



Informatica® PowerCenter
10.5

변환 가이드

이 소프트웨어와 설명서는 사용 및 공개에 대한 제한 사항이 포함되어 있는 별도의 사용권 계약에 따라서는 제공됩니다. 본 문서의 어떤 부분도 Informatica LLC의 사전 통지 없이 어떠한 형태나 수단(전자적, 사진 복사, 녹음 등)으로 복제되거나 전송될 수 없습니다.

Informatica, Informatica 로고 및 PowerCenter는 미국과 전 세계 여러 관할 국가에서 Informatica LLC의 상표 또는 등록 상표입니다. Informatica 상표의 현재 목록은 <https://www.informatica.com/trademarks.html>에서 확인할 수 있습니다. 다른 회사 및 제품명은 해당 소유자의 상표 또는 등록 상표일 수 있습니다.

이 소프트웨어 및/또는 설명서 중 일부는 타사 저작권의 적용을 받으며, 이에 국한되지 않습니다. 저작권 DataDirect Technologies. 모든 권리 보유. 저작권 (c) Sun Microsystems. 모든 권리 보유. 저작권 (c) RSA Security Inc. 모든 권리 보유. 저작권 (c) Ordinal Technology Corp. 모든 권리 보유. 저작권 (c) Aandacht c.v. 모든 권리 보유. 저작권 Genivia, Inc. 모든 권리 보유. 저작권 Isomorphic Software. 모든 권리 보유. 저작권 (c) Meta Integration Technology, Inc. 모든 권리 보유. 저작권 (c) Intalio. 모든 권리 보유. 저작권 (c) Oracle. 모든 권리 보유. 저작권 (c) Adobe Systems Incorporated. 모든 권리 보유. 저작권 (c) DataArt, Inc. 모든 권리 보유. 저작권 (c) ComponentSource. 모든 권리 보유. 저작권 (c) Microsoft Corporation. 모든 권리 보유. 저작권 (c) Rogue Wave Software, Inc. 모든 권리 보유. 저작권 (c) Teradata Corporation. 모든 권리 보유. 저작권 (c) Yahoo! Inc. 모든 권리 보유. 저작권 (c) Glyph & Cog, LLC. 모든 권리 보유. 저작권 (c) Thinkmap, Inc. 모든 권리 보유. 저작권 (c) Clearpace Software Limited. 모든 권리 보유. 저작권 (c) Information Builders, Inc. 모든 권리 보유. 저작권 (c) OSS Nokalva, Inc. 모든 권리 보유. 저작권 Edifecs, Inc. 모든 권리 보유. 저작권 Cleo Communications, Inc. 모든 권리 보유. 저작권 (c) International Organization for Standardization 1986. 모든 권리 보유. 저작권 (c) ej-technologies GmbH. 모든 권리 보유. 저작권 (c) Jaspersoft Corporation. 모든 권리 보유. 저작권 (c) International Business Machines Corporation. 모든 권리 보유. 저작권 (c) yWorks GmbH. 모든 권리 보유. 저작권 (c) Lucent Technologies. 모든 권리 보유. 저작권 (c) University of Toronto. 모든 권리 보유. 저작권 (c) Daniel Veillard. 모든 권리 보유. 저작권 (c) Unicode, Inc. 저작권 IBM Corp. 모든 권리 보유. 저작권 (c) MicroQuill Software Publishing, Inc. 모든 권리 보유. 저작권 (c) PassMark Software Pty Ltd. 모든 권리 보유. 저작권 (c) LogiXML, Inc. 모든 권리 보유. 저작권 (c) 2003-2010 Lorenzi Davide, 모든 권리 보유. 저작권 (c) Red Hat, Inc. 모든 권리 보유. 저작권 (c) The Board of Trustees of the Leland Stanford Junior University. 모든 권리 보유. 저작권 (c) EMC Corporation. 모든 권리 보유. 저작권 (c) Flexera Software. 모든 권리 보유. 저작권 (c) Jinfonet Software. 모든 권리 보유. 저작권 (c) Apple Inc. 모든 권리 보유. 저작권 (c) Telerik Inc. 모든 권리 보유. 저작권 (c) BEA Systems. 모든 권리 보유. 저작권 (c) PDFlib GmbH. 모든 권리 보유. 저작권 (c) Orientation in Objects GmbH. 모든 권리 보유. 저작권 (c) Tanuki Software, Ltd. 모든 권리 보유. 저작권 (c) Ricebridge. 모든 권리 보유. 저작권 (c) Sencha, Inc. 모든 권리 보유. 저작권 (c) Scalable Systems, Inc. 모든 권리 보유. 저작권 (c) jQWidgets. 모든 권리 보유. 저작권 (c) Tableau Software, Inc. 모든 권리 보유. 저작권 (c) MaxMind, Inc. 모든 권리 보유. 저작권 (c) TMat Software s.r.o. 모든 권리 보유. 저작권 (c) MapR Technologies Inc. 모든 권리 보유. 저작권 (c) Amazon Corporate LLC. 모든 권리 보유. 저작권 (c) Highsoft. 모든 권리 보유. 저작권 (c) Python Software Foundation. 모든 권리 보유. 저작권 (c) BeOpen.com. 모든 권리 보유. 저작권 (c) CNRI. 모든 권리 보유.

이 제품에는 Apache Software Foundation(<http://www.apache.org/>)에서 개발한 소프트웨어 및/또는 Apache License의 다양한 버전("라이선스")에 따라 사용이 허가된 기타 소프트웨어가 포함되어 있습니다. <http://www.apache.org/licenses/>에서 이러한 라이선스의 복사본을 얻을 수 있습니다. 관련 법규 또는 서면 동의에 명시되어 있지 않은 경우, 이러한 라이선스에 따라 배포되는 소프트웨어는 어떠한 종류의 명시적이거나 묵시적인 보증 또는 조건 없이 "있는 그대로" 배포됩니다. 사용 권한에 대한 특정 언어별 라이선스 및 해당 라이선스에 따른 제한 사항을 참조하십시오.

이 제품에는 Mozilla(<http://www.mozilla.org/>)에서 개발한 소프트웨어, JBoss Group, LLC(저작권 JBoss Group, LLC, 모든 권리 보유.)가 저작권을 소유한 소프트웨어, Bruno Lowagie and Paulo Soares(저작권 (c) 1999-2006 by Bruno Lowagie and Paulo Soares)가 저작권을 소유한 소프트웨어 및 GNU Lesser General Public License Agreement(<http://www.gnu.org/licenses/lgpl.html>)의 다양한 버전에 따라 라이선스가 부여된 기타 소프트웨어가 포함되어 있습니다. 해당 정보는 상품성 및 특정 목적에의 적합성에 대한 묵시적 보증을 포함하여 이에 국한되지 않는 어떠한 종류의 명시적이거나 묵시적인 보증 없이 "있는 그대로" 제공되며, Informatica는 어떠한 책임도 지지 않습니다.

이 제품에는 Douglas C. Schmidt와 Washington University, University of California, Irvine, Vanderbilt University의 연구팀(저작권 ((c)) 1993-2006, 모든 권리 보유.)이 저작권을 소유한 ACE(TM) 및 TAO(TM) 소프트웨어가 포함되어 있습니다.

이 제품에는 OpenSSL Toolkit(저작권 The OpenSSL Project. 모든 권리 보유)에서 사용할 수 있도록 OpenSSL Project에서 개발한 소프트웨어가 포함되어 있으며 이 소프트웨어의 재배포는 <http://www.openssl.org> 및 <http://www.openssl.org/source/license.html>의 조항에 따라 변경될 수 있습니다.

이 제품에는 Curl 소프트웨어(저작권 1996-2013, Daniel Stenberg, <daniel@haxx.se>. 모든 권리 보유.)가 포함되어 있습니다. 이 소프트웨어와 관련된 사용 권한 및 제한 사항은 <http://curl.haxx.se/docs/copyright.html>에 명시된 조항에 따라 변경될 수 있습니다. 위와 같은 저작권 고지 및 이러한 허가 고지가 모든 제품에 표시되어 있는 경우 목적 및 사용료 유무에 관계없이 이 소프트웨어를 사용, 복사, 수정 및 배포할 수 있는 사용 권한이 부여됩니다.

이 제품에는 MetaStuff, Ltd(저작권 2001-2005 ((C)) MetaStuff, Ltd. 모든 권리 보유.)가 저작권을 소유한 소프트웨어가 포함되어 있습니다. 이 소프트웨어와 관련된 사용 권한 및 제한은 <http://www.dom4j.org/license.html>의 조항에 따라 변경될 수 있습니다.

이 제품에는 Per Bothner(저작권 (c) 1996-2006 Per Bothner. 모든 권리 보유.)가 저작권을 소유한 소프트웨어가 포함되어 있습니다. 이러한 정보를 사용할 수 있는 권리는 <http://www.gnu.org/software/kawa/Software-License.html>의 라이선스에 설명되어 있습니다.

이 제품에는 OSSP UUID 소프트웨어(저작권 (c) 2002 Ralf S. Engelschall, 저작권 (c) 2002 The OSSP Project 저작권 (c) 2002 Cable & Wireless Deutschland)가 포함되어 있습니다. 이 소프트웨어와 관련된 사용 권한 및 제한은 <http://www.opensource.org/licenses/mit-license.php>의 조항에 따라 변경될 수 있습니다.

이 제품에는 Boost(<http://www.boost.org/>)에서 개발하거나 Boost 소프트웨어 라이선스에 따라 개발된 소프트웨어가 포함되어 있습니다. 이 소프트웨어와 관련된 사용 권한 및 제한은 http://www.boost.org/LICENSE_1_0.txt의 조항에 따라 변경될 수 있습니다.

이 제품에는 University of Cambridge(저작권 (c) 1997-2007 University of Cambridge)가 저작권을 소유한 소프트웨어가 포함되어 있습니다. 이 소프트웨어와 관련된 사용 권한 및 제한은 <http://www.pcre.org/license.txt>의 조항에 따라 변경될 수 있습니다.

이 제품에는 Eclipse Foundation(저작권 (c) 2007 The Eclipse Foundation. 모든 권리 보유.)이 저작권을 소유한 소프트웨어가 포함되어 있습니다. 이 소프트웨어와 관련된 사용 권한 및 제한은 <http://www.eclipse.org/org/documents/epl-v10.php> 및 <http://www.eclipse.org/org/documents/edl-v10.php>의 조항에 따라 변경될 수 있습니다.

이 제품에는 <http://www.tcl.tk/software/tcltk/license.html>, <http://www.bosrup.com/web/overlib?License>, <http://www.stlport.org/doc/license.html>, <http://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/licenses-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, <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/IODBCLicense>, <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.schneier.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> 및 <https://sourceforge.net/p/xmlunit/code/HEAD/tree/trunk/LICENSE.txt>.

이 제품에는 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>), 새 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>) 및 Initial Developer's Public License 버전 1.0(<http://www.firebirdsql.org/en/initial-developer-s-public-license-version-1-0/>)에 따라 라이선스가 부여된 소프트웨어가 포함되어 있습니다.

이 제품에는 Joe Walnes와 XStream Committers(저작권 (c) 2003-2006 Joe Walnes, 2006-2007 XStream Committers. 모든 권리 보유.)가 저작권을 소유한 소프트웨어가 포함되어 있습니다. 이 소프트웨어와 관련된 사용 권한 및 제한은 <http://xstream.codehaus.org/license.html>의 조항에 따라 변경될 수 있습니다. 이 제품에는 Indiana University Extreme! Lab에서 개발한 소프트웨어가 포함되어 있습니다. 자세한 내용을 확인하려면 <http://www.extreme.indiana.edu/>를 방문하십시오.

이 제품에는 Frank Balluffi 및 Markus Moeller(저작권 (c) 2013 Frank Balluffi and Markus Moeller. 모든 권리 보유.)가 저작권을 소유한 소프트웨어가 포함되어 있습니다. 이 소프트웨어와 관련된 사용 권한 및 제한 사항은 MIT license에 명시된 조항에 따라 변경될 수 있습니다.

<https://www.informatica.com/legal/patents.html>에서 특허를 참조하십시오.

고지 사항: Informatica LLC는 비침해, 상품성 또는 특정 목적에 따른 사용에 대한 묵시적 보증을 포함하여 이에 국한되지 않는 어떠한 종류의 명시적이거나 묵시적인 보증 없이 이 문서를 "있는 그대로" 제공합니다. Informatica LLC는 이 소프트웨어나 문서에 오류가 없음을 보장하지 않습니다. 이 소프트웨어나 설명서에 제공된 정보에는 기술적 오류나 인쇄 오류가 있을 수 있습니다. 이 소프트웨어 및 설명서의 정보는 언제든지 예고 없이 변경될 수 있습니다.

고지 사항

이 Informatica 제품(이하 "소프트웨어")에는 Progress Software Corporation(이하 "DataDirect")의 운영 회사인 DataDirect Technologies의 특정 드라이버(이하 "DataDirect Drivers")가 포함되어 있습니다. 이러한 드라이버에는 다음 조건이 적용됩니다.

