します。

トランスフォーメーションは、次の動的 SQL クエリによって、顧客データを CUST テーブルに挿入します。

```
INSERT INTO CUST VALUES (?CustomerId?,?CustomerName?,?PhoneNumber?,?Email?);
```

SQL トランスフォーメーションによって、クエリーのパラメータはトランスフォーメーションの入力ポートからの顧客データで置換されます。例として、次のソース行を挙げます。この行に格納されている顧客情報は、顧客番号 1 のものです。

```
1,John Smith,6502345677,jsmith@catgary.com,US
```

SQL トランスフォーメーションは、DBORA_US 接続オブジェクトを使用してデータベースに接続し、次の SQL クエリを実行します。

```
INSERT INTO CUST VALUES (1,'John Smith','6502345677','jsmith@catgary.com');
```

セッション属性の設定

次のソースおよびターゲットにアクセスするように、セッションを設定してください。

属性	値
ソースファイル名	Customer.dat
Output File Name	Error_File.dat

注: セッション用のデータベース接続は、設定しないでください。SQL トランスフォーメーションは、入力行処理時に毎々、別のデータベースに接続する可能性があります。

ターゲットデータの結果

サンプルデータを使ってセッションを実行すると、United States データベース、United Kingdom データベース、または Canadian データベース内の Cust テーブルにデータが移入されます。

United States データベースには、次の行が含まれています。

CustomerID	CustomerName	PhoneNumber	電子メール
1	John Smith	6502345677	jsmith@catagary.com
4	Robert Gregson	5423123453	rgregson@networkmail.com

United Kingdom データベースには、次の行が含まれています。

CustomerID	CustomerName	PhoneNumber	電子メール
2	Nigel Whitman	5123456754	nwhitman@nwhitman.com

Canadian データベースには、次の行が含まれています。

CustomerID	CustomerName	PhoneNumber	電子メール
3	Girish Goyal	5423123453	ggoyal@netcompany.com

何らかのエラーがデータベースから SQL トランスフォーメーションに返された場合、エラーテキストが Error_File ターゲットに格納されます。

第 28 章

ストアドプロシージャトランスフォーマーション

この章では、以下の項目について説明します。

- [ストアドプロシージャトランスフォーマーションの概要, 465 ページ](#)
- [マッピングでのストアドプロシージャの使用, 469 ページ](#)
- [ストアドプロシージャの記述, 470 ページ](#)
- [ストアドプロシージャトランスフォーマーションの作成, 472 ページ](#)
- [接続されたトランスフォーマーションの設定, 477 ページ](#)
- [接続されていないトランスフォーマーションの設定, 477 ページ](#)
- [エラー処理, 481 ページ](#)
- [サポートされるデータベース, 482 ページ](#)
- [式のルール, 483 ページ](#)
- [ストアドプロシージャトランスフォーマーションに関するヒント, 484 ページ](#)
- [ストアドプロシージャトランスフォーマーションのトラブルシューティング, 484 ページ](#)

ストアドプロシージャトランスフォーマーションの概要

ストアドプロシージャトランスフォーマーションは、データベースに値を格納したり、データベースの保守を行ったりするための重要なツールです。データベース管理者は、標準 SQL 文で実現するには複雑すぎる処理をストアドプロシージャを作成することによって自動化することができます。ストアドプロシージャトランスフォーマーションは、パッシブトランスフォーマーションです。接続されたまたは接続されていないストアドプロシージャトランスフォーマーションを設定できます。

ストアドプロシージャは、一連の Transact-SQL、PL-SQL、または他のデータベースの手続き文、およびオプションのフロー制御文をプリコンパイルしたもので、実行形式のスクリプトに似ています。ストアドプロシージャは、データベース内に格納され、データベース内で実行されます。ストアドプロシージャは、SQL 文の実行と同様に、データベースクライアントツールの EXECUTE SQL 文を使用して実行することができます。ただし標準 SQL とは異なり、ストアドプロシージャでは、ユーザー定義変数、条件文、およびその他のパワフルなプログラミング機能を使用することができます。

ストアドプロシージャはすべてのデータベースがサポートしているとは限りません。またストアドプロシージャの構文はデータベースによって異なります。ストアドプロシージャを使用して、以下の作業を実行できます。

- データをターゲットデータベースにロードする前にターゲットデータベースの状態をチェックする。
- データベース内に十分な領域があるかどうかを調べる。
- 特殊な計算を行う。
- インデックスの削除と再作成。

ストアドプロシージャは SQL 文よりもはるかに柔軟性を持っているため、データベースの開発者やプログラマは、データベース内でいろいろな処理を行うためにストアドプロシージャを使用します。またストアドプロシージャは、重要な処理に不可欠なエラー処理およびロギング機能を提供します。開発者は、データベースに提供されるクライアントツールを使用して、データベース内にストアドプロシージャを作成します。

ストアドプロシージャは、ストアドプロシージャトランスフォーメーションを作成する前にデータベース内に存在していなければなりません。ストアドプロシージャは、ソース、ターゲット、または Integration Service への有効な接続を持つ任意のデータベース内に存在することができます。

ストアドプロシージャを使用して、マッピングの一部を構成するようなクエリまたは計算を行います。たとえば、消費税を計算するための十分にテストしたストアドプロシージャがあれば、式トランスフォーメーションで同じ計算を再作成する代わりにそのストアドプロシージャを使用して計算を行うことができます。

入力データと出力データ

ストアドプロシージャで最も便利な機能の 1 つは、ストアドプロシージャとの間でデータをやり取りできるということです。Integration Service とストアドプロシージャの間で受け渡しされるデータには、次の 3 種類があります。

- 入出力パラメータ
- 戻り値
- 状態コード

データの受け渡しについてはデータベースの実装に応じていくつかの制限があります。この制限については、この章での解説を通じて説明します。また、すべてのストアドプロシージャがデータのやり取りを行うわけではありません。たとえばセッションの実行後にデータベースインデックスを再構築するストアドプロシージャの場合、データを受け取ることはできません。セッションは既に完了しているからです。

入出力パラメータ

多くのストアドプロシージャでは、値を渡してその戻り値を受け取ります。これらの値は、入力パラメータおよび出力パラメータと呼ばれます。たとえば消費税を計算するストアドプロシージャは、商品の値段といった 1 つの入力パラメータを取ります。計算を行ったあと、ストアドプロシージャは、税金の金額と税を含む商品の合計金額の 2 つの出力パラメータを返します。

ストアドプロシージャトランスフォーメーションは、ポートや変数を使用したり、式に値（たとえば 10 または SALES）を入力したりすることによって、入力パラメータと出力パラメータをやり取りします。

戻り値

ほとんどのデータベースは、ストアドプロシージャを実行したあとに戻り値を提供します。データベースの実装に応じて、戻り値はユーザー定義可能であったり（1 つの出力パラメータと同様の動作をすることを意味します）、あるいは整数値だけが返されたりします。

ストアドプロシージャトランスフォーメーションは、入出力パラメータが取得される方法に応じて、入出力パラメータと同じような方法で戻り値を取得します。場合によってはパラメータまたは戻り値のいずれかだけを取得することができます。

ストアードプロシージャが単一の戻り値ではなく一連の結果を返す場合、ストアードプロシージャトランスフォーメーションはプロシージャの最初の戻り値のみを受け取ります。

注: Oracle のストアード関数は、出力パラメータまたは戻り値をサポートしていることを除けば Oracle のストアードプロシージャに似ています。この章のストアードプロシージャに関する説明は、特に明記していない限りストアード関数にも当てはまります。

状態コード

Integration Service は、状態コードによって、ワークフロー実行中にエラーを処理できます。ストアードプロシージャは、その処理が正常に完了したかどうかを通知する状態コードを発行します。ユーザーはこの値を見ることができません。Integration Service はこの値を使用して、セッションの実行を続けるか中止するかを決定します。Workflow Manager のオプションの設定により、ストアードプロシージャでエラーが発生したときにセッションを続行するかまたは中止するかを指定することができます。

接続されたモードと接続されていないモード

ストアードプロシージャは、接続されたモードまたはコネクต์されていないモードのどちらかで実行されます。使用するモードは、ストアードプロシージャが実行する処理やセッション内でのストアードプロシージャの使用方法により決定されます。マッピングで、接続されたストアードプロシージャトランスフォーメーションおよびコネクต์されていないストアードプロシージャトランスフォーメーションを設定できます。

- **接続されたモード。** 接続されたモードでは、マッピングを通過するデータはストアードプロシージャトランスフォーメーションも通過します。入力ポートからトランスフォーメーションに入力されるすべてのデータが、ストアードプロシージャに影響を及ぼします。接続されたストアードプロシージャトランスフォーメーションは、入力ポートのデータを入力パラメータとしてストアードプロシージャに渡す場合、またはストアードプロシージャの結果を別のトランスフォーメーションに出力パラメータとして渡す場合に使用してください。
- **接続されていないモード。** 接続されていないストアードプロシージャトランスフォーメーションは、マッピングのフローに直接接続されません。接続されていないストアードプロシージャトランスフォーメーションは、セッションの前または後で実行されます。あるいはマッピングに含まれる別のトランスフォーメーションの式によって呼び出されることもあります。

以下の表に、接続されたトランスフォーメーションとコネクต์されていないトランスフォーメーションの比較を示します。

目的	使用するモード
ストアードプロシージャをセッションの前または後で実行する	接続されていない
セッション実行前または実行後に起動されるストアードプロシージャのように、マッピング中にストアードプロシージャを 1 回だけ実行する	接続されていない
行がストアードプロシージャトランスフォーメーションを通過するごとにストアードプロシージャを実行する	接続されている、または接続されていない
マッピングを通過するデータに基づいてストアードプロシージャを実行する（たとえば特定のポートが NULL 値を含んでいないとき）	接続されていない
パラメータをストアードプロシージャに渡し、1 つの出力パラメータを受け取る	接続されている、または接続されていない

目的	使用するモード
パラメータをストアードプロシージャに渡し、複数の出力パラメータを受け取る 注: 接続されていないストアードプロシージャトランスフォーメーションから複数の出力パラメータを取得するには、出力パラメータそれぞれに対して変数を作成する必要があります。	接続されている、または接続されていない
ネストしているストアードプロシージャを実行する	接続されていない
マッピング内で複数回呼び出す	接続されていない

関連項目：

- [「接続されていないトランスフォーメーションの設定」 \(ページ 477\)](#)
- [「接続されたトランスフォーメーションの設定」 \(ページ 477\)](#)

ストアードプロシージャの実行タイミングの指定

ストアードプロシージャトランスフォーメーションのモードの指定に加え、ストアードプロシージャトランスフォーメーションをいつ実行するかを指定します。上記のコネクトされていないストアードプロシージャの場合、式トランスフォーメーションがこのストアードプロシージャを参照しています。これは、式トランスフォーメーションを行が通過するごとにストアードプロシージャが実行されることを意味します。ただし、ストアードプロシージャトランスフォーメーションを参照しているトランスフォーメーションがない場合は、セッションの前か後にストアードプロシージャを 1 回だけ実行するように指定できます。

ストアードプロシージャトランスフォーメーションの実行用オプションには次のものがあります。

- **ノーマル。**ストアードプロシージャは、マッピング内のトランスフォーメーションが存在する位置で行ごとに実行されます。このオプションは、入力ポートに対して計算を実行する場合など、マッピングを通過するそれぞれのデータ行に対してストアードプロシージャを呼び出す場合に便利です。接続されたストアードプロシージャは、このモードでのみ実行されます。
- **ソースのロード前。**ストアードプロシージャは、セッションがソースからデータを取得する前に実行されます。このオプションは、テーブルの存在を検査したり、一時テーブル内でデータの結合を実行したりする場合に役立ちます。
- **ソースのロード後。**ストアードプロシージャは、セッションがソースからデータを取得した後に実行されます。このオプションは、一時テーブルを削除するのに役立ちます。
- **ターゲットのロード前。**ストアードプロシージャは、セッションがターゲットにデータを渡す前に実行されます。このオプションは、ターゲットテーブルやターゲットシステム上のディスク領域を検査するのに役立ちます。
- **ターゲットのロード後。**ストアードプロシージャは、セッションがターゲットにデータを渡した後に実行されます。このオプションは、データベースのインデックスを再作成するのに便利です。

複数のストアードプロシージャトランスフォーメーションを、同じマッピング内で異なるモードで実行することができます。たとえば Source Pre Load ストアードプロシージャでテーブルの整合性をチェックし、ノーマルストアードプロシージャでテーブルに値を格納し、Post Load ストアードプロシージャでデータベースのインデックスを再構築する、といった処理を行えます。ただし、1 つのストアードプロシージャトランスフォーメーションの同じインスタンスを 1 つのマッピング内で接続されたモードとコネクトされていないモードの両方で実行することはできません。その場合は、トランスフォーメーションのインスタンスを別個に作成する必要があります。

マッピングがマッピング内の複数のソースまたはターゲットのロード前またはロード後のストアードプロシージャを呼び出すと、Integration Service はマッピングで指定された実行順でストアードプロシージャを実行します。

Integration Service は、トランスフォーメーションプロパティに指定されたデータベース接続を使用して、各ストアードプロシージャを実行します。Integration Service は、最初のストアードプロシージャに遭遇したときにデータベース接続を開きます。データベース接続は、Integration Service がその接続に対するストアードプロシージャをすべて処理し終えるまで開いたままの状態です。Integration Service は、別のデータベース接続を使用するストアードプロシージャに遭遇すると、データベース接続を閉じて新しい接続を開きます。

同じデータベース接続を使用する複数のストアードプロシージャを実行する場合、それらが連続して実行されるように設定します。連続して実行されるように設定しなかった場合、ターゲットに予期しない結果が現れる場合があります。たとえば、2つのストアードプロシージャ（ストアードプロシージャ A とストアードプロシージャ B）があるとした場合。ストアードプロシージャ A はトランザクションを開始し、ストアードプロシージャ B はトランザクションをコミットします。ストアードプロシージャ B の前に別のデータベース接続を使用してストアードプロシージャ C を実行すると、Integration Service はストアードプロシージャ C を実行するときにデータベース接続を閉じるため、ストアードプロシージャ B はトランザクションをコミットできません。

データベース接続内で複数のストアードプロシージャを実行する際は、下記のガイドラインに従ってください。

- ストアードプロシージャのプロパティで定義されたのと同じデータベース接続文字列を使用する。
- ストアードプロシージャが連続した順序で実行されるように設定する。
- ストアードプロシージャタイプが同じである。
 - Source Pre Load
 - Source Post Load
 - Target Pre Load
 - Target Post Load

マッピングでのストアードプロシージャの使用

マッピング内でストアードプロシージャトランスフォーメーションを使用するためには、いくつかの作業を行う必要があります。ストアードプロシージャはデータベース内に存在するため、マッピングやセッションだけでなく、データベース内のストアードプロシージャも設定する必要があります。

ストアードプロシージャトランスフォーメーションを使用するには、次の手順を実行します。

1. ストアードプロシージャをデータベース内に作成します。

Designer を使用してトランスフォーメーションを作成する前に、ストアードプロシージャをデータベース内に作成する必要があります。また、提供されているデータベースクライアントツールを使用して、ストアードプロシージャをテストしなければなりません。
2. ストアードプロシージャトランスフォーメーションをインポートするか、新たに作成します。

Designer を使用してストアードプロシージャトランスフォーメーションをインポートするか作成します。必要な入出力および戻り値のためのポートを設定します。
3. 接続されたトランスフォーメーションまたはコネクタされていないトランスフォーメーションのどちらを使用するかを決定します。

ストアードプロシージャがどのようにマッピングに関係するかをトランスフォーメーションを設定する前に決めなければなりません。
4. 接続されたトランスフォーメーションの場合は、適切な入力ポートおよび出力ポートをマッピングします。

接続されたストアードプロシージャトランスフォーメーションは、他のトランスフォーメーションの場合と同じように使用します。適切な入力フローポートをクリックしてからトランスフォーメーションにドラッグし、出力ポートから他のトランスフォーメーションへのマッピングを作成します。

5. コネクトされていないトランスフォーメーションの場合は、セッション実行前または実行後に起動されるようにストアドプロシージャを設定するか、または別のトランスフォーメーションの式から起動されるように設定します。

ストアドプロシージャはセッションの前でも後でも実行できるため、そのコネクトされていないトランスフォーメーションをいつ実行するかを指定しなければなりません。一方、別のトランスフォーメーションからストアドプロシージャを呼び出す場合は、ストアドプロシージャを呼び出すトランスフォーメーション内で式を記述します。式には変数を含むことができます。戻り値は含んでも含まなくてもかまいません。

6. セッションを設定します。

Workflow Manager のセッションプロパティには、ストアドプロシージャを実行したときのエラー処理や、SQL 上書きのためのいくつかのオプションが用意されています。

ストアドプロシージャの記述

SQL 文を使用してストアドプロシージャをデータベース内に作成します。ほかの Transact-SQL 文やデータベース固有の関数を追加することもできます。これらには、ユーザー定義のデータタイプや、実行順序を指定する文も含まれます。

ストアドプロシージャの例

次の例では、従業員の ID 番号を入力パラメータとして取り、従業員名を出力パラメータとして返すストアドプロシージャがソースデータベースにあります。また、ストアドプロシージャが正常に完了したことの通知として戻り値 0 が返されます。

従業員 ID と従業員名を含むデータベーステーブルは次のようになっています。

従業員 ID	従業員名
101	Bill Takash
102	Louis Li
103	Sarah Ferguson

ストアドプロシージャは、従業員 ID 「101」を入力パラメータとして受け取り、名前「Bill Takash」を返します。マッピングがどのようにこのストアドプロシージャを呼び出すかに応じて、ID のいくつかまたは全部がストアドプロシージャに渡されます。

構文はデータベースごとに異なるため、このストアドプロシージャを作成するための SQL 文もいろいろあります。また、SQL 文をデータベースに渡すためのクライアントツールも異なります。ほとんどのデータベースでは、標準 SQL エディタを含む、一式のクライアントツールを提供しています。Microsoft SQL Server など一部のデータベースは、いくつかの初期 SQL 文を作成するツールを提供しています。

注: Large Object を伴うストアドプロシージャ引数が含まれる場合、Integration Service はセッションに失敗します。

Informix

Informix の場合、出力パラメータを宣言するための構文がほかのデータベースと異なります。ほとんどのデータベースの場合、変数の宣言に IN または OUT を使用してそれが入力パラメータであるかまたは出力パラメータであるかを指定します。Informix では、キーワード RETURNING を使用します。これにより、入出力パラメ

ータと戻り値を区別するのが難しくなっています。たとえば、RETURN コマンドを使用して 1 つまたは複数の出力パラメータを返します。

```
CREATE PROCEDURE GET_NAME_USING_ID (nID integer)
RETURNING varchar(20);
define nID integer;
define outVAR as varchar(20);
SELECT FIRST_NAME INTO outVAR FROM CONTACT WHERE ID = nID
return outVAR;
END PROCEDURE;
```

この場合、RETURN 文が outVAR の値を渡すことに注意してください。ただし他のデータベースとは異なり、outVAR は戻り値ではなく、出力パラメータです。複数の出力パラメータは次のように返されます。

```
return outVAR1, outVAR2, outVAR3
```

Informix も戻り値を渡します。戻り値はユーザー定義ではなく、エラーチェック用の値として生成されます。トランスフォーメーションにおいては、[R] 値をチェックしなければなりません。

Oracle

Oracle の場合、値を返すストアードプロシージャはすべてストアード関数と呼ばれます。CREATE PROCEDURE 文を使用して例に基づく新しいストアードプロシージャを作成する代わりに、CREATE FUNCTION 文を使用します。この例において、変数は IN および OUT として宣言されていますが、Oracle では INOUT パラメータ型もサポートされています。INOUT を使用すると、パラメータに値を渡し、それを変更し、変更後の値を返すことができます。

```
CREATE OR REPLACE FUNCTION GET_NAME_USING_ID (
nID IN NUMBER,
outVAR OUT VARCHAR2)
RETURN VARCHAR2 IS
RETURN_VAR varchar2(100);
BEGIN
SELECT FIRST_NAME INTO outVAR FROM CONTACT WHERE ID = nID;
RETURN_VAR := 'Success';
RETURN (RETURN_VAR);
END;
/
```

戻り値が VARCHAR2 データタイプの文字列値 (Success) であることに注意してください。Oracle は、文字列データタイプの戻り値を許可している唯一のデータベースです。

Sybase ASE および Microsoft SQL Server

次の構文が示すとおり、Sybase と Microsoft のストアードプロシージャは同じです。

```
CREATE PROCEDURE GET_NAME_USING_ID @nID int = 1, @outVar varchar(20) OUTPUT
AS
SELECT @outVar = FIRST_NAME FROM CONTACT WHERE ID = @nID
return 0
```

戻り値は変数である必要はありません。この例の場合、SELECT 文が成功すれば戻り値として 0 が返されます。

IBM DB2

以下は、IBM DB2 における SQL ストアドプロシージャの例です。

```
CREATE PROCEDURE get_name_using_id ( IN id_in int,
                                     OUT emp_out char(18),
                                     OUT sqlcode_out int)
LANGUAGE SQL
P1: BEGIN
  -- Declare variables
  DECLARE SQLCODE INT DEFAULT 0;
  DECLARE emp_TMP char(18) DEFAULT ' ';
  -- Decler handler
```

```

DECLARE EXIT HANDLER FOR SQLEXCEPTION
SET SQLCODE_OUT = SQLCODE;
select employee into emp_TMP
from doc_employee
where id = id_in;
SET emp_out = EMP_TMP;
SET sqlcode_out = SQLCODE;
END P1

```

Teradata

下記は Teradata における SQL ストアドプロシージャの例です。このストアドプロシージャは、入力パラメータとして従業員 ID をとり、出力パラメータとして従業員名を返します。

```

CREATE PROCEDURE GET_NAME_USING_ID (IN nID integer, OUT outVAR varchar(40))
BEGIN
  SELECT FIRST_NAME INTO :outVAR FROM CONTACT where ID = :nID;
END;

```

ストアドプロシージャトランスフォーメーションの作成

データベース内にストアドプロシージャを設定してテストしたら、Mapping Designer でストアドプロシージャトランスフォーメーションを作成します。ストアドプロシージャトランスフォーメーションを設定するには次の 2 つの方法があります。

- [ストアドプロシージャのインポート] ダイアログボックスを使用して、ストアドプロシージャが使用するポートを設定する。
- 入力パラメータまたは出力パラメータの適切なポートを作成して、手作業でトランスフォーメーションを設定する。

ストアドプロシージャトランスフォーメーションは、デフォルトで [Normal] タイプとして作成されます。これは、セッションの前後に実行されるのではなく、マッピング中に実行されることを意味します。

新しいストアドプロシージャトランスフォーメーションは、再利用可能なトランスフォーメーションとしては作成されません。再利用可能なトランスフォーメーションを作成するには、トランスフォーメーションを作成したあと、そのトランスフォーメーションのプロパティで [再利用可能にする] をクリックします。

注: 再利用可能なトランスフォーメーションのプロパティは、Mapping Designer ではなく Transformation Developer で設定し、トランスフォーメーションに対する全体的な変更としなければなりません。

ストアドプロシージャのインポート

ストアドプロシージャをインポートすると、Designer はストアドプロシージャの入力パラメータおよび出力パラメータに基づいてポートを作成します。ストアドプロシージャはできる限りインポートするようにしてください。

Mapping Designer でストアドプロシージャをインポートするには、次の 3 つの方法があります。

- ストアドプロシージャのアイコンを選択し、ストアドプロシージャトランスフォーメーションを追加する。
- [トランスフォーメーション] - [ストアドプロシージャのインポート] を選択します。
- [トランスフォーメーション] - [作成] を選択し、次に [ストアドプロシージャ] を選択します。

ストアドプロシージャ名にピリオド (.) を含むストアドプロシージャをインポートすると、Designer はストアドプロシージャトランスフォーメーション名でピリオドをアンダスコア (_) に置き換えます。

ストアドプロシージャをインポートする手順

1. Mapping Designer で、[トランスフォーメーション] - [ストアドプロシージャのインポート] を選択します。
2. ストアドプロシージャが格納されているデータベースを ODBC ソースのリストの中から選択します。データベースに接続するためのユーザー名、オーナー名、およびパスワードを入力し、[接続] をクリックします。

ダイアログに FUNCTIONS というフォルダが表示されます。このフォルダ内に表示されたストアドプロシージャには、入力パラメータ、出力パラメータ、または戻り値が含まれています。パラメータまたは戻り値を含まないストアドプロシージャがデータベースに存在する場合、それらは PROCEDURES というフォルダ内に表示されます。これは主に Oracle のストアドプロシージャに当てはまります。標準の接続されたストアドプロシージャをこの関数リストに表示させるためには、入力ポートと出力ポートがそれぞれ 1 つ以上なければなりません。

ヒント: <スキップ>を選択すると、ストアドプロシージャをインポートせずにストアドプロシージャトランスフォーメーションを追加することができます。この場合、トランスフォーメーション内でポートおよび接続情報を手作業で追加する必要があります。

3. 必要に応じて、検索フィールドを使用して、表示されるプロシージャの数を制限します。
4. インポートするプロシージャを選択し、[OK] をクリックします。

ストアドプロシージャトランスフォーメーションがマッピング内に表示されます。ストアドプロシージャトランスフォーメーションの名前は、選択したストアドプロシージャと同じです。ストアドプロシージャが入力パラメータ、出力パラメータ、または戻り値を含む場合、ストアドプロシージャトランスフォーメーション内でそれぞれのパラメータまたは戻り値に対応するポートが表示されます。

このストアドプロシージャトランスフォーメーションの場合、ストアドプロシージャは次の値とパラメータを含んでいます。

- 出力ポートを持つ、RETURN_VALUE という Integer 型の戻り値
- 入力ポートを持つ、nNAME という String 型の入力パラメータ
- 入出力ポートを持つ、outVar という Integer 型の出力パラメータ

注: このトランスフォーメーション名を変更する場合は、トランスフォーメーションのプロパティでストアドプロシージャの名前を設定する必要があります。マッピング内に同じストアドプロシージャのインスタンスが複数ある場合は、ストアドプロシージャの名前の設定も行う必要があります。

5. トランスフォーメーションを開き、[プロパティ] タブをクリックします。

ストアドプロシージャが格納されているデータベースを [接続情報] 行から選択します。ストアドプロシージャトランスフォーメーションの名前をそのストアドプロシージャ以外の名前に変更した場合は、[ストアドプロシージャ名] に名前を入力します。

6. [OK] をクリックします。

手動によるストアドプロシージャトランスフォーメーションの作成

手作業でストアドプロシージャトランスフォーメーションを作成する場合は、ストアドプロシージャの入力パラメータ、出力パラメータ、戻り値を知っておく必要があります。また、これらのパラメータのデータタイプやストアドプロシージャの名前も把握しておく必要があります。これらすべては、[ストアドプロシージャのインポート] で設定されます。

ストアードプロシージャトランスフォーメーションを作成するには：

1. Mapping Designer で [トランスフォーメーション] - [作成] を選択し、次に [ストアードプロシージャ] を選択します。

ストアードプロシージャトランスフォーメーションには、自動的にストアードプロシージャの名前が付けられます。このトランスフォーメーション名を変更する場合は、[トランスフォーメーションのプロパティ] でストアードプロシージャの名前を設定する必要があります。マッピング内に同じストアードプロシージャのインスタンスが複数ある場合は、この操作を行う必要があります。

2. [スキップ] をクリックします。

ストアードプロシージャトランスフォーメーションが Mapping Designer に表示されます。

3. トランスフォーメーションを開き、[ポート] タブをクリックします。

ポートは、ストアードプロシージャの入力パラメータ、出力パラメータ、および戻り値に基づいて作成する必要があります。ストアードプロシージャトランスフォーメーションのポートは、下記のストアードプロシージャパラメータそれぞれについて作成します。

- 1 つの Integer 型の入力パラメータ
- 1 つの String 型の出力パラメータ
- 1 つの戻り値

Integer 型の入力パラメータに対しては、Integer 型の入力ポートを作成します。パラメータとポートは、同じデータタイプおよび精度でなければなりません。出力パラメータと戻り値に対して同様の操作を行います。

戻り値の場合は、出力ポートのほかに [R] カラムを選択する必要があります。複数のパラメータを持つストアードプロシージャの場合、ストアードプロシージャにリストされているのと同じ順にポートを一覧表示する必要があります。

4. [プロパティ] タブをクリックします。

ストアードプロシージャの名前を [ストアードプロシージャ名] 行に入力します。次に、ストアードプロシージャが格納されているデータベースを [接続情報] 行から選択します。

5. [OK] をクリックします。

リポジトリがマッピングを検証して保存しても、Designer は手作業で入力したストアードプロシージャトランスフォーメーションを検証しません。すなわち、適切なパラメータまたは戻り値がストアードプロシージャに存在するかどうかを確認するチェックは行われません。ストアードプロシージャトランスフォーメーションの設定が正しくない場合、セッションは失敗します。

ストアードプロシージャのオプションの設定

以下の表では、ストアードプロシージャトランスフォーメーションのプロパティについて説明します。

設定	説明
ストアードプロシージャ名	データベース内のストアードプロシージャの名前。トランスフォーメーションの名前がデータベース内の実際のストアードプロシージャ名と異なる場合、Integration Service はこのテキストを使用してストアードプロシージャを呼び出します。トランスフォーメーション名をストアードプロシージャ名と同じにする場合は、このフィールドを空白のままにしておきます。[ストアードプロシージャのインポート] 機能を使用すると、この名前はストアードプロシージャ名と同じになります。
接続情報	<p>ストアードプロシージャを含んでいるデータベースを指定します。データベースは、マッピング、セッション、またはパラメータファイルの中で定義できます。</p> <ul style="list-style-type: none">- Mapping。リレーショナル接続オブジェクトを選択します。- セッション。接続変数 <code>\$Source</code> または <code>\$Target</code> を使用します。このどちらかの変数を使用する場合、ストアードプロシージャは、セッションを実行したときに指定したソースデータベースまたはターゲットデータベース内に存在しなければなりません。セッションプロパティで、各変数のデータベース接続を指定します。- パラメータファイル。セッションパラメータ <code>\$DBConnection</code> 名前を使用し、これをパラメータファイルに定義します。 <p>デフォルトでは、Designer はノーマルのストアードプロシージャタイプに対して <code>\$Target</code> を設定します。Source Pre Load および Source Post Load の場合、Designer は <code>\$Source</code> を指定します。Target Pre Load および Target Post Load の場合、Designer は <code>\$Target</code> を指定します。これらの値は、セッションプロパティで上書きできます。</p>
呼び出しテキスト	<p>ストアードプロシージャを呼び出すためのテキスト。[ストアードプロシージャタイプ] が [ノーマル] でない場合にのみ使用します。呼び出しテキストには、ストアードプロシージャに渡される入力パラメータをすべて含める必要があります。</p> <p>このほか、呼び出しテキストに PowerCenter パラメータまたは変数を使用することもできます。パラメータファイルで定義可能なパラメータまたは変数タイプを使用します。</p>
ストアードプロシージャタイプ	Integration Service がいつストアードプロシージャを呼び出すかを指定します。オプションとして、[ノーマル]（マッピング内）のほかに、ソースデータベースまたはターゲットデータベース上でのロード前または後の実行を指定することができます。デフォルトは [ノーマル] です。
トレースレベル	<p>セッションログにレポートされるトランザクション詳細の量。以下のトレースレベルを使用します。</p> <ul style="list-style-type: none">- Terse- ノーマル- Verbose Initialization- Verbose Data <p>デフォルトは [ノーマル] です。</p>
実行順序	同じマッピング内のほかのストアードプロシージャと関連して、Integration Service がこのトランスフォーメーション内で使用されているストアードプロシージャを呼び出す順序。[ストアードプロシージャタイプ] が [ノーマル] 以外に設定されていて、かつ複数のストアードプロシージャがある場合にのみ使用します。

設定	説明
サブ秒の精度	<p>日時ポートのサブ秒の精度を指定します。</p> <p>日時データの位取りを編集できるデータベースでは、精度を変更できます。サブ秒の精度を変更できるデータタイプは、Oracle Timestamp、Informix Datetime、および Teradata Timestamp です。</p> <p>0 から 9 までの正の整数値を入力します。デフォルトは、6（マイクロ秒）です。</p>
出力が再現可能	<p>トランスフォーメーションが行を生成する順序がセッション間で同じかどうかを示します。Integration Service が最後のチェックポイントからセッションを再開できるのは、出力が再現可能で決定性のある場合です。使用する値は次のとおりです。</p> <ul style="list-style-type: none"> - Always。入力データの順序がセッションの実行ごとに異なる場合でも、出力データの順序は常に同じです。 - 入力順による。すべての入力グループの入力データの順序がセッションの実行の度に常に同じである場合は、再現可能なデータを作成します。入力グループの入力データが順序付けられていない場合、出力は順序付けられません。 - Never。出力データの順序はセッションの実行ごとに異なります。トランスフォーメーションが再現可能なデータを作成しない場合、最後のチェックポイントから再開するようにリカバリを設定することはできません。 <p>デフォルトは「入力順による」です。</p>
出力が確定的かどうか	<p>トランスフォーメーションが、セッションの実行ごとに一貫した出力データを生成するかどうかを指定します。このトランスフォーメーションを使用するセッションでリカバリを実行するには、このプロパティを有効にする必要があります。</p> <p>デフォルトでは無効になっています。</p>

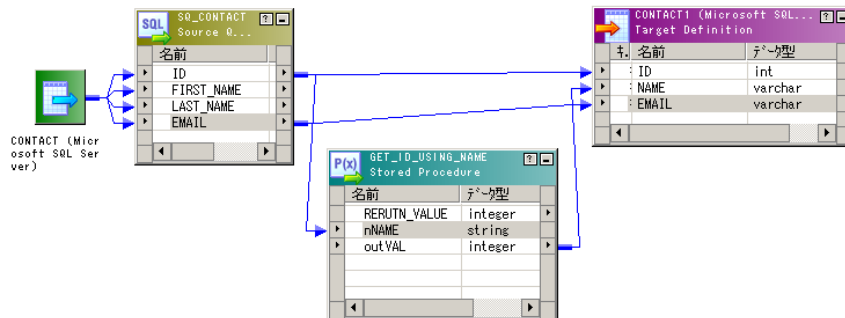
警告: トランスフォーメーションを繰り返し可能で一意に定まるものとして設定する場合は、データが繰り返し可能で一意に定まることを保証する必要があります。セッションとリカバリで同じデータが生成されないトランスフォーメーションを使用してセッションをリカバリしようすると、リカバリプロセスを実行した結果、データが破損する可能性があります。

ストアードプロシージャの変更

ストアードプロシージャのパラメータの数または戻り値が変更された場合、ストアードプロシージャをインポートし直すか、または手作業でストアードプロシージャトランスフォーメーションを編集します。Designer は、マッピングを開くたびにストアードプロシージャトランスフォーメーションを検査しません。また、トランスフォーメーションをインポートまたは作成したあと、Designer はそのストアードプロシージャを自動的に検証しません。ストアードプロシージャがトランスフォーメーションの設定と一致しない場合、セッションは失敗します。

接続されたトランスフォーメーションの設定

以下の図に、ソース修飾子からストアードプロシージャトランスフォーメーションの入力パラメータに ID を送るマッピングを示します。



ストアードプロシージャトランスフォーメーションは、出力パラメータをターゲットに渡します。ソース修飾子トランスフォーメーションのデータの各行は、ストアードプロシージャトランスフォーメーションを介してデータを渡します。

必須条件ではありませんが、ほとんどすべての接続されたストアードプロシージャトランスフォーメーションは、入力パラメータと出力パラメータを含んでいます。必須の入力パラメータは、ストアードプロシージャトランスフォーメーションの入力ポートとして指定されています。出力パラメータは、トランスフォーメーションの出力ポートとして示されます。戻り値もやはり出力ポートであり、トランスフォーメーションの [ポート] 設定において [R] 値が選択されています。標準の接続されたストアードプロシージャをこの関数リストに表示させるためには、入力ポートと出力ポートがそれぞれ 1 つ以上なければなりません。

ストアードプロシージャからの出力パラメータと戻り値は、トランスフォーメーション内のほかの出力ポートと同じように使用されます。これらのポートを、他のトランスフォーメーションやターゲットにリンクできます。

接続されたストアードプロシージャトランスフォーメーションを設定するには：

1. マッピング内に Stored Procedure トランスフォーメーションを作成します。
2. Stored procedure トランスフォーメーションの出力ポートを別のトランスフォーメーションまたはターゲットにドラッグします。
3. Stored Procedure トランスフォーメーションを開き、[プロパティ] タブを選択してください。
4. トランスフォーメーションの作成時にデータベースを選択していない場合は、[Connection Information] で適切なデータベースを選択します。
5. トランスフォーメーションのトレースレベルを選択します。
マッピングをテストする場合は、[Verbose Initialization] オプションを選択して、トランスフォーメーションが失敗した場合に多くの情報を取得できるようにします。
6. [OK] をクリックします。

接続されていないトランスフォーメーションの設定

コネクต์されていないストアードプロシージャトランスフォーメーションは、マッピングのデータフローに直接接続されません。ストアードプロシージャは次のどちらかで実行されます。

- **式から。** マッピングに含まれる別のトランスフォーメーション内で、式エディタで記述された式から呼び出されます。

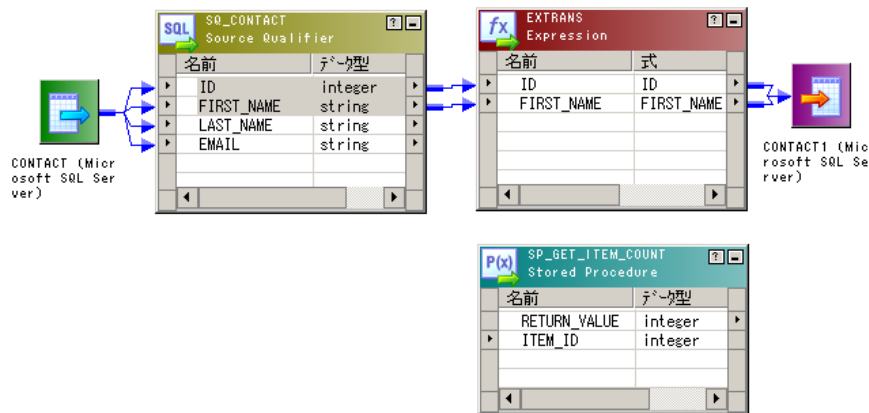
- **セッション実行前/実行後。**セッションの実行前または実行後に実行されます。

接続されていないストアードプロシージャトランスフォーメーションの実行方法を下記で説明します。

式からストアードプロシージャを呼ぶ場合

コネクต์されていないマッピングでは、ストアードプロシージャトランスフォーメーションはパイプラインに接続されません。

以下の図に、ストアードプロシージャトランスフォーメーションを参照する式トランスフォーメーションとのマッピングを示します。



ただし、接続されたマッピングの場合と同様に、マッピングを介してストアードプロシージャをデータフローに適用することができます。むしろ、式を使用してストアードプロシージャを呼び出すということは入力パラメータとしてストアードプロシージャに渡すデータを選択できることを意味するため、柔軟性がさらに高まります。

式の中でコネクต์されていないストアードプロシージャトランスフォーメーションを使用している場合は、出力パラメータの値をポートに返す手段が必要となります。以下のいずれかの方法を使って出力値を取得します。

- 出力値をローカル変数に割り当てる。
- 出力値をシステム変数 PROC_RESULT に割り当てる。

PROC_RESULT を使用することにより、戻りパラメータの値を直接出力ポートに割り当て、それを直接ターゲットに割り当てることができます。また、この2つのオプションを組み合わせ、1つの出力パラメータをPROC_RESULTに割り当て、他のパラメータを変数として割り当てることもできます。

PROC_RESULT は、式の中でのみ使用します。PROC_RESULT も変数も使用しない場合、その式を含むポートはNULLを取得します。PROC_RESULT は、接続されたルックアップトランスフォーメーション内やストアードプロシージャトランスフォーメーションの [Call Text] 内で使用することはできません。

ストアードプロシージャをネストして、1つのストアードプロシージャの出力パラメータを別のストアードプロシージャに渡す場合は、PROC_RESULT を使ってその値を渡します。

Integration Service は、式トランスフォーメーションから接続されていないストアードプロシージャトランスフォーメーションを呼び出します。ストアードプロシージャトランスフォーメーションに2つの入力ポートと1つの出力ポートがあることに注意してください。これらの3つのポートはすべてStringデータタイプです。

式からストアードプロシージャを呼び出すには：

1. マッピング内にストアードプロシージャトランスフォーメーションを作成します。
2. 出力ポートと変数ポートをサポートする任意のトランスフォーメーション内で、ストアードプロシージャを呼び出す新しい出力ポートを作成します。出力ポートに名前を付けます。

ストアードプロシージャを呼び出す出力ポートは、式をサポートしていなければなりません。式をどのように設定するかによって、出力ポートが出力パラメータの値または戻り値のどちらを含むかが決まります。

3. そのポートに対して式のエディタを開きます。

式のエディタ内でトランスフォーメーション言語の:SP キーワードを使用して、新しいポートの値をストアードプロシージャ呼び出しとして設定します。これを適切に設定する最も簡単な方法は、式エディタ内で[ストアードプロシージャ] ノードを選択し、一覧表示されているストアードプロシージャトランスフォーメーションの名前をクリックすることです。標準の接続されたストアードプロシージャをこの関数リストに表示させるためには、入力ポートと出力ポートがそれぞれ1つ以上なければなりません。

式のエディタ内に、一対の空のかっこと共にストアードプロシージャが表示されます。必要な入力パラメータや出力パラメータが、式のエディタの左下部分に表示されます。

4. 入力パラメータを渡して出力パラメータまたは戻り値を取得するように式を設定します。

その際、式のエディタに表示されているパラメータが入力パラメータであるか出力パラメータであるかがわかっている必要があります。変数またはポート名を、ストアードプロシージャに表示される順序でかっこ内に挿入します。ポートおよび変数のデータタイプは、ストアードプロシージャに渡されるパラメータのデータタイプと一致しなければなりません。

たとえば、ストアードプロシージャをクリックすると次のような表示が現れます。

```
:SP.GET_NAME_FROM_ID()
```

このストアードプロシージャは入力パラメータとして整数値を取り、出力パラメータとして文字列値を返します。出力パラメータまたは戻り値がどのように取得されるかは、出力パラメータの数や、戻り値を取得する必要があるかどうかに応じて変わります。

ストアードプロシージャが1つの出力パラメータまたは戻り値（両方ではない）を返すのであれば、予約変数 PROC_RESULT を出力変数として使用します。上記の例では、次のような式になります。

```
:SP.GET_NAME_FROM_ID(inID, PROC_RESULT)
```

inID は、トランスフォーメーションの入力ポートまたは変数です。PROC_RESULT の値が式の出力ポートに適用されます。

ストアードプロシージャが複数の出力パラメータを返す場合は、それぞれの出力パラメータのための変数を作成しなければなりません。たとえばストアードプロシージャ式に対して varOUTPUT2 というポートと varOUTPUT1 という変数を作成した場合、この式は次のようになります。

```
:SP.GET_NAME_FROM_ID(inID, varOUTPUT1, PROC_RESULT)
```

2 番目の出力ポートの値は式の出力ポートに適用され、最初の出力ポートの値は varOUTPUT1 に適用されます。出力パラメータは、ストアードプロシージャ内で宣言された順序で返されます。

これらの式において、ポートおよび変数のデータタイプは、入出力変数および戻り値のデータタイプにすべて一致しなければなりません。

5. [検証] をクリックして式を検査します。次に [OK] をクリックして式のエディタを閉じます。

式を検証することによって、ストアードプロシージャ内のパラメータのデータタイプが式のパラメータのデータタイプと一致していることが保証されます。

6. [OK] をクリックします。

マッピングを保存する際、Designer はストアードプロシージャ式を検証しません。ストアードプロシージャの式の設定が正しくない場合、セッションは失敗します。ストアードプロシージャを使用するマッピングをテストする場合は、[Override Tracing] セッションオプションを Verbose モードに設定し、ストアードプロシージャが失敗した場合に実行を停止するように [On Stored Procedure] セッションオプションを設定します。これらのセッションオプションは、セッションプロパティの [設定オブジェクト] タブの [Error Handling] 設定で設定します。

ポートに対して入力された式内のストアードプロシージャは、そのポートを通過するすべての値に作用する必要はありません。たとえば IIF 文を使用して、5 から始まる ID 番号のような特定の値だけをストアードプロシージャに渡し、それ以外の値はスキップするように設定することができます。また、ネストしたストアードプロシ

ジャを設定して、1つのストアードプロシージャの戻り値を2番目のストアードプロシージャの入力パラメータとすることもできます。

セッション実行前または実行後のストアードプロシージャの呼び出し

ストアードプロシージャをセッションごとに1回だけ実行したい場合もあります。たとえばマッピングを実行する前にターゲットデータベースにテーブルが存在することを確認する必要がある場合は、ターゲットのロード前に実行されるストアードプロシージャを使用してテーブルをチェックし、その結果に基づいてワークフローの実行を続行するか中止するかを判断することができます。ストアードプロシージャは、ソース、ターゲット、またはその他の接続されているデータベースに対して実行することができます。

ロード前または後に実行されるストアードプロシージャを作成するには：

1. マッピング内にストアードプロシージャトランスフォーメーションを作成します。
2. ストアードプロシージャトランスフォーメーションをダブルクリックし、次に[プロパティ] タブを選択します。
3. ストアードプロシージャの名前を入力します。
ストアードプロシージャをインポートした場合、ストアードプロシージャ名はデフォルトで表示されます。ストアードプロシージャを手作業で設定した場合は、ストアードプロシージャの名前を入力します。
4. [接続情報] で、ストアードプロシージャが格納されているデータベースを選択します。
5. ストアードプロシージャの呼び出し文字列を入力します。

呼び出しテキストでは、ストアードプロシージャの名前に続けてかっこ内に適切な入力パラメータを指定します。入力パラメータがない場合は中が空白の一對のかっこを記述します。そうでないとストアードプロシージャ呼び出しが失敗します。SQL文EXECを書く必要はなく、:SP キーワードを使用する必要もありません。たとえば check_disk_space というストアードプロシージャを呼び出すには、次のテキストを入力します。

```
check_disk_space()
```

文字列の入力パラメータを渡す場合はそれを引用符で囲まずに入力します。文字列の中に空白が含まれている場合は、パラメータを二重引用符で囲みます。たとえばストアードプロシージャ check_disk_space が入力パラメータとしてマシン名を取る場合は、次のように入力します。

```
check_disk_space(oracle_db)
```

セッション実行前または実行後に起動されるストアードプロシージャを介して日時値を渡す場合、Informatica デフォルト日付フォーマットに従い、二重引用符で囲んで、次のような値にする必要があります。

```
SP("12/31/2000 11:45:59")
```

PowerCenter パラメータおよび変数は、呼び出しテキストの中に使用できます。パラメータファイルで定義可能なパラメータまたは変数タイプを使用します。パラメータまたは変数は、呼び出しテキストの中に入力することも、あるいは呼び出しテキストとして使用することもできます。たとえば、セッションパラメータ \$ParamMyCallText を呼び出しテキストとして使用することも、あるいは \$ParamMyCallText をパラメータファイル内の呼び出しテキストに設定することもできます。

注: セッション前およびセッション後のプロシージャの入力パラメータについては、プロシージャパラメータではなく値を入力する必要があります。

6. ストアードプロシージャのタイプを選択します。
ストアードプロシージャのタイプには以下のオプションがあります。
 - **ソースのロード前。** ストアードプロシージャは、セッションがソースからデータを取得する前に実行されます。このオプションは、テーブルの存在を検査したり、一時テーブル内でデータの結合を実行したりする場合に役立ちます。

- **ソースのロード後。**ストアードプロシージャは、セッションがソースからデータを取得した後に実行されます。このオプションは、一時テーブルを削除するのに役立ちます。
 - **ターゲットのロード前。**ストアードプロシージャは、セッションがターゲットにデータを渡す前に実行されます。このオプションは、ターゲットテーブルやターゲットシステム上のディスク領域を検査するのに役立ちます。
 - **ターゲットのロード後。**ストアードプロシージャは、セッションがターゲットにデータを渡した後に実行されます。このオプションは、データベースのインデックスを再作成するのに便利です。
7. [Execution Order] を選択し、必要であれば上向き矢印または下向き矢印をクリックして順序を変更します。

セッションにおいて同時に実行される複数のストアードプロシージャ（例えば「ソースのロード後」で実行される2つのプロシージャ）を追加した場合、ストアードプロシージャ実行計画を設定して、Integration Service がそのストアードプロシージャを呼び出す順序を決定できます。変更したい各ストアードプロシージャに対してこの手順を繰り返す必要があります。

8. [OK] をクリックします。

リポジトリがマッピングを検証して保存しても、Designer はストアードプロシージャの式がエラーなしで実行されることを保証しません。ストアードプロシージャの式の設定が正しくない場合、セッションは失敗します。ストアードプロシージャを使用するマッピングをテストする場合は、[Override Tracing] セッションオプションを Verbose モードに設定し、ストアードプロシージャが失敗した場合に実行を停止するように [On Stored Procedure] セッションオプションを設定します。これらのセッションオプションは、セッションプロパティの [設定オブジェクト] タブの [Error Handling] 設定で設定します。

セッション実行前に起動されるストアードプロシージャおよびセッション実行後に起動されるストアードプロシージャの実行中に呼び出された出力パラメータまたは戻り値は、その値を取得することができないために失われてしまいます。そのような値を取得する必要がある場合は、ストアードプロシージャがデータベース内のテーブルに値を保存するように設定しなければなりません。

エラー処理

ストアードプロシージャは、「ゼロによる除算」や「行の終わり」といったデータベースエラーを返すことがあります。ストアードプロシージャ実行中に発生したデータベースエラーの最終的な結果は、ストアードプロシージャを実行したタイミングとセッションの設定によって変わります。

セッションは、セッション実行前またはセッション実行後に起動されるストアードプロシージャでエラーを検出したときにそのセッションの実行を中止するかまたは続行するように設定することができます。デフォルトでは、セッション実行前またはセッション実行後に起動されるストアードプロシージャでデータベースエラーが発生すると、Integration Service はセッションを中止します。

セッション実行前のエラー

読み込み前およびロード前のストアードプロシージャは、セッション実行前に起動されるストアードプロシージャとみなされます。どちらのストアードプロシージャも、Integration Service がソースデータの読み込みを開始する前に実行されます。セッション実行前に起動されるストアードプロシージャの実行中にデータベースエラーが発生した場合、Integration Service はセッションの設定に応じて異なる処理を行います。

- ストアードプロシージャエラーの発生時に実行を中止するようにセッションが設定されている場合、Integration Service はそのセッションに失敗します。
- ストアードプロシージャエラーの発生時に実行を継続するようにセッションが設定されている場合、Integration Service はそのセッションを続行します。

セッション実行後のエラー

読み込み後およびロード後のストアードプロシージャは、セッション実行後に起動されるストアードプロシージャとみなされます。どちらのストアードプロシージャも、Integration Service がすべてのデータをデータベースにコミットした後で実行されます。セッション実行後に起動されるストアードプロシージャの実行中にデータベースエラーが発生した場合、Integration Service はセッションの設定に応じて異なる処理を行います。

- ストアドプロシージャエラーの発生時に実行を中止するようにセッションが設定されている場合、Integration Service はそのセッションに失敗します。

ただし、この時点で Integration Service はすでにすべてのデータをセッションターゲットにコミットしています。

- ストアドプロシージャエラーの発生時に実行を継続するようにセッションが設定されている場合、Integration Service はそのセッションを続行します。

セッションのエラー

セッション実行中に発生した接続されているストアードプロシージャまたは接続されていないストアードプロシージャのエラーは、セッションエラー処理オプションの影響を受けません。データベースが特定の行に対してエラーを返すと、Integration Service はその行をスキップして次の行を処理します。他の行トランスフォーマーセッションエラーの場合と同様に、スキップされた行はセッションログに記録されます。

サポートされるデータベース

この節では、Oracle をはじめ、Informix、Microsoft SQL Server、Sybase などのデータベースについてサポートされているオプションを説明します。

関連項目：

- [「ストアードプロシージャの記述」 \(ページ 470\)](#)

SQL の宣言

データベースでは、ストアードプロシージャを作成する文は、次の Oracle ストアドプロシージャのようになります。

```
create or replace procedure sp_combine_str
(str1_inout IN OUT varchar2,
str2_inout IN OUT varchar2,
str_out OUT varchar2)
is
begin
str1_inout := UPPER(str1_inout);
str2_inout := upper(str2_inout);
str_out := str1_inout || ' ' || str2_inout;
end;
```

この例では、Oracle 文は CREATE OR REPLACE PROCEDURE で始まっています。Oracle はストアードプロシージャとストアード関数の両方をサポートしているため、Oracle だけがオプションの CREATE FUNCTION 文を使用します。

パラメータの種類

ストアードプロシージャには3種類のパラメータがあります。

- **IN**。ストアードプロシージャに渡す必要があるパラメータを定義します。
- **OUT**。ストアードプロシージャの戻り値となるパラメータを定義します。
- **INOUT**。入力と出力の両方としてパラメータを定義します。このタイプのパラメータをサポートしているのは Oracle だけです。

マッピングにおける入出力ポート

Oracle は INOUT パラメータをサポートするため、ストアードプロシージャトランスフォーメーションのポートを、同じストアードプロシージャパラメータに対して入力ポートと出力ポートの両方として機能させることができます。ほかのデータベースでは、1つのポートに対して入力と出力の両方のチェックボックスを選択することはできません。

サポートされる戻り値の型

データベースが異なれば、サポートされる戻り値のデータタイプも異なります。Informix だけはユーザー定義の戻り値をサポートしていません。

式のルール

コネクタされていないストアードプロシージャトランスフォーメーションは、別のトランスフォーメーション内の式から呼び出すことができます。式を設定するときは、次の規則とガイドラインに従う必要があります。