1. DataDirect Drivers는 상품성, 특정 목적에의 적합성 및 비침해에 대한 묵시적 보증을 포함하여 이에 국한되지 않는 어떠한 종류의 명시적이거나 묵시적인 보증 없이 "있는 그대로" 제공됩니다.
2. DataDirect 또는 그 타사 공급자는 손해의 발생 가능성을 사전에 알고 있었는지 여부에 관계없이 ODBC 드라이버의 사용으로 발생하는 직접, 간접, 부수적, 특별, 결과적 또는 기타 손해에 대해 어떠한 경우에도 최종 사용자에게 책임을 지지 않습니다. 이러한 제한 사항은 계약 위반, 보증 불이행, 과실, 무과실 책임, 허위 진술 및 기타 불법 행위를 포함하여 이에 국한되지 않는 모든 소송 사유에 적용됩니다.

이 설명서의 정보는 예고 없이 변경될 수 있습니다. 이 문서에서 문제가 발견되는 경우 infa_documentation@informatica.com으로 보고해 주십시오.

Informatica 제품은 제품이 제공될 당시의 계약 조건에 따라 보증됩니다. Informatica는 상품성과 특정 목적에의 적합성에 대한 보증 그리고 비침해에 대한 보증 또는 조건을 포함하여 어떠한 종류의 명시적이거나 묵시적인 보증 없이 이 문서의 정보를 "있는 그대로" 제공합니다.

발행 날짜: 2021-10-13

목차

서문	23
Informatica 리소스	23
Informatica 네트워크	23
Informatica 기술 자료	23
Informatica 설명서	24
Informatica Product Availability Matrix	24
Informatica Velocity	24
Informatica Marketplace	24
Informatica 글로벌 고객 지원 센터	24
장 1: 변환 작업	25
변환 개요	25
활성 변환	25
수동 변환	26
연결되지 않은 변환	26
원시 변환 및 비원시 변환	26
변환 설명	26
변환 작성	29
변환 구성	29
변환 이름 바꾸기	30
변환 포트	30
포트 생성	30
포트 구성	30
포트 연결	30
다중 그룹 변환	31
식 작업	31
식 편집기 사용	33
식 평가	34
식 평가	35
식 평가 제한 사항	35
로컬 변수	36
임시로 데이터 저장 및 복잡한 식 간소화	36
행의 값 저장	36
저장 프로시저의 값 캡처	37
변수 포트 구성 지침	37
포트의 기본값	38
사용자 정의 기본값	39
사용자 정의 기본 입력값	40
사용자 정의 기본 출력값	42

기본값에 대한 일반 규칙.....	43
기본값 유효성 검사.....	44
변환의 추적 수준 구성.....	44
재사용 가능 변환.....	44
인스턴스 및 상속된 변경 내용.....	45
식의 매핑 변수.....	45
재사용 가능 변환 작성.....	45
재사용 불가능 변환 승격.....	46
재사용 가능 변환의 재사용 불가능 인스턴스 작성.....	46
매핑에 재사용 가능 변환 추가.....	46
재사용 가능 변환 수정.....	47

장 2: 집계 변환..... 48

집계 변환 개요.....	48
집계 변환의 구성 요소.....	48
집계 변환 속성 구성.....	49
집계 변환 포트 구성.....	49
집계 캐시 구성.....	50
집계 식.....	50
집계 함수.....	50
중첩 집계 함수.....	51
조건절.....	51
비집계 함수.....	51
집계 함수의 Null 값.....	51
그룹 기준 포트.....	51
비집계 식.....	53
기본값.....	53
정렬된 입력 사용.....	53
정렬된 입력 조건.....	54
데이터 정렬.....	54
집계 변환 작성.....	55
집계 변환에 대한 팁.....	55
집계 변환 문제 해결.....	56

장 3: 사용자 지정 변환..... 57

사용자 지정 변환 개요.....	57
사용자 지정 변환을 기반으로 작성되는 변환을 사용한 작업.....	58
코드 페이지 호환성.....	58
사용자 지정 변환 프로시저 배포.....	59
사용자 지정 변환 작성.....	59
사용자 지정 변환에 대한 규칙 및 지침.....	59
사용자 지정 변환 구성 요소.....	60

그룹 및 포트 작업.....	60
그룹 및 포트 작성.....	60
그룹 및 포트 편집.....	60
포트 관계 정의.....	61
포트 특성 작업.....	61
포트 특성 값 편집.....	62
사용자 지정 변환 속성.....	62
업데이트 전략 설정.....	64
스레드별 프로시저 코드 작업.....	64
트랜잭션 제어 작업.....	64
변환 범위.....	65
트랜잭션 생성.....	65
트랜잭션 경계 작업.....	65
입력 데이터 차단.....	66
데이터를 차단하는 프로시저 코드 작성.....	66
사용자 지정 변환을 차단 변환으로 구성.....	67
사용자 지정 변환이 포함된 매핑의 유효성 검사.....	67
프로시저 속성 작업.....	68
사용자 지정 변환 프로시저 작성.....	68
1단계. 사용자 지정 변환 작성.....	69
2단계. C 파일 생성.....	71
3단계. 변환 논리를 포함하여 코드 작성.....	71
4단계. 모듈 빌드.....	76
5단계. 매핑 생성.....	78
6단계. 워크플로우의 세션 실행.....	78
장 4: 사용자 지정 변환 함수.....	79
사용자 지정 변환 함수 개요.....	79
핸들 작업.....	79
함수 참조.....	80
행 작업.....	83
행 기반 및 배열 기반 데이터 액세스 모드에 대한 규칙 및 지침.....	84
생성된 함수.....	84
초기화 함수.....	85
알림 함수.....	86
초기화 취소 함수.....	88
API 함수.....	89
데이터 액세스 모드 설정 함수.....	90
탐색 함수.....	91
속성 함수.....	93
데이터 유형 재바인딩 함수.....	100
데이터 처리 함수(행 기반 모드).....	102

통과 포트 설정 함수	104
출력 알림 함수	105
데이터 경계 출력 알림 함수	105
오류 함수	106
세션 로그 메시지 함수	106
오류 개수 증분 함수	107
종료됨 함수	107
차단 함수	108
포인터 함수	109
문자열 모드 변경 함수	109
데이터 코드 페이지 설정 함수	110
행 전략 함수(행 기반 모드)	110
기본 행 전략 변경 함수	111
배열 기반 API 함수	112
최대 행 수 함수	112
행 수 함수	114
올바른 행임 함수	114
데이터 처리 함수(배열 기반 모드)	115
행 전략 함수(배열 기반 모드)	117
입력 오류 행 설정 함수	118

장 5: 데이터 마스킹 변환 121

데이터 마스킹 변환	121
마스킹 속성	122
로컬	122
마스킹 유형	122
반복 가능 출력	123
시드	123
매핑 매개 변수	123
연결된 O/P	124
키 마스킹	124
문자열 값 마스킹	124
숫자 값 마스킹	125
날짜/시간 값 마스킹	126
대체 마스킹	126
사전	126
저장소 테이블	127
대체 마스킹 속성	128
관계형 사전	128
연결 요구 사항	129
대체 마스킹에 대한 규칙 및 지침	129
종속 마스킹	129

중속 마스킹 예제.....	130
반복 가능 중속 마스킹.....	131
무작위 마스킹.....	131
숫자 값 마스킹.....	131
문자열 값 마스킹.....	131
날짜 값 마스킹.....	132
마스킹 규칙 적용.....	132
마스크 형식.....	133
소스 문자열 문자.....	134
결과 문자열 대체 문자.....	134
범위.....	135
블러팅.....	135
식 마스킹.....	136
반복 가능한 식 마스킹.....	136
식 마스킹에 대한 규칙 및 지침.....	138
형식 유지 암호화.....	138
특수 마스크 형식.....	139
사회 보장 번호 마스킹.....	139
사회 보장 번호 형식.....	140
지역 번호 요구 사항.....	140
반복 가능한 사회 보장 번호 마스킹.....	140
신용 카드 번호 마스킹.....	140
전화 번호 마스킹.....	141
전자 메일 주소 마스킹.....	141
고급 전자 메일 마스킹.....	141
사회 보험 번호 마스킹.....	143
SIN 시작 자릿수.....	143
반복 가능 SIN 번호.....	143
IP 주소 마스킹.....	143
URL 주소 마스킹.....	143
기본값 파일.....	144
데이터 마스킹 변환 세션 속성.....	144
데이터 마스킹 변환에 대한 규칙 및 지침.....	145
장 6: 데이터 마스킹 예제.....	146
이름 및 주소 조회 파일.....	146
조회 변환을 사용하여 데이터 대체.....	146
식 변환을 사용하여 데이터 마스킹.....	149
장 7: 식 변환.....	152
식 변환 개요.....	152
식 변환 구성 요소.....	152

포트 구성.	153
값 계산.	153
식 변환 작성.	153
장 8: 외부 프로시저 변환.	155
외부 프로시저 변환 개요.	155
코드 페이지 호환성.	155
외부 프로시저와 외부 프로시저 변환.	156
외부 프로시저 변환 속성.	156
COM과 Informatica 외부 프로시저.	156
BankSoft 예제.	157
외부 프로시저 변환 속성 구성.	157
COM 프로시저 개발.	159
COM 프로시저를 작성하기 위한 단계.	159
COM 외부 프로시저 서버 유형.	159
Visual C++를 사용하여 COM 프로시저 개발.	159
Visual Basic을 사용하여 COM 프로시저 개발.	164
Informatica 외부 프로시저 개발.	166
1단계. 외부 프로시저 변환 작성.	166
2단계. C++ 파일 생성.	168
3단계. 구현을 포함하여 메서드 스텝 작성.	170
4단계. 모듈 작성.	171
5단계. 매핑 생성.	172
6단계. 세션 실행.	172
외부 프로시저 배포.	173
COM 프로시저 배포.	173
Informatica 모듈 배포.	174
개발 참고 사항.	175
COM 데이터 유형.	175
행 수준 프로시저.	175
프로시저의 반환 값.	176
프로시저 호출의 예외.	176
프로시저에 대한 메모리 관리.	176
기존 C/C++ 라이브러리 또는 VB 함수에 대한 래퍼 클래스.	176
오류 및 추적 메시지 생성.	177
연결되지 않은 외부 프로시저 변환.	178
COM 및 Informatica 모듈 초기화.	179
TX에서 배포 및 사용되는 다른 파일.	180
초기화 속성의 서비스 프로세스 변수.	181
외부 프로시저 인터페이스.	181
디스패치 함수.	181
외부 프로시저 함수.	182

속성 액세스 함수.....	182
매개 변수 액세스 함수.....	183
코드 페이지 액세스 함수.....	185
변환 이름 액세스 함수.....	185
프로시저 액세스 함수.....	186
파티션 관련 함수.....	186
추적 수준 함수.....	186
장 9: 필터 변환.....	187
필터 변환 개요.....	187
필터 변환 구성 요소.....	188
필터 변환 포트 구성.....	188
필터 조건.....	188
Null 값을 가진 행 필터링.....	189
필터 변환 작성 단계.....	189
필터 변환에 대한 팁.....	189
장 10: HTTP 변환.....	191
HTTP 변환 개요.....	191
인증.....	192
HTTP 서버에 연결.....	192
HTTP 변환 작성.....	192
속성 탭 구성.....	192
HTTP 탭 구성.....	193
메서드 선택.....	194
그룹 및 포트 구성.....	194
URL 구성.....	197
예.....	199
GET 예제.....	199
POST 예제.....	200
SIMPLE POST 예제.....	201
SIMPLE PATCH 예.....	202
SIMPLE PUT 예.....	203
SIMPLE DELETE 예.....	204
장 11: ID 확인 변환.....	206
ID 확인 변환 개요.....	206
변환 작성 및 구성.....	206
검색 서버 연결.....	206
시스템 및 검색 구성.....	207
보기 선택.....	208
ID 확인 변환 탭.....	209

그룹 및 포트.....	209
입력 그룹 및 포트.....	209
출력 그룹 및 포트.....	210

장 12: Java 변환..... 211

Java 변환 개요.....	211
Java 변환 정의 단계.....	212
활성 및 수동 Java 변환.....	212
데이터 유형 변환.....	212
Java 코드 탭 사용.....	213
포트 구성.....	214
그룹 및 포트 작성.....	214
기본 포트 값 설정.....	214
Java 변환 속성 구성.....	215
트랜잭션 제어 작업.....	217
업데이트 전략 설정.....	217
Java 코드 개발.....	218
Java 코드 조각 작성.....	218
Java 패키지 가져오기.....	218
도우미 코드 정의.....	219
입력 행에서 탭.....	220
데이터 끝에서 탭.....	220
트랜잭션 수신 시 탭.....	220
Java 코드를 사용하여 플랫폼 파일 구문 분석.....	221
Java 변환 설정 구성.....	221
클래스 경로 구성.....	221
많은 전체 자릿수 활성화.....	223
초 단위 이하 처리.....	223
Java 변환 컴파일.....	223
컴파일 오류 수정.....	224
컴파일 오류의 원인 찾기.....	224
컴파일 오류의 소스 식별.....	224

장 13: Java 변환 API 참조..... 226

Java 변환 API 메서드 개요.....	226
커밋.....	227
failSession.....	228
generateRow.....	228
getInRowType.....	229
getMetadata.....	229
incrementErrorCount.....	230
isNull.....	231

logError.	231
logInfo.	232
롤백.	232
setNull.	233
setOutRowType.	233
storeMetadata.	234

장 14: Java 식..... 235

Java 식 개요.	235
식 함수 유형.	235
식 정의 대화 상자를 사용하여 식 정의.	236
1단계. 함수 구성.	236
2단계. 식 작성 및 유효성 검사.	237
3단계. 식에 대한 Java 코드 생성.	237
식 정의 대화 상자를 사용한 식 작성 및 Java 코드 생성.	237
Java 식 템플릿.	237
단순 인터페이스 작업.	238
invokeJExpression.	238
단순 인터페이스 예제.	239
고급 인터페이스 작업.	239
고급 인터페이스를 사용하여 식 호출.	240
고급 인터페이스 작업에 대한 규칙 및 지침.	240
EDatatype 클래스.	240
JExprParamMetadata 클래스.	241
defineJExpression.	241
JExpression 클래스.	242
고급 인터페이스 예제.	243
JExpression 클래스 API 참조.	243
getBytes.	244
getDouble.	244
getInt.	244
getLong.	244
getResultDataType.	245
getResultMetadata.	245
getStringBuffer.	245
invoke.	245
isResultNull.	246

장 15: Java 변환 예제..... 247

Java 변환 예제 개요.	247
1단계. 매핑 가져오기.	248
2단계. 변환 작성 및 포트 구성.	248

3단계. Java 코드 입력.	249
패키지 가져오기 탭.	249
도우미 코드 탭.	249
입력 행에서 탭.	250
4단계. Java 코드 컴파일.	251
5단계. 세션 및 워크플로우 작성.	252
샘플 데이터.	252
장 16: 조이너 변환.	254
조이너 변환 개요.	254
조이너 변환 작업.	254
조이너 변환 속성.	255
조인 조건 정의.	256
조인 유형 정의.	257
일반 조인.	257
마스터 외부 조인.	258
세부 외부 조인.	258
전체 외부 조인.	259
정렬된 입력 사용.	259
정렬 순서 구성.	259
매핑에 변환 추가.	260
조이너 변환 구성.	260
조인 조건 정의.	260
단일 소스의 데이터 조인.	261
동일한 파이프라인의 분기 2개 조인.	261
동일한 소스의 인스턴스 2개 조인.	262
단일 소스의 데이터 조인에 대한 지침.	263
소스 파이프라인 차단.	263
정렬되지 않은 조이너 변환.	263
정렬된 조이너 변환.	263
트랜잭션 작업.	264
단일 파이프라인의 트랜잭션 경계 유지.	264
세부 파이프라인에서 트랜잭션 경계 유지.	265
두 파이프라인의 트랜잭션 경계 삭제.	265
조이너 변환 작성.	265
조이너 변환에 대한 팁.	266
장 17: 조회 변환.	267
조회 변환 개요.	267
조회 소스 유형.	268
관계형 조회.	268
플랫 파일 조회.	268

파이프라인 조회.....	269
연결된 조회 및 연결되지 않은 조회.....	271
연결된 조회.....	271
연결되지 않은 조회.....	272
조회 구성 요소.....	272
조회 소스.....	272
조회 포트.....	273
조회 속성.....	274
조회 조건.....	274
조회 속성.....	274
세션에서 조회 속성 구성.....	278
조회 쿼리.....	280
기본 조회 쿼리.....	280
조회 쿼리 재정의.....	280
캐시되지 않은 조회에 대한 SQL 재정의.....	282
조회 소스 필터.....	282
조회 조건.....	283
캐시되지 않거나 정적 캐시.....	284
동적 캐시.....	284
여러 일치 항목 처리.....	284
조회 캐시.....	285
여러 행 반환.....	286
여러 행 반환에 대한 규칙 및 지침.....	286
연결되지 않은 조회 변환 구성.....	286
1단계. 입력 포트 추가.....	287
2단계. 조회 조건 추가.....	287
3단계. 반환 값 지정.....	287
4단계. 식을 통해 조회 호출.....	288
데이터베이스 교착 상태 복원력.....	288
조회 변환 작성.....	289
재사용 가능 파이프라인 조회 변환 작성.....	290
재사용 불가능 파이프라인 조회 변환 작성.....	290
조회 변환에 대한 팁.....	290
장 18: 조회 캐시.....	292
조회 캐시 개요.....	292
캐시 비교.....	293
연결된 조회 캐시 빌드.....	294
순차 캐시.....	294
동시 캐시.....	295
지속형 조회 캐시 사용.....	295
비지속형 캐시 사용.....	295

지속형 캐시 사용.	295
조회 캐시 다시 빌드.	296
캐시되지 않은 조회 또는 정적 캐시 작업.	297
조회 캐시 공유.	297
명명되지 않은 조회 캐시 공유.	297
명명된 조회 캐시 공유.	299
조회 캐시에 대한 팁.	303
장 19: 동적 조회 캐시.	304
동적 조회 캐시 개요.	304
동적 조회 속성.	305
NewLookupRows.	306
연결된 식.	306
Null 값.	307
비교 시 포트 무시.	308
SQL 재정의.	308
조회 변환 값.	309
초기 캐시 값.	309
입력 값.	310
조회 값.	310
출력 값.	310
동적 조회 캐시 업데이트.	311
삽입 기타 항목 업데이트.	312
업데이트 기타 항목 삽입.	312
동적 조회를 사용하는 매핑.	313
업스트림 업데이트 전략 변환 구성.	313
다운스트림 변환 구성.	314
동적 조회 캐시를 사용하여 세션 구성.	315
조건부 동적 캐시 업데이트.	315
세션 처리.	315
조건부 동적 캐시 조회 구성.	316
식 결과로 동적 캐시 업데이트.	316
Null 식 값.	316
세션 처리.	317
동적 캐시 업데이트를 위한 식 구성.	317
조회 소스와 캐시 동기화.	317
NewLookupRow.	318
동적 캐시 동기화 구성.	318
동적 조회 캐시 예제.	318
동적 조회 캐시에 대한 규칙 및 지침.	319

장 20: 노멀라이저 변환.....	321
노멀라이저 변환 개요.....	321
노멀라이저 변환 구성 요소.....	322
포트 탭.....	322
속성 탭.....	323
노멀라이저 탭.....	323
노멀라이저 변환 생성된 키.....	325
생성된 키 값 저장.....	325
생성된 키 값 변경.....	325
VSAM 노멀라이저 변환.....	325
VSAM 노멀라이저 포트 탭.....	327
VSAM 노멀라이저 탭.....	328
VSAM 노멀라이저 변환 작성 단계.....	329
파이프라인 노멀라이저 변환.....	330
파이프라인 노멀라이저 포트 탭.....	331
파이프라인 노멀라이저 탭.....	332
파이프라인 노멀라이저 변환 작성 단계.....	333
매핑에서 노멀라이저 변환 사용.....	333
키 값 생성.....	335
노멀라이저 변환 문제 해결.....	336
 장 21: 순위 변환.....	 338
순위 변환 개요.....	338
문자열 값 순위 지정.....	339
순위 캐시.....	339
순위 변환 속성.....	339
순위 변환의 포트.....	339
순위 인덱스.....	340
그룹 정의.....	340
순위 변환 작성.....	341
 장 22: 라우터 변환.....	 343
라우터 변환 개요.....	343
그룹 작업.....	344
입력 그룹.....	344
출력 그룹.....	344
그룹 필터 조건 사용.....	345
그룹 추가.....	346
포트 작업.....	347
매핑에서 라우터 변환 연결.....	347
라우터 변환 작성.....	347

장 23: 시퀀스 생성기 변환.....	349
시퀀스 생성기 변환 개요.....	349
시퀀스 생성기 포트.....	349
NEXTVAL 포트.....	350
CURRVAL.....	352
시퀀스 생성기 변환 속성.....	353
시작 값.....	354
증분 범위.....	354
끝 값.....	355
값 범위 반복.....	355
현재 값.....	355
총 캐시된 값.....	355
재사용 불가능 시퀀스 생성기.....	356
재사용 가능 시퀀스 생성기.....	356
시퀀스 생성기 고급 속성.....	357
재설정.....	357
시퀀스 생성기 변환 작성.....	357
시퀀스 생성기 변환 - 비원시 환경.....	358
시퀀스 생성기 변환 - Blaze 엔진.....	358
시퀀스 생성기 변환 - Spark 엔진.....	358
 장 24: 분류기 변환.....	 359
분류기 변환 개요.....	359
데이터 정렬.....	359
분류기 변환 속성.....	360
분류기 캐시 크기.....	361
대/소문자 구분.....	361
작업 디렉터리.....	361
고유 출력 행.....	361
추적 수준.....	361
Null을 Low로 처리.....	362
변환 범위.....	362
분류기 변환 작성.....	362
 장 25: 소스 한정자 변환.....	 363
소스 한정자 변환 개요.....	363
변환 데이터 유형.....	364
대상 로드 순서.....	364
날짜/시간 값.....	364
매개 변수 및 변수.....	365
소스 한정자 변환 속성.....	365

기본 쿼리.....	366
기본 쿼리 보기.....	367
기본 쿼리 재정의.....	367
소스 데이터 조인.....	367
기본 조인.....	367
사용자 지정 조인.....	368
다른 유형 조인.....	368
키 관계 작성.....	369
SQL 쿼리 추가.....	369
사용자 정의 조인 입력.....	370
외부 조인 지원.....	371
Informatica 조인 구문.....	371
외부 조인 작성.....	376
일반적인 데이터베이스 구문 제한 사항.....	377
소스 필터 입력.....	378
정렬된 포트 사용.....	379
고유 항목 선택.....	379
세션에서 고유 항목 선택 재정의.....	380
사전 세션 및 사후 세션 SQL 명령 추가.....	380
소스 한정자 변환 작성.....	381
수동으로 소스 한정자 변환 작성.....	381
소스 한정자 변환 옵션 구성.....	381
소스 한정자 변환 문제 해결.....	381
장 26: SQL 변환.....	383
SQL 변환 개요.....	383
스크립트 모드.....	384
예제.....	384
스크립트 모드에 대한 규칙 및 지침.....	385
쿼리 모드.....	385
정적 SQL 쿼리 사용.....	385
동적 SQL 쿼리 사용.....	387
통과 포트 구성.....	389
수동 모드 구성.....	389
쿼리 모드에 대한 규칙 및 지침.....	390
데이터베이스에 연결.....	390
정적 데이터베이스 연결 사용.....	390
논리적 데이터베이스 연결 전달.....	391
전체 연결 정보 전달.....	391
데이터베이스 연결에 대한 규칙 및 지침.....	393
세션 처리.....	393
트랜잭션 제어.....	394

고가용성.....	394
SQL 쿼리 로그.....	396
입력 행 대 출력 행의 카디널리티.....	396
쿼리 문 처리.....	397
영향을 받는 행 수.....	397
최대 출력 행 개수.....	398
오류 행 이해.....	398
SQL 오류에 대해 계속.....	400
SQL 변환 속성.....	400
속성 탭.....	400
SQL 설정 탭.....	402
SQL 포트 탭.....	402
SQL 문.....	403
SQL 변환 작성.....	404

장 27: 매핑에서 SQL 변환 사용..... 406

SQL 변환 예제 개요.....	406
동적 업데이트 예제.....	406
소스 파일 정의.....	407
대상 정의 작성.....	408
데이터베이스 테이블 작성.....	408
식 변환 구성.....	409
SQL 변환 정의.....	409
세션 특성 구성.....	411
대상 데이터 결과.....	411
동적 연결 예제.....	411
소스 파일 정의.....	412
대상 정의 작성.....	412
데이터베이스 테이블 작성.....	413
데이터베이스 연결 작성.....	413
식 변환 구성.....	413
SQL 변환 정의.....	414
세션 특성 구성.....	415
대상 데이터 결과.....	415

장 28: 저장 프로시저 변환..... 416

저장 프로시저 변환 개요.....	416
입력 및 출력 데이터.....	417
연결됨 및 연결되지 않음.....	418
저장 프로시저 실행 시기 지정.....	418
매핑에서 저장 프로시저 사용.....	419
저장 프로시저 작성.....	420

샘플 저장 프로시저.....	420
저장 프로시저 변환 작성.....	422
저장 프로시저 가져오기.....	422
수동으로 저장 프로시저 변환 작성.....	423
저장 프로시저 설정 옵션.....	424
저장 프로시저 변경.....	425
연결된 변환 구성.....	426
연결되지 않은 변환 구성.....	426
식에서 저장 프로시저 호출.....	426
사전 세션 또는 사후 세션 저장 프로시저 호출.....	428
오류 처리.....	430
사전 세션 오류.....	430
사후 세션 오류.....	430
세션 오류.....	430
지원되는 데이터베이스.....	430
SQL 선언.....	430
매개 변수 유형.....	431
매핑의 입력/출력 포트.....	431
지원되는 반환 값 유형.....	431
식 규칙.....	431
저장 프로시저 변환에 대한 팁.....	432
저장 프로시저 변환 문제 해결.....	432