- 1つの出力パラメータが PROC_RESULT 変数を使って返されます。
- 式内でストアードプロシージャを使用するときは、:SP 参照修飾子を使用します。タイプミスを防ぐためには、式のエディタ内で [ストアードプロシージャ] ノードを選択し、ストアードプロシージャの名前をダブルクリックします。
- ただし、1つのストアードプロシージャトランスフォーメーションの同じインスタンスを1つのマッピング内でコネクタされたモードとコネクタされていないモードの両方で実行することはできません。その場合は、トランスフォーメーションのインスタンスを別個に作成する必要があります。
- 式内の入出力パラメータは、ストアードプロシージャトランスフォーメーション内の入出力ポートと一致しなければなりません。ストアードプロシージャに入力パラメータがある場合は、ストアードプロシージャトランスフォーメーションにも入力ポートがなければなりません。
- ストアードプロシージャを含める式を書く場合は、ストアードプロシージャおよびストアードプロシージャトランスフォーメーションに示されているのと同じ順にパラメータを記述する必要があります。
- 式のパラメータには、ストアードプロシージャトランスフォーメーションのすべてのパラメータを含まなければなりません。入力パラメータを省略することはできません。必要であれば、ダミーの変数をストアードプロシージャに渡します。
- 式内の引数は、ストアードプロシージャトランスフォーメーションの引数と同じデータタイプおよび精度を持たなければなりません。
- PROC_RESULT を使用して、ストアードプロシージャの式の出力パラメータをターゲットへ直接適用します。出力パラメータに変数を使用して結果をターゲットへ直接渡すことはできません。同じトランスフォーメーション内で結果を出力ポートへ渡すにはローカル変数を使用します。

- ストアドプロシージャをネストすることにより、1つのストアドプロシージャの戻り値を別のストアドプロシージャの入力パラメータとして渡すことができます。たとえば次の2つのストアドプロシージャがあるとします。

- get_employee_id (employee_name)

- get_employee_salary (employee_id)

get_employee_id の戻り値が従業員 ID 番号であるとしてします。このストアドプロシージャを次のようにネストすることができます。

```
:sp.get_employee_salary (:sp.get_employee_id (employee_name))
```

ストアドプロシージャは、複数レベルでネストすることができます。

- 文字列のパラメータを一重引用符で囲んではなりません。空白を含まない入力パラメータに対しては、引用符を使用しません。入力パラメータに空白が含まれる場合は、二重引用符を使用します。

ストアドプロシージャトランスフォーメーションに関するヒント

ストアドプロシージャの不必要なインスタンスを使用しないでください。

マッピングにおいてストアドプロシージャが実行されるたびに、セッションはそのストアドプロシージャがデータベース内で完了するのを待たなければなりません。これを防ぐためには次の2つの方法があります。

- **行数を減らす。** ストアドプロシージャトランスフォーメーションの前にアクティブなトランスフォーメーションを使用して、ストアドプロシージャへ渡さなければならない行の数を少なくします。あるいは、ストアドプロシージャへ渡される値をその前にテストする式を作成し、本当に渡す必要がある値かどうかを確認します。
- **式を作成する。** ストアドプロシージャに使用されるロジックのほとんどは、Designer で式を使用してレプリケートすることができます。

ストアドプロシージャトランスフォーメーションのトラブルシューティング

セッションログファイルに「stored procedure not found」というエラーが記録されました。

ストアドプロシージャが正しいデータベース上で実行されることを確認してください。デフォルトでは、ストアドプロシージャトランスフォーメーションはターゲットデータベースを使用してストアドプロシージャを実行します。マッピング内でトランスフォーメーションをダブルクリックし、[プロパティ] タブを選択します。[接続情報] でどのデータベースが選択されているかを調べてください。

Microsoft SQL Server ストアドプロシージャを使用したところ、出力パラメータが返されません。

戻り値を保持するパラメータが、ストアードプロシージャで OUTPUT として宣言されていないか確認してください。Microsoft SQL Server の場合、OUTPUT は入出力ポートを意味します。マッピングにおいて、ポートに対して [I] ボックスと [O] ボックスの両方にチェックマークを付けている可能性があります。入力ポートのチェックマークを外してください。

以前はエラーのなかったセッションがストアードプロシージャで失敗するようになりました。

ストアードプロシージャトランスフォーメーションに関係する問題で最も一般的な原因は、データベース内のストアードプロシージャに変更が加えられたことにあります。ストアードプロシージャ内の入出力パラメータまたは戻り値が変更されると、ストアードプロシージャトランスフォーメーションは無効になります。ストアードプロシージャをインポートし直すか、またはストアードプロシージャに対して適切なポートを手作業で追加、削除、または変更しなければなりません。

マッピングを編集したらセッションが無効になりました。これはなぜですか？

ストアードプロシージャトランスフォーメーションに変更を加えたことが原因でセッションが無効になってしまふことがあります。この最も一般的な理由には、ストアードプロシージャの種類を変更してしまった（たとえばノーマルタイプのストアードプロシージャを Post-load Source タイプのストアードプロシージャに変更した）ことが挙げられます。

第 29 章

トランザクション制御トランスフォーマーション

この章では、以下の項目について説明します。

- [トランザクション制御トランスフォーマーションの概要, 486 ページ](#)
- [トランザクション制御トランスフォーマーションのプロパティ, 487 ページ](#)
- [マッピングでのトランザクション制御トランスフォーマーションの使用, 489 ページ](#)
- [マッピングのガイドラインと検証, 491 ページ](#)
- [トランザクション制御トランスフォーマーションの作成, 492 ページ](#)

トランザクション制御トランスフォーマーションの概要

PowerCenter では、トランザクション制御トランスフォーマーションを通過する行のセットに基づいてトランザクションのコミットおよびロールバックを制御することができます。トランザクション制御トランスフォーマーションはアクティブなトランスフォーマーションです。トランザクションは、コミット行またはロールバック行によって関連付けられた行のセットです。入力行の数の変化に応じてトランザクションを定義できます。トランザクションは、従業員 ID や注文入力日といった共通のキーによって指定される行のグループに基づいて定義することもできます。

PowerCenter では、以下のレベルでトランザクション制御を定義できます。

- **マッピング内。**マッピング内では、トランザクション制御トランスフォーマーションを使ってトランザクションを定義します。トランザクション制御トランスフォーマーション内で式を使ってトランザクションを定義します。その式の戻り値に基づいて、コミットするかロールバックするか、あるいはトランザクションの変更を行わずに継続するかを選択できます。
- **セッション内。**セッションの設定時に、セッションをユーザー定義コミット用に設定します。トランザクションのコミットまたはロールバックは、Integration Service での行の変換またはターゲットへの行の書き込みが失敗した場合に選択することができます。

セッションの実行時に、Integration Service はトランスフォーマーションに入力される各行について式を評価します。コミット行が評価されると、トランザクション内のすべての行がターゲットにコミットされます。Integration Service は、ロールバック行を評価すると、トランザクションのすべての行をターゲットからロールバックします。

マッピングのターゲットがフラットファイルの場合は、Integration Service が新しいトランザクションを開始するたびに出力ファイルを生成できます。ターゲットフラットファイル名は動的に指定できます。

注: また、他のトランスフォーメーションプロパティでトランスフォーメーション範囲を使ってトランスフォーメーションを定義することもできます。

トランザクション制御トランスフォーメーションのプロパティ

トランザクション制御トランスフォーメーションは、コミットする条件を定義し、トランザクションターゲットからトランザクションをロールバックするのに使用します。トランザクションターゲットとしては、リレーショナルターゲット、XML ターゲット、動的 MQSeries ターゲットなどが挙げられます。これらのパラメータは、[プロパティ] タブのトランザクション制御式で定義します。トランザクションは、コミット行またはロールバック行によって関連付けられた行または行のセットです。行の数は、各トランザクションによって異なります。

トランザクション制御トランスフォーメーションの設定時に、以下のコンポーネントを定義します。

- **[トランスフォーメーション] タブ。** このタブでは、トランスフォーメーションの名前を変更したり、説明を追加することができます。
- **[ポート] タブ。** トランザクション制御トランスフォーメーションに入出力ポートを追加することができます。
- **[プロパティ] タブ。** トランザクションに対して、コミット、ロールバック、または何もしないことを意味するフラグ設定をするトランザクション制御式を定義できます。
- **[メタデータエクステンション] タブ。** 情報をトランザクション制御トランスフォーメーションに関連付けることで、リポジトリに格納されているメタデータを拡張することができます。

[プロパティ] タブ

[プロパティ] タブでは、以下のプロパティを設定することができます。

- トランザクション制御式
- トレースレベル

[トランザクション制御条件] フィールドにトランザクション制御式を入力します。トランザクション制御式は、IIF 関数を使用して、各行が条件を満たすかどうかをテストします。式には以下の構文を使用します。

IIF (condition, value1, value2)

式には、条件の戻り値に基づいて Integration Service が実行する動作を表す値が含まれます。Integration Service は、行ごとに条件を評価します。戻り値に基づき、Integration Service が行をコミットするか、行をロールバックするか、あるいは行のトランザクション変更を実行しないかが決定されます。Integration Service は、式の戻り値に基づいてコミットまたはロールバックを発行すると、新しいトランザクションを開始します。トランザクション制御式を作成するときには、式エディタで以下の組み込み変数を使用します。

- **TC_CONTINUE_TRANSACTION。** Integration Service は、この行にはトランザクション変更を行いません。これは式のデフォルト値です。
- **TC_COMMIT_BEFORE。** Integration Service はトランザクションをコミットし、新しいトランザクションを開始し、現在の行をターゲットに書き出します。現在の行は、新しいトランザクション内にあります。
- **TC_COMMIT_AFTER。** Integration Service は、現在の行をターゲットに書き出し、トランザクションをコミットし、新しいトランザクションを開始します。現在の行は、コミットされたトランザクション内にあります。

- **TC_ROLLBACK_BEFORE**。Integration Service は、現在のトランザクションをロールバックし、新しいトランザクションを開始し、現在の行をターゲットに書き出します。現在の行は、新しいトランザクション内にあります。
- **TC_ROLLBACK_AFTER**。Integration Service は、現在の行をターゲットに書き出し、トランザクションをロールバックし、新しいトランザクションを開始します。現在の行は、ロールバックされたトランザクション内にあります。

トランザクション制御式がコミット、ロールバック、または継続以外の値に評価されると、Integration Service はセッションに失敗します。

例

トランザクション制御を使って、注文入力日に基づいて注文情報を書き出したいとします。所定の日付に入力されたすべての注文が、同じトランザクション内でターゲットにコミットされる必要があります。これを行うためには、以下のトランスフォーメーションを含むマッピングを作成します。

- **ソータトランスフォーメーション**。ソースデータを注文入力日でソートします。
- **式トランスフォーメーション**。入力された日付が新しい日付かどうかを判定するためにローカル変数を使用します。

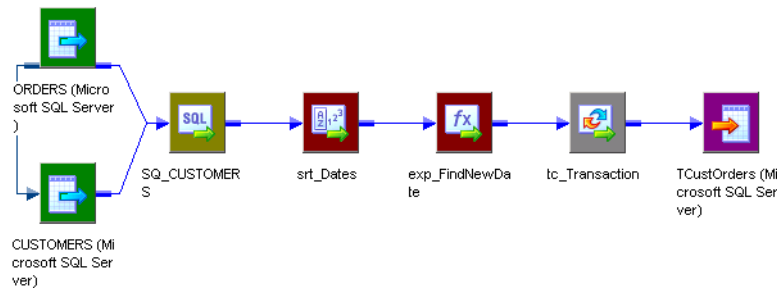
次の表に、式トランスフォーメーションのポートを示します。

ポート名	式	説明
DATE_ENTERED	DATE_ENTERED	入出力ポート。 入力された日付を受け取って、渡します。
NEW_DATE	IIF (DATE_ENTERED=PREVDATE, 0,1)	変数ポート。 DATE_ENTERED のカレント値と、変数ポート PREV_DATE に格納されている DATE_ENTERED の値を比較します。
PREV_DATE	DATE_ENTERED	変数ポート。 Integration Service が NEW_DATE ポート を評価した後で、DATE_ENTERED の値を受け取ります。
DATE_OUT	NEW_DATE	出力ポート。 NEW_DATE からのフラグを Transaction Control トランスフォーメーションに渡します。
注: Integration Service は依存関係によってポートを評価します。 トランスフォーメーションに表示されるポートの順序は、評価の順序（入力ポート、変数ポート、出力ポート）に一致している必要があります。		

- **トランザクション制御トランスフォーメーション**。Integration Service が新しい注文入力日を検出したときにデータをコミットするように、以下のトランザクション制御式を作成します。

IIF(NEW_DATE = 1, TC_COMMIT_BEFORE, TC_CONTINUE_TRANSACTION)

以下の図に、トランザクション制御トランスフォーメーションを使用するマッピングの例を示します。



マッピングでのトランザクション制御トランスフォーメーションの使用

トランザクション制御トランスフォーメーションはトランザクションジェネレータです。トランザクション制御トランスフォーメーションは、マッピングでトランザクション境界を定義および再定義します。また、先行するアクティブソースまたはトランザクションジェネレータから渡されるトランザクション境界を削除したり、新しいトランザクション境界を後続に渡したりします。

トランザクションを生成するように設定したカスタムトランスフォーメーションを使ってトランザクション境界を定義することもできます。

トランザクション制御トランスフォーメーションは、マッピング内の後続のトランスフォーメーションおよびターゲットに対して有効な場合と無効な場合があります。受け取ったトランザクション境界を削除するトランスフォーメーションをトランザクション制御トランスフォーメーションの後に配置した場合、トランザクション制御トランスフォーメーションは後続のトランスフォーメーションまたはターゲットに対して無効になります。トランザクション境界を削除するものには、以下のアクティブソースおよびトランスフォーメーションが含まれます。

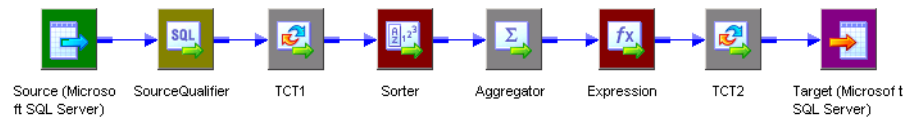
- トランスフォーメーション範囲がすべての入力レベルのアグリゲータトランスフォーメーション
- トランスフォーメーション範囲がすべての入力レベルのジョイナトランスフォーメーション
- トランスフォーメーション範囲がすべての入力レベルのランクトランスフォーメーション
- トランスフォーメーション範囲がすべての入力レベルのソータトランスフォーメーション
- トランスフォーメーション範囲が「すべての入力」レベルであるカスタムトランスフォーメーション
- トランザクションを生成するように設定されたカスタムトランスフォーメーション
- トランザクション制御トランスフォーメーション
- 先行する複数のトランザクション制御ポイントに接続されている、カスタムトランスフォーメーションなどの複数の入力グループを持つトランスフォーメーション

ターゲットに対して無効なトランザクション制御トランスフォーメーションを含むマッピングは、有効になる場合と無効になる場合があります。マッピングを保存または検証すると、Designer はトランザクション制御トランスフォーメーションがターゲットに対して無効であることを示すメッセージを表示します。

トランザクション制御トランスフォーメーションは、ターゲットに対して無効であっても、後続のトランスフォーメーションに対しては有効である場合があります。トランザクション範囲が「トランザクション」レベルの後続のトランスフォーメーションでは、先行するトランザクション制御トランスフォーメーションによって定義されたトランザクション境界を使用することができます。

以下の図に示した有効なマッピングでは、トランザクション制御トランスフォーメーションはソーストランスフォーメーションに対しては有効ですが、ターゲットに対しては無効です。

この例では、TCT1 トランスフォーメーションはターゲットに対しては無効ですが、ソーストランスフォーメーションに対しては有効です。 ソーストランスフォーメーションのトランスフォーメーション範囲プロパティは [トランザクション] です。 TCT1 で定義されたトランザクション境界が使用されます。 アグリゲータのトランスフォーメーション範囲プロパティは [すべての入力] です。 TCT1 で定義されたトランザクション境界が削除されます。 TCT2 トランスフォーメーションは、ターゲットに対する有効なトランザクション制御トランスフォーメーションです。

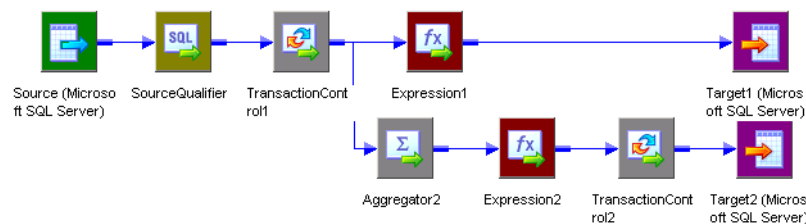


複数のターゲットを持つトランザクション制御マッピングの例

トランザクション制御トランスフォーメーションは、あるターゲットに対しては有効、別のターゲットに対しては無効となる場合があります。

各ターゲットが有効なトランザクション制御トランスフォーメーションに接続されている場合、そのマッピングは有効です。マッピング内のターゲットのうち 1 つでも有効なトランザクション制御トランスフォーメーションに接続されていないものがある場合、そのマッピングは無効です。

以下の図に、有効と無効の両方のトランザクション制御トランスフォーメーションを含む、有効なマッピングを示します。

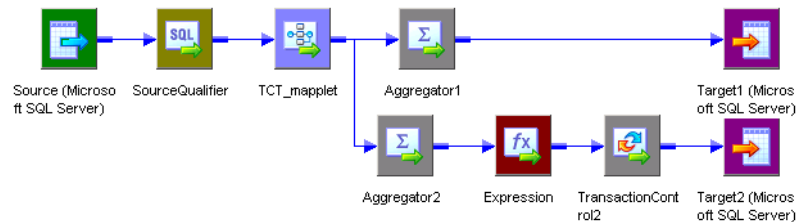


Integration Service は TransactionControl1 を処理し、トランザクション制御式を評価して、トランザクション境界を作成します。マッピングには、TransactionControl1 と Target1 の間にトランザクション境界を削除するトランスフォーメーションは含まれず、TransactionControl1 は Target1 に対して有効になります。Integration Service は、Target1 に対して TransactionControl1 が定義したトランザクション境界を使用します。

しかし、マッピングには TransactionControl1 と Target2 の間にトランザクション境界を削除するトランスフォーメーションが含まれているため、TransactionControl1 は Target2 に対しては無効になります。Integration Service でトランスフォーメーション範囲を [すべての入力] に設定してアグリゲータ 2 を処理すると、TransactionControl1 で定義されたトランザクション境界が削除され、オープントランザクション内のすべての行が出力されます。次に Integration Service は、TransactionControl2 を評価し、トランザクション境界を作成して、そのトランザクション境界を Target2 に対して使用します。

TransactionControl1 でロールバックが発生した場合、Integration Service は Target1 からの行だけをロールバックします。Target2 からの行はロールバックしません。

以下の図に、有効と無効の両方のトランザクション制御トランスフォーメーションを含む、無効なマッピングを示します。



マップレット TCT_mapplet には、トランザクション制御トランスフォーメーションが含まれています。これは Target1 および Target2 に対して無効です。アグリゲータ 1 トランスフォーメーション範囲プロパティは [すべての入力] です。これは Target1 に対するアクティブソースです。アグリゲータ 2 トランスフォーメーション範囲プロパティは [すべての入力] です。これは Target2 に対するアクティブソースです。TransactionControl2 トランスフォーメーションは Target2 に対して有効です。

Target1 が有効なトランザクション制御トランスフォーメーションに接続していないため、このマッピングは無効です。

マッピングのガイドラインと検証

トランザクション制御トランスフォーメーションを含むマッピングを作成するときには、以下の規則とガイドラインに従ってください。

- マッピングに XML ターゲットが含まれ、コミット時に追加または新しいドキュメントを作成するように選択した場合、入力グループは同じトランザクション制御点からデータを受け取る必要があります。
- リレーショナルターゲット、XML ターゲット、および動的な MQSeries ターゲット以外のターゲットに接続されているトランザクション制御トランスフォーメーションは、リレーショナルターゲット、XML ターゲット、および動的な IBM MQSeries ターゲットに対して無効になります。
- 各ターゲットインスタンスは必ずトランザクション制御トランスフォーメーションに接続する必要があります。
- 複数のターゲットを 1 つのトランザクション制御トランスフォーメーションに接続することができます。
- 1 つのターゲットに接続できる有効なトランザクション制御トランスフォーメーションは 1 つだけです。
- シーケンスジェネレータトランスフォーメーションで始まるパイプラインブランチには、トランザクション制御トランスフォーメーションを配置できません。
- 動的ルックアップトランスフォーメーションとトランザクション制御トランスフォーメーションを同じマッピングで使う場合、ロールバックされたトランザクションはターゲットデータと同期しない可能性があります。
- トランザクション制御トランスフォーメーションは、あるターゲットに対しては有効、別のターゲットに対しては無効となる場合があります。各ターゲットが有効なトランザクション制御トランスフォーメーションに接続されている場合、そのマッピングは有効です。
- マッピング内のすべてのターゲットを有効なトランザクション制御トランスフォーメーションに接続する必要があります。あるいは、どのターゲットも接続しない必要があります。

トランザクション制御トランスフォーメーションの作成

トランザクション制御トランスフォーメーションをマッピングに追加するには、以下の手順を実行します。

1. Mapping Designer で、[トランスフォーメーション] - [作成] をクリックします。[トランザクション制御] トランスフォーメーションを選択します。
2. トランスフォーメーションの名前を入力します。
トランザクション制御トランスフォーメーションの命名規則は、「TC_トランスフォーメーション名」です。
3. トランスフォーメーションの説明を入力します。
この説明は Repository Manager でトランスフォーメーションの詳細情報を表示すると確認でき、トランスフォーメーションの処理内容がわかるようになっています。
4. [作成] をクリックします。
Designer がトランザクション制御トランスフォーメーションを作成します。
5. [完了] をクリックします。
6. ポートをトランスフォーメーションにドラッグします。
Designer は、対象ポートのそれぞれについて入出力ポートを作成します。
7. [トランスフォーメーションの編集] ダイアログボックスを開き、[ポート] タブを選択します。
ポートの追加、ポート名の変更、ポート説明の追加、およびデフォルト値の入力が行えます。
8. [プロパティ] タブを選択します。コミットとロールバックの動作を定義するトランザクション制御式を入力します。
9. [メタデータエクステンション] タブを選択します。トランザクション制御トランスフォーメーションのメタデータエクステンションを作成または編集します。
10. [OK] をクリックします。

第 30 章

共有体トランスフォーメーション

この章では、以下の項目について説明します。

- [共有体トランスフォーメーションの概要, 493 ページ](#)
- [グループおよびポートに関する作業, 494 ページ](#)
- [共有体トランスフォーメーションの作成, 494 ページ](#)
- [マッピングにおける共有体トランスフォーメーションの使用, 495 ページ](#)

共有体トランスフォーメーションの概要

共有体トランスフォーメーションは、複数のパイプラインまたはパイプラインブランチからのデータを 1 つのパイプラインブランチに結合する場合に使用する、複数の入力グループを持つトランスフォーメーションです。共有体トランスフォーメーションが複数のソースからデータを結合するやり方は、UNION ALL SQL 文を使って 2 つ以上の SQL 文の結果を結合するのと似ています。共有体トランスフォーメーションは、UNION ALL 文と同様に、重複行を削除しません。共有体トランスフォーメーションはアクティブなトランスフォーメーションです。

Data Integration Service は、すべての入力グループを並列に処理します。また、共有体トランスフォーメーションに接続されたソースを読み込むと同時に、データブロックをトランスフォーメーションの入力グループにプッシュします。共有体トランスフォーメーションは、Integration Service からブロックを受け取る順番に基づいてデータブロックを処理します。

共有体トランスフォーメーションには、異なる種類のソースを接続できます。トランスフォーメーションは、一致するポートを使ってソースを結合し、入力グループと同じポートを使って 1 つの出力グループからデータを出力します。

共有体トランスフォーメーションは、カスタムトランスフォーメーションを使用して開発されています。

共有体トランスフォーメーションのルールとガイドライン

共有体トランスフォーメーションを使用するときには、以下の規則とガイドラインに従ってください。

- 複数の入力グループを作成できますが、出力グループは 1 つだけしか作成できません。
- すべての入力グループと出力グループは、一致するポートを持っている必要があります。精度、データタイプ、および位取りは、すべてのグループで同じである必要があります。
- 共有体トランスフォーメーションは、重複行を削除しません。重複行を削除するには、ルータやフィルタなどの別のトランスフォーメーションを追加する必要があります。
- 共有体トランスフォーメーションは、トランザクションを生成しません。

共有体トランスフォーメーションのコンポーネント

共有体トランスフォーメーションの設定時には、以下のコンポーネントを定義します。

- **【トランスフォーメーション】 タブ。**トランスフォーメーションの名前の変更、および説明の追加を行うことができます。
- **【プロパティ】 タブ。**トレースレベルを指定できます。
- **【グループ】 タブ。**入力グループの作成と削除を行うことができます。Designer は、作成したグループを【ポート】 タブに表示します。
- **【グループポート】 タブ。**入力グループのポートの作成と削除を行うことができます。Designer は、作成したポートを【ポート】 タブに表示します。

共有体トランスフォーメーションでは【ポート】 タブを変更することはできません。

グループおよびポートに関する作業

共有体トランスフォーメーションには複数の入力グループと 1 つの出力グループがあります。【グループ】 タブで入力グループを作成し、【グループポート】 タブでポートを作成します。

【グループ】 タブでは 1 つ以上の入力グループを作成できます。Designer は、デフォルトで 1 つの出力グループを作成します。この出力グループを編集または削除することはできません。

ポートを作成する場合、トランスフォーメーションからポートをコピーすることも、あるいは手動でポートを作成することもできます。【グループポート】 タブでポートを作成する際に、Designer は各入力グループに入力ポートを作成し、出力グループに出力ポートを作成します。Designer は、【グループポート】 タブで指定したポート名を各入力ポートおよび出力ポートに使用し、トランスフォーメーション内で一意となるように各ポート名に番号を追加します。また、各ポートには、データタイプ、精度、および位取りなどのメタデータは同じものを使用します。

【ポート】 タブには、作成したグループとポートが表示されます。【ポート】 タブのグループとポートの情報は編集できません。【グループ】 タブと【グループポート】 タブを使ってグループとポートを編集します。

共有体トランスフォーメーションの作成

以下の手順に従って、共有体トランスフォーメーションを作成します。

1. Mapping Designer で、【トランスフォーメーション】 - 【作成】 を選択します。
2. 【共有体トランスフォーメーション】 を選択し、トランスフォーメーションの名前を入力します。
共有体トランスフォーメーションの命名規則は、「UN_ トランスフォーメーション名」です。
3. トランスフォーメーションの説明を入力します。【作成】 をクリックし、次に【完了】 をクリックします。
4. 【グループ】 タブをクリックします。
5. 結合したい各パイプラインまたはパイプラインブランチに入力グループを追加します。
Designer は各グループにデフォルト名を割り当てます。ただし、名前は変更できます。
6. 【グループポート】 タブをクリックします。
7. 結合したい各データ行に新しいポートを追加します。
8. 名前やデータタイプなどのポートプロパティを入力します。