장 29: 트랜잭션 제어 변환..... 434

트랜잭션 제어 변환 개요.....	434
트랜잭션 제어 변환 속성.....	434
속성 탭.....	435
예제.....	435
매핑에서 트랜잭션 제어 변환 사용.....	436
여러 대상이 있는 샘플 트랜잭션 제어 매핑.....	437
매핑 지침 및 유효성 검사.....	438
트랜잭션 제어 변환 작성.....	439

장 30: 합집합 변환..... 440

합집합 변환 개요.....	440
합집합 변환에 대한 규칙 및 지침.....	440
합집합 변환 구성 요소.....	441
그룹 및 포트 작업.....	441
합집합 변환 작성.....	441
매핑에서 합집합 변환 사용.....	442

장 31: 구조화되지 않은 Data Transformation.....	443
구조화되지 않은 Data Transformation 개요.....	443
구조화되지 않은 데이터 옵션 구성.....	444
Data Transformation 리포지토리 디렉터리 구성.....	445
Data Transformation 서비스 유형.....	445
구조화되지 않은 Data Transformation 구성 요소.....	445
속성 탭.....	446
UDT 설정 탭.....	447
상태 추적 메시지 보기.....	448
구조화되지 않은 Data Transformation 포트.....	448
입력 및 출력 유형.....	449
추가적인 구조화되지 않은 Data Transformation 포트.....	449
Data Transformation 서비스에서 포트 작성.....	450
구조화되지 않은 Data Transformation 서비스 이름.....	450
관계형 계층.....	451
계층 스키마 내보내기.....	451
매핑.....	452
관계형 테이블을 위한 Word 문서 구문 분석.....	452
XML에서 Excel 시트 작성.....	452
XML 파일 출력 분할.....	453
구조화되지 않은 데이터 매핑에 대한 규칙 및 지침.....	453
구조화되지 않은 Data Transformation 작성.....	454
 장 32: 업데이트 전략 변환.....	 455
업데이트 전략 변환 개요.....	455
업데이트 전략 설정.....	455
매핑 내 행에 플래그 지정.....	456
거부된 행 전달.....	456
업데이트 전략 식.....	456
집계 및 업데이트 전략 변환.....	457
조회 및 업데이트 전략 변환.....	457
세션의 업데이트 전략 설정.....	458
모든 행에 대한 작업 지정.....	458
개별 대상 테이블에 대한 작업 지정.....	459
업데이트 전략 검사 목록.....	459
 장 33: XML 변환.....	 461
XML 소스 한정자 변환.....	461
XML 파서 변환.....	461
XML 생성기 변환.....	462

인덱스.....	463
----------	-----

서문

*PowerCenter® 변환 가이드*에서는 Informatica 변환의 구성, 지침, 사용 및 런타임 동작에 대해 알아볼 수 있습니다. 변환은 통합 서비스가 데이터에 대해 수행하는 작업을 나타내는 리포지토리 개체입니다.

데이터 생성, 변경 또는 전송을 위한 데이터 작업을 수행하는 방법을 이 가이드에서 알아볼 수 있습니다. 매핑을 실행하는 위치와 방법에 따라 각 변환에 대한 지원을 확인하십시오. 데이터는 변환 포트를 통해 매핑 또는 맵셋에서 연결한 대상 및 다른 변환으로 전달됩니다. 활성 변환의 경우 식 편집기를 사용하여 식을 입력할 수 있습니다. 변환에는 수신 또는 발신 데이터의 행을 정의하는 포트 집합인 여러 개의 입력 및 출력 그룹이 포함될 수 있습니다.

Informatica 리소스

Informatica는 Informatica Network 및 기타 온라인 포털을 통해 다양한 범위의 제품 리소스를 제공합니다. 리소스를 통해 Informatica 제품 및 솔루션을 최대한 활용하고 다른 Informatica 사용자 및 주제별 전문가로부터 배울 수 있습니다.

Informatica 네트워크

Informatica Network는 Informatica 기술 자료, Informatica 글로벌 고객 지원 센터 등 여러 리소스로 연결되는 관문입니다. Informatica Network를 시작하려면 <https://network.informatica.com>을 방문하십시오.

Informatica Network 멤버인 경우 다음 옵션이 가능합니다.

- 기술 자료에서 제품 리소스를 검색할 수 있습니다.
- 제품 사용 가능 여부에 대한 정보를 봅니다.
- 지원 사례를 생성하고 검토할 수 있습니다.
- 거주 지역의 Informatica 사용자 그룹 네트워크를 검색하고 동료와 협업 관계 유지

Informatica 기술 자료

Informatica 기술 자료를 사용하여 사용 방법 문서, 모범 사례, 비디오 자습서, 자주 묻는 질문에 대한 답변 등 제품 리소스를 확인할 수 있습니다.

기술 자료를 검색하려면 <https://search.informatica.com>을 방문하십시오. 기술 자료에 대한 질문, 의견 또는 아이디어가 있는 경우 KB_Feedback@informatica.com을 통해 Informatica 기술 자료 팀에 문의해 주시기 바랍니다.

Informatica 설명서

Informatica 설명서 포털에서 확장된 설명서 라이브러리를 탐색하여 현재 및 최근 제품 릴리스를 확인할 수 있습니다. 설명서 포털을 탐색하려면 <https://docs.informatica.com>을 방문하십시오.

제품 설명서에 대한 질문, 의견 또는 아이디어가 있는 경우 infa_documentation@informatica.com에서 Informatica 설명서 팀에 문의해 주시기 바랍니다.

Informatica Product Availability Matrix

PAM(Product Availability Matrix)은 제품 릴리스에서 지원하는 운영 체제 버전, 데이터베이스 및 데이터 소스 유형과 대상을 나타냅니다.

<https://network.informatica.com/community/informatica-network/product-availability-matrices>에서 Informatica PAM을 찾을 수 있습니다.

Informatica Velocity

Informatica Velocity는 수백 가지 데이터 관리 프로젝트의 실제 경험을 토대로 Informatica 전문 서비스업에서 개발한 팁과 모범 사례 모음입니다. Informatica Velocity는 전 세계의 조직과 협력하여 성공적인 데이터 관리 솔루션을 계획, 개발, 배포 및 유지 관리하는 Informatica 컨설턴트의 포괄적인 지식을 보여줍니다.

Informatica Velocity 리소스는 <http://velocity.informatica.com>에서 확인할 수 있습니다. Informatica Velocity에 대한 질문, 주석 또는 아이디어가 있으시면 Informatica 전문 서비스업(ips@informatica.com)에 문의하십시오.

Informatica Marketplace

Informatica Marketplace는 Informatica 구현을 확대 및 개선하기 위한 솔루션을 찾을 수 있는 포럼입니다.

Marketplace에서 Informatica 개발자와 파트너가 제공하는 수백 개의 솔루션을 활용하여 생산성을 향상시키고 프로젝트의 구현에 걸리는 시간을 줄일 수 있습니다. <https://marketplace.informatica.com>에서 Informatica Marketplace를 찾을 수 있습니다.

Informatica 글로벌 고객 지원 센터

전화 또는 Informatica Network를 통해 글로벌 지원 센터에 문의할 수 있습니다.

해당 지역의 Informatica 글로벌 고객 지원 전화 번호는 Informatica 웹 사이트 (<https://www.informatica.com/services-and-training/customer-success-services/contact-us.html>)를 방문하여 찾을 수 있습니다.

Informatica Network에서 온라인 지원 리소스를 찾으려면 <https://network.informatica.com>을 방문하고 eSupport 옵션을 선택하십시오.

제 1 장

변환 작업

이 장에 포함된 항목:

- [변환 개요, 25](#)
- [변환 작성, 29](#)
- [변환 구성, 29](#)
- [변환 포트, 30](#)
- [다중 그룹 변환, 31](#)
- [식 작업, 31](#)
- [로컬 변수, 36](#)
- [포트의 기본값, 38](#)
- [변환의 추적 수준 구성, 44](#)
- [재사용 가능 변환, 44](#)

변환 개요

변환은 데이터를 생성하거나 수정하거나 전달하는 리포지토리 개체입니다. 디자이너는 특정 함수를 수행하는 변환 집합을 제공합니다. 예를 들어 집계 변환은 데이터 그룹에 대한 계산을 수행합니다.

매핑에서 변환은 통합 서비스가 데이터에 대해 수행하는 작업을 나타냅니다. 데이터는 사용자가 매핑 또는 맵렛에서 연결하는 변환 포트를 통해 전달됩니다.

변환은 다음 유형일 수 있습니다.

- 활성 또는 수동
- 연결됨 또는 연결되지 않음
- 원시 또는 비원시

활성 변환

활성 변환은 다음과 같은 작업을 수행할 수 있습니다.

- **변환을 통해 통과되는 행 수 변경.** 예를 들어 필터 변환은 필터 조건을 만족하지 않는 행을 제거하므로 활성입니다. 모든 다중 그룹 변환은 변환을 통해 통과되는 행의 수를 변경할 수 있으므로 활성입니다.
- **트랜잭션 경계 변경.** 예를 들어 트랜잭션 제어 변환은 각 행에 대해 평가되는 식에 따라 커밋 또는 롤백 트랜잭션을 정의하므로 활성입니다.

- **행 유형 변경.** 예를 들어 업데이트 전략 변환은 행에 삽입, 삭제, 업데이트 또는 거부 플래그를 지정하므로 활성입니다.

통합 서비스가 활성 변환에서 전달되는 행을 연결하지 못할 수 있기 때문에 디자이너에서 여러 활성 변환 또는 활성 및 수동 변환을 동일한 다운스트림 변환 또는 변환 입력 그룹에 연결할 수 없습니다. 예를 들어 행에 삭제 플래그를 지정하는 업데이트 전략 변환이 매핑의 한 분기에 있다고 가정합니다. 행에 삽입 플래그를 지정하는 업데이트 전략 변환이 또 다른 분기에 있습니다. 이러한 변환을 단일 변환 입력 그룹에 연결하면 통합 서비스가 행에 대한 삭제 및 삽입 작업을 결합할 수 없습니다.

시퀀스 생성기 변환에는 이 규칙이 적용되지 않습니다. 디자이너에서 시퀀스 생성기 변환 및 활성 변환을 동일한 다운스트림 변환 또는 변환 입력 그룹에 연결할 수 있습니다. 시퀀스 생성기 변환은 데이터를 수신하지 않습니다. 이 변환은 고유한 숫자 값을 생성합니다. 따라서 통합 서비스가 시퀀스 생성기 변환 및 활성 변환에서 전달된 행을 연결할 때 문제가 발생하지 않습니다.

수동 변환

수동 변환은 변환을 통과하는 행 수를 변경하지 않고 트랜잭션 경계를 유지 관리하고 행 유형을 유지 관리합니다.

업스트림 분기의 모든 변환이 수동인 경우 여러 변환을 동일한 다운스트림 변환 또는 동일한 변환 입력 그룹에 연결할 수 있습니다. 분기를 발생시키는 변환은 활성이거나 수동일 수 있습니다.

연결되지 않은 변환

변환은 데이터 흐름에 연결될 수도 있고 연결되지 않을 수도 있습니다. 연결되지 않은 변환은 매핑의 다른 변환과 연결되어 있지 않습니다. 연결되지 않은 변환은 다른 변환 내에서 호출되며 값을 해당 변환에 반환합니다.

원시 변환 및 비원시 변환

원시 변환은 디자이너에 제공되는 변환 집합입니다. 비원시 변환은 사용자가 사용자 지정 변환을 사용하여 작성하는 변환입니다. 디자이너에는 **Java**, **SQL** 및 합집합 변환 같은 비원시 변환도 일부 제공됩니다. 사용자 지정 변환에 적용되는 규칙은 사용자 지정 변환을 사용하여 구성된 비원시 변환에도 적용됩니다.

변환 설명

다음 테이블에는 각 변환에 대한 간략한 설명이 나와 있습니다.

변환	유형	설명
집계	- 활성 - 연결됨 - 원시	집계 계산을 수행합니다.
응용 프로그램 소스 한정자	- 활성 - 연결됨 - 비원시	통합 서비스가 세션을 실행할 때 ERP 소스 등의 응용 프로그램에서 읽는 행을 나타냅니다.
사용자 지정	- 활성 또는 수동 - 연결됨 - 비원시	공유 라이브러리 또는 DLL에 있는 프로시저를 호출합니다.