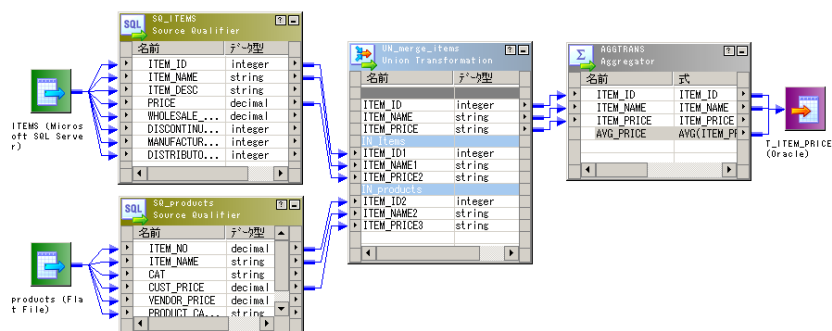
9. [プロパティ] タブをクリックし、トレースレベルを設定します。
10. [OK] をクリックします。

マッピングにおける共有体トランスフォーメーションの使用

共有体トランスフォーメーションは、非ブロッキング複数入力グループトランスフォーメーションです。入力グループは、1つのパイプラインの異なるブランチ、あるいは異なるソースパイプラインに接続することができます。

共有体トランスフォーメーションをマッピングに追加する際には、すべての入力グループで同じポートに接続していることを確認する必要があります。1つの入力グループのポートはすべて接続していても、別の入力グループのポートを接続していない場合、Integration Service は未接続のポートに NULL を渡します。

以下の図に、共有体トランスフォーメーションとのマッピングを示します。



マッピング内の共有体トランスフォーメーションが1つのトランザクションジェネレータからのデータを受け取った場合、Integration Service はトランザクション境界をプロパゲートします。トランスフォーメーションが複数のトランザクションジェネレータからデータを受け取った場合、Integration Service はすべての入力トランザクション境界を削除し、オープントランザクションに行を出力します。

第 31 章

構造化されていないデータのトランスフォーメーション

この章では、以下の項目について説明します。

- [構造化されていないデータのトランスフォーメーションの概要, 496 ページ](#)
- [構造化されていないデータオプションの設定, 497 ページ](#)
- [Data Transformation サービスのタイプ, 498 ページ](#)
- [構造化されていないデータのトランスフォーメーションのコンポーネント, 499 ページ](#)
- [構造化されていないデータのトランスフォーメーションポート, 502 ページ](#)
- [構造化されていないデータのトランスフォーメーションサービス名, 504 ページ](#)
- [リレーショナル階層, 505 ページ](#)
- [マッピング, 506 ページ](#)
- [構造化されていないデータのトランスフォーメーションの作成, 508 ページ](#)

構造化されていないデータのトランスフォーメーションの概要

構造化されていないデータのトランスフォーメーションは、メッセージング形式、HTML ページ、PDF ドキュメントなどの非構造化ファイル形式および半構造化ファイル形式を処理するトランスフォーメーションです。また、ACORD、HIPAA、HL7、EDI-X12、EDIFACT、SWIFT などの構造化形式も変換します。

構造化されていないデータのトランスフォーメーションは、PowerCenter セッションから Data Transformation サービスを呼び出します。Data Transformation は、非構造化ファイル形式および半構造化ファイル形式を変換するアプリケーションです。構造化されていないデータのトランスフォーメーションからのデータを Data Transformation サービスに渡し、データを変換してからパイプラインに返すことができます。構造化されていないデータのトランスフォーメーションは、アクティブまたはパッシブなトランスフォーメーションにすることができます。

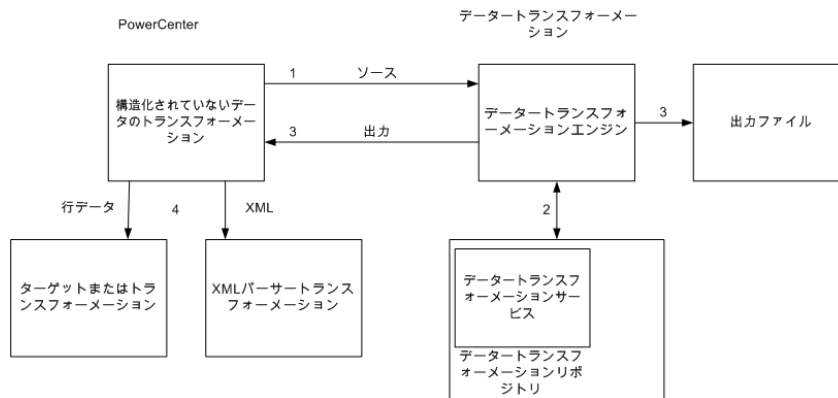
Data Transformation には以下のコンポーネントがあります。

- Informatica Developer トランスフォーメーションプロジェクトの設計および構成機能を提供する開発ツールです。
- Data Transformation サービス。Data Transformation リポジトリにデプロイされ、実行の準備が完了している Data Transformation プロジェクトです。

- Data Transformation リポジトリ。Informatica Developer で作成した実行可能なサービスを格納するためのディレクトリです。テストサービスや実稼動サービス用のリポジトリなど、さまざまなリポジトリにプロジェクトをデプロイできます。
- Data Transformation エンジン。リポジトリにデプロイするサービスを実行するプロセッサです。

サービスを実行した Data Transformation エンジンは、出力データを書き込んだり、出力データを Integration Service に返したりします。Integration Service に出力を返すときは、XML データを返します。Data TransformationXML を出力ポートに返すように構造化されていないデータトランスフォーメーションを設定したり、行データを返すように出力グループを設定したりできます。

以下の図に、PowerCenter 構造化されていないデータのトランスフォーメーションと、PowerCenter セッションからのデータを変換する Data Transformation サービス間のインタフェースを示します。



PowerCenter セッションから Data Transformation サービスを呼び出すと、以下のイベントが発生します。

1. 構造化されていないデータのトランスフォーメーションは、ソースデータを Data Transformation エンジンに渡します。
2. Data Transformation エンジンは、そのデータを変換する Data Transformation サービスを実行します。Data Transformation サービスは、Data Transformation リポジトリフォルダーにあります。
3. Data Transformation エンジンは、変換後のデータを直接出力ファイルに書き込むか、または変換後のデータを、構造化されていないデータのトランスフォーメーションに返します。
4. 構造化されていないデータトランスフォーメーションは、XML データまたはデータ行を返します。構造化されていないデータトランスフォーメーションが XML を返す場合は、構造化されていないデータトランスフォーメーションをマッピングの XML パーサートランスフォーメーションに接続します。

構造化されていないデータオプションの設定

構造化されていないデータトランスフォーメーションは、PowerCenter にインストールされます。データトランスフォーメーションには固有のインストーラがあります。PowerCenter をインストールした後、データトランスフォーメーションサーバーおよびクライアントコンポーネントをインストールします。

構造化されていないデータオプションをインストールするには、以下の手順を実行します。

1. PowerCenter をインストールします。
2. データトランスフォーメーションをインストールします。

データトランスフォーメーションをインストールする場合、Java Runtime Environment を選択する必要があります。プロンプトが表示されたら、PowerCenter が使用する JRE を参照します。

デフォルトでは、セットアップはデータトランスフォーメーションエンジンをプロセス内で実行するように設定します。データトランスフォーメーションを構造化されていないデータのトランスフォーメーションと共に使用する場合は、データトランスフォーメーションをプロセス外で実行しないように設定する必要があります。

3. データトランスフォーメーションリポジトリフォルダを設定します。

Data Transformation リポジトリディレクトリの設定

Data Transformation リポジトリには、実行可能な Data Transformation サービスが含まれています。Data Transformation をインストールするとき、インストーラにより、次のフォルダーにリポジトリが作成されます。

```
<Data_Transformation_install_dir>\DataTransformation\ServiceDB
```

リポジトリフォルダーの場所を別の場所にするには、[スタート] メニューから Data Transformation を開きます。Data Transformation

[CM 設定] > [CM リポジトリ] > [ファイルシステム] > 基礎パス

Informatica Developer がリモートファイルシステムにアクセスできる場合は、Data Transformation リポジトリをリモートの場所に変更し、Informatica Developer から統合サービスを実行するリモートコンピュータにサービスを直接デプロイできます。

カスタムファイルを、<Data_Transformation_install_dir>\DataTransformation\autoInclude\user ディレクトリおよび<Data_Transformation_install_dir>\DataTransformation\externLibs\user ディレクトリから、Integration Service を実行するノード上の同じディレクトリにコピーします。

Data Transformation サービスのタイプ

Informatica Developer でデータプロセッサトランスフォーメーションを作成する場合は、オブジェクトまたはコンポーネントを選択してトランスフォーメーションサービスタイプを定義します。Data Transformation には、以下のタイプのデータ変換サービスがあります。

- パーサー。ソースドキュメントを XML または JSON 形式に変換します。パーサーの出力は常に XML になります。入力には、テキスト、HTML、Word、PDF、HL7 など任意の形式を使用できます。
- シリアライザ。XML または JSON ファイルを任意の形式の出力ドキュメントに変換します。シリアライザの出力は、テキストドキュメント、HTML ドキュメント、PDF など、任意の形式にすることができます。
- マッパーと XMap。XML または JSON ソースドキュメントを別の XML または JSON 構造体に変換します。マッパーは、シリアライザと同じように入力を処理します。また、パーサーと同じように出力を生成します。入力および出力は、完全に構造化された XML または JSON 形式です。
- トランスフォーマ。任意の形式のデータを変更します。テキストを追加、削除、変換、または変更します。パーサ、マッパ、またはシリアライザと共にトランスフォーマを使用します。また、トランスフォーマをスタンドアロンコンポーネントとして実行することもできます。
- ストリーマ。マルチギガバイトのデータストリームなどサイズの大きな入力ドキュメントをセグメントに分割します。Streamer は、HIPAA ファイルや EDI ファイルなど、複数のメッセージやレコードを含むドキュメントを処理します。

構造化されていないデータのトランスフォーメーションのコンポーネント

構造化されていないデータのトランスフォーメーションには、以下のタブがあります。

- トランスフォーメーション。トランスフォーメーションの名前および説明を入力します。構造化されていないデータトランスフォーメーションの命名規則は UD_トランスフォーメーション名です。また、構造化されていないデータトランスフォーメーションを再利用可能にすることもできます。
- プロパティ。[パーティション化可能] や [出力が再現可能] など、構造化されていないデータのトランスフォーメーションの一般的なプロパティを設定します。
- UDT 設定。入力タイプ、出力タイプ、サービス名など、構造化されていないデータのトランスフォーメーション設定を変更します。
- UDT ポート。構造化されていないデータのトランスフォーメーションのポートおよび属性を設定します。
- 入力階層構造化されていないデータのトランスフォーメーションがデータ行を読み込めるように、入力グループおよびポートの階層を定義します。
- 出力階層。構造化されていないデータのトランスフォーメーションが行をリレーショナルターゲットに書き込めるように、出力グループおよび出力ポートの階層を定義します。

[プロパティ] タブ

[プロパティ] タブでは、構造化されていないデータのトランスフォーメーションの全般的なプロパティを設定します。

以下の表では、**[プロパティ]** タブで設定可能なプロパティについて説明します。

プロパティ	説明
トレースレベル	このトランスフォーメーションを含むセッションを実行したときにセッションログに記録される情報の詳細度。デフォルトは [Normal] です。
IsPartitionable	トランスフォーメーションは複数のパーティションで実行できます。次のいずれかのオプションを選択します。 <ul style="list-style-type: none">- いいえ。トランスフォーメーションはパーティション化できません。- ローカルで。トランスフォーメーションをパーティション化することはできますが、同じノード上のパイプラインですべてのパーティションが実行される必要があります。- グリッドをまたがる。トランスフォーメーションをパーティション化することができ、各パーティションは異なるノードに配分されます。 デフォルトは [グリッドをまたがる] です。

プロパティ	説明
Output is Repeatable	<p>出力データの順序はセッションの実行ごとに一致しています。</p> <ul style="list-style-type: none"> - Never。出力データの順序はセッションの実行ごとに異なります。 - Based On Input Order。入力データの順序がセッションの実行ごとに一致している場合、出力順序をセッションの実行ごとに一致させます。 - Always。入力データの順序がセッションの実行ごとに異なる場合でも、出力データの順序は常に同じです。 <p>アクティブなトランスフォーメーションの場合、デフォルトは [Never] です。パッシブなトランスフォーメーション実行の場合、デフォルトは [Based On Input Order(入力順による)] です。</p>
Output is Deterministic	<p>トランスフォーメーションが、セッションの実行ごとに一貫した出力データを生成するかどうかを指定します。このトランスフォーメーションを使用するセッションでリカバリを実行するには、このプロパティを有効にします。</p>

警告: トランスフォーメーションを繰り返し可能で一意に定まるものとして設定する場合は、データが繰り返し可能で一意に定まることを保証する必要があります。セッションとリカバリで同じデータが生成されないトランスフォーメーションを使用してセッションをリカバリしようとする、リカバリプロセスを実行した結果、データが破損する可能性があります。

[UDT 設定] タブ

[UDT 設定] タブでは、構造化されていないデータのトランスフォーメーションの属性を設定します。

以下の表では、**[UDT 設定]** タブに用意されている属性について説明します。

属性	説明
入力タイプ	<p>構造化されていないデータのトランスフォーメーションが Data Transformation エンジンに渡す入力データのタイプ。以下の入力タイプのいずれかを選択します。</p> <ul style="list-style-type: none"> - バッファ。構造化されていないデータのトランスフォーメーションは、InputBuffer ポートでソースデータを受け取り、そのデータをポートから Data Transformation エンジンに渡します。 - ファイル。構造化されていないデータのトランスフォーメーションは、InputBuffer ポートでソースファイルパスを受け取り、そのソースファイルパスを Data Transformation エンジンに渡します。Data Transformation エンジンがソースファイルを開きます。
出力タイプ	<p>構造化されていないデータトランスフォーメーションまたは Data Transformation エンジンが返す出力データのタイプ。以下の出力タイプのいずれかを選択します。</p> <ul style="list-style-type: none"> - バッファ。出力ポートのリレーショナル階層を設定していない限り、構造されていないデータトランスフォーメーションは OutputBuffer ポートを介して XML データを返します。ポートのリレーショナル階層を設定している場合、構造されていないデータトランスフォーメーションは OutputBuffer ポートに書き込みません。 - ファイル。Data Transformation エンジンがファイルに出力を書き込みます。構造されていないデータトランスフォーメーションでポートのリレーショナル階層を設定していない限り、データトランスフォーメーションエンジンは構造されていないデータトランスフォーメーションにデータを返しません。 - スプリット。構造されていないデータトランスフォーメーションは、OutputBuffer ポートに収まるように大きな XML 出力ファイルを小さいファイルに分割します。分割した XML ファイルは、XML パーサトランスフォーメーションに渡す必要があります。
サービス名	<p>実行する Data Transformation サービスの名前。サービスは、ローカルの Data Transformation リポジトリに存在する必要があります。</p>

属性	説明
ストリーマチャンクサイズ	Data Transformation サービスがストリーマを実行するときに、構造化されていないデータのトランスフォーメーションが Data Transformation エンジンに渡すデータのバッファサイズ。有効な値は 1~1,000,000KB です。デフォルトは、256KB です。
動的サービス名	<p>入力行ごとに異なる Data Transformation サービスを実行します。動的サービス名を有効にすると、構造化されていないデータトランスフォーメーションはサービス名入力ポートでサービス名を受け取ります。</p> <p>動的サービス名を無効にすると、構造化されていないデータトランスフォーメーションはどの入力行でも同じサービスを実行します。[UDT Settings] のサービス名属性には、サービス名が含まれている必要があります。デフォルトでは無効になっています。</p>
ステータストレースレベル	<p>Data Transformation サービスからのステータスメッセージのレベルを設定します。</p> <ul style="list-style-type: none"> - 説明のみ。Data Transformation サービスの成否を示すステータスコードおよび簡単な説明を返します。 - フルステータス。Data Transformation サービスからステータスコードおよびステータスメッセージを XML で返します。 - なし。Data Transformation サービスからステータスを返しません。デフォルトは [なし] です。
入力フラッシュを許可	<p>ポートの入力グループを設定する場合は、[入力フラッシュを許可] を有効にします。</p> <p>構造化されていないデータのトランスフォーメーションでは、グループのすべてのデータが含まれている場合に XML が作成されます。グループ別にソートされたデータをトランスフォーメーションが受信するようにマッピングを設定します。</p> <p>入力フラッシュが有効になっていない場合、構造化されていないデータのトランスフォーメーションでは、データがメモリに保存され、すべてのグループのデータを受信した後で XML が作成されます。</p>

ステータストレースメッセージの表示

Data Transformation サービスからステータスメッセージを表示できます。ステータストレースレベルを [説明のみ] または [フルステータス] に設定します。Designer は、Unstructured Data トランスフォーメーションに UDT_Status_Code ポートおよび UDT_Status_Message 出力ポートを作成します。

Data Transformation を選択すると、データトランスフォーメーションエンジンはステータスコードと以下のステータスメッセージのいずれかを返します。

Status Code	ステータスメッセージ
1	成功
2	Warning
3	失敗
4	Error
5	致命的なエラー

[フルステータス] を選択すると、Data Transformation エンジン は Data Transformation サービスからのステータスコードおよびエラーメッセージを返します。メッセージは XML フォーマットです。

構造化されていないデータのトランスフォーメーションポート

Unstructured Data トランスフォーメーションを作成すると、Designer がデフォルトポートを作成します。Unstructured Data トランスフォーメーションの設定方法によっては、それ以外のポートも作成します。構造化されていないデータのトランスフォーメーションの入力タイプおよび出力タイプによって、構造化されていないデータのトランスフォーメーションが Data Transformation エンジンとどのようにデータを受け渡しするかが決まります。

以下の表に、構造化されていないデータのトランスフォーメーションのデフォルトのポートを示します。

ポート	入出力	説明
InputBuffer	入力	入力タイプがバッファである場合、ソースデータを受け取ります。 入力タイプがファイルである場合、ソースファイル名およびソースファイルパスを受け取ります。
OutputBuffer	出力	出力タイプがバッファである場合、XML データを返します。 出力タイプがファイルである場合、出力ファイル名を返します。 ポートの階層的な出力グループを設定している場合、データを返しません。

以下の表では、構造化されていないデータのトランスフォーメーションを設定している場合に Designer が作成する構造化されていないデータのトランスフォーメーションのデフォルト以外のポートについて説明します。

ポート	入出力	説明
OutputFileName	入力	出力タイプがファイルである場合、出力ファイルの名前を受け取ります。
サービス名	入力	動的サービス名を有効にしている場合、Data Transformation サービスの名前を受け取ります。
UDT_Status_Code	出力	ステータストレースレベルが「説明のみ」または「フルステータス」である場合、Data Transformation エンジンからのステータスコードを返します。
UDT_Status_Message	出力	ステータストレースレベルが「説明のみ」または「フルステータス」である場合、Data Transformation エンジンからのステータスメッセージを返します。

注: 【入力階層】 タブまたは 【出力階層】 タブでは、リレーショナルターゲットの出力ポートのグループを追加できます。ポートのグループを設定すると、【UDT ポート】 タブには階層的なグループとポートが別のタブに定義されているというメッセージが表示されます。

入力タイプと出力タイプ

入力タイプによって、Integration Service が Data Transformation エンジンに渡すデータのタイプが決まります。入力タイプによって、入力がデータであるかソースファイルパスであるかが決まります。

以下の入力タイプのいずれかを設定します。

- バッファ。構造化されていないデータのトランスフォーメーションは、InputBuffer ポートでソースデータを受け取ります。Integration Service は、InputBuffer ポートから Data Transformation エンジンにソース行を渡します。
- ファイル。構造化されていないデータのトランスフォーメーションは、InputBuffer ポートでソースファイルパスを受け取ります。Integration Service は、ソースファイルパスを Data Transformation エンジンに渡します。Data Transformation エンジンがソースファイルを開きます。ファイル入力タイプは、Microsoft Excel ファイルまたは Microsoft Word ファイルなどのバイナリファイルを解析するのに使用します。

出力グループおよび出力ポートを定義しないと、構造化されていないデータトランスフォーメーションは出力タイプに基づいてデータを返します。

以下の出力タイプのいずれかを設定します。

- バッファ。構造化されていないデータのトランスフォーメーションは、Outputbuffer ポートを介して XML を返します。XML パーサトランスフォーメーションを Outputbuffer ポートに接続する必要があります。
- ファイル。Data Transformation エンジンには、Integration Service にデータを渡すのではなく、出力ファイルにデータを書き込みます。Data Transformation エンジンには、OutputFilename ポートのファイル名を基に、出力ファイルに名前を付けます。XML データを PDF ファイルや Microsoft Excel ファイルなどのバイナリデータに変換するには、出力タイプとしてファイルを選択します。

Integration Service は、ソース行ごとに OutputBuffer ポートに出力ファイル名を返します。出力ファイル名が空白の場合、Integration Service は行エラーを返します。エラーが発生すると、Integration Service は OutputBuffer に NULL 値を書き込み、行エラーを返します。

- スプリット。構造化されていないデータのトランスフォーメーションは、Data Transformation エンジンからの XML データを複数のセグメントに分割します。構造化されていないデータトランスフォーメーションから OutputBuffer ポートに返される XML ファイルが大きすぎる場合には、出力タイプとしてスプリットを選択します。出力タイプとしてスプリットを設定しているときには、XML データを XML パーサトランスフォーメーションに渡します。このとき、複数の XML 行を 1 つの XML ファイルに処理するよう、XML パーサトランスフォーメーションを設定します。

追加の構造化されていないデータのトランスフォーメーションポート

Data Transformation では、複数の入力ファイル、ファイル名、およびパラメータが必要になることがあります。その場合、複数の出力ファイルを返すことができます。構造化されていないデータトランスフォーメーションを作成すると、Designer によって InputBuffer ポートが 1 つと、OutputBuffer ポートが 1 つ作成されます。構造化されていないデータのトランスフォーメーションと Data Transformation エンジンの間で追加のファイルまたはファイル名を渡す必要がある場合は、入力ポートまたは出力ポートを追加します。ポートは手動で追加することも、Data Transformation サービスから追加することもできます。

以下の表では、**[UDT ポート]** タブで作成できるポートについて説明します。

ポートタイプ	入力/出力	説明
追加の入力（バッファ）	Input	Data Transformation エンジンに渡す入力データを受け取ります。
追加の入力（ファイル）	Input	開く Data Transformation エンジンのファイル名とパスを受け取ります。
サービスパラメータ	Input	Data Transformation サービスの入力パラメータを受け取ります。

ポートタイプ	入力/出力	説明
追加の出力（バッファ）	Output	Data Transformation エンジンから XML データを受け取ります。
追加の出力（ファイル）	Output	Data Transformation エンジンから出力ファイル名を受け取ります。
パススルー	入力/ アウトプット	構造化されていないデータのトランスフォーメーションを介してデータをそのままの状態で渡します。

Data Transformation サービスからのポートの作成

Data Transformation サービスでは、入力パラメータ、追加の入力ファイル、またはユーザー定義変数が必要になることがあります。その場合、Unstructured Data トランスフォーメーションに複数の出力ファイルを返すことができます。パラメータ、追加の入力ファイル、および追加の出力ファイルを渡すポートを追加できます。Designer は、Data Transformation サービスのポートに対応するポートを作成します。

注: サービスからポートを読み込むには、サービス名を設定する必要があります。

1. 構造化されていないデータのトランスフォーメーションで **【ポート】** タブをクリックします。
2. **【サービスから読み込む】** をクリックします。

Designer には、Data Transformation サービスからのサービスパラメータ、追加の入力ポート要件、および追加の出力ポート要件が表示されます。サービスパラメータには、Data Transformation のシステム変数およびユーザー定義変数が含まれます。

3. 作成するポートを選択し、各ポートをバッファポートまたはファイルポートとして設定します。
4. **【読み込み】** をクリックして、選択したポートを作成します。表示されているポートをすべて選択することができます。

構造化されていないデータのトランスフォーメーションサービス名

構造化されていないデータのトランスフォーメーションを作成するとき、Designer には Data Transformation リポジトリにある Data Transformation サービスのリストが表示されます。構造化されていないデータのトランスフォーメーションから呼び出す Data Transformation サービスの名前を選択します。トランスフォーメーションを作成した後、サービス名を変更できます。サービス名は、**[UDT 設定]** タブに表示されます。

ソース行ごとに異なる Data Transformation サービスを実行するには、動的サービス名属性を有効にします。各ソース行と共にサービス名を渡します。動的サービス名を有効にすると、Designer が ServiceName 入力ポートを作成します。

動的サービス名を有効にした場合、Data Transformation サービスからポートを作成することはできません。

入力ポートのリレーショナル構造を定義する場合、動的サービス名を有効にすることはできません。入力行ごとに異なる Data Transformation サービスが必要です。

リレーショナル階層

ポートのグループを定義し、グループのリレーショナル構造を定義できます。入力ポートの階層を作成するには、**【入力階層】** タブでポートを設定します。行データをリレーショナルテーブルなどのターゲットに渡すには、**【出力階層】** タブで出力ポートを設定します。

入力ポートのリレーショナル構造を定義する場合、構造化されていないデータのトランスフォーメーションでは、Data Transformation サービスに渡す XML が生成されます。パフォーマンスを向上させるには、PowerCenter Integration Service が構造化されていないデータのトランスフォーメーションに対して XML をフラッシュできるようにします。入力フラッシュを有効にすると、PowerCenter Integration Service はルート値のすべてのデータの受信後に各グループから XML をフラッシュします。例えば、1 つの従業員グループと 1 つの従業員住所グループがあるとします。PowerCenter Integration Service は、異なる従業員がデータに追加されるたびに、両方のグループから構造化されていないデータのトランスフォーメーションに対してデータをフラッシュできます。各グループのデータは、ルートグループのプライマリキーでソートする必要があります。同じキーを持たないグループがある場合は、パイプライン内のグループを結合できます。

出力グループを設定すると、その出力グループがリレーショナルテーブル、つまり出力データを渡すターゲットとなります。Data Transformation エンジン、OutputBuffer ポートに XML ファイルを書き込むのではなく、グループポートに行を返します。トランスフォーメーションは、出力タイプに基づいて行を書き込みます。

【出力階層】 タブの左側のペインで、グループの階層を作成します。どのグループもルートグループの下にあります。ルートは削除できません。各グループには、ポートおよび他のグループを含めることができます。グループ構造は、ターゲットテーブル間のリレーションを表します。グループ内にグループを定義すると、そのグループ間に親子リレーションが定義されます。Designer は、生成キーでグループ間にプライマリキーと外部キーのリレーションを定義します。

グループを選択して、グループのポートを表示します。グループのポートを追加または削除できます。ポートを追加すると、Designer がデフォルトポート設定を作成します。ポート名、データ型、および精度を変更します。ポートにデータを含める必要がある場合は、**【非 NULL】** を選択します。それ以外の場合、出力データはオプションです。

構造化されていないデータトランスフォーメーションをワークスペースに表示すると、トランスフォーメーショングループの各ポートにはグループ名からなる接頭辞が付きます。

グループを削除する場合は、グループ内のポートと子グループも削除します。

階層スキーマのエクスポート

構造化されていないデータのトランスフォーメーションに階層的な出力グループを定義するときは、その同じ構造をデータ変換用に作成する Data Transformation サービスに定義する必要があります。構造化されていないデータトランスフォーメーションから階層構造を XML スキーマファイルとしてエクスポートします。スキーマをデータプロセッサトランスフォーメーションにインポートします。インポート後、データプロセッサトランスフォーメーションで、ソースドキュメントの内容を XML 要素および属性にマッピングできます。

【リレーショナル階層】 タブからグループ階層をエクスポートするには、**【XML スキーマにエクスポート】** をクリックします。XSD ファイルの名前および場所を選択します。Informatica Developer を使用してスキーマをインポートするときにアクセスできる場所を選択します。

Designer は、以下の名前空間を持つスキーマを作成します。

```
"www.informatica.com/UDT/XSD/<mappingName_<Transformation_Name>"
```

スキーマには以下のコメントが含まれています。

```
<!-- ===== AUTO-GENERATED FILE - DO NOT EDIT ===== -->
<!-- ===== This file has been generated by Informatica PowerCenter ===== -->
```

スキーマを変更した場合、Data Transformation エンジンが出力ポートとは異なる形式のデータを構造化されていないデータのトランスフォーメーションに返す可能性があります。

スキーマの XML 要素は、階層内の出力ポートを表しています。NULL 値を含めることができるカラムには、minOccurs=0 および maxOccurs=1 となる XML 属性があります。

マッピング

マッピングを作成する場合、実行する Data Transformation サービスのタイプに応じてマッピングを設計します。例えば、Data Transformation のパーサおよびマッパーは XML データを生成します。構造されていないデータトランスフォーメーションは、XML データから行を返すように設定したり、XML ファイルを返すように設定したりできます。

Data Transformation シリアルライザコンポーネントは、XML からどの出力でも生成できます。HTML ファイルの他、Microsoft Word や Microsoft Excel などのバイナリファイルも生成できます。出力がバイナリデータの場合、Data Transformation エンジンでは出力を構造化されていないデータのトランスフォーメーションに渡すのではなく、ファイルに書き込むことができます。

以降の例では、構造されていないデータトランスフォーメーションでマッピングを設定する方法を示します。

リレーショナルテーブル用の Word ドキュメントの解析

Microsoft Word ドキュメントから注文情報を抽出して、注文ヘッダテーブルおよび注文詳細テーブルに書き込むことができます。データトランスフォーメーションパーサーサービス呼び出すように構造されていないデータトランスフォーメーションを設定し、解析対象の各 Word ドキュメントの名前を渡します。Data Transformation エンジンでは、Word ドキュメントを開いて解析し、文書の行を構造化されていないデータのトランスフォーメーションに返します。構造されていないデータトランスフォーメーションは、注文のヘッダと詳細をリレーショナルターゲットに渡します。

マッピングには次のオブジェクトがあります。

- ソース修飾子トランスフォーメーション。各 Microsoft Word ファイル名を Unstructured Data トランスフォーメーションに渡します。ソースファイル名には、注文情報を含むファイルへの完全パスが含まれます。
- 構造化されていないデータのトランスフォーメーション。入力タイプはファイルです。出力タイプはバッファです。トランスフォーメーションには、注文ヘッダ出力グループおよび注文詳細出力グループが含まれています。グループには、プライマリキーと外部キーのリレーションがあります。

構造されていないデータトランスフォーメーションは、InputBuffer ポートでソースファイル名を受け取ります。次に、受け取ったファイル名を Data Transformation エンジンに渡します。Data Transformation エンジンでは、Word ドキュメントから注文ヘッダおよび注文詳細行を抽出するパーサーサービスを実行します。Data Transformation エンジンでは、抽出したデータを構造化されていないデータのトランスフォーメーションに返します。構造されていないデータトランスフォーメーションは、注文ヘッダグループおよび注文詳細グループからリレーショナルターゲットにデータを渡します。

- リレーショナルターゲット。構造化されていないデータのトランスフォーメーションから行を受け取ります。

XML からの Excel シートの作成

XML ファイルから従業員の名前と住所を抽出し、従業員名一覧の Microsoft Excel シートを作成できます。

マッピングには、以下のようなコンポーネントがあります。

- XML ソースファイル。従業員の名前と住所が含まれています。
- ソース修飾子トランスフォーメーション。XML データおよび出力ファイル名を構造化されていないデータトランスフォーメーションに渡します。XML ファイルには、従業員名が含まれています。

- 構造化されていないデータのトランスフォーメーション。入力タイプはバッファで、出力タイプはファイルです。構造化されていないデータトランスフォーメーションは InputBuffer ポートで XML ファイルを受け取り、OutputFileName ポートでファイル名を受け取ります。次に、XML データとファイル名を Data Transformation エンジンに渡します。

Data Transformation エンジン、シリアルライザサービスを実行して XML データを Microsoft Excel ファイルに変換します。このとき、OutputFilename の値に基づく名前の Excel ファイルにデータが書き込まれます。

構造化されていないデータのトランスフォーメーションは、Data Transformation エンジンから出力ファイル名のみを受け取ります。構造化されていないデータトランスフォーメーションの OutputBuffer ポートは、OutputFilename の値を返します。

- フラットファイルターゲット。出力ファイル名を受け取ります。

分割 XML ファイルの出力

Data Transformation のコンポーネントであるパーサおよびマップを使用すると、どの形式のデータでも変換して、XML データを生成できます。XML データが大きい場合には、XML をセグメントに分割し、各セグメントを XML パーサトランスフォーメーションに渡すことができます。XML パーサトランスフォーメーションは、セグメントを受け取り、XML データを 1 つのドキュメントとして処理します。

XML 出力を分割するように構造化されていないデータトランスフォーメーションを設定すると、OutputBuffer ポートのサイズに基づいて XML データが返されます。XML ファイルのサイズが出力ポートの精度を上回る場合は、Integration Service によってポートサイズ以下のファイルに分割されます。XML パーサトランスフォーメーションは、XML を解析し、行をリレーショナルテーブルなどのターゲットに渡します。

例えば、Data Transformation パーササービスを使用して、Microsoft Word ドキュメントから注文ヘッダと詳細の情報を抽出できます。

マッピングには、以下のようなコンポーネントがあります。

- ソース修飾子トランスフォーメーション。Word 文書のファイル名を、構造化されていないデータのトランスフォーメーションに渡します。ソースファイル名には、注文情報を含むファイルへの完全パスが含まれます。
- 構造化されていないデータのトランスフォーメーション。入力タイプはファイルです。出力タイプはスプリットです。構造化されていないデータトランスフォーメーションは、InputBuffer ポートでソースファイル名を受け取ります。次に、受け取ったファイル名を Data Transformation エンジンに渡します。Data Transformation エンジンがソースファイルを開いて解析し、構造化されていないデータのトランスフォーメーションに XML データを返します。

構造化されていないデータトランスフォーメーションは、XML データを受け取って小さいファイルに分割し、各セグメントを XML パーサトランスフォーメーションに渡します。次に、OutputBuffer ポートサイズより小さいセグメントでデータを返します。XML データを複数のセグメントに分けて返す場合には、行ごとに同じパススルーデータを生成します。また、行が成功しても失敗しても、データをパススルーポートに返します。

- XML パーサトランスフォーメーション。入力ストリーミング有効化セッションプロパティは有効になっています。XML パーサトランスフォーメーションは、DataInput ポートで XML データを受け取ります。入力データはセグメントに分かれています。XML パーサトランスフォーメーションは、XML データを解析して注文ヘッダ行と詳細行を生成します。次に、注文ヘッダ行と詳細行をリレーショナルターゲットに渡します。続いて、フィルタトランスフォーメーションにパススルーデータを返します。
- フィルタトランスフォーメーション。パススルーデータをリレーショナルターゲットに渡す前に、重複するデータを削除します。
- リレーショナルターゲット。XML パーサトランスフォーメーションおよびフィルタトランスフォーメーションの各グループからデータを受け取ります。

非構造化データマッピングのルールおよびガイドライン

非構造化データマッピングを作成するときは、以下の規則とガイドラインに従ってください。

- 出力ポートの階層グループを設定すると、Integration Service は OutputBuffer ポートに書き込むのではなく、ポートグループに書き込みます。トランスフォーメーションに定義している出力タイプに関係なく、ポートグループに書き込みます。
- 構造化されていないデータのトランスフォーメーションの出力タイプがファイルで、まだグループ出力ポートが定義されていない場合は、OutputBuffer ポートをダウンストリームトランスフォーメーションにリンクさせる必要があります。これを行わない場合は、マッピングが無効になります。Data Transformation サービスが出力ファイルを書き込むとき、OutputBuffer ポートには出力ファイル名が含まれています。
- サービス名入力ポートで構造化されていないデータトランスフォーメーションにサービス名を渡すには、動的サービス名を有効にします。動的サービス名を有効にすると、Designer がサービス名入力ポートを作成します。
- 構造化されていないデータトランスフォーメーションでサービス名を設定するか、または動的サービス名オプションを有効にする必要があります。これを行わない場合は、マッピングが無効になります。
- 構造化されていないデータトランスフォーメーションから返された XML 出力を XML パーサトランスフォーメーションにリンクします。

構造化されていないデータのトランスフォーメーションの作成

構造化されていないデータのトランスフォーメーションを作成するには、PowerCenter Transformation Developer または Mapping Designer を使用します。

- Mapping Designer または Transformation Developer で、**【トランスフォーメーション】 > 【作成】** の順にクリックします。
- トランスフォーメーションのタイプとして **【構造化されていないデータのトランスフォーメーションを選択します】** を選択します。
- トランスフォーメーションの名前を入力します。
- 【作成】** をクリックします。

【構造化されていないデータのトランスフォーメーション】 ダイアログボックスが表示されます。

- 以下のプロパティを設定します。

プロパティ	説明
サービス名	使用する Data Transformation サービスの名前。Designer には、Data Transformation リポジトリフォルダにある Data Transformation サービスが表示されます。後で動的サービス名を有効にする場合には、名前を選択しないでください。トランスフォーメーションを作成した後、[UDT Settings] タブでサービス名を追加できます。
入力タイプ	Data Transformation エンジンが入力データを受け取る方法について説明したものです。デフォルト値は Buffer です。
Output Type	Data Transformation が出力データを返す方法について説明したものです。デフォルト値は Buffer です。

6. **[OK]** をクリックします。
7. **[UDT 設定]** タブでは、サービス名、入力タイプ、および出力タイプを変更できます。
8. **[プロパティ]** タブでは、構造化されていないデータのトランスフォーメーションのプロパティを設定します。
9. Data Transformation サービスに複数の入力ファイルまたは出力ファイルがある場合、またはデータトランスフォーメーションで入力パラメータが必要になる場合は、**[UDT ポート]** タブでポートを追加できます。このタブで、パススルーポートを追加することもできます。
10. 構造化されていないデータのトランスフォーメーションから XML データではなく行データを返すようにする場合は、**[リレーショナル階層]** タブで出力ポートグループを作成します。
11. ポートグループを作成した場合は、**[リレーショナル階層]** タブから各グループについて記述したスキーマをエクスポートします。
12. そのスキーマを Data Transformation プロジェクトにインポートして、プロジェクト出力を定義します。

第 32 章

アップデイトストラテジトランスフォーメーション

この章では、以下の項目について説明します。

- [アップデイトストラテジトランスフォーメーションの概要, 510 ページ](#)
- [マッピング内の行のフラグ設定, 511 ページ](#)
- [セッションのアップデイトストラテジの設定, 513 ページ](#)
- [アップデイトストラテジチェックリスト, 515 ページ](#)

アップデイトストラテジトランスフォーメーションの概要

アップデイトストラテジトランスフォーメーションはアクティブなトランスフォーメーションです。データウェアハウスを設計する場合、ターゲットに格納する情報のタイプを決定する必要があります。ターゲットテーブルの設計の一環として、履歴データをすべて保持するのか、最新の変更だけ保持するのかを決定する必要があります。

たとえば、ターゲットテーブル T_CUSTOMERS に顧客データが格納されているとします。ある顧客の住所に変更があった場合、顧客行の該当部分を更新する代わりに、テーブルに変更前の住所も保存しておきたいことがあります。このような場合には、新しい住所の入った新規行を作成し、顧客の古い住所の入った元の行はそのまま保存しておくことになります。この方法で、履歴情報をターゲットテーブルに保持できます。しかし、T_CUSTOMERS テーブルを現在の顧客データのスナップショットとしたい場合は、既存の顧客行を更新し、それによって古い住所は失われることになります。

選択するモデルによって、既存の行の変更を処理する方法が決まります。PowerCenter では、アップデイトストラテジを次の 2 つのレベルで設定します。

- **セッション内。** セッションを設定する場合、すべての行を同じ方法で処理する（例えば、すべての行を挿入行として処理する）か、セッションのマッピング内にコーディングされた指示を適用してデータベース操作ごとに行にフラグを設定するかを、Integration Service で指定できます。
- **マッピング内。** マッピング内では、アップデイトストラテジトランスフォーメーションを使って、挿入、削除、更新、あるいは拒否のフラグを行に設定します。

注: カスタムトランスフォーメーションを使って、挿入、削除、更新、または拒否のフラグを設定することもできます。

アップデートストラテジの設定

更新方式を定義するには、以下の手順を実行します。

1. マッピング内の行に対する挿入、更新、削除、または拒否のフラグ設定を制御する場合は、マッピングにアップデートストラテジトランスフォーメーションを追加します。同じターゲットに対する行に異なるデータベース操作のフラグを設定したい場合、あるいは行を拒否できるようにしたい場合には、アップデートストラテジトランスフォーメーションは必須です。
2. セッションの設定を行うときに、行のフラグ設定の方法を指定します。すべての行に挿入、削除、または更新のフラグを設定するか、またはデータドリブンオプションを選択することができます。データドリブンオプションを選択した場合、Integration Service はセッションのマッピング内のアップデートストラテジトランスフォーメーションにコーディングされた指示に従います。
3. セッションの設定を行うときに、ターゲットごとに挿入、更新、および削除のオプションを定義します。ターゲットごとに挿入または削除を有効または無効に設定できます。また、更新の取り扱い方法を3つの方法から選択できます。

関連項目：

- [「セッションのアップデートストラテジの設定」](#) (ページ 513)

マッピング内の行のフラグ設定

Update Strategy を最大限にコントロールするために、アップデートストラテジトランスフォーメーションをマッピングに追加します。このトランスフォーメーションの最も重要な機能はアップデートストラテジ式で、個々の行に挿入、削除、更新、または拒否のフラグを設定するために使用されます。

以下の表に、各データベース操作を表す定数とその数値を示します。

操作	定数	数値
挿入	DD_INSERT	0
更新	DD_UPDATE	1
削除	DD_DELETE	2
Reject	DD_REJECT	3

Integration Service は、上記以外の値をすべて挿入として扱います。

拒否された行の転送

拒否された行を次のトランスフォーメーションに渡すか、または削除するように、アップデートストラテジトランスフォーメーションを設定することができます。デフォルトでは、Integration Service は拒否された行を次のトランスフォーメーションに転送します。Integration Service は、行に拒否のフラグを設定し、セッション拒否ファイルに書き込みます。[拒否された行の転送]を選択しなかった場合、Integration Service は拒否された行を削除してセッションログファイルに書き込みます。

行エラー処理を有効にした場合、Integration Service は拒否された行と削除された行を行エラーログファイルに書き込みます。拒否ファイルは作成しません。削除された行を、行エラーログだけでなくセッションログにも書き出したい場合は、Verbose Data トレースを有効にします。

アップデイトストラテジ式

アップデイトストラテジ式ではしばしばトランスフォーメーション言語の IIF または DECODE 関数を使って、各行が特定の条件に一致しているかどうかをテストします。条件に一致している場合は、各行に数値コードを割り当てることにより、特定のデータベース操作に対応するフラグを設定できます。たとえば次の IIF 文は、入力日付が適用日付よりあとの場合は行に拒否のフラグを設定します。そうでない場合は行に更新のフラグを設定します。

```
IIF( ( ENTRY_DATE > APPLY_DATE), DD_REJECT, DD_UPDATE )
```

アップデイトストラテジトランスフォーメーションを作成するには：

1. Mapping Designer でマッピングにアップデイトストラテジトランスフォーメーションを追加します。
2. [レイアウト] - [カラムのリンク] をクリックします。
3. アップデイトストラテジトランスフォーメーションを介して渡されるデータを表す他のトランスフォーメーションから、すべてのポートをドラッグします。

アップデイトストラテジトランスフォーメーションでは、Designer はドラッグした各ポートのコピーを作成します。また Designer は、新しいポートを元のポートに接続します。アップデイトストラテジトランスフォーメーションの各ポートは、入出力ポートです。

通常、特定のターゲットに対して使用するすべてのカラムを選択します。これらのカラムがアップデイトストラテジトランスフォーメーションを通過すると、この情報には更新、挿入、削除、または拒否のフラグが設定されます。
4. アップデイトストラテジトランスフォーメーションを開き、名前を変更します。

アップデイトストラテジトランスフォーメーションの命名規則は、「UPD_トランスフォーメーション名」です。
5. [プロパティ] タブをクリックします。
6. [アップデイトストラテジトランスフォーメーション] フィールド内のボタンをクリックします。

式エディタが表示されます。
7. 行に、挿入、削除、更新、または拒否のフラグを設定するアップデイトストラテジ式を入力します。
8. 式を検証し、[OK] をクリックします。
9. [OK] をクリックします。
10. アップデイトストラテジトランスフォーメーション内のポートを、別のトランスフォーメーションまたはターゲットインスタンスに接続します。

アグリゲータトランスフォーメーションとアップデイトストラテジトランスフォーメーション

同じパイプライン内でアグリゲータトランスフォーメーションとアップデイトストラテジトランスフォーメーションを連結する場合はアグリゲータを配置してからアップデイトストラテジトランスフォーメーションを配置します。この順番では、Integration Service は集計計算を実行し、計算結果を含む行に、挿入、更新、削除、または拒否のフラグを設定します。

アグリゲータトランスフォーメーションの前にアップデイトストラテジトランスフォーメーションを配置する場合は、アグリゲータトランスフォーメーションが、異なる演算子をフラグ付された行をどのように処理するかを考慮する必要があります。この順番では、Integration Service は挿入、更新、削除、または拒否のフラグを行に設定してから集計計算を実行します。行にどのようにフラグを設定するかによって、アグリゲータトランスフォーメーションが計算で使用する行の値をどのように扱うかが決まります。例えば、行に削除のフラグを設定した後で、その行を使用して合計を計算すると、Integration Service はこの行の値を合計から除算します。行に拒否のフラグを設定した後で、その行を使用して合計を計算すると、Integration Service はこの行の値を合計には加えません。行に挿入または更新のフラグを設定した後で、その行を使用して合計を計算すると、Integration Service はこの行の値を合計に加えます。

ルックアップトランスフォーメーションとアップデートストラテジトランスフォーメーション

動的ルックアップキャッシュを使用するルックアップトランスフォーメーションを使ってマッピングを作成する場合、アップデートストラテジトランスフォーメーションを使用してターゲットテーブルの行にフラグを設定する必要があります。アップデートストラテジトランスフォーメーションと動的ルックアップキャッシュを使用してセッションを設定する場合、定義しなければならないセッションプロパティがあります。

[Treat Source Rows As] オプションを [Data Driven] に定義する必要があります。このオプションは、セッションプロパティの中の [プロパティ] タブで指定します。

また、下記のアップデートストラテジターゲットテーブルオプションも定義する必要があります。

- [Insert] を選択。
- [更新として更新] を選択。
- [Delete] を非選択。

これらのアップデートストラテジターゲットテーブルオプションにより、Integration Service は確実に「更新」とマークされた行を更新し、「挿入」とマークされた行を挿入します。

[データドリブン] を選択しなかった場合、Integration Service は、すべての行に対して [ソース行の扱い] オプションで指定したデータベース操作のフラグを設定します。行に対するフラグの設定には、マッピング内のアップデートストラテジトランスフォーメーションは使用しません。Integration Service は、正しい行を挿入および更新しません。[更新として更新] を選択しない場合、Integration Service はターゲットテーブルで更新行としてフラグを設定された行を正しく更新しません。その結果、ルックアップキャッシュとターゲットテーブルが同期しなくなる可能性があります。

セッションのアップデートストラテジの設定

セッションの設定を行う際は、更新などのデータベース操作の処理に関するオプションがいくつかあります。

すべての行に対する操作の指定

セッションを設定する際、[ソース行の扱い] 設定を使用してすべての行に対するデータベース操作を 1 つ選択できます。

以下の表に、[ソース行の扱い] 設定のオプションを示します。

設定	説明
挿入	すべての行を挿入として扱います。行の挿入がデータベースにおけるプライマリキーまたは外部キーの制約に違反する場合、Integration Service はその行を拒否します。
削除	すべての行を削除として扱います。行ごとにターゲットテーブル内の対応する行が（プライマリキー値に基づいて）見つかった場合、Integration Service はその行を削除します。リポトリ内のターゲット定義にプライマリキー制約が存在していなければならないことに注意してください。

設定	説明
更新	すべての行を更新として扱います。Integration Service は行ごとにターゲットテーブル内で一致するプライマリーキー値を検索します。一致するプライマリーキーを検出すると、Integration Service はその行を更新します。ターゲット定義にプライマリーキー制約が存在していなければなりません。
データドリブン	Integration Service はセッションのマッピングにおけるアップデイトストラテジトランスフォーメーションおよびカスタムトランスフォーメーションにコーディングされている指示に従って、挿入、削除、更新、または拒否のフラグを行に設定する方法を決定します。 セッションのマッピングにアップデイトストラテジトランスフォーメーションが含まれている場合、このフィールドはデフォルトで「データドリブン」に設定されます。 マッピングにアップデイトストラテジトランスフォーメーションまたはカスタムトランスフォーメーションが含まれる場合に「データドリブン」が選択されていないと、Workflow Manager は警告を表示します。セッションを実行すると、Integration Service ではマッピング内のアップデイトストラテジトランスフォーメーションまたはカスタムトランスフォーメーションの指示に従わずに、行フラグの設定方法が決定されます。

以下の表に、各設定についてのアップデイトストラテジを示します。

設定	用途
挿入	ターゲットテーブルに初めてデータを入れる場合、またはデータウェアハウスの履歴を保持する場合。後者の場合は、選択したターゲットテーブルのグループに対してだけでなく、データウェアハウス全体にこの方式を設定しなければなりません。
削除	ターゲットテーブルをクリアする場合。
更新	ターゲットテーブルを更新する場合。データウェアハウスに履歴データがスナップショットが含まれる場合、この設定を選択します。あとで、個々のターゲットテーブルの更新方法を設定するときに、更新された行を新規行として挿入するか、更新された情報を使ってターゲット内の既存の行を上書きするかを指定できます。
データドリブン	行に挿入、削除、更新、または拒否のフラグを設定する方法を詳細に制御したい場合。同じテーブルの行に、場合によって異なる操作（あるときは更新、あるときは拒否のように）のフラグを設定しなければならないときに、この設定を選択します。また、この設定は、行に拒否のフラグを設定できる唯一の方法を提供します。

個々のターゲットテーブルに対する操作の指定

セッション内のすべての行の扱い方が決まったら、個々のターゲットについても更新方式オプションを設定する必要があります。セッションプロパティの「マッピング」タブの「トランスフォーメーション」ビューで更新方式オプションを定義します。

次のアップデイトストラテジオプションを設定できます。

- **挿入。** このオプションを選択すると、ターゲットテーブルに行が挿入されます。
- **削除。** このオプションを選択すると、テーブルから行が削除されます。
- **更新。** これには下記のオプションがあります。
 - **更新として更新。** 行がターゲットテーブルに存在する場合、更新のフラグが設定されている各行を更新します。
 - **挿入として更新。** 更新のフラグが設定されている各行を挿入します。
 - **更新でなければ挿入。** 行が存在する場合、更新します。存在しない場合は挿入します。

- **テーブルを切り詰め。**このオプションを選択すると、データをロードする前にターゲットテーブルを切り詰めます。

アップデイトストラテジチェックリスト

更新方式の選択には、セッションで正しいオプションを設定する必要があり、場合によってアップデイトストラテジトランスフォーメーションをマッピングに追加する必要があります。この節には、各種の更新方式を実装するために必要な事項をまとめてあります。

- ターゲットテーブルに挿入だけを行う。
セッションの設定を行うときに、[Treat Source Rows As] セッションプロパティで [Insert] を選択します。また、セッション内のすべてのターゲットインスタンスに対して [Insert] オプションを選択する必要があります。
- ターゲットテーブルのすべての行を削除する。
セッションの設定を行うときに、[Treat Source Rows As] セッションプロパティで [Delete] を選択します。また、セッション内のすべてのターゲットインスタンスに対して [Delete] オプションを選択する必要があります。
- ターゲットテーブルの内容の更新だけを行う。
セッションの設定を行うときに、[Treat Source Rows As] セッションプロパティで [Update] を選択します。各ターゲットテーブルインスタンスの更新オプションを設定するときに、どのターゲットインスタンスに対しても [Update] オプションを選択する必要があります。
- 同じターゲットテーブルに対して、行によって異なるデータベース操作を行う。
マッピングにアップデイトストラテジトランスフォーメーションを追加します。トランスフォーメーションのアップデイトストラテジ式を記述するときに、DECODE または IIF 関数を使って異なる操作（挿入、削除、更新、または拒否）のフラグを行に設定します。このマッピングを使用するセッションの設定を行う場合は、[ソース行の扱い] セッションプロパティについて [データドリブン] を選択します。各ターゲットテーブルインスタンスに対して、[Insert]、[Delete]、およびいずれかの [Update] オプションを選択しておく必要があります。
- データを拒否する。
マッピングにアップデイトストラテジトランスフォーメーションを追加します。トランスフォーメーションのアップデイトストラテジ式に、DECODE または IIF を使って行を拒否する条件を指定します。このマッピングを使用するセッションの設定を行う場合は、[ソース行の扱い] セッションプロパティについて [データドリブン] を選択します。

第 33 章

XML トランスフォーメーション

この章では、以下の項目について説明します。

- [XML ソース修飾子トランスフォーメーション, 516 ページ](#)
- [XML パーサートランスフォーメーション, 516 ページ](#)
- [XML ジェネレータトランスフォーメーション, 517 ページ](#)

XML ソース修飾子トランスフォーメーション

XML ソース修飾子トランスフォーメーションをマッピングに追加するには、XML ソース定義を Mapping Designer ワークスペースにドラッグするか、手動で XML ソース定義を作成します。マッピングに XML ソース定義を追加する際、XML ソース修飾子トランスフォーメーションに XML ソース定義を接続する必要があります。XML ソース修飾子トランスフォーメーションは、Integration Service がセッション実行時に読み込むデータ要素を定義します。これによって、PowerCenter によるソースデータの読み込み方法が決まります。XML ソース修飾子トランスフォーメーションはアクティブなトランスフォーメーションです。

XML ソース修飾子トランスフォーメーションは、XML ソースの各カラムに対して必ず入力ポートまたは出力ポートを 1 つ持っています。ソース定義の XML ソース修飾子トランスフォーメーションを作成すると、Designer は XML ソース定義の各ポートを XML ソース修飾子トランスフォーメーションのポートにリンクします。リンクの削除や編集はできません。XML ソース定義をマッピングから削除すると、Designer は対応する XML ソース修飾子トランスフォーメーションも削除します。1 つの XML ソース定義を、1 つの XML ソース修飾子トランスフォーメーションにリンクすることができます。

1 つの XML ソース修飾子グループのポートを他のトランスフォーメーションのポートにリンクし、別々のデータフローを作成することができます。ただし、XML ソース修飾子トランスフォーメーションの複数のグループのポートを同じターゲットトランスフォーメーションのポートにリンクすることはできません。