변환	유형	설명
데이터 마스킹	- 수동 - 연결됨 - 비원시	비프로덕션 환경에 대해 중요한 프로덕션 데이터를 실제 테스트 데이터로 대체합니다.
식	- 수동 - 연결됨 - 원시	값을 계산합니다.
외부 프로시저	- 수동 - 연결됨 또는 연결되지 않음 - 원시	공유 라이브러리 또는 Windows의 COM 계층에 있는 프로시저를 호출합니다.
필터	- 활성 - 연결됨 - 원시	데이터를 필터링합니다.
HTTP	- 수동 - 연결됨 - 비원시	HTTP 서버에 연결하여 데이터를 읽거나 업데이트합니다.
입력	- 수동 - 연결됨 - 원시	맵렛 입력 행을 정의합니다. Mapplet Designer에서 사용할 수 있습니다.
Java	- 활성 또는 수동 - 연결됨 - 비원시	Java로 코딩된 사용자 논리를 실행합니다. 사용자 논리의 바이트 코드는 리포지토리에 저장됩니다.
조이너	- 활성 - 연결됨 - 원시	여러 데이터베이스 또는 플랫폼 파일 시스템의 데이터를 조인합니다.
조회	- 활성 또는 수동 - 연결됨 또는 연결되지 않음 - 원시	플랫폼 파일, 관계형 테이블, 보기 또는 동의어에서 데이터를 조회하고 반환합니다.
노멀라이저	- 활성 - 연결됨 - 원시	COBOL 소스에 대한 소스 한정자입니다. 관계형 또는 플랫폼 파일 소스에서 반환된 데이터를 정규화하기 위해 파이프라인에서 사용될 수도 있습니다.
출력	- 수동 - 연결됨 - 원시	맵렛 출력 행을 정의합니다. Mapplet Designer에서 사용할 수 있습니다.
순위	- 활성 - 연결됨 - 원시	레코드를 상위 또는 하위 범위로 제한합니다.
라우터	- 활성 - 연결됨 - 원시	그룹 조건에 따라 데이터를 여러 변환으로 라우팅합니다.

변환	유형	설명
시퀀스 생성기	- 수동 - 연결됨 - 원시	기본 키를 생성합니다.
분류기	- 활성 - 연결됨 - 원시	정렬 키에 따라 데이터를 정렬합니다.
소스 한정자	- 활성 - 연결됨 - 원시	통합 서비스가 세션을 실행할 때 관계형 또는 플랫폼 파일 소스에서 읽는 행을 나타냅니다.
SQL	- 활성 또는 수동 - 연결됨 - 비원시	데이터베이스에 대해 SQL 쿼리를 실행합니다.
저장 프로시저	- 수동 - 연결됨 또는 연결되지 않음 - 원시	저장 프로시저를 호출합니다.
트랜잭션 제어	- 활성 - 연결됨 - 원시	커밋 및 롤백 트랜잭션을 정의합니다.
합집합	- 활성 - 연결됨 - 비원시	여러 데이터베이스 또는 플랫폼 파일 시스템의 데이터를 병합합니다.
구조화되지 않은 데이터	- 활성 또는 수동 - 연결됨 - 비원시	구조화되지 않은 형식 및 반구조화된 형식의 데이터를 변환합니다.
업데이트 전략	- 활성 - 연결됨 - 원시	행을 삽입, 삭제, 업데이트 또는 거부할지 여부를 지정합니다.
XML 생성기	- 활성 - 연결됨 - 원시	하나 이상의 입력 포트에서 데이터를 읽고 단일 출력 포트를 통해 XML을 출력합니다.
XML 파서	- 활성 - 연결됨 - 원시	입력 포트 1개에서 XML을 읽고 1개 이상의 출력 포트에 데이터를 출력합니다.
XML 소스 한정자	- 활성 - 연결됨 - 원시	통합 서비스가 세션을 실행할 때 XML 소스에서 읽는 행을 나타냅니다.

매핑을 작성할 때 변환을 추가하고 비즈니스 용도에 맞게 데이터를 처리하도록 변환을 구성합니다. 변환을 매핑에 통합하려면 다음 태스크를 완료하십시오.

1. **변환을 작성합니다.** 변환을 매핑 디자이너에서 매핑의 일부로 또는 Maplet Designer에서 맵렛의 일부로 작성하거나 변환 개발자에서 재사용 가능 변환으로 작성합니다.

2. **변환을 구성합니다.** 각 변환 유형에는 구성 가능한 고유한 옵션 집합이 있습니다.
3. **다른 트랜잭션 및 대상 정의에 변환 연결.** 포트 하나를 다른 포트로 끌어 와서 매핑 또는 맵렛에서 서로 연결합니다.

변환 작성

다음과 같은 디자이너 도구를 사용하여 변환을 작성할 수 있습니다.

- **매핑 디자이너.** 소스를 대상에 연결하는 변환을 작성합니다. 매핑의 변환은 재사용 가능하게 구성하지 않는 한 다른 매핑에서 사용할 수 없습니다.
- **변환 개발자.** 여러 매핑에서 사용되는 재사용 가능 변환이라는 개별 변환을 작성합니다.
- **Mapplet Designer.** 여러 매핑에서 사용되는 맵렛이라는 일련의 변환을 작성하고 구성합니다.

매핑 디자이너, 변환 개발자 및 Mapplet Designer에서 동일한 프로세스를 사용하여 변환을 작성할 수 있습니다. 변환을 작성하려면

1. 해당하는 Designer 도구를 엽니다.
2. 매핑 디자이너에서 매핑을 열거나 작성합니다. Mapplet Designer에서 맵렛을 열거나 작성합니다.
3. 변환 > 작성을 클릭하고 작성할 변환 유형을 선택합니다.
4. 변환을 배치할 매핑의 부분을 끌어서 선택합니다.

새 변환이 작업 공간에 나타납니다. 다음으로, 새 포트를 추가하고 그 외의 속성을 설정하여 변환을 구성해야 합니다.

변환 구성

변환을 작성한 후에 해당 변환을 구성할 수 있습니다. 모든 변환에는 다음과 같은 공통 탭이 포함되어 있습니다.

- **변환.** 변환의 이름 또는 설명을 추가합니다.
- **포트.** 포트를 추가하고 구성합니다.
- **속성.** 변환에 고유한 속성을 구성합니다.

일부 변환에는 조건 탭과 같은 기타 탭이 포함될 수 있으며, 이러한 탭에서 조이너 또는 노멀라이저 변환의 조건을 입력합니다.

변환을 구성할 때 다음 태스크를 완료할 수 있습니다.

- **포트 추가.** 변환 간에 이동되는 데이터 열을 정의합니다.
- **그룹 추가.** 일부 변환에서 변환으로 들어오거나 나가는 데이터 행을 정의하는 입력 또는 출력 그룹을 정의합니다.
- **식 입력.** 데이터를 변환하는 일부 변환에서 SQL과 비슷한 식을 입력합니다.
- **로컬 변수 정의.** 임시로 데이터를 저장하는 일부 변환에서 로컬 변수를 정의합니다.
- **기본값 재정의.** 입력 Null 및 출력 변환 오류를 처리하도록 포트에 대한 기본값을 구성합니다.
- **추적 수준 입력.** 변환에 대한 세션 로그에 통합 서비스가 기록하는 세부 정보 양을 선택합니다.

변환 이름 바꾸기

변환의 이름을 바꾸려면 이름 바꾸기 단추를 클릭하고 변환에 대해 쉽게 알 수 있는 이름을 입력한 다음 확인을 클릭합니다.

변환 포트

변환을 생성한 다음 포트를 정의합니다. 포트를 생성하고 포트 속성을 정의합니다.

일부 변환을 생성하는 경우 모든 포트를 수동으로 생성할 필요가 없습니다. 예를 들어 조회 변환을 생성하고 조회 테이블을 참조할 수 있습니다. 변환 포트를 보면 변환에 참조한 테이블의 각 열에 대한 출력 포트가 있는지 확인할 수 있습니다. 해당 포트는 정의할 필요가 없습니다.

포트 생성

일부 변환을 생성하는 경우 모든 포트를 수동으로 생성할 필요가 없습니다. 예를 들어 조회 변환을 생성하고 조회 테이블을 참조할 수 있습니다. 변환 포트를 보면 변환에 참조한 테이블의 각 열에 대한 출력 포트가 있는지 확인할 수 있습니다. 해당 포트는 정의할 필요가 없습니다.

다음과 같은 방법으로 포트를 생성합니다.

- **다른 변환에서 포트 끌기.** 다른 변환에서 포트를 끌어오면 디자이너가 동일한 속성을 포함한 포트를 생성하고 두 포트를 연결합니다. 레이아웃 > 열 복사를 클릭하여 포트 복사를 활성화합니다.
- **포트 탭에서 추가 단추를 클릭합니다.** 디자이너가 구성할 수 있는 빈 포트를 생성합니다.

포트 구성

변환 포트를 정의할 경우 포트 속성을 정의하십시오. 포트 속성에는 포트 이름, 데이터 유형, 포트 유형 및 기본값이 포함됩니다.

다음 포트 속성을 구성하십시오.

- **포트 이름.** 포트의 이름입니다. 포트 이름을 지정할 경우 다음 규칙을 사용하십시오.
 - 싱글바이트 또는 더블바이트 문자나 싱글바이트 또는 더블바이트 밑줄(_)로 시작합니다.
 - 포트 이름에는 싱글바이트 또는 더블바이트 문자의 문자, 숫자, 밑줄(_), \$, # 또는 @가 포함될 수 있습니다.
- **데이터 유형, 전체 자릿수 및 소수 자릿수.** 식 또는 조건을 입력할 경우 데이터 유형이 식의 반환 값과 일치하는지 확인하십시오.
- **포트 유형.** 변환에 입력, 출력, 입력/출력 및 변수 포트 유형의 조합이 포함될 수 있습니다.
- **기본값.** Null 값 또는 출력 변환 오류를 포함하는 포트에 대한 기본값을 할당합니다. 일부 포트의 기본값을 재정의할 수 있습니다.
- **설명.** 포트에 대한 설명입니다.
- **기타 속성.** 일부 변환에는 식 또는 그룹 기준 속성과 같이 해당 변환과 관련된 속성이 있습니다.

포트 연결

매핑에서 변환을 추가하고 구성한 후에 이 변환을 대상 및 다른 변환에 연결합니다. 포트를 통해 매핑 개체를 연결합니다. 데이터는 다음과 같은 포트를 통해 매핑 내부와 외부로 전달됩니다.

- **입력 포트.** 데이터를 수신합니다.

- **출력 포트.** 데이터를 전달합니다.
- **입력/출력 포트.** 데이터를 수신하고 변경되지 않은 상태로 전달합니다.

포트를 연결하려면 서로 다른 매핑 개체의 포트 간에 씁니다. 디자이너가 링크의 유효성을 검사하고 링크가 유효성 검사 요구 사항을 충족하는 경우에만 링크를 작성합니다.

다중 그룹 변환

변환에 여러 입력 및 출력 그룹이 있을 수 있습니다. 그룹은 수신 또는 전송 데이터의 행을 정의하는 포트 집합입니다.

그룹은 관계형 소스 또는 대상 정의의 테이블과 비슷합니다. 대부분의 변환에는 하나의 입력 그룹과 하나의 출력 그룹이 있습니다. 그러나 일부 변환에는 여러 입력 그룹, 여러 출력 그룹 또는 둘 다 있습니다. 그룹은 변환으로 들어오거나 나가는 데이터 행의 표현입니다.

모든 다중 그룹 변환은 활성 변환입니다. 여러 활성 변환 또는 활성 및 수동 변환을 동일한 다운스트림 변환 또는 변환 입력 그룹에 연결할 수 없습니다.

일부 다중 입력 그룹 변환은 통합 서비스가 다른 입력 그룹의 행을 기다리는 동안 통합 서비스가 입력 그룹에서 데이터를 차단하도록 요구합니다. 차단 변환은 수신 데이터를 차단하는 다중 입력 그룹 변환입니다. 다음 변환은 차단 변환입니다.

- 입력이 차단할 수 있음 속성이 활성화된 사용자 지정 변환
- 정렬되지 않은 입력에 구성된 조이너 변환

매핑을 저장하거나 유효성을 검증하는 경우 활성 또는 차단 변환이 포함된 일부 매핑이 유효하지 않을 수 있습니다.

식 작업

일부 변환의 경우 식 편집기를 사용하여 식을 입력할 수 있습니다. 다음 함수를 포함하는 식을 작성합니다.

- **변환 언어 함수.** 공통 식을 처리하기 위해 설계된 SQL 같은 함수입니다.
- **사용자 정의 함수.** 변환 언어 함수를 기반으로 사용자가 PowerCenter에서 작성한 함수입니다.
- **사용자 지정 함수.** 사용자 지정 함수 API를 사용하여 작성한 함수입니다.

입력 또는 입력/출력 포트의 데이터 값을 사용하는 식을 포트에 입력합니다. 예를 들어 모든 직원의 급여가 들어 있는 입력 포트 IN_SALARY를 포함하는 변환이 있을 수 있습니다. IN_SALARY 열의 값을 나중에 매핑에서 사용할 수 있고, 이 변환을 통해 계산한 총 급여 및 평균 급여를 사용할 수도 있습니다. 이러한 이유 때문에 디자이너는 각 계산 값에 대해 별도의 출력 포트를 작성하도록 요구합니다.

다음 표에는 식을 입력할 수 있는 변환이 나열되어 있습니다.

변환	식	반환 값
집계	변환을 통해 전달되는 모든 데이터를 기반으로 집계 계산을 수행합니다. 또는 집계 계산에서 레코드에 대한 필터를 지정하여 특정 유형의 레코드를 제외할 수 있습니다. 예를 들어 이 변환을 사용하여 지점에 있는 모든 직원의 총 수와 평균 급여를 찾을 수 있습니다.	포트에 대한 집계 계산의 결과입니다.
데이터 마스킹	입력 또는 출력 포트 값을 기반으로 행에 대해 계산을 수행합니다. 식은 데이터 마스킹 변환에서 프로덕션 데이터를 마스킹하기 위한 수단입니다.	입력 또는 출력 포트를 사용한 행 수준 계산의 결과입니다.
식	단일 행 내의 값을 기반으로 계산을 수행합니다. 예를 들어 특정 품목의 가격 및 수량을 기반으로 주문에서 해당 품목의 총 구입 가격을 계산할 수 있습니다.	포트에 대한 행 수준 계산의 결과입니다.
필터	이 변환을 통해 전달되는 행을 필터링하는 데 사용되는 조건을 지정합니다. 예를 들어 미납금이 있는 고객에 대해 고객 데이터를 BAD_DEBT 테이블에 기록하려는 경우 필터 변환을 사용하여 고객 데이터를 필터링할 수 있습니다.	행이 지정된 조건을 만족하는지 여부에 따라 TRUE 또는 FALSE입니다. TRUE를 반환하는 행만 이 변환을 통해 전달됩니다. 변환은 변환을 통과하는 각 행에 이 값을 적용합니다.
순위	순위에 포함되는 행에 대한 조건을 설정합니다. 예를 들어 회사에 고용된 상위 10명의 판매자에 대한 순위를 지정할 수 있습니다.	포트에 대한 조건 또는 계산의 결과입니다.
라우터	그룹 식에 따라 데이터를 여러 변환으로 라우팅합니다. 예를 들어 이 변환을 사용하여 세 가지 다른 급여 수준에서 직원의 급여를 비교할 수 있습니다. 라우터 변환에서 세 개의 그룹을 작성하여 이 작업을 수행할 수 있습니다. 예를 들어 각 급여 범위에 대한 그룹 식을 한 개 작성합니다.	행이 지정된 그룹 식을 만족하는지 여부에 따라 TRUE 또는 FALSE입니다. TRUE를 반환하는 행만 이 변환의 각 사용자 정의 그룹을 통해 전달됩니다. FALSE를 반환하는 행은 기본 그룹을 통해 전달됩니다.
업데이트 전략	업데이트, 삽입, 삭제 또는 거부할 행에 플래그를 지정합니다. 사용자가 적용하는 조건에 따라 대상에 대한 업데이트를 제어하려면 이 변환을 사용합니다. 예를 들어 메일 주소가 변경된 경우 업데이트 전략 변환을 사용하여 모든 고객 행에 업데이트 플래그를 지정하거나 더 이상 회사에 근무하지 않는 사람과 관련된 모든 직원 행에 거부 플래그를 지정할 수 있습니다.	업데이트, 삽입, 삭제 또는 거부를 나타내는 숫자 코드입니다. 변환은 변환을 통과하는 각 행에 이 값을 적용합니다.
트랜잭션 제어	커밋, 롤백 또는 트랜잭션 변경 없음 중 어떤 작업을 통합 서비스가 수행할지 결정하는 데 사용되는 조건을 지정합니다. 변환을 통과하는 행 또는 행 집합을 기반으로 커밋 및 롤백 트랜잭션을 제어하려는 경우 이 변환을 사용합니다. 예를 들어 이 변환을 사용하면 주문 입력 날짜를 기준으로 행 집합을 커밋할 수 있습니다.	행이 지정된 조건을 충족하는지 여부에 따라 다음 기본 제공 변수 중 하나입니다. <ul style="list-style-type: none"> - TC_CONTINUE_TRANSACTION - TC_COMMIT_BEFORE - TC_COMMIT_AFTER - TC_ROLLBACK_BEFORE - TC_ROLLBACK_AFTER 통합 서비스가 반환 값에 따라 작업을 수행합니다.

식 편집기 사용

식 편집기를 사용하여 SQL과 유사한 문을 작성합니다. 식을 수동으로 입력할 수도 있지만 가리키고 클릭하는 방법을 사용합니다. 식을 작성할 때 실수를 최소화할 수 있도록 가리키고 클릭 인터페이스를 통해 함수, 포트, 변수 및 연산자를 선택하십시오. 식에 포함할 수 있는 최대 문자 수는 32,767자입니다.

식 변환의 식 편집기에서 구성하는 식은 평가할 수 있습니다.

식을 테스트하는 경우 샘플 데이터를 입력한 다음 식을 평가할 수 있습니다. 재사용 가능한 변환의 식은 변환 디자이너에서 평가합니다. 재사용 불가능한 변환의 식은 매핑 디자이너에서 평가합니다. 일부 함수는 포트 데이터 유형을 이진 또는 날짜/시간으로 설정한 경우 식 평가를 지원하지 않습니다.

식에 포트 이름 입력

연결된 변환의 경우 식에 포트 이름을 사용하면 변환에서 포트 이름을 변경할 때 디자이너가 해당 식을 업데이트합니다. 예를 들어 두 날짜 `Date_Promised`와 `Date_Delivered` 간의 차이를 확인하는 올바른 식을 작성한다고 가정합니다. 나중에 `Date_Promised` 포트 이름을 `Due_Date`로 변경하면 디자이너가 식에서 `Date_Promised` 포트 이름을 `Due_Date`로 변경합니다.

참고: 매핑에서 이 포트를 사용하는 다른 재사용 불가능한 변환에 이름 `Due_Date`를 전달할 수 있습니다.

설명 추가

식에 대한 설명 정보를 제공하는 설명을 식에 추가하거나 식에 대한 비즈니스 설명서에 액세스할 수 있는 유효한 URL을 지정할 수 있습니다.

다음과 같은 방식 중 하나로 설명을 추가할 수 있습니다.

- 식 내에서 설명을 추가하려면 -- 또는 // 설명 표시기를 사용합니다.
- 대화 상자에 설명을 추가하려면 설명 단추를 클릭합니다.

식 유효성 검사

유효성 검사 단추를 사용하여 식의 유효성을 검사합니다. 식의 유효성을 검사하지 않으면 식 편집기를 닫을 때 디자이너가 식의 유효성을 검사합니다. 식이 유효하지 않으면 디자이너에서 경고를 표시합니다. 유효하지 않은 식을 저장할 수도 있고 수정할 수도 있습니다. 올바르지 않은 식을 포함하는 매핑에 대해 세션을 실행할 수는 없습니다.

식 편집기 표시

식 편집기는 구문 식을 다른 색상으로 표시하여 가독성을 개선할 수 있습니다. 최신 Rich Edit 컨트롤 `riched20.dll`이 시스템에 설치된 경우 식 편집기에서 식 함수는 파란색, 설명은 회색, 따옴표로 묶인 문자열은 녹색으로 표시됩니다.

식 편집기의 크기를 조정할 수 있습니다. 테두리에서 끌어서 대화 상자를 확대합니다. 디자이너가 대화 상자의 새로운 크기를 클라이언트 설정으로 저장합니다.

포트에 식 추가

데이터 마스킹 변환에서 식을 입력 포트에 추가할 수 있습니다. 다른 모든 변환의 경우 식을 출력 포트에 추가합니다.

다음 단계를 수행하여 식을 포트에 추가하십시오.

1. 변환에서 포트를 선택하고 식 편집기를 엽니다.
2. 식을 입력합니다.

함수 탭과 포트 탭 및 연산자 키를 사용합니다.

3. 식에 설명을 추가합니다.

설명 표시기 -- 또는 //를 사용합니다.

4. 식의 유효성을 검사합니다.

유효성 검사 단추를 사용하여 식의 유효성을 검사합니다.

매개 변수 파일에 식 문자열 정의

통합 서비스는 식을 구문 분석한 후 식의 매핑 매개 변수 및 변수를 확장합니다. 자주 변경되는 식이 있을 경우 매개 변수 파일에서 식 문자열을 정의하면 식이 변경될 때 식을 사용하는 매핑을 업데이트할 필요가 없습니다.

매개 변수 파일에서 식 문자열을 정의하려면 식 문자열을 저장할 매핑 매개 변수 또는 변수를 작성하고 매개 변수 또는 변수를 매개 변수 파일의 식 문자열로 설정합니다. 작성하는 매개 변수 또는 변수에서 **IsExprVar**를 **true**로 설정해야 합니다. **IsExprVar**가 **true**이면 통합 서비스가 식을 구문 분석하기 전에 매개 변수 또는 변수를 확장합니다.

예를 들어 매개 변수 파일에서 식 **IIF(color='red',5)**를 정의하려면 다음 단계를 수행하십시오.

1. 식을 사용하는 매핑에서 매핑 매개 변수 **\$\$Exp**를 작성합니다. **IsExprVar**를 **true**로 설정하고 데이터 유형을 문자열로 설정합니다.

2. 식 편집기에서 다음과 같이 매핑 매개 변수의 이름으로 식을 설정합니다.

\$\$Exp

3. 매개 변수 파일을 사용하도록 세션 또는 워크플로우를 구성합니다.

4. 매개 변수 파일에서 다음과 같이 **\$\$Exp**의 값을 식 문자열로 설정합니다.

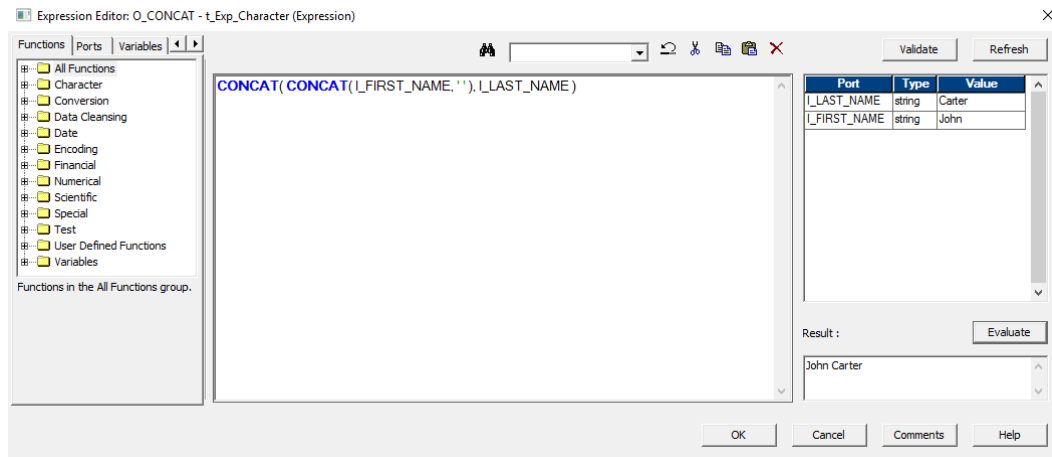
\$\$Exp=IIF(color='red',5)

식 평가

식 변환의 식 편집기에서 구성하는 식은 평가할 수 있습니다. 식을 테스트하는 경우 샘플 데이터를 입력한 다음 식을 평가할 수 있습니다.

식 조건을 편집하는 경우 테스트 데이터를 평가하기 전에 테스트 패널에서 새로 고침을 선택해야 합니다. 유효한 식을 입력하지 않으면 테스트 패널에 포트가 입력되지 않습니다. 시스템 정의 매개 변수를 지정할 때 입력하는 값은 런타임 값이 아닙니다.

다음 이미지에는 식 변환의 식 편집기 안에서 샘플 식을 평가하는 방법이 나와 있습니다.



참고: 날짜/시간 또는 이진 데이터 유형의 식은 평가할 수 없습니다.

예

모든 결과를 대상에 로드하기 전에 각 고객에 대해 받은 주문의 총 수를 기반으로 홍보용 제공품을 계산해야 합니다. 식 변환 내에 식을 정의하여 제공품을 계산하는 매핑을 개발해야 합니다. 연산자가 매핑을 실행하기 전에 식을 평가하여 결과를 확인하십시오.

식 평가

식 변환의 식 편집기에서 식을 테스트하고 확인할 수 있습니다.

1. 식 변환의 식 편집기에서 식을 입력합니다.
2. 식의 유효성을 검사하려면 유효성 검사를 클릭합니다.
3. 오른쪽 창의 식 테스트 섹션에 식 조건의 최신 변경 내용을 반영하려면 새로 고침을 클릭합니다.
4. 오른쪽 창에서 식에 사용된 입력 포트의 샘플 값을 입력합니다.
5. 식을 평가하려면 평가를 클릭합니다.

식 평가 제한 사항

식을 평가하는 경우 특정 제한 사항이 적용됩니다.

식을 평가할 때는 다음과 같은 제한 사항을 고려해야 합니다.