プロパティのいくつかを編集し、メタデータエクステンションを XML ソース修飾子トランスフォーメーションに追加することができます。

XML パーサートランスフォーメーション

XML パーサートランスフォーメーションを使用して、パイプライン内部で XML を抽出します。XML パーサートランスフォーメーションを使用すると、TIBCO や MQ Series などのメッセージングシステム、あるいはファイルやデータベースなどの他のソースから XML データを抽出できます。XML パーサートランスフォーメーションの機能は、パイプライン内で XML を解析することを除いて、XML ソースの機能と同様です。例えば、

TIBCO ソースから XML データを抽出し、そのデータをリレーショナルターゲットに渡すことができます。XML パーサートランスフォーメーションはアクティブなトランスフォーメーションです。

XML パーサートランスフォーメーションは、1 つの入力ポートから XML データを読み込み、1 つ以上の出力ポートにデータを書き出します。

XML ジェネレータトランスフォーメーション

XML ジェネレータトランスフォーメーションを使用して、パイプライン内部で XML を作成します。XML ジェネレータトランスフォーメーションを使用すると、TIBCO や MQ Series などのメッセージングシステム、あるいはファイルやデータベースなどの他のソースからデータを読み込むことができます。XML ジェネレータトランスフォーメーションの機能は、パイプライン内で XML を生成することを除いて、XML ターゲットの機能と同様です。例えば、リレーショナルソースからデータを抽出し、その XML データをターゲットに渡すことができます。XML ジェネレータトランスフォーメーションはアクティブなトランスフォーメーションです。

XML ジェネレータトランスフォーメーションは、複数のポートからデータを受け取り、1 つの出力ポートに XML を書き出します。

索引

記号

DB2

IBM DB2 を参照 [438](#)

A

ABORT 関数

使用 [41](#)

address.dic

データのマスクング [155](#)

API メソッド

Java トランスフォーメーション [244](#)

API 関数

カスタムトランスフォーメーション [96](#)

ASCII

エクスターナルプロシージャトランスフォーメーション [165](#)

カスタムトランスフォーメーション [62](#)

ASCII モード

ジョイナトランスフォーメーション用ソート順の設定 [283](#)

AutoCommit

[SQL 設定] タブ [450](#)

B

BankSoft の例

Informatica エクスターナルプロシージャ [175](#)

概要 [166](#)

BigDecimal データタイプ

Java トランスフォーメーション [240](#)

C

C/C++

Integration Service へのリンク [187](#)

CLASSPATH

Java トランスフォーメーション、設定 [239](#)

COBOL

VSAM ノーマライザトランスフォーメーション [360](#)

COBOL ソース定義

OCCURS 文 [360](#)

ノーマライザトランスフォーメーションの作成 [363](#)

company_names.dic

データのマスクング [155](#)

COM エクスターナルプロシージャ

Informatica エクスターナルプロシージャとの比較 [166](#)

Visual Basic での開発 [174](#)

Visual C++での開発 [169](#), [172](#)

概要 [168](#)

行レベルのプロシージャ [186](#)

作成 [168](#)

作成にあたっての注意 [185](#)

サーバータイプ [168](#)

COM エクスターナルプロシージャ (続く)

初期化 [189](#)

接続されていない [189](#)

ソースの作成 [172](#)

ターゲットの作成 [172](#)

データタイプ [185](#)

デバッグ [187](#)

配布 [184](#)

メモリ管理 [187](#)

戻り値 [186](#)

リポジトリに追加 [172](#)

リポジトリによる登録 [172](#)

例外処理 [187](#)

COM サーバー

COM エクスターナルプロシージャのタイプ [168](#)

CURRVAL ポート

シーケンスジェネレータトランスフォーメーション [389](#)

D

Data Integration Service)

再起動モード [252](#)

defineJExpression

Java 式の API メソッド [264](#)

defineJExpression メソッド

Java トランスフォーメーション [246](#)

Detail Outer ジョイン

説明 [282](#)

DLL (ダイナミックリンクライブラリ)

エクスターナルプロシージャのコンパイル [181](#)

double データタイプ

Java トランスフォーメーション [240](#)

E

EDataType クラス

Java 式 [263](#)

ERROR 関数

使用 [41](#)

F

failSession メソッド

Java トランスフォーメーション [247](#)

firstnames.dic

データのマスクング [155](#)

フラットファイル

データの結合 [277](#)

フラットファイル

ルックアップ [294](#)

Full Outer ジョイン

定義 [282](#)

G

generateRow メソッド
 Java トランスフォーメーション [247](#)
getBytes メソッド
 Java トランスフォーメーション [266](#)
getDouble メソッド
 Java トランスフォーメーション [267](#)
getInRowType メソッド
 Java トランスフォーメーション [248](#)
getInt メソッド
 Java トランスフォーメーション [267](#)
getLong メソッド
 Java トランスフォーメーション [267](#)
getMetada メソッド
 Java トランスフォーメーション [249](#)
getResultDataType メソッド
 Java トランスフォーメーション [267](#)
getResultMetadata メソッド
 Java トランスフォーメーション [267](#)
getStringBuffer メソッド
 Java トランスフォーメーション [268](#)
Group By ポート
 アグリゲータトランスフォーメーション [55](#)
 デフォルト値の使用 [57](#)
 非集計式 [57](#)

H

HTTP トランスフォーメーション
 HTTP タブの設定 [206](#)
 応答コード [203](#)
 グループ [207](#)
 グループおよびポートの設定 [207](#)
 作成 [204](#)
 スレッド特有のコード [205](#)
 認証 [204](#)
 パーティション化可能（プロパティ） [205](#)
 [パーティションごとに 1 つのスレッドを要求します] プロパティ [205](#)
 プロパティの設定 [206](#)
 ベース URL [211](#)
 例 [213](#)

I

IBM DB2
 接続文字列の構文 [438](#)
IDispatch インタフェース
 クラスの定義 [168](#)
IDM_Dictionary
 データマスキング接続 [137](#)
IDM_Storage
 データマスキング接続 [137](#)
IIF 関数
 シーケンスジェネレータトランスフォーメーションを使用した欠落
 キーの置き換え [388](#)
incrementErrorCount メソッド
 Java トランスフォーメーション [249](#)
Informatica エクスターナルプロシージャ
 COM との比較 [166](#)
 C++コードの生成 [178](#)
 開発 [175](#)
 行レベルのプロシージャ [186](#)
 作成にあたっての注意 [185](#)
 初期化 [189](#)

Informatica エクスターナルプロシージャ (続く)

 接続されていない [189](#)
 デバッグ [187](#)
 配布 [185](#)
 メモリ管理 [187](#)
 戻り値 [186](#)
 例外処理 [187](#)
Informix
 ストアドプロシージャに関する注意 [470](#)
InputBuffer ポート
 構造化されていないデータのトランスフォーメーション [502](#)
Integration Service
 ストアドプロシージャのエラー処理 [481](#)
 データタイプ [185](#)
 トランザクション境界 [71](#)
invokeJExpression
 API メソッド [260](#)
invokeJExpression メソッド
 Java トランスフォーメーション [250](#)
isNull メソッド
 Java トランスフォーメーション [251](#)
isResultNull メソッド
 Java トランスフォーメーション [268](#)

J

Java Classpath
 セッションのプロパティ [239](#)
Java コードスニペット
 Java トランスフォーメーション用に作成 [234](#)
 例 [272](#)
Java コードタブ
 使用 [228](#)
Java コードの生成
 Java 式 [259](#)
Java 式
 EDatatype クラス [263](#)
 invokeJExpression API メソッド [260](#)
 Java コードの生成 [259](#)
 Java トランスフォーメーション [257](#)
 JExpression クラス [265](#), [266](#)
 JExprParaMetadata クラス [263](#)
 カスタム関数の使用 [258](#)
 [関数の定義] ダイアログボックスで作成 [259](#)
 関数の設定 [259](#)
 高度なインタフェース [262](#)
 高度なインタフェースを使用した呼び出し [262](#)
 高度なインタフェースの例 [265](#)
 作成 [259](#)
 式の関数タイプ [258](#)
 [式の定義] ダイアログボックスで作成 [259](#)
 生成 [258](#)
 設定 [258](#)
 単純なインタフェース [260](#)
 単純なインタフェースの例 [261](#)
 単純なインタフェースを使用した呼び出し [260](#)
 トランスフォーメーション言語関数の使用 [258](#)
 ユーザー定義関数の使用 [258](#)
 呼び出し [250](#)
 呼び出しに関するルールおよびガイドライン [250](#)
 ルールおよびガイドライン [260](#), [262](#)
Java 式の API メソッド
 defineJExpression [264](#)
 getBytes [266](#)
 getDouble [267](#)
 getInt [267](#)
 getLong [267](#)

Java 式の API メソッド (続く)
 getResultDataType [267](#)
 getResultMetadata [267](#)
 getStringBuffer [268](#)
 isResultNull [268](#)
 呼び出し [268](#)
Java トランスフォーメーション
 API メソッド [244](#)
 CLASSPATH の設定 [239](#)
 defineJExpression メソッド [246](#)
 failSession メソッド [247](#)
 generateRow メソッド [247](#)
 getInRowType メソッド [248](#)
 getMetadata メソッド [249](#)
 [Helper コード] タブ [236](#)
 incrementErrorCount メソッド [249](#)
 invokeJExpression メソッド [250](#)
 isNull メソッド [251](#)
 Java コードスニペットの作成 [234](#)
 Java コードタブ [228](#)
 Java コードの作成 [233](#)
 Java プリミティブデータ型 [226](#)
 logError メソッド [251](#)
 logInfo メソッド [252](#)
 NULL 値の設定 [254](#)
 NULL 値のチェック [251](#)
 resetNotification メソッド [252](#)
 setNull メソッド [254](#)
 storeMetadata メソッド [255](#)
 アクティブ [226](#)
 アップデートストラテジの設定 [233](#)
 アップデートストラテジ (プロパティ) [233](#)
 [インポート] タブ [235](#)
 エラーの検出 [242](#)
 概要 [225](#)
 クラス名 (プロパティ) [230](#)
 グループの作成 [229](#)
 言語 (プロパティ) [230](#)
 コンパイル [241](#)
 コンパイルエラー [241](#)
 コンパイルエラーの原因の特定 [242](#)
 サブ秒の処理 [241](#)
 出力行タイプの設定 [254](#)
 [出力のソート] プロパティ [230](#)
 [出力は確定的] プロパティ [230](#)
 セッションの失敗 [247](#)
 セッションレベルのクラスパス [239](#)
 セッションログ [252](#)
 データ型の変換 [226](#)
 [データの終わりに達したとき] タブ [237](#)
 デバッグ [241](#)
 デフォルトのポート値 [229](#)
 トランザクション制御 [232](#)
 トランザクションの生成 (プロパティ) [233](#)
 [トランザクションの生成] プロパティ [230](#)
 [トランザクションを受け取ったとき] タブ [238](#)
 トランスフォーメーション範囲 (プロパティ) [230](#), [232](#)
 トランスフォーメーションレベルのクラスパス [239](#)
 トレースレベル (プロパティ) [230](#)
 入力行タイプの取得 [248](#)
 [入力行に達したとき] タブ [237](#)
 入力はブロック (プロパティ) [230](#)
 [パッケージのインポート] タブ [235](#)
 パッシブ [226](#)
 パーティション化可能 (プロパティ) [230](#)
 [パーティションごとに 1 つのスレッドを要求します] プロパティ [230](#)
 非ユーザーコードのエラー [243](#)

Java トランスフォーメーション (続く)
 フラットファイルの解析 [238](#)
 プロパティ [230](#)
 [ヘルパ] タブ [236](#)
 変数のリセット [252](#)
 ポートの作成 [229](#)
 マッピングの失敗 [247](#)
 メタデータの格納 [255](#)
 メタデータの取得 [249](#)
 ユーザーコードのエラー [242](#)
 例 [270](#)
 ログ [251](#), [252](#)
 出力ポート [230](#)
 入力ポート [230](#)
Java トランスフォーメーション API メソッド
 setOutRowType [254](#)
 コミット [245](#)
 ロールバック [253](#)
Java パッケージ
 インポート [235](#)
Java プリミティブデータ型
 Java トランスフォーメーション [226](#)
JDK
 Java トランスフォーメーション [225](#)
JExpression クラス
 Java 式 [265](#), [266](#)
JExprParaMetadata クラス
 Java 式 [263](#)
JRE
 Java トランスフォーメーション [225](#)

L
logError メソッド
 Java トランスフォーメーション [251](#)
LogicalConnectionObject
 SQL トランスフォーメーションの例 [462](#)
 説明 [437](#)
logInfo メソッド
 Java トランスフォーメーション [252](#)

M
Mapping Designer
 再利用可能なトランスフォーメーションの追加 [49](#)
Master Outer ジョイン
 説明 [282](#)
 トランザクション境界の保持 [289](#)
MFC AppWizard
 概要 [181](#)
Microsoft SQL Server
 ストアドプロシージャに関する注意 [471](#)
 接続文字列の構文 [438](#)

N
NaN
 1 への変換#QNAN [428](#)
NEXTVAL ポート
 シーケンスジェネレーター [385](#)
Normal ジョイン
 構文 [418](#)
 作成 [418](#)
 定義 [281](#)

Normal トレースレベル

概要 [46](#)

NULL 値

Java トランスフォーメーションの設定 [254](#)

Java トランスフォーメーションでのチェック [251](#)

集計関数 [55](#)

集計関数を使用した置換 [57](#)

スキップ [43](#)

ソータトランスフォーメーション [405](#)

定数を使用した置き換え [41](#)

フィルタリング [201](#)

NULL の詳細な順序付け (プロパティ)

ジョイナトランスフォーメーション [278](#)

NULL を無視 (プロパティ)

選択 [339](#)

NumRowsAffected ポート

SQL トランスフォーメーション [444](#)

O

OCCURS 文

COBOL の例 [360](#)

Oracle

ストアドプロシージャに関する注意 [471](#)

接続文字列の構文 [438](#)

ORDER BY

上書き [308](#)

オーバーライド [309](#)

ルックアップクエリ [308](#)

OutputBuffer ポート

構造化されていないデータのトランスフォーメーション [502](#)

OutputFileName ポート

構造化されていないデータのトランスフォーメーション [502](#)

出力タイプ

構造化されていないデータのトランスフォーメーション [500](#)

P

パフォーマンス

アグリゲータトランスフォーメーション [57](#)

ジョイナトランスフォーメーション [291](#)

フィルタの向上 [202](#)

ルックアップトランスフォーメーション [320](#)

R

resetNotification メソッド

Java トランスフォーメーション [252](#)

rollback

Java トランスフォーメーション API メソッド [253](#)

S

ScriptError ポート

説明 [430](#)

ScriptName ポート

SQL トランスフォーメーション [430](#)

ScriptResults ポート

SQL トランスフォーメーション [430](#)

Sequence Generator トランスフォーメーション

Blaze エンジン [401](#)

Spark エンジン [401](#)

setNull メソッド

Java トランスフォーメーション [254](#)

setOutRowType

Java トランスフォーメーション API メソッド [254](#)

SIN 番号

再現可能なデータマスキング [151](#)

社会保険番号のマスキング [151](#)

Source Analyzer

キー関係の作成 [414](#)

SQL

カスタムクエリの追加 [414](#)

デフォルトクエリの上書き [411](#)

デフォルトクエリのオーバーライド [414](#)

デフォルトのクエリを表示 [411](#)

SQL オーバーライド

デフォルトクエリ [411](#)

SQL クエリ

カスタムクエリの追加 [414](#)

デフォルトクエリの上書き [411](#)

デフォルトクエリのオーバーライド [414](#)

デフォルトのクエリを表示 [411](#)

動的更新の例 [454](#)

動的接続の例 [459](#)

[SQL 設定] タブ

SQL トランスフォーメーション [450](#)

SQL トランスフォーメーション

HA リカバリのガイドライン [442](#)

NumRowsAffected [444](#)

PM_REC_STATE テーブル

SQL トランスフォーメーションリカバリ [442](#)

ScriptError ポート [430](#)

ScriptName ポート [430](#)

ScriptResults ポート [430](#)

SELECT クエリ [433](#)

SQL 属性の設定 [450](#)

SQL ポートの説明 [450](#)

Verbose Data の設定 [448](#)

クエリモード [432](#)

サポートされている SQL 文 [451](#)

詳細オプション [439](#)

スクリプトモード [430](#)

静的 SQL クエリ [432](#)

静的クエリポート [433](#)

接続の設定 [437](#)

説明 [429](#)

データベース回復機能 [443](#)

データベースのレジリエンス [442](#)

動的 SQL クエリ [434](#)

動的クエリの例 [454](#)

動的接続の例 [459](#)

トランザクション制御 [441](#)

ネイティブデータタイプカラム [433](#)

パススルーポート [435](#)

パッシブモード [436](#), [437](#)

フル接続情報を渡す [438](#)

プロパティ [448](#)

メッセージを確実に 1 度だけ配信 [442](#)

文字列の置換の使用 [434](#)

SQL 文

SQL トランスフォーメーションでサポート [451](#)

[SQL ポート] タブ

SQL トランスフォーメーション [450](#)

storeMetadata メソッド

Java トランスフォーメーション [255](#)

surnames.dic

データのマスキング [155](#)

Sybase ASE

ORDER BY の制限 [309](#)

ストアドプロシージャに関する注意 [471](#)

接続文字列の構文 [438](#)

T

TC_COMMIT_AFTER 定数
説明 [487](#)
TC_COMMIT_BEFORE 定数
説明 [487](#)
TC_CONTINUE_TRANSACTION 定数
説明 [487](#)
TC_ROLLBACK_AFTER 定数
説明 [487](#)
TC_ROLLBACK_BEFORE 定数
説明 [487](#)
Teradata
接続文字列の構文 [438](#)
Terse トレースレベル
定義 [46](#)
TINFParm パラメータタイプ
定義 [188](#)
Transformation Developer
再利用可能なトランスフォーメーション [47](#)
Transformation Exchange (TX)
定義 [164](#)
TX-prefixed ファイル
エクスターナルプロシージャ [178](#)

U

UDT 設定
構造化されていないデータのトランスフォーメーション [500](#)
Unicode モード
エクスターナルプロシージャトランスフォーメーション [165](#)
カスタムトランスフォーメーション [62](#)
ジョイナトランスフォーメーション用ソート順の設定 [283](#)
URL
ビジネス文書へのリンクを使用した追加 [33](#)

V

[Verbose Data] トレースレベル
SQL トランスフォーメーション [448](#)
概要 [46](#)
Verbose Initialization トレースレベル
概要 [46](#)
Vertica
接続文字列の構文 [438](#)
Visual Basic
Application Setup ウィザード [184](#)
COM エクスターナルプロシージャの開発 [174](#)
COM データタイプ [185](#)
Integration Service への関数の追加 [187](#)
エクスターナルプロシージャのコード [165](#)
手動でのプロシージャの配布 [184](#)
ラッパークラス [187](#)
Visual C++
COM エクスターナルプロシージャの開発 [169](#)
COM データタイプ [185](#)
Integration Service へのライブラリの追加 [187](#)
手動でのプロシージャの配布 [184](#)
ラッパークラス [187](#)
VSAM ノーマライザトランスフォーメーション
作成 [363](#)
説明 [360](#)
[ノーマライザ] タブ [362](#)
[ポート] タブ [362](#)

W

Web リンク
式への追加 [33](#)
Windows システム
DLL のコンパイル [181](#)

X

XML
大きい出力の分割 [507](#)
XML ジェネレータトランスフォーメーション
概要 [517](#)
XML スキーマへのエクスポート
構造化されていないデータのトランスフォーメーション [505](#)
XML トランスフォーメーション
XML ジェネレータ [517](#)
XML パーサー [516](#)
ソース修飾子 [516](#)
XML 入力ストリーミング有効化
XML パーサーセッションプロパティ [507](#)
XML の分割
構造化されていないデータのトランスフォーメーション [507](#)
XML パーサートランスフォーメーション
概要 [516](#)
分割入力の有効化 [507](#)

あ

アウタージョイン
Extract Override として作成 [422](#)
Integration Service でサポートされるタイプ [416](#)
「結合タイプ [アウタージョイン]」も参照
aab] [422](#)
作成 [422](#)
ジョイン上書きとして作成 [422](#)
アクティブなトランスフォーメーション
Java [225](#), [226](#)
XML ジェネレータ [517](#)
XML ソース修飾子 [516](#)
XML パーサー [516](#)
アグリゲータ [51](#)
アップデートストラテジ [510](#)
概要 [25](#)
カスタム [61](#)
ジョイナ [277](#)
ソース修飾子 [407](#), [429](#)
ソータ [402](#)
ランザクション制御 [487](#)
ノーマライザ [355](#)
フィルタ [199](#)
ランク [373](#)
ルータ [378](#)
共有体 [493](#)
アクティブ (プロパティ)
Java トランスフォーメーション [230](#)
アグリゲータトランスフォーメーション
Group By ポート [55](#)
NULL 値 [55](#)
STDDEV (標準偏差) 関数 [54](#)
VARIANCE 関数 [54](#)
アップデートストラテジの組み合わせ [512](#)
概要 [51](#)
関数リスト [54](#)
コンポーネント [52](#)
作成 [59](#)

アグリゲータトランスフォーメーション (続く)

- 式トランスフォーメーションとの比較 [51](#)
- ジョイナトランスフォーメーションでの使用 [284](#)
- ソート済みポート [57](#)
- トラブルシューティング [60](#)
- トレースレベル [52](#)
- ネストされた集計 [54](#)
- パフォーマンスの最適化 [59](#)
- 非集計関数の例 [55](#)
- ポート [53](#)
- 条件句の例 [55](#)
- 変数の使用 [36](#)

値

- 式トランスフォーメーションを使用した計算 [162](#)

アップデートストラテジ

- Java トランスフォーメーションでの設定 [233](#)
- カスタムトランスフォーメーションでの設定 [69](#)
- 行ストラテジ関数 (行ベース) [118](#)

アップデートストラテジトランスフォーメーション

- アグリゲータの組み合わせ [512](#)

概要 [510](#)

拒否された行の転送 [511](#)

作成 [511](#)

式の入力 [512](#)

セッションのオプション設定 [513](#), [514](#)

設定手順 [511](#)

チェックリスト [515](#)

ルックアップの組み合わせ [513](#)

アップデートストラテジトランスフォーメーション (プロパティ)

- Java トランスフォーメーション [230](#)

い

依存関係

- カスタムトランスフォーメーションのポート [65](#)

一意の出力

- データマスキングトランスフォーメーション [136](#)

インクリメント

- シーケンス間隔の設定 [392](#), [393](#)

インスタンス

- 再利用可能なトランスフォーメーションの作成 [48](#)

インスタンス変数

- Java トランスフォーメーション [236-238](#)

インデックス

- ルックアップ条件 [320](#)

- ルックアップテーブル [299](#), [320](#)

引用識別子

- 予約語 [410](#)

う

ウィザード

- ATL COM AppWizard [169](#)

- MFC AppWizard [181](#)

- Visual Basic Application Setup ウィザード [184](#)

え

永続ルックアップキャッシュ

概要 [326](#)

共有 [328](#)

データベースからの再キャッシュ [324](#)

名前付きおよび名前なし [326](#)

名前付きファイル [328](#)

エクスターナルプロシージャ

- Informatica エクスターナルプロシージャの配布 [185](#)

インタフェース関数 [192](#)

作成にあたっての注意 [185](#)

デバッグ [187](#)

配布 [184](#)

エクスターナルプロシージャトランスフォーメーション

- 64 ビットエクスターナルプロシージャトランスフォーメーション

64 [194](#)

ATL オブジェクト [169](#)

BankSoft の例 [166](#)

BankSoft を使用する Informatica エクスターナルプロシージャの例 [175](#)

COM エクスターナルプロシージャ [168](#)

COM データタイプ [185](#)

COM と Informatica のタイプ比較 [166](#)

C++エクスターナルプロシージャ用ライブラリの構築 [171](#)

Designer での作成 [176](#)

IDispatch インタフェース [168](#)

Informatica エクスターナルプロシージャ用ライブラリの構築 [181](#)

Informatica エクスターナルプロシージャ [175](#)

MFC AppWizard [181](#)

Visual Basic [174](#)

Visual C++ [169](#)

Visual Basic エクスターナルプロシージャ用ライブラリの構築 [175](#)

インタフェース関数 [192](#)

エクスターナルプロシージャ関数 [193](#)

概要 [164](#)

行レベルのプロシージャ [186](#)

コードページアクセス関数 [196](#)

作成にあたっての注意 [185](#)

実行時位置 (プロパティ) [166](#)

出力が再現可能 (プロパティ) [166](#)

出力は確定的 (プロパティ) [166](#)

初期化 [189](#)

セッション [173](#)

接続されていない [189](#)

説明 [165](#)

ディスパッチ関数 [192](#)

デバッグ [187](#)

トレースレベル関数 [197](#)

トレースレベル (プロパティ) [166](#)

パーティション化可能 (プロパティ) [166](#)

パーティション関連の関数 [197](#)

パラメータアクセス関数 [194](#)

必要なファイル [191](#)

プログラム識別子 (プロパティ) [166](#)

プロセス変数のサポート [191](#)

プロパティ [165](#), [166](#)

プロパティアクセス関数 [193](#)

マッピングでの使用 [173](#)

マルチスレッドコード [164](#)

メモリ管理 [187](#)

メンバ変数 [194](#)

モジュール (プロパティ) [166](#)

戻り値 [186](#)

ラッパークラス [187](#)

例外処理 [187](#)

エラー

- Java トランスフォーメーションにおけるしきい値の増加 [249](#)

式エディタでの検証 [34](#)

処理 [43](#)

動的ルックアップキャッシュ [347](#)

エラー数

- Java トランスフォーメーションでの増分 [249](#)

エラー行

- SQL トランスフォーメーション [446](#)

エラー処理

- ストアドプロシージャ [481](#)
- 動的ルックアップキャッシュ [347](#)

エラーメッセージ

- エクスターナルプロシージャのトレース [187](#)
- エクスターナルプロシージャ用 [187](#)

演算子

- ルックアップ条件 [313](#)

お

大文字小文字の区別（プロパティ）

- ソータ変換 [404](#)

オーバーライド

- デフォルトソース修飾子 SQL クエリ [414](#)

か

開始桁

- 社会保険番号 [151](#)

開始値

- シーケンスジェネレータ変換 [392](#)

開発

- COM エクスターナルプロシージャ [168](#)
- Informatica エクスターナルプロシージャ [175](#)

外部キー

- シーケンスジェネレータ変換を使用した作成 [388](#)

格上げ

- 再利用不可能な変換 [49](#)

確実に 1 度だけ配信

- SQL 変換 [442](#)

格納テーブル

- 置換データマスキング [135](#)
- 式マスキング [145](#)

カスタム関数

- Java 式での使用 [258](#)

カスタム変換

- アップデートストラテジの設定 [69](#)
- アップデートストラテジプロパティ [69](#)

概要

関数

グループの作成

コードファイルの生成

コードページ

コンポーネント

作成

初期化プロパティ

スレッド

スレッド特有のコード

データのブロック

トランザクション境界

トランザクション制御

【トランザクションの生成】プロパティ

変換範囲のプロパティ

【入力ブロック】プロパティ

配布

パーティション化可能（プロパティ）

- 【パーティションごとに 1 つのスレッドを要求します】プロパティ [67](#)