변환 제한 사항

다음 변환의 식은 평가할 수 없습니다.

- 집계
- 데이터 마스킹
- 필터
- 순위
- 라우터
- 저장 프로시저
- 트랜잭션 제어
- 업데이트 전략

데이터 유형 제한 사항

입력 포트 유형 또는 반환 함수에 이진 또는 날짜/시간 데이터 유형이 포함되는 경우 식을 평가할 수 없습니다.

포트 제한 사항

조회 또는 저장 프로시저 포트를 사용하는 식은 평가할 수 없습니다.

로컬 변수

집계, 식 및 순위 변환에서 로컬 변수를 사용하여 성능을 개선할 수 있습니다. 식에서 변수를 참조하거나 변수를 사용하여 임시로 데이터를 저장할 수 있습니다.

변수를 사용하여 다음 작업을 완료할 수 있습니다.

- 데이터를 임시로 저장합니다.
- 복잡한 식을 간소화합니다.
- 이전 행의 값을 저장합니다.
- 저장 프로시저의 여러 반환 값을 캡처합니다.
- 값을 비교합니다.
- 연결되지 않은 조회 변환의 결과를 저장합니다.

임시로 데이터 저장 및 복잡한 식 간소화

동일한 변환에서 관련 식을 여러 번 입력하는 경우 변수를 사용하면 성능이 향상됩니다. 동일한 식 구성 요소를 변환에서 여러 번 구문 분석하고 유효성 검사를 수행하는 대신 구성 요소를 변수로 정의할 수 있습니다.

예를 들어 집계 변환이 합계 및 평균을 계산하기 전에 동일한 필터 조건을 사용하는 경우 이 조건을 변수로 정의한 다음 두 집계 계산에서 조건을 재사용할 수 있습니다.

복잡한 식을 간소화할 수 있습니다. 집계에서 여러 식에 동일한 계산이 포함된 경우 계산 결과를 저장하기 위한 변수를 작성하면 성능을 향상시킬 수 있습니다.

예를 들어 다음 식을 작성하여 동일한 데이터로 평균 급여와 총 급여를 모두 찾을 수 있습니다.

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

두 계산에 대해 동일한 인수를 입력하는 대신 이 계산에서 각 조건에 대해 변수 포트를 작성한 다음 식을 변경하여 변수를 사용할 수 있습니다.

다음 테이블은 변수를 사용하여 복잡한 식을 간소화하고 임시로 데이터를 저장하는 방법을 보여줍니다.

포트	값
V_CONDITION1	JOB_STATUS = 'Full-time'
V_CONDITION2	OFFICE_ID = 1000
AVG_SALARY	AVG(SALARY, (V_CONDITION1 AND V_CONDITION2))
SUM_SALARY	SUM(SALARY, (V_CONDITION1 AND V_CONDITION2))

행의 값 저장

소스 행의 데이터를 저장하도록 변환의 변수를 구성할 수 있습니다. 변환 식에서 변수를 사용할 수 있습니다.

예를 들어 소스 파일에 다음과 같은 행이 있습니다.

```
California  
California  
California  
Hawaii
```



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

각 행은 상태를 포함합니다. 행 수를 카운트하고 각 상태에 대해 행 카운트를 반환해야 합니다.

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

상태별로 소스 행을 그룹화하고 각 그룹의 행 수를 카운트하도록 집계 변환을 구성할 수 있습니다. 행 카운트를 저장하도록 집계 변환의 변수를 구성합니다. 다른 변수를 정의하여 이전 행의 상태 이름을 저장합니다.

집계 변환에는 다음 포트가 있습니다.

포트	포트 유형	식	설명
상태	통과	n/a	상태의 이름입니다. 소스 행은 상태 이름에 따라 그룹화됩니다. 집계 변환이 각 상태에 대해 하나의 행을 반환합니다.
State_Count	변수	IIF (PREVIOUS_STATE = STATE, STATE_COUNT +1, 1)	현재 State의 행 카운트입니다. 현재 State 열의 값이 Previous_State 열과 동일한 경우 통합 서비스가 State_Count를 증분합니다. 그렇지 않은 경우 State_Count를 1로 재설정합니다.
Previous_State	변수	주	이전 행의 State 열 값입니다. 통합 서비스가 행을 처리할 때 State 값이 Previous_State로 이동합니다.
State_Counter	출력	State_Count	상태에 대해 처리된 집계 변환의 행 수입니다. 통합 서비스는 각 상태에 대해 State_Counter를 한 번 반환합니다.

저장 프로시저의 값 캡처

변수를 사용하면 저장 프로시저에서 여러 열의 반환 값을 캡처할 수 있습니다.

변수 포트 구성 지침

변환에서 변수 포트를 구성할 경우 다음 요소를 고려하십시오.

- **포트 순서.** 통합 서비스는 포트를 종속성에 따라 평가합니다. 변환의 포트 순서는 평가 순서와 일치해야 합니다. 즉 입력 포트, 변수 포트, 출력 포트 순서입니다.
- **데이터 유형.** 선택한 데이터 유형에 따라 입력한 식의 반환 값이 반영됩니다.
- **변수 초기화.** 통합 서비스는 변수 포트에 초기 값을 설정하므로 변수 포트에서 카운터를 작성할 수 있습니다.

포트 순서

통합 서비스는 먼저 입력 포트를 평가합니다. 통합 서비스는 변수를 그 다음에 평가하고 출력 포트를 마지막으로 평가합니다.

통합 서비스는 다음 순서로 포트를 평가합니다.

1. **입력 포트.** 통합 서비스는 입력 포트가 다른 포트에 종속되지 않기 때문에 첫 번째로 모든 입력 포트를 평가합니다. 따라서 원하는 순서로 입력 포트를 작성할 수 있습니다. 통합 서비스는 입력 포트가 다른 포트를 참조하지 않기 때문에 입력 포트의 순서를 지정하지 않습니다.

2. **변수 포트.** 변수 포트는 입력 포트 및 변수 포트를 참조할 수 있지만 출력 포트는 참조할 수 없습니다. 변수 포트가 입력 포트를 참조할 수 있기 때문에 통합 서비스는 입력 포트 다음에 변수 포트를 평가합니다. 변수가 다른 변수를 참조할 수 있으므로 변수 포트의 표시 순서는 통합 서비스가 각 변수를 평가하는 순서와 동일합니다.
예를 들어 빌딩의 원래 값을 계산한 다음 감가상각에 대해 적용하는 경우 원래 값 계산을 변수 포트에 작성할 수 있습니다. 이 변수 포트는 감가상각에 적용하는 포트 전에 나타나야 합니다.
3. **출력 포트.** 출력 포트가 입력 포트 및 변수 포트를 참조할 수 있기 때문에 통합 서비스는 출력 포트를 마지막으로 평가합니다. 출력 포트는 다른 출력 포트를 참조할 수 없기 때문에 출력 포트의 표시 순서는 중요하지 않습니다. 출력 포트가 포트의 목록 아래에 표시되는지 확인합니다.

데이터 유형

포트를 변수로 구성하는 경우 모든 식 또는 조건을 포트에 입력할 수 있습니다. 이 포트에 대한 데이터 유형은 입력한 식의 반환 값에 따라 선택합니다. 변수 포트를 통해 조건을 지정하는 경우 데이터 유형이 숫자인 모든 포트는 TRUE(0이 아닌 경우) 및 FALSE(0인 경우)의 값을 반환합니다.

변수 초기화

통합 서비스는 변수의 초기 값을 NULL로 설정하지 않습니다.

통합 서비스는 변수의 초기 값을 설정하기 위해 다음 지침을 사용합니다.

- 숫자 포트의 경우 0
- 문자열 포트의 경우 빈 문자열
- 날짜/시간 포트의 경우 01/01/0001

그러므로 초기 값이 필요한 카운터로 변수를 사용합니다. 예를 들어 다음 식을 사용하여 숫자 변수를 작성할 수 있습니다.

```
VAR1 + 1
```

이 식은 VAR1 포트의 행 수를 카운트합니다. 변수의 초기 값이 NULL로 설정되면 식이 항상 NULL로 평가됩니다. 그래서 초기 값이 0으로 설정되는 것입니다.

포트의 기본값

모든 변환은 기본값을 사용하여 입력 Null 값 및 출력 변환 오류에 대한 통합 서비스의 처리 방식을 결정합니다.

입력, 출력 및 입력/출력 포트에는 시스템 기본값이 있으며 이러한 기본값은 사용자 정의 기본값으로 재정의할 수 있습니다. 기본값의 함수는 포트 유형에 따라 다릅니다.

- **입력 포트.** Null 입력 포트에 대한 시스템 기본값은 NULL입니다. 기본값은 변환에서 공백으로 표시됩니다. 입력 값이 NULL인 경우 통합 서비스가 NULL로 유지합니다.
- **출력 포트.** 출력 변환 오류에 대한 시스템 기본값은 ERROR입니다. 기본값은 변환에서 ERROR(“변환 오류”)로 표시됩니다. 변환 오류가 발생하면 통합 서비스가 행을 건너뛸 것입니다. 통합 서비스는 ERROR 함수가 건너뛰는 모든 입력 행을 로그 파일에 기록합니다.

다음 오류는 변환 오류입니다.

- 숫자를 날짜 함수에 전달하는 것과 같은 데이터 변환 오류.
- 0으로 나누는 것과 같은 식 평가 오류.

- ERROR 함수에 대한 호출.

- **통과 포트.** Null 입력에 대한 시스템 기본값은 입력 포트와 동일한 NULL입니다. 시스템 기본값은 변환에서 공백으로 표시됩니다. 출력 변환 오류에 대한 기본값은 출력 포트와 동일합니다. 출력 변환 오류에 대한 기본값은 변환에 표시되지 않습니다.

참고: Java 변환은 Java 변환 포트 유형에 따라 PowerCenter® 데이터 유형을 Java 데이터 유형으로 변환합니다. Null 입력에 대한 기본값은 Java 데이터 유형에 따라 다릅니다.

다음 표에는 연결된 변환의 포트에 대한 시스템 기본값이 표시됩니다.

포트 유형	기본값	통합 서비스 동작	지원되는 사용자 정의 기본값
입력, 통과	NULL	통합 서비스가 모든 입력 Null 값을 NULL로 전달합니다.	입력, 입력/출력
출력, 통과	ERROR	통합 서비스가 출력 포트 변환 오류에 대해 ERROR 함수를 호출합니다. 통합 서비스는 오류가 있는 행을 건너뛰고 입력 데이터 및 오류 메시지를 로그 파일에 기록합니다.	출력

변수 포트는 기본값을 지원하지 않습니다. 통합 서비스는 데이터 유형에 따라 변수 포트를 초기화합니다.

일부 기본값을 재정의하여 Null 입력 값 및 출력 변환 오류 발생 시의 통합 서비스 동작을 변경할 수 있습니다.

사용자 정의 기본값

연결된 변환 내의 지원되는 입력, 통과 및 출력 포트에 대해 사용자 정의 기본값을 사용하여 시스템 기본값을 재정의할 수 있습니다.

포트의 시스템 기본값을 재정의하려면 다음 규칙 및 지침을 사용합니다.

- **입력 포트.** 통합 서비스가 NULL 값을 NULL로 처리하지 않게 하려는 경우 입력 포트에 대해 사용자 정의 기본값을 입력할 수 있습니다. 입력 포트에 NULL가 전달되면 통합 서비스가 NULL를 기본값으로 바꿉니다.
- **출력 포트.** 통합 서비스가 행을 건너뛰지 않게 하거나 건너뀀 행과 함께 특정 메시지를 로그에 쓰게 하려는 경우 출력 포트에 사용자 정의 기본값을 입력할 수 있습니다. 출력 포트에 기본값이 정의된 경우 출력 포트에서 변환 오류가 발생하면 통합 서비스가 행을 기본값으로 바꿉니다.
- **통과 포트.** 통합 서비스가 NULL 값을 NULL로 처리하지 않게 하려는 경우 통과 포트에 대해 사용자 정의 기본값을 입력할 수 있습니다. 통과 포트의 출력 변환 오류에 대해서는 사용자 정의 기본값을 입력할 수 없습니다.

참고: 통합 서비스는 연결되지 않은 변환의 사용자 정의 기본값을 무시합니다. 예를 들어 조회 또는 저장 프로시저 변환을 식을 통해 호출하는 경우 통합 서비스는 사용자 정의 기본값을 무시하고 시스템 기본값을 적용합니다.

다음 옵션을 사용하여 사용자 정의 기본값을 입력합니다.

- **상수 값.** NULL을 포함하여 상수(숫자 또는 텍스트)를 사용합니다.
- **상수 식.** 상수 매개 변수를 사용하는 변환 함수를 포함할 수 있습니다.
- **ERROR.** 변환 오류를 생성합니다. 세션 로그 또는 행 오류 로그에 행 및 메시지를 씁니다.
- **ABORT.** 세션을 중단합니다.

상수 값

모든 상수 값을 기본값으로 입력할 수 있습니다. 상수 값은 포트의 데이터 유형과 일치해야 합니다.