プロセスに行を渡す

プロセスのコンパイル

プロセスの作成

プロセスのプロパティ

プロパティ

プロパティ ID

カスタム変換 (続)

ポート属性

ポートの関係の定義

ポートの作成

メタデータエクステンション

モジュールの構築

ルールおよびガイドライン

カスタム変換関数

API

エラー

エラーカウンターの増分

行ストラテジ（行ベース）

行ストラテジ（配列ベース）

行の数

行有効検証

最大行数

終了要求検査

出力通知

初期化

初期化解除

生成済み

セッションログ

通知

データタイプの再関連付け

データアクセスモード設定

データ境界の出力

データコードページ設定

データ操作（行ベース）

データ操作（配列ベース）

デフォルト行ストラテジ変更

ナビゲーション

入力エラー行の設定

配列ベース

パススルーポートの設定

ハンドルに関する作業

プロパティ

ポインタ

文字列モード変更

ロジックのブロック

カスタム変換プロセス

行に関する作業

コードファイルの生成

作成

スレッド特有

例

環境変数

Java パッケージ用の設定

関数

カスタム変換 API

集計

非集計

完全なデータベース接続

SQL 変換への受け渡し

関連するポート

シーケンス ID

ルックアップ変換 [338](#)

き

キー

- シーケンスジェネレータ変換を使用した作成 [388](#)

ジョイン用に作成

ソース定義

キーマスキング

数値

数値のマスキング

キーマスキング (続く)

- 説明 [132](#)
- 日時の値のマスキング [134](#)
- 文字列値のマスキング [133](#)
- キータイプ属性
 - ノーマライズトランスフォーメーション [362](#)
- キャッシュ
 - 共有ルックアップ [328](#)
 - コンカレント [324](#), [325](#)
 - ジョイナトランスフォーメーション [288](#)
 - ジョイナトランスフォーメーションにおけるマスタ行 [288](#)
 - 静的ルックアップキャッシュ [328](#)
 - 動的ルックアップキャッシュ [336](#)
 - 名前付き永続ルックアップ [328](#)
 - ルックアップトランスフォーメーション [322](#)
 - 連続した [324](#), [325](#)
- キャッシュサイズ
 - データマスキングトランスフォーメーション [153](#)
- キャッシュされる値の数
 - シーケンスジェネレータのプロパティ値 [390](#)
 - シーケンスジェネレータトランスフォーメーションプロパティ [394](#)
- キャッシュディレクトリ
 - データマスキングトランスフォーメーション [153](#)
 - ルックアップキャッシュディレクトリ名の設定 [301](#)
- キャッシュディレクトリ (プロパティ)
 - ジョイナトランスフォーメーション [278](#)
- キャッシュファイル名のプレフィックス
 - 概要 [328](#)
- キャッシュファイル名
 - 永続ルックアップキャッシュでの指定 [301](#)
- キャッシュファイル名のプレフィックス (プロパティ)
 - 説明 [301](#)
- 行
 - 更新のフラグ設定 [511](#)
 - 削除 [515](#)
- 行順序の維持
 - シーケンスジェネレータトランスフォーメーション [396](#)
- 行ストラテジ関数
 - 行ベース [118](#)
 - 配列ベース [125](#)
- 行トランスフォーメーション範囲
 - ジョイナトランスフォーメーションでの動作 [288](#)
- 行内の SQL エラー時でも処理を継続する (プロパティ)
 - SQL トランスフォーメーション [450](#)
- 行のフィルタリング
 - トランスフォーメーション [199](#), [402](#)
 - フィルタとしてのソース修飾子 [202](#)
- 行のルーティング
 - トランスフォーメーション [378](#)
- 行ベース関数
 - 行ストラテジ [118](#)
 - データ操作 [109](#)
- 共有
 - 名前付きルックアップキャッシュ [331](#)
 - 名前なしルックアップキャッシュ [328](#)
- 共有ストレージテーブル
 - データマスキングトランスフォーメーション [153](#)
- 拒否された行の転送
 - オプション [511](#)
 - 設定 [511](#)
- 拒否ファイル
 - アップデートストラテジ [511](#)

ク

- クエリ
 - ソース修飾子トランスフォーメーション [410](#), [414](#)

クエリ (続く)

- ルックアップトランスフォーメーション [308](#)
- ルックアップのオーバーライド [308](#)
- クエリモード
 - SQL トランスフォーメーション [432](#)
 - ルールおよびガイドライン [436](#)
- クラス名 (プロパティ)
 - Java トランスフォーメーション [230](#)
- グループ
 - HTTP トランスフォーメーション [207](#)
 - Java トランスフォーメーション [229](#)
 - カスタムトランスフォーメーション [64](#)
 - カスタムトランスフォーメーションのルール [65](#)
 - 共有体トランスフォーメーション [494](#)
 - 追加 [382](#)
 - ユーザー定義 [380](#)
 - ルータトランスフォーメーション [380](#)
- グループフィルタ条件
 - ルータトランスフォーメーション [380](#)

け

- 計算
 - 式トランスフォーメーションの使用 [161](#)
 - 集計 [51](#)
 - 変数の使用 [37](#)
- 結果文字列の置換文字
 - データマスキングトランスフォーメーション [143](#)
- 結合
 - Informatica 構文 [417](#)
 - カスタム [413](#)
 - キー関係の作成 [414](#)
 - ソース修飾子のデフォルト [412](#)
 - ユーザー定義 [416](#)
- 結合タイプ
 - Detail Outer ジョイン [282](#)
 - Full Outer ジョイン [282](#)
 - Master Outer ジョイン [282](#)
 - Normal ジョイン [281](#)
 - ジョイナのプロパティ [280](#)
 - ソース修飾子トランスフォーメーション [416](#)
 - ライトアウタージョイン [416](#)
 - レフトアウタージョイン [416](#)
- 結合タイプ (プロパティ)
 - ジョイナトランスフォーメーション [278](#)
- 欠落値
 - シーケンスジェネレータを使用した置き換え [388](#)
- 言語 (プロパティ)
 - Java トランスフォーメーション [230](#)
- 現在の値
 - シーケンスジェネレータトランスフォーメーション [393](#)
- 検証
 - 式 [34](#)
 - デフォルト値 [46](#)

こ

- 更新でなければ挿入 (プロパティ)
 - 説明 [345](#), [514](#)
- 更新として更新 (プロパティ)
 - 説明 [514](#)
- 高精度
 - Java トランスフォーメーションにおける有効化 [240](#)
- 構造化されていないデータのトランスフォーメーション
 - InputBuffer ポート [502](#)
 - OutputBuffer ポート [502](#)

構造化されていないデータのトランスフォーメーション (続く)

- OutputFileName ポート [502](#)
- [UDT 設定] タブ [500](#)
- XML 出力の分割 [507](#)
- XML スキーマへのエクスポート [505](#)
- XML ファイルへの書き込み [507](#)
- 概要 [496](#)
- コンポーネント [499](#)
- ステータストレースレベルの設定 [501](#)
- 設定 [508](#)
- データトランスフォーメーションからのポートの読み込み [504](#)
- 入力データのフラッシュ [505](#)
- ファイル出力タイプの例 [506](#)
- ポート [503](#)
- リレーショナルターゲットへの書き込み [505](#), [506](#)

高度なインタフェース

- EDataType クラス [263](#)
- Java 式 [262](#)
- Java 式の呼び出し [262](#)
- JExpression クラス [265](#), [266](#)
- JExprParaMetadata クラス [263](#)
- 例 [265](#)

構文

- Normal ジョインの作成 [418](#)
- 共通するデータベース制約 [423](#)
- ライトアウトジョインの作成 [421](#)
- レフトアウトジョインの作成 [419](#)

コードスニペット

- Java トランスフォーメーション用に作成 [234](#)

コード全体ウィンドウ

- Java コンパイルエラー [242](#)

コードページ

- アクセス関数 [196](#)
- エクスターナルプロシージャトランスフォーメーション [165](#)
- カスタムトランスフォーメーション [62](#)

コードページ ID

- カスタムトランスフォーメーション、変更 [118](#)

個別に選択

- セッションでの上書き [426](#)
- [ソース修飾子] オプション [425](#)

コミット

- Java トランスフォーメーション API メソッド [245](#)

コメント

- 式への追加 [33](#)

コンカレントキャッシュ

- 「キャッシュ」を参照 [324](#)

コンパイル

- Java トランスフォーメーション [241](#)
- Windows システム上の DLL [181](#)
- カスタムトランスフォーメーションプロシージャ [82](#)

コンパイルエラー

- Java トランスフォーメーションのソースの識別 [242](#)

さ

サイクル

- シーケンスジェネレータトランスフォーメーションプロパティ [392](#)

サイクル (プロパティ)

- シーケンスジェネレータトランスフォーメーションプロパティ [390](#)

再現可能な出力

- データマスキングトランスフォーメーション [131](#)

最大出力行数 (プロパティ)

- SQL トランスフォーメーション [450](#)

再利用可能

- シーケンスジェネレータトランスフォーメーション [396](#)

再利用可能なトランスフォーメーション

- 概要 [47](#)

再利用可能なトランスフォーメーション (続く)

- 再利用不可能なインスタンスの作成 [49](#)

- 作成 [48](#)

- 変更 [50](#)

- マッピングへの追加 [49](#)

- マッピング変数 [48](#)

作業ディレクトリ

- ソートトランスフォーメーション、指定 [404](#)

作成

- COM エクスターナルプロシージャ [168](#)

- Informatica エクスターナルプロシージャ [175](#)

- アグリゲータトランスフォーメーション [59](#)

- アップデートストラテジトランスフォーメーション [511](#)

- カスタムトランスフォーメーション [63](#), [74](#)

- 共有体トランスフォーメーション [494](#)

- 再利用可能なトランスフォーメーション [48](#)

- 再利用可能なトランスフォーメーションの再利用不可能なインスタンス [49](#)

- シーケンスデータオブジェクト [397](#)

- 式トランスフォーメーション [163](#)

- シーケンスジェネレータトランスフォーメーション [398](#), [399](#)

- ジョイナトランスフォーメーション [290](#)

- ストアドプロシージャトランスフォーメーション [472](#)

- トランスフォーメーション [28](#)

- フィルタトランスフォーメーション [201](#)

- ポート [30](#)

- リンクトランスフォーメーション [376](#)

- ルータトランスフォーメーション [383](#)

サブ秒

- Java トランスフォーメーションでの処理 [241](#)

サブ秒の精度

- ストアドプロシージャトランスフォーメーション [475](#)

- ルックアップトランスフォーメーション [301](#)

し

シーケンスデータオブジェクト

- プロパティ [396](#)

- 作成 [397](#)

シード値

- データマスキングトランスフォーメーション [131](#)

式

- Java トランスフォーメーション [257](#)

- アグリゲータトランスフォーメーション [54](#)

- アップデートストラテジ [512](#)

- 簡略化 [36](#)

- 検証 [34](#)

- ストアドプロシージャトランスフォーメーションのルール [483](#)

- ストアドプロシージャの呼び出し [478](#)

- 入力 [31](#), [33](#)

- 非集計 [57](#)

- フィルタ条件 [200](#)

- 戻り値 [31](#)

- ルックアップの呼び出し [317](#)

式エディタ

- Java 式での使用 [259](#)

- 概要 [33](#)

- 構文の色 [34](#)

- 式の検証 [34](#)

シーケンス ID

- ルックアップトランスフォーメーション [338](#)

シーケンスジェネレータトランスフォーメーション

- CURRVAL ポート [389](#)

- IIF 関数を使用した欠落キーの置き換え [388](#)

- NEXTVAL ポート [385](#)

- 値の範囲 [393](#)

- 開始値 [392](#)

シーケンスジェネレータトランスフォーメーション (続く)

- 概要 [384](#)
- キャッシュされる値の数 [394](#)
- 行順序の維持 [396](#)
- 現在の値 [393](#)
- サイクル [392](#), [393](#)
- 再利用可能 [395](#)
- 再利用不可 [395](#)
- 作成 [398](#), [399](#)
- [増分] プロパティ [392](#), [393](#)
- 複合キーの作成 [388](#)
- プロパティ [390](#), [396](#)
- ポート [385](#)
- リセット [396](#)
- 非ネイティブ環境 [401](#)
- 実行時位置 (プロパティ)
 - エクスターナルプロシージャトランスフォーメーション [166](#)
- 実行順序 (プロパティ)
 - ストアドプロシージャトランスフォーメーション [475](#)
- 自動コミット
 - SQL トランスフォーメーションを使用した設定 [441](#)
 - 説明 [450](#)
- 社会保障番号
 - 再現可能なデータマスキング [148](#)
 - 地域コードマスキング [148](#)
- 集計関数
 - NULL 値 [55](#)
 - 概要 [54](#)
 - 式の使用 [54](#)
 - リスト [54](#)
- [出力階層] タブ
 - 構造化されていないデータのトランスフォーメーション [505](#)
- 出力が再現可能 (プロパティ)
 - エクスターナルプロシージャトランスフォーメーション [166](#)
 - ストアドプロシージャトランスフォーメーション [475](#)
- 出力行
 - Java トランスフォーメーションでの行タイプの設定 [254](#)
- 出力のソート (プロパティ)
 - Java トランスフォーメーション [230](#)
- 出力は確定的 (プロパティ)
 - Java トランスフォーメーション [230](#)
 - エクスターナルプロシージャトランスフォーメーション [166](#)
 - ストアドプロシージャトランスフォーメーション [475](#)
- 出力パラメータ
 - ストアドプロシージャ [466](#)
 - 接続されていないストアドプロシージャトランスフォーメーション [478](#)
- 出力ポート
 - Java トランスフォーメーション [230](#)
 - エラー処理 [39](#)
 - 概要 [30](#)
 - 式トランスフォーメーションに必要 [162](#)
 - デフォルト値 [39](#)
 - 変数として使用 [237](#)
- ジョイナキャッシュ
 - ジョイナトランスフォーメーション [288](#)
- ジョイナトランスフォーメーション
 - ASCII モードでのソート順の設定 [283](#)
 - Unicode モードでのソート順の設定 [283](#)
 - 同じソースからのデータの結合 [285](#)
 - 概要 [277](#)
 - キャッシュ [288](#)
 - 行トランスフォーメーション範囲 [288](#)
 - 行トランスフォーメーション範囲の動作 [288](#)
 - 結合タイプ [280](#)
 - 作成 [290](#)
 - 条件 [280](#)
 - すべての入力トランスフォーメーション範囲 [288](#)

ジョイナトランスフォーメーション (続く)

- すべての入力トランスフォーメーション範囲の動作 [288](#)
- ソースデータのブロック [287](#)
- ソータトランスフォーメーションでの使用 [283](#)
- ソート順の設定 [283](#)
- ソートの基点ポートを使用するためのジョイン条件の設定 [283](#)
- トランザクション [288](#)
- トランザクション境界の削除 [290](#)
- トランザクション境界の保持 [289](#)
- トランザクショントランスフォーメーション範囲 [288](#)
- トランザクショントランスフォーメーション範囲の動作 [288](#)
- トランスフォーメーション範囲 [288](#)
- 入力ルール [277](#)
- パフォーマンスのヒント [291](#)
- 複数のデータベースの結合 [277](#)
- プロパティ [278](#)
- リアルタイムデータ [288](#)
- リアルタイムデータの処理 [290](#)
- ジョイナのインデックスキャッシュサイズ
 - ジョイナトランスフォーメーションプロパティ [278](#)
- ジョイナのデータキャッシュサイズ
 - ジョイナトランスフォーメーションプロパティ [278](#)
- ジョインオーバーライド
 - Normal ジョイン構文 [418](#)
 - ライトアウタージョイン構文 [421](#)
 - レフトアウタージョイン構文 [419](#)
- ジョイン構文
 - Normal ジョイン [418](#)
 - ライトアウタージョイン [421](#)
 - レフトアウタージョイン [419](#)
- ジョイン条件
 - 概要 [280](#)
 - ソートの基点ポートの使用 [283](#)
 - 定義 [284](#)
- 条件
 - ジョイナトランスフォーメーション [280](#)
 - フィルタトランスフォーメーション [200](#)
 - ルータトランスフォーメーション [380](#)
 - ルックアップトランスフォーメーション [312](#), [316](#)
- 詳細オプション
 - SQL トランスフォーメーション [439](#)
- 初期化
 - Integration Service 変数のサポート [191](#)
 - エクスターナルプロシージャ [189](#)
 - カスタムトランスフォーメーションプロシージャ [73](#)
 - 変数 [39](#)
- 初期化解除関数
 - カスタムトランスフォーメーション [94](#)
- 初期化関数
 - カスタムトランスフォーメーション [91](#)
- シリアライザ
 - データトランスフォーメーション [498](#)

す

- 数値
 - キーマスキング [133](#)
 - ランダムマスキング [140](#)
- スクリプトモード
 - SQL トランスフォーメーション [430](#)
 - ルールおよびガイドライン [431](#)
- スクリプトロケールオプション
 - SQL トランスフォーメーション [450](#)
- 状態コード
 - ストアドプロシージャトランスフォーメーション [467](#)
- ステータストレースレベル
 - 構造化されていないデータのトランスフォーメーション [501](#)

ストアドプロシージャ
IBM DB2 の例 [471](#)
Informix の例 [470](#)
Microsoft の例 [471](#)
Oracle の例 [471](#)
Sybase の例 [471](#)
Teradata の例 [472](#)
インポート [472](#)
エラー処理 [481](#)
記述 [470](#)
サポートされるデータベース [482](#)
処理の順序、指定 [468](#)
セッション実行後のエラー [482](#)
セッション実行前のエラー [481](#)
セッション実行前またはセッション実行後のセッションの作成 [480](#)
セッションのエラー [482](#)
タイプの設定 [475](#)
タイプのロード [480](#)
定義 [465](#)
データベース固有の構文に関する注意 [470](#)
変更パラメータ [476](#)
変数に書き込み [38](#)
ストアドプロシージャタイプ (プロパティ)
ストアドプロシージャトランスフォーメーション [475](#)
ストアドプロシージャトランスフォーメーション
インポートを使用した作成 [472](#)
オプションの設定 [475](#)
概要 [465](#)
式のルール [483](#)
実行時の指定 [468](#)
実行順序 (プロパティ) [475](#)
出力が再現可能 (プロパティ) [475](#)
出力データ [466](#)
出力は確定的 (プロパティ) [475](#)
手動による作成 [473](#), [475](#)
状態コード [467](#)
ストアドプロシージャタイプ (プロパティ) [475](#)
ストアドプロシージャのインポート [472](#)
セッション実行時、指定 [468](#)
セッション前/後 [480](#)
セッション前またはセッション後に実行 [480](#)
接続された [467](#)
接続されたストアドプロシージャの設定 [477](#)
接続されていない [467](#), [477](#)
接続情報 (プロパティ) [475](#)
設定 [469](#)
トラブルシューティング [484](#)
トレースレベル (プロパティ) [475](#)
日時ポートのサブ秒の精度 [475](#)
入出力パラメータ [466](#)
入力データ [466](#)
パフォーマンスのヒント [484](#)
プロパティ [475](#)
変更 [476](#)
戻り値 [466](#)
呼び出しテキスト (プロパティ) [475](#)
接続されていないストアドプロシージャの設定 [477](#)
ストリーマコンポーネント
データトランスフォーメーション [498](#)
ストリーマチャンクサイズ
構造化されていないデータのトランスフォーメーション [500](#)
ストレージの暗号化
データマスキングトランスフォーメーション [153](#)
ストレージのコミット間隔
データマスキングトランスフォーメーション [153](#)
ストレージの暗号化キー
データマスキングトランスフォーメーション [153](#)

すべての入力トランスフォーメーション範囲
ジョイナトランスフォーメーションでの動作 [288](#)
スレッド
カスタムトランスフォーメーション [69](#)
スレッド特有の操作
HTTP トランスフォーメーション [205](#)
カスタムトランスフォーメーション [67](#)
記述 [69](#)

せ

生成キー
ノーマライザトランスフォーメーション [359](#)
生成されたカラム ID
ノーマライザトランスフォーメーション [356](#)
生成済み関数
カスタムトランスフォーメーション [90](#)
静的 SQL クエリ
SQL トランスフォーメーション [432](#)
ポートの設定 [433](#)
静的コード
Java トランスフォーメーション [236](#)
静的データベース接続
説明 [437](#)
パフォーマンスに関する考慮事項 [440](#)
静的変数
Java トランスフォーメーション [236](#)
静的ルックアップキャッシュ
概要 [328](#)
セッション
\$\$\$SessStartTime [409](#)
アップデートストラテジの設定 [513](#)
エクスターナルプロシージャトランスフォーメーション [173](#)
個別に選択の上書き [426](#)
差分集計 [51](#)
ストアドプロシージャ前/後、実行 [480](#)
ストアドプロシージャトランスフォーメーション [467](#)
ストアドプロシージャのエラー処理の設定 [481](#)
セッション実行後
エラー [482](#)
ストアドプロシージャ [480](#)
セッション実行前
エラー [481](#)
ストアドプロシージャ [480](#)
セッション実行前/実行後 SQL
ソース修飾子トランスフォーメーション [426](#)
セッションの失敗
Java トランスフォーメーション [247](#)
セッションログ
Java トランスフォーメーション [252](#)
接続
SQL トランスフォーメーション [437](#)
接続文字列の例 [438](#)
接続されたトランスフォーメーション
Java [225](#)
SQL [429](#)
XML ジェネレータ [517](#)
XML ソース修飾子 [516](#)
XML パーサー [516](#)
アグリゲータ [51](#)
アップデートストラテジ [510](#)
カスタム [61](#)
共有体トランスフォーメーション [493](#)
式 [161](#)
シーケンスジェネレータ [384](#)
ジョイナ [277](#)
ストアドプロシージャ [465](#)

接続されたトランスフォーメーション (続く)

ソース修飾子 [407](#)
ノーマライザ [355](#)
フィルタ [199](#)
ランク [373](#)
ルータ [378](#)
ルックアップ [292](#)

接続されたルックアップ

概要 [297](#)
作成 [318](#)
説明 [296](#)

接続されていないトランスフォーメーション

エクスターナルプロシージャトランスフォーメーション [189](#)
ストアドプロシージャトランスフォーメーション [465](#)
ルックアップ [292](#)
ルックアップトランスフォーメーション [298](#), [316](#)

接続されていないルックアップ

概要 [298](#), [316](#)
式による呼び出し [317](#)
説明 [296](#)
ルックアップ条件の追加 [316](#)
戻り値の指定 [317](#)

接続されていないルックアップトランスフォーメーション

入力ポート [316](#)
戻りポート [317](#)

接続設定

SQL トランスフォーメーション [438](#)

接続プールの使用 (プロパティ)

SQL トランスフォーメーション [450](#)

接続文字列

構文 [438](#)

設定

ポート [29](#), [30](#)

そ

挿入でなければ更新 (プロパティ)

説明 [344](#)

挿入として更新 (プロパティ)

説明 [514](#)

ソース

同じソースからのデータの結合 [285](#)
結合 [277](#)
統合 [493](#)
複数の結合 [277](#)

\$Source

ストアドプロシージャトランスフォーメーション [ソース] [475](#)
ルックアップトランスフォーメーション [301](#)

ソースフィルタ

ソース修飾子への追加 [423](#)

ソート済みポート

アグリゲータトランスフォーメーション [57](#)
キャッシュ要件 [53](#)
使用しない理由 [58](#)
ソース修飾子 [424](#)
ソート順 [424](#)
ソート前データ [58](#)

ソース行の扱い

アップデートストラテジ [513](#)

ソース行のフィルタリング

ルックアップトランスフォーメーション [311](#), [312](#)

ソース修飾子トランスフォーメーション

\$\$\$SessStartTime [409](#)
SQL オーバーライド [414](#)
XML ソース修飾子 [516](#)
アウタージョインのサポート [416](#)
アグリゲータを使用したソート順 [58](#)

ソース修飾子トランスフォーメーション (続く)

概要 [407](#)
キー関係の作成 [414](#)
結合 [414](#)
[個別に選択] オプション [425](#)
セッション実行前/実行後 SQL [426](#)
設定 [427](#)
ソースフィルタの入力 [423](#)
ソースデータの結合 [412](#)
ソート済みオプションの数 [424](#)
ターゲットのロード順 [408](#)
データタイプ [408](#)
デフォルトクエリ [410](#)
デフォルトクエリの上書き [411](#)
デフォルトクエリのオーバーライド [414](#)
デフォルトジョイン [412](#)
デフォルトのクエリを表示 [411](#)
トラブルシューティング [428](#)
プロパティ [427](#)
マッピングパラメータとマッピング変数 [409](#)
ユーザー作成のジョイン [413](#)
ユーザー定義ジョインの入力 [416](#)
ルックアップソースとしての [295](#)

ソース文字列の文字

デタマスキングトランスフォーメーション [142](#)

ソータトランスフォーメーション

概要 [402](#)
作業ディレクトリ [404](#)
作成 [405](#)
ジョイナトランスフォーメーションでの使用 [283](#)
設定 [403](#)
ソータキャッシュサイズの設定 [404](#)
プロパティ [403](#)

ソート順

アグリゲータトランスフォーメーション [58](#)
ジョイナトランスフォーメーションの設定 [283](#)
ソース修飾子トランスフォーメーション [424](#)

ソート済みデータ

ジョイナトランスフォーメーションでの使用 [283](#)
パーティション化されたパイプラインからの結合 [283](#)
ソート済みデータの結合
結合処理のパフォーマンスを最適化するための設定 [283](#)
ソータトランスフォーメーションの使用 [283](#)
ソート済みフラットファイルの使用 [283](#)
ソート済みリレーショナルデータの使用 [283](#)

ソート済み入力

フラットファイルルックアップ [294](#)

ソート済み入力 (プロパティ)

アグリゲータトランスフォーメーション [57](#)
ジョイナトランスフォーメーション [278](#)

ソート済みフラットファイル

ジョイナトランスフォーメーションでの使用 [283](#)

ソート済みリレーショナルデータ

ジョイナトランスフォーメーションでの使用 [283](#)

ソートの基点

使用するジョイン条件の設定 [283](#)
定義 [283](#)

た

ターゲット

更新 [510](#)

\$Target

ストアドプロシージャトランスフォーメーション [ターゲット]
[475](#)
ルックアップトランスフォーメーション [ターゲット] [301](#)

ターゲットのロード順
ソース修飾子 [408](#)
タイプのロード
ストアドプロシージャ [480](#)
ターゲットテーブル
アップデーストラテジの設定 [514](#)
行の削除 [515](#)
挿入 [515](#)
単純なインタフェース
Java 式 [260](#)
Java トランスフォーメーション API メソッド [260](#)
例 [261](#)

ち

置換マスキング
説明 [134](#)
データマスキングのルールおよびガイドライン [137](#)
マスキングプロパティ [135](#)
リレーショナルディクショナリの設定 [137](#)
重複しない出力行
ソートトランスフォーメーション [404](#)

つ

追加
グループ [382](#)
式へのコメント [33](#)
通知関数
カスタムトランスフォーメーション [92](#)

て

定義
カスタムトランスフォーメーションでのポートの依存関係 [65](#)
ディクショナリ
再現可能な式マスキング [145](#)
置換データマスキング [134](#)
ディクショナリ情報
データマスキングトランスフォーメーション [136](#)
定数
NULL 値との置き換え [41](#)
ディスパッチ関数
説明 [192](#)
データ
アップデーストラテジトランスフォーメーションを通じた拒否
[515](#)
一時保存 [36](#)
結合 [277](#)
個別に選択 [425](#)
ソート前 [58](#)
データタイプ
COM [185](#)
ソース修飾子 [408](#)
トランスフォーメーション [185](#)
データタイプの再関連付け関数
カスタムトランスフォーメーション関数 [108](#)
データ型
Java トランスフォーメーション [226](#)
テーブル
キー関係の作成 [414](#)
データアクセスモード関数
カスタムトランスフォーメーション関数 [97](#)
データコードページ設定関数
カスタムトランスフォーメーション関数 [118](#)

データ操作関数
行ベース [109](#)
配列ベース [123](#)
データトランスフォーメーション
概要 [496](#)
リボジトリの場所 [498](#)
データドリブン
概要 [513](#)
[データの終わりに達したとき] タブ
Java トランスフォーメーション [237](#)
データのブロック
カスタムトランスフォーメーション [71](#)
カスタムトランスフォーメーション関数 [116](#)
ジョイナトランスフォーメーション [287](#)
データのマスキング
サンプルの会社名 [155](#)
サンプルの住所 [155](#)
サンプルの姓 [155](#)
サンプルの名 [155](#)
ルックアップデータのサンプル [155](#)
データベース
異なるデータベースからのデータの結合 [277](#)
サポートされているオプション [482](#)
データベースからの再キャッシュ
概要 [324](#)
名前付きキャッシュ [326](#)
名前なしキャッシュ [326](#)
データベース接続
SQL トランスフォーメーション [437](#)
データベースのレジリエンス
SQL トランスフォーメーション [442](#)
データベースへの接続
SQL トランスフォーメーション [437](#)
データマスキング
式トランスフォーメーションを使用したマッピング [159](#)
ルックアップトランスフォーメーションを使用したマッピング [155](#)
データマスキングトランスフォーメーション
IP アドレスのマスキング [151](#)
URL のマスキング [152](#)
一意の出力 [153](#)
格納テーブル [135](#), [145](#)
キャッシュサイズ [153](#)
キャッシュディレクトリ [153](#)
共有ストレージテーブル [153](#)
再現可能な SIN 番号 [151](#)
再現可能な SSN [148](#)
再現可能な式マスキング [145](#)
式マスキングのガイドライン [146](#)
ストレージのコミット間隔 [153](#)
セッションプロパティ [153](#)
接続要件 [137](#)
ソース文字列の文字 [142](#)
置換マスキング [134](#)
置換マスキングのディクショナリ [134](#)
置換マスキングプロパティ [135](#), [136](#)
ディクショナリ名の式マスキング [145](#)
データマスキングトランスフォーメーション [153](#)
デフォルト値ファイル [152](#)
電話番号のマスキング [149](#)
範囲 [143](#)
日付値のマスキング [144](#)
ブラー [144](#)
マスキングプロパティ [130](#)
マスク形式 [141](#)
マッピングパラメータの使用 [131](#)
ランダムマスキング [139](#)
リレーショナルディクショナリの設定 [137](#)
ルールおよびガイドライン [153](#)

データマスキングトランスフォーメーション (続く)

- 依存データマスキング [138](#)
- 再現可能な依存マスキング [139](#)
- 式マスキング [145](#)
- 社会保険番号のマスキング [151](#)
- 社会保険番号のマスキング [147](#)
- 電子メールアドレスのマスキング [149](#)

デバッグ

- Java トランスフォーメーション [241](#)
- エクスターナルプロシージャ [187](#)

デフォルトクエリ

- 上書きの方法 [411](#)
- 概要 [410](#)
- ソース修飾子を使用したオーバーライド [414](#)
- 表示 [410](#)

デフォルトグループ

- ルータトランスフォーメーション [380](#)

デフォルトジョイン

- ソース修飾子 [412](#)

デフォルト値

- アグリゲータ Group By ポート [57](#)
- 概要 [39](#)
- 検証 [46](#)
- 出力ポート [39](#), [40](#)
- データマスキング [152](#)
- 入力 [46](#)
- 入力ポート [39](#), [40](#)
- パススルーポート [39](#), [40](#)
- ユーザー定義 [40](#)
- ルール [45](#)

と

統計出力ポートの追加 (プロパティ)

- SQL トランスフォーメーション [450](#)

統合サービス

- データの集計 [55](#)
- デバッグモードでの実行 [187](#)
- 変数のサポート [191](#)

動的 SQL クエリ

- SQL トランスフォーメーション [434](#)
- SQL トランスフォーメーションの例 [454](#)

動的サービス名

- 構造化されていないデータのトランスフォーメーション [500](#)

動的接続

- SQL トランスフォーメーション [438](#)
- SQL トランスフォーメーションの例 [459](#)
- パフォーマンスに関する考慮事項 [440](#)

動的ルックアップキャッシュ

- エラーしきい値 [347](#)
- 説明 [336](#)
- ターゲットとの同期 [347](#)
- フラットファイルソースの使用 [336](#)
- ルックアップ SQL オーバーライド [341](#)
- ロードの拒否 [347](#)

登録

- COM プロシージャをリポジトリへ [172](#)

トラブルシューティング

- Java トランスフォーメーション [241](#)
- アグリゲータトランスフォーメーション [60](#)
- ストアドプロシージャトランスフォーメーション [484](#)
- ソース修飾子トランスフォーメーション [428](#)
- ノーマライザトランスフォーメーション [372](#)

トランザクション

- ジョイナトランスフォーメーションでの作業 [288](#)
- 生成 [70](#), [233](#), [245](#)
- 定義 [487](#)

トランザクション境界

- カスタムトランスフォーメーション [71](#)
- ジョイナトランスフォーメーションでの削除 [290](#)
- ジョイナトランスフォーメーションでの保持 [289](#)

トランザクション制御

- Java トランスフォーメーション [232](#)
- SQL トランスフォーメーション [441](#)
- 概要 [486](#)
- カスタムトランスフォーメーション [70](#)
- 式 [487](#)
- トランスフォーメーション [487](#)
- 例 [488](#)

トランザクション制御トランスフォーメーション

- 概要 [487](#)
- 作成 [492](#)
- プロパティ [487](#)
- マッピング内 [489](#)
- マッピングの検証 [491](#)
- 無効 [489](#)
- 有効 [489](#)

トランザクショントランスフォーメーション範囲

- ジョイナトランスフォーメーションでの動作 [288](#)

トランザクションの生成

- Java トランスフォーメーション [232](#), [233](#), [245](#)
- カスタムトランスフォーメーション [70](#)

トランザクションの生成 (プロパティ)

- Java トランスフォーメーション [230](#)

[トランザクションを受け取ったとき] タブ

- Java トランスフォーメーション [238](#)

トランスフォーマコンポーネント

- データトランスフォーメーション [498](#)

トランスフォーメーション

- Java [225](#)
- SQL [429](#)
- XML ジェネレータ [517](#)
- XML ソース修飾子 [516](#)
- XML パーサー [516](#)
- アクティブとパッシブ [25](#)
- アグリゲータ [51](#)
- アップデートストラテジ [510](#)
- エラー処理 [43](#)
- 概要 [24](#)
- カスタム [61](#)
- 共有体 [493](#)
- 再利用可能なトランスフォーメーション [47](#)
- 再利用可能に格上げ [49](#)
- 再利用可能にする [49](#)
- 作成 [28](#)
- 式 [161](#)
- 式で利用できるタイプ [31](#)
- シーケンスジェネレータ [384](#)
- ジョイナ [277](#)
- ストアドプロシージャ [465](#)
- 接続された [25](#)
- 接続されていない [25](#)
- 説明 [26](#)
- ソース修飾子 [407](#)
- タイプ [24](#)
- 定義 [24](#)
- トレースレベル [46](#)
- ネイティブと非ネイティブ [25](#)
- ノーマライザ [355](#)
- フィルタ [199](#)
- 複数グループ [31](#)
- マッピングへの追加 [28](#)
- ランク [373](#)
- ルータ [378](#)
- ルックアップ [292](#)

トランスフォーメーション言語

Java 式での使用 [258](#)

集計関数 [54](#)

トランスフォーメーションのブロック

説明 [31](#)

トランスフォーメーション範囲

Java トランスフォーメーション [232](#)

カスタムトランスフォーメーション [70](#)

ジョイナトランスフォーメーションでの行トランスフォーメーション範囲 [288](#)

ジョイナトランスフォーメーションでのすべての入力トランスフォーメーション範囲 [288](#)

ジョイナトランスフォーメーションでのトランザクショントランスフォーメーション範囲 [288](#)

ジョイナトランスフォーメーションの定義 [288](#)

ジョイナトランスフォーメーションプロパティ [278](#)

ソータトランスフォーメーション [405](#)

トランスフォーメーション範囲 (プロパティ)

Java トランスフォーメーション [230](#)

トレースレベル

Java トランスフォーメーションプロパティ [230](#)

Terse [46](#)

Verbose Data [46](#)

Verbose Initialization [46](#)

上書き [46](#)

エクスターナルプロシージャトランスフォーメーションプロパティ [166](#)

概要 [46](#)

ジョイナトランスフォーメーションプロパティ [278](#)

ストアドプロシージャトランスフォーメーションのプロパティ [475](#)

セッションのプロパティ [52](#)

ソータトランスフォーメーションプロパティ [405](#)

ノーマル [46](#)

トレースレベル関数

説明 [197](#)

な

名前付き永続ルックアップキャッシュ

概要 [328](#)

共有 [331](#)

名前付きキャッシュ

永続 [326](#)

共有 [331](#)

データベースからの再キャッシュ [326](#)

名前なしキャッシュ

永続 [326](#)

共有 [328](#)

データベースからの再キャッシュ [326](#)

に

日時の値

ソース修飾子トランスフォーメーション [408](#)

データーマスキング [134](#)

入出力ポート

概要 [30](#)

入力

SQL クエリオーバーライド [414](#)

式 [31](#), [33](#)

ソースフィルタ [423](#)

ユーザー定義ジョイン [416](#)

[入力階層] タブ

構造化されていないデータのトランスフォーメーション [505](#)

入力行

行タイプの取得 [248](#)

[入力行に達したとき] タブ

Java トランスフォーメーション [237](#)

例 [273](#)

入力タイプ

構造化されていないデータのトランスフォーメーション [500](#)

入力ブロック (プロパティ)

Java トランスフォーメーション [230](#)

入力パラメータ

ストアドプロシージャ [466](#)

入力ポート

Java トランスフォーメーション [230](#)

概要 [30](#)

デフォルト値 [39](#)

変数として使用 [237](#)

ね

ネイティブデータタイプ

SQL トランスフォーメーション [433](#)

ネイティブトランスフォーメーション

概要 [25](#)

の

ノーマルジョイン

トランザクション境界の保持 [289](#)

ノーマライザトランスフォーメーション

VSAM ノーマライザ [360](#)

VSAM ノーマライザトランスフォーメーションの作成 [363](#)

概要 [355](#)

キータイプ属性 [362](#)

生成キー [359](#)

生成されたカラム ID [356](#)

トラブルシューティング [372](#)

[ノーマライザ] タブ [358](#)

パイプラインノーマライザ [364](#)

パイプラインノーマライザトランスフォーメーションの作成 [367](#)

パイプラインノーマライザの [ポート] タブ [366](#)

発生属性 [358](#)

[プロパティ] タブ [357](#)

[ポート] タブ [356](#)

マッピングの例 [368](#)

例 [368](#)

レベル属性 [362](#), [367](#)

は

パーサー

データトランスフォーメーション [498](#)

データトランスフォーメーションの出力 [507](#)

配布

エクスターナルプロシージャ [184](#)

カスタムトランスフォーメーションプロシージャ [63](#)

パイプライン

共有体トランスフォーメーションとの結合 [493](#)

パイプラインのパーティション化

HTTP トランスフォーメーション [205](#)

カスタムトランスフォーメーション [67](#)

パイプラインノーマライザトランスフォーメーション

作成 [367](#)

説明 [364](#)

[ノーマライザ] タブ [366](#)

[ポート] タブ [366](#)

パイプラインルックアップ

再利用可能なトランスフォーメーション [320](#)

パイプラインルックアップ (続く)
 再利用不可能なトランスフォーメーション [320](#)
 セッションプロパティの設定 [308](#)
 説明 [292](#)
 マッピング [295](#)
 マッピングの例 [295](#)
 ルックアップトランスフォーメーションの例 [295](#)
 ルックアップトランスフォーメーションプロパティ [295](#)
 配列ベース関数
 概要 [120](#)
 行ストラテジ [125](#)
 行の数 [122](#)
 行有効検証 [123](#)
 最大行数 [121](#)
 データ操作 [123](#)
 入力エラー行の設定 [127](#)
 パススルーポート
 SQL トランスフォーメーションへの追加 [435](#)
 デフォルト値 [39](#)
 パッシブトランスフォーメーション
 Java [225](#), [226](#)
 SQL トランスフォーメーションの設定 [436](#)
 概要 [25](#)
 式 [161](#)
 シーケンスジェネレータ [384](#)
 ストアードプロシージャ [465](#)
 ルックアップ [292](#)
 パッシブモード
 SQL トランスフォーメーション [437](#)
 発生属性
 ノーマライザトランスフォーメーション [358](#)
 パーティション化可能 (プロパティ)
 HTTP トランスフォーメーション [205](#)
 Java トランスフォーメーション [230](#)
 エクスターナルプロシージャトランスフォーメーション [166](#)
 カスタムトランスフォーメーション [67](#)
 パーティション化されたパイプライン
 ソート済みデータの結合 [283](#)
 パーティション関連の関数
 説明 [197](#)
 パーティションごとに 1 つのスレッドを要求します (プロパティ)
 HTTP トランスフォーメーション [205](#)
 Java トランスフォーメーション [230](#)
 カスタムトランスフォーメーション [67](#)
 パフォーマンス
 静的データベース接続 [440](#)
 論理データベース接続 [440](#)
 変数を使用して改善する [36](#)
 パラメータアクセス関数
 64-bit [194](#)
 説明 [194](#)
 パラメータのバインド
 SQL トランスフォーメーションクエリ [432](#)
 パラメータのマスキング
 データマスキング [131](#)
 範囲
 数値のマスキング [143](#)
 ハンドル
 カスタムトランスフォーメーション [85](#)

ひ

比較において無視 (プロパティ)
 説明 [340](#)
 非集計関数
 例 [55](#)

非集計式
 概要 [57](#)
 日付の値
 ランダムデータマスキング [140](#)
 非ネイティブトランスフォーメーション
 概要 [25](#)
 非ユーザーコードのエラー
 Java トランスフォーメーション [243](#)

ふ

ファイル出力タイプ
 構造化されていないデータのトランスフォーメーション [506](#)
 ファイル入力タイプ
 構造化されていないデータのトランスフォーメーション [502](#)
 フィルタトランスフォーメーション
 開発のヒント [202](#)
 概要 [199](#)
 作成 [201](#)
 条件 [200](#)
 パフォーマンスのヒント [202](#)
 例 [199](#)
 複合キー
 シーケンスジェネレータトランスフォーメーションを使用した作成
 [388](#)
 複数グループ
 トランスフォーメーション [31](#)
 複数の行を返す
 ルックアップトランスフォーメーション [315](#)
 部分クエリ
 置換 [435](#)
 ブラー
 数値 [144](#)
 日付の値 [144](#)
 プライマリキー
 シーケンスジェネレータトランスフォーメーションを使用した作成
 [388](#)
 フラットファイルルックアップ
 説明 [294](#)
 ソート済み入力 [294](#)
 プール内の最大接続数 (プロパティ)
 SQL トランスフォーメーション [450](#)
 プログラム識別子 (プロパティ)
 エクスターナルプロシージャトランスフォーメーション [166](#)
 プロセス変数
 初期化プロパティ [191](#)
 ブロック
 ジョイナトランスフォーメーションにおける明細行 [287](#)
 プロパティ ID
 カスタムトランスフォーメーション [101](#)
 プロパティアクセス関数
 説明 [193](#)
 プロパティ関数
 カスタムトランスフォーメーション [100](#)

へ

ベース URL
 設定 [211](#)
 変数ポート
 概要 [36](#)

ほ

ポート

Group By [55](#)
HTTP トランスフォーメーション [207](#)
Java トランスフォーメーション [229](#)
アグリゲータトランスフォーメーション [53](#)
カスタムトランスフォーメーション [64](#)
共有体トランスフォーメーション [494](#)
構造化されていないデータのトランスフォーメーション [503](#)
作成 [29](#), [30](#)
シーケンスジェネレータトランスフォーメーション [385](#)
設定 [29](#), [30](#)
ソース修飾子 [424](#)
ソート済み [57](#), [424](#)
ソート済みポートオプション [424](#)
デフォルト値の概要 [39](#)
評価順 [38](#)
変数ポート [36](#)
ランクトランスフォーメーション [374](#)
ルータトランスフォーメーション [382](#)
ルックアップトランスフォーメーション [299](#)

ポート属性

概要 [66](#)
編集 [66](#)

ポート値

Java トランスフォーメーション [229](#)

ポートの依存性

カスタムトランスフォーメーション [65](#)

ポートの読み込み

構造化されていないデータのトランスフォーメーション [504](#)

ま

「マスキングプロパティ」タブ

データマスキングトランスフォーメーション [130](#)

マスキングルール

結果文字列の置換文字 [143](#)
ソース文字列の文字 [142](#)
適用 [141](#)
範囲 [143](#)
ブラー [144](#)
マスク形式 [141](#)

マスク形式

文字列値のマスキング [141](#)

マスタ行

キャッシュ [288](#)
ソート済みジョイナトランスフォーメーションでの処理 [288](#)
未ソートのジョイナトランスフォーメーションでの処理 [287](#)

マスタでの NULL の順序付け（プロパティ）

ジョイナトランスフォーメーション [278](#)

マスタのソート順（プロパティ）

ジョイナトランスフォーメーション [278](#)

マップ

データトランスフォーメーション [498](#)

マッピング

COBOL ソースの追加 [360](#)
エクスターナルプロシージャトランスフォーメーションの使用 [173](#)
行に対する更新のフラグ設定 [511](#)
再利用可能なトランスフォーメーションの追加 [49](#)
再利用可能なトランスフォーメーションの変更 [50](#)
ストアドプロシージャからの影響 [467](#)
接続されたストアドプロシージャトランスフォーメーションの設定 [477](#)
接続されていないストアドプロシージャトランスフォーメーションの設定 [477](#)
トランスフォーメーションの追加 [28](#)

マッピング (続く)

ルータトランスフォーメーションの使用 [383](#)

ルックアップのコンポーネント [299](#)

マッピングの失敗

Java トランスフォーメーション [247](#)

マッピングパラメータ

ソース修飾子トランスフォーメーション [409](#)

ルックアップ SQL オーバーライド [308](#)

マッピング変数

再利用可能なトランスフォーメーション [48](#)

ソース修飾子トランスフォーメーション [409](#)

ルックアップ SQL オーバーライド [308](#)

み

未ソートのジョイナトランスフォーメーション

マスタ行の処理 [287](#)

明細行の処理 [287](#)

む

無効なトランザクション制御トランスフォーメーション

定義 [489](#)

め

明細行

ジョイナトランスフォーメーションでのブロック [287](#)

ソート済みジョイナトランスフォーメーションでの処理 [288](#)

未ソートのジョイナトランスフォーメーションでの処理 [287](#)

メソッド

Java トランスフォーメーション [237](#), [238](#)

Java トランスフォーメーション API [244](#)

メタデータエクステンション

カスタムトランスフォーメーション内 [73](#)

メッセージのトレース

エクスターナルプロシージャ用 [187](#)

メモリ管理

エクスターナルプロシージャ用 [187](#)

も

モジュール（プロパティ）

エクスターナルプロシージャトランスフォーメーション [166](#)

文字列

リンク付け [374](#)

文字列値

カスタムデータマスキング [140](#)

キーデータマスキング [133](#)

文字列の置換

SQL トランスフォーメーションクエリ [434](#)

戻り値

エクスターナルプロシージャから [186](#)

ストアドプロシージャトランスフォーメーション [466](#)

ルックアップトランスフォーメーション [317](#)

戻りポート

ルックアップトランスフォーメーション [299](#), [317](#)

ゆ

有効なトランザクション制御トランスフォーメーション

定義 [489](#)

ユーザーコードのエラー
Java トランスフォーメーション [242](#)
ユーザー定義関数
Java 式での使用 [258](#)
ユーザー定義グループ
ルータトランスフォーメーション [380](#)
ユーザー定義ジョイン
入力 [416](#)
ユーザー定義メソッド
Java トランスフォーメーション [236-238](#)

よ

呼び出し
Java 式の API メソッド [268](#)
呼び出しテキスト
ストアドプロシージャ、入力 [480](#)
呼び出しテキスト (プロパティ)
ストアドプロシージャトランスフォーメーション [475](#)
予約語
resword.txt [410](#)
SQL の生成 [410](#)
ルックアップクエリ [309](#)

ら

ライトアウトジョイン
構文 [421](#)
作成 [421](#)
ライブラリ
C++エクスターナルプロシージャ用 [171](#)
Informatica エクスターナルプロシージャ用 [181](#)
VB エクスターナルプロシージャ用 [175](#)
ラッパークラス
既存ライブラリ用または既存関数用 [187](#)
ランク付け
データグループ [375](#)
文字列値 [374](#)
ランクトランスフォーメーション
RANKINDEX ポート [375](#)
オプション [374](#)
概要 [373](#)
グループの定義 [375](#)
作成 [376](#)
ポート [374](#)
変数の使用 [36](#)
ランダムマスキング
数値 [139](#)
日付値のマスキング [140](#)
文字列値のマスキング [140](#)

り

リアルタイムデータ
ジョイナトランスフォーメーション [288](#)
ジョイナトランスフォーメーションによる処理 [290](#)
リセット
シーケンスジェネレータトランスフォーメーション [396](#)
リポジトリ
COM エクスターナルプロシージャ [172](#)
COM プロシージャの登録 [172](#)
リレーショナルデータベース
結合 [277](#)
リレーショナル階層
構造化されていないデータのトランスフォーメーション [505](#)

る

ルータトランスフォーメーション
概要 [378](#)
グループ [380](#)
グループフィルタ条件 [380](#)
作成 [383](#)
ノーマライザデータのフィルタリング [368](#)
ポート [382](#)
マッピングでの接続 [383](#)
例 [380](#)
ルール
デフォルト値 [45](#)
ルックアップ SQL オーバーライド
動的キャッシュ [341](#)
ルックアップキャッシュ
[ルックアップキャッシュパーシステント] プロパティ [301](#)
[ルックアップのデータキャッシュサイズ] プロパティ [301](#)
ORDER BY のオーバーライド [309](#)
永続 [326](#)
概要 [322](#)
キャッシュの事前作成 [301](#)
キャッシュの有効化 [301](#)
共有 [328](#)
最初および最後の値の処理 [313](#)
静的 [328](#)
定義 [322](#)
データベースからの再キャッシュ [324](#)
動的 [336](#)
動的、エラーしきい値 [347](#)
動的、ターゲットとの同期 [347](#)
動的、ルックアップソースとの同期 [351](#)
名前付き永続キャッシュ [328](#)
名前なしキャッシュを使用したパーティション化のガイドライン
[329](#)
名前なしルックアップの共有 [329](#)
ロードの拒否 [347](#)
ルックアップキャッシュが有効 (プロパティ)
説明 [301](#)
ルックアップキャッシュディレクトリ名 (プロパティ)
説明 [301](#)
ルックアップキャッシュの再初期化
「データベースからの再キャッシュ」を参照 [324](#)
ルックアップキャッシュの事前作成
ルックアッププロパティ [301](#)
ルックアップキャッシュパーシステント (プロパティ)
説明 [301](#)
ルックアップクエリ
ORDER BY [308](#)
Sybase ORDER BY の制限 [309](#)
オーバーライド [308](#)
説明 [308](#)
デフォルトクエリ [308](#)
予約語 [309](#)
ルックアップ条件
概要 [313](#)
設定 [312](#)
定義 [300](#)
データマスキングトランスフォーメーション [136](#)
ルックアップソースフィルタ
ルックアップの制限 [311](#), [312](#)
説明 [301](#)
ルックアップテーブル
インデックス [299](#), [320](#)
名前 [301](#)
ルックアップトランスフォーメーション
NULL を無視 [339](#)
アップデートストラテジの組み合わせ [513](#)

ルックアップトランスフォーメーション (続く)

- 永続キャッシュ [326](#)
- エラーしきい値 [347](#)
- 概要 [292](#)
- カスタムクエリの入力 [310](#)
- 関連入力ポート [338](#)
- キャッシュ [322](#)
- キャッシュの共有 [328](#)
- 再利用可能なパイプライン [320](#)
- 再利用不可能なパイプライン [320](#)
- 式 [317](#)
- シーケンス ID [338](#)
- 出力ポート [299](#)
- 条件 [312](#), [316](#)
- 接続 [296](#), [297](#)
- 接続されたルックアップの作成 [318](#)
- 接続されていない [316](#)
- データベースからの再キャッシュ [324](#)
- デフォルトクエリ [308](#)
- デフォルトのクエリのオーバーライド [308](#)
- 動的キャッシュ [336](#)
- 動的キャッシュとターゲットとの同期 [347](#)
- 名前付き永続キャッシュ [328](#)
- 入力ポート [299](#)
- のコンポーネント [299](#)
- パイプラインルックアップセッションプロパティの設定 [308](#)
- パイプラインルックアップの説明 [292](#)
- パイプラインルックアップの例 [295](#)
- パフォーマンスのヒント [320](#)
- 比較において無視 [340](#)
- 複数の一致 [313](#)
- 複数の行を返す [315](#)
- フラットファイルルックアップ [292](#), [294](#)
- プロパティ [301](#)
- ポート [299](#)
- マッピングでのパイプラインの使用 [295](#)
- マッピングパラメータとマッピング変数 [308](#)
- 未接続 [296](#), [298](#)
- ルックアップソース [292](#)
- ルックアップソースとの動的キャッシュの同期 [351](#)
- ルックアップポート [299](#)

ルックアップトランスフォーメーション (続く)

- ロードの拒否 [347](#)
- 日時ポートのサブ秒の精度 [301](#)
- 戻り値 [317](#)
- ルックアッププロパティ
- セッションでの設定 [306](#)
- ルックアップポート
- 説明 [299](#)

れ

例外

- エクスターナルプロシージャから [187](#)

レジリエンス

- SQL トランスフォーメーションデータベース [443](#)

レフトアウタージョイン

- 構文 [419](#)

- 作成 [419](#)

レベル属性

- VSAM ノーマライゼイトランスフォーメーション [362](#)

- パイプラインノーマライゼイトランスフォーメーション [367](#)

連続したキャッシュ

- キャッシュを参照 [325](#)

- 「キャッシュ」を参照 [324](#)

ろ

ローカル変数

- 概要 [36](#)

ロード順

- ソース修飾子 [408](#)

ログ

- Java トランスフォーメーション [251](#), [252](#)

ロールバック行の生成

- Java トランスフォーメーション [253](#)

論理データベース接続

- SQL トランスフォーメーションへの受け渡し [437](#)

- パフォーマンスに関する考慮事項 [440](#)