예를 들어 숫자 포트의 기본값은 숫자 상수여야 합니다. 일부 상수 값은 다음과 같습니다.

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

상수 식

상수 식은 집계 함수를 제외한 변환 함수를 사용하여 상수 식을 작성하는 식입니다. 상수 식에서 입력, 입력/출력 또는 변수 포트의 값을 사용할 수 없습니다.

일부 유효한 상수 식에는 다음이 포함됩니다.

```
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
```

통합 서비스가 세션을 초기화할 경우 전체 매핑에 대한 기본값을 할당하기 때문에 식 내에서 포트의 값을 사용할 수 없습니다.

다음 예제에서는 포트의 값을 사용하기 때문에 올바르지 않습니다.

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

참고: 기본값 식에서 저장 프로시저 또는 조회 테이블을 호출할 수 없습니다.

ERROR 및 ABORT 함수

ERROR 및 ABORT 함수를 입력 및 출력 포트의 기본값으로 사용하고 입력/출력 포트의 입력 값으로 사용할 수 있습니다. 통합 서비스가 ERROR 함수를 만나면 행을 건너뛵니다. ABORT 함수를 만나면 세션을 중단합니다.

사용자 정의 기본 입력값

통합 서비스가 NULL 값을 NULL로 처리하지 않게 하려는 경우 사용자 정의 기본 입력 값을 입력할 수 있습니다.

NULL 값을 재정의하려면 다음 태스크 중 하나를 완료하십시오.

- NULL 값을 상수 값 또는 상수 식으로 바꿉니다.
- ERROR 함수를 사용하여 NULL 값을 건너뛵니다.
- ABORT 함수를 사용하여 세션을 중단합니다.

다음 테이블에는 통합 서비스가 입력 및 입력/출력 포트에 대해 NULL 입력을 처리하는 방식이 요약되어 있습니다.

기본값	기본값 유형	설명
NULL(빈 상태로 표시)	시스템	통합 서비스가 NULL을 전달합니다.
상수 또는 상수 식	사용자 정의	통합 서비스가 NULL 값을 상수 또는 상수 식 값으로 바꿉니다.

기본값	기본값 유형	설명
오류	사용자 정의	통합 서비스가 변환 오류로 처리합니다. - 변환 오류 카운트를 1씩 늘립니다. - 행을 건너 뛰고 오류 메시지를 로그 파일 또는 행 오류 로그에 씁니다. 통합 서비스가 거부 파일에 행을 쓰지 않습니다.
ABORT	사용자 정의	통합 서비스에서 Null 입력 값이 발생하면 세션이 중단됩니다. 통합 서비스가 오류 카운트를 늘리거나 행을 거부 파일에 쓰지 않습니다.

Null 값 바꾸기

상수 값 또는 식을 사용하여 포트의 Null 값을 지정된 값으로 대체할 수 있습니다.

예를 들어 입력 문자열 포트의 이름이 DEPT_NAME이고 Null 값을 문자열 'UNKNOWN DEPT'로 바꾸려는 경우 기본값을 'UNKNOWN DEPT'로 설정할 수 있습니다. 변환에 따라 통합 서비스가 'UNKNOWN DEPT'를 변환 내의 변수 또는 식으로 전달하거나 데이터 흐름의 다음 변환으로 전달합니다.

예를 들어 통합 서비스가 포트의 모든 Null 값을 문자열 'UNKNOWN DEPT'로 바꿉니다.

DEPT_NAME	REPLACED VALUE
Housewares	Housewares
NULL	UNKNOWN DEPT
Produce	Produce

Null 레코드 건너뛰기

null 값이 변환으로 전달되지 않도록 하려면 ERROR 함수를 기본값으로 사용합니다. DEPT_NAME의 입력 값이 NULL일 때 행을 건너뛰려는 경우를 예로 들어 보겠습니다. 다음 식을 기본값으로 사용할 수 있습니다.

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

ERROR 함수를 기본값으로 사용할 때 통합 서비스는 null 값이 포함된 행을 건너뛵니다. 통합 서비스는 ERROR 함수가 건너뛴 모든 행을 로그 파일에 씁니다. 그러나 이러한 행을 거부 파일에 쓰지는 않습니다.

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

다음 로그는 통합 서비스가 null 값이 포함된 행을 건너뛰는 경우를 보여 줍니다.

```
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"
)
```

세션 정보

통합 서비스에 Null 입력 값이 발생할 경우 세션을 중단하려면 ABORT 함수를 사용하십시오.

사용자 정의 기본 출력 값

사용자 정의 기본값을 생성하여 출력 포트에 대한 시스템 기본값을 재정의할 수 있습니다.

통합 서비스에서 오류가 발생한 행을 건너뛰지 않도록 하거나 건너뛴 행이 포함된 특정 메시지를 로그에 쓰도록 하려는 경우 출력 포트에 대해 사용자 정의 기본값을 입력할 수 있습니다. 통합 서비스에서 출력 변환 오류가 발생하면 기본값을 입력하여 다음 함수를 완료할 수 있습니다.

- 오류는 상수 값이나 상수 식으로 바꾸십시오. 통합 서비스에서 행을 건너뛰지 않습니다.
- ABORT 함수를 사용하여 세션을 중단합니다.
- 변환 오류에 대한 구체적인 메시지를 로그에 씁니다.

입력/출력 포트에 대해서는 사용자 정의 기본 출력 값을 입력할 수 없습니다.

다음 테이블에는 통합 서비스가 변환의 출력 포트 변환 오류와 기본값을 처리하는 방법이 요약되어 있습니다.

기본값	기본값 유형	설명
변환 오류	시스템	기본값을 재정의하지 않은 상태에서 변환 오류가 발생하면 통합 서비스는 다음 태스크를 수행합니다. <ul style="list-style-type: none">- 변환 오류 카운트를 1씩 늘립니다.- 행을 건너뛰고 세션 구성에 따라 오류 및 입력 행을 세션 로그 파일이나 행 오류 로그에 씁니다. 통합 서비스가 행을 거부 파일에 쓰지 않습니다.
상수 또는 상수 식	사용자 정의	통합 서비스가 오류를 기본값으로 바꿉니다. 통합 서비스가 오류 수를 늘리거나 메시지를 로그에 쓰지 않습니다.
ABORT	사용자 정의	세션이 중단되며 통합 서비스가 메시지를 세션 로그에 씁니다. 통합 서비스가 오류 카운트를 늘리거나 행을 거부 파일에 쓰지 않습니다.

대체 오류

변환 오류 발생 시 통합 서비스가 행을 건너뛰지 않도록 하려면 출력 포트의 기본값으로 상수 또는 상수 식을 사용합니다.

예를 들어 NET_SALARY라는 숫자 출력 포트가 있는 경우 변환 오류 발생 시 상수 값 '9999'를 사용하려면 NET_SALARY 포트에 기본값 9999를 할당합니다. NET_SALARY의 값을 계산하는 동안 0으로 나누는 등의 변환 오류가 발생하면 통합 서비스는 기본값인 9999를 사용합니다.

세션 중단

변환 오류를 허용하지 않으려는 경우 ABORT 함수를 출력 포트의 기본값으로 사용합니다.

세션 로그 또는 행 오류 로그에 메시지 쓰기

통합 서비스가 특정 메시지를 세션 로그에 기록하고 행을 건너뛰도록 하려는 경우, 사용자 정의 기본값을 출력 포트에 입력할 수 있습니다. 시스템 기본값은 ERROR ('변환 오류')이며, 통합 서비스는 세션 로그에 'transformation error' 메시지를 기록하고 행을 건너뛵니다. 다른 메시지를 쓰려는 경우 '변환 오류'를 바꿀 수 있습니다.

행 오류 로깅을 활성화한 경우, 통합 서비스는 세션 로그 대신 오류 로그에 오류 메시지를 기록하며 트랜잭션 제어 변환 롤백 또는 커밋 오류를 기록하지 않습니다. 행을 세션 로그 외에 행 오류 로그에도 기록하려면 자세한 정보 표시 데이터 추적을 활성화합니다.

출력 포트 식의 ERROR 함수

ERROR 함수를 사용하는 식을 입력할 경우 출력 포트의 사용자 정의 기본값이 식의 ERROR 함수를 재정의할 수 있습니다.

예를 들어, 통합 서비스에 오류가 발생할 경우 'Negative Sale' 값을 사용하도록 지시하는 다음 식을 입력합니다.

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

다음 예제에서는 사용자 정의 기본값이 식의 ERROR 함수를 재정의하는 방법을 보여 줍니다.

- **상수 값 또는 식.** 상수 값 또는 식이 출력 포트 식의 ERROR 함수를 재정의합니다.

예를 들어, '0'을 기본값으로 입력할 경우 통합 서비스에서 출력 포트 식의 ERROR 함수를 재정의합니다. 통합 서비스에서 오류가 발생할 경우 값 0을 전달합니다. 행을 건너뛰거나 로그에 'Negative Sale'을 기록하지 않습니다.

- **ABORT.** ABORT 함수가 출력 포트 식의 ERROR 함수를 재정의합니다.

ABORT 함수를 기본값으로 사용할 경우 변환 오류가 발생하면 통합 서비스가 중단됩니다. ABORT 함수가 출력 포트 식의 ERROR 함수를 재정의합니다.

- **ERROR.** ERROR 함수를 기본값으로 사용할 경우 통합 서비스의 로그에 다음 정보가 포함됩니다.

- 기본값의 오류 메시지
- 출력 포트 식의 ERROR 함수에 표시된 오류 메시지
- 건너뛴 행

예를 들어, 다음 ERROR 함수로 기본값을 재정의할 수 있습니다.

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

통합 서비스에서 행을 건너뛰고, 두 오류 메시지를 로그에 포함합니다.

```
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')
```

기본값에 대한 일반 규칙

기본값을 작성할 경우 다음 규칙 및 지침을 사용하십시오.

- 기본값은 NULL, 상수 값, 상수 식, ERROR 함수 또는 ABORT 함수여야 합니다.
- 입력/출력 포트의 경우 통합 서비스에서 기본값을 사용하여 Null 입력 값을 처리합니다. 입력/출력 포트의 출력 기본값은 항상 ERROR('변환 오류')입니다.
- 변수 포트에서는 기본값을 사용하지 않습니다.
- 집계 및 순위 변환에서 그룹 기준 포트에 기본값을 할당할 수 있습니다.
- 모든 변환의 일부 포트 유형에서만 사용자 정의 기본값을 허용합니다. 포트에서 사용자 정의 기본값을 허용하지 않을 경우 기본값 필드가 비활성화됩니다.
- 일부 변환에서만 사용자 정의 기본값을 허용합니다.
- 변환이 매핑 데이터 흐름에 연결되지 않은 경우 통합 서비스에서 사용자 정의 기본값을 무시합니다.
- 입력 포트가 연결되지 않은 경우 해당 값은 NULL로 추정되고 통합 서비스가 해당 입력 포트에 대한 기본값을 사용합니다.
- 입력 포트 기본값에 ABORT 함수가 포함되고 입력 값이 NULL일 경우 통합 서비스가 즉시 세션을 중지합니다. ABORT 함수를 기본값으로 사용하여 Null 입력 값을 제한합니다. 입력 포트의 첫 번째 Null 값에서 세션을 중지합니다.

- 출력 포트 기본값에 **ABORT** 함수가 포함되고 해당 포트에 대한 변환 오류가 발생할 경우, 세션이 즉시 중지됩니다. 변환 오류에 대해 엄격한 규칙을 적용하려면 **ABORT** 함수를 기본값으로 사용하십시오. 이 포트에 대한 첫 번째 변환 오류에서 세션을 중지합니다.
- ABORT** 함수, 상수 값 및 상수 식이 출력 포트 식에 구성된 **ERROR** 함수를 재정의합니다.

기본값 유효성 검사

기본값을 입력할 때 기본값의 유효성을 검사할 수 있습니다. 디자이너에는 유효한 기본값을 보장할 수 있는 유효성 검사 단추가 있습니다. 기본값이 유효한지 표시하는 메시지가 나타납니다.

매핑을 저장할 때에도 디자이너에서 기본값 유효성을 검사합니다. 잘못된 기본값을 입력할 경우 디자이너에서 매핑을 올바르게 않은 매핑으로 표시합니다.

변환의 추적 수준 구성

변환을 구성할 때 통합 서비스가 세션 로그에 기록하는 세부 정보의 양을 설정할 수 있습니다.

다음 테이블에는 세션 로그 추적 수준이 설명되어 있습니다.

추적 수준	설명
보통	통합 서비스가 변환 행 오류로 인한 초기화/상태 정보, 발생한 오류 및 건너뛴 행을 기록합니다. 세션 결과를 요약하지만 개별 행 수준이 아닙니다.
간단	통합 서비스가 초기화 정보와 거부된 데이터에 대한 알림 및 오류 메시지를 기록합니다.
자세한 정보 표시 초기화	일반 추적과 함께 통합 서비스가 추가적인 초기화 세부 정보, 사용된 인덱스 및 데이터 파일의 이름 및 자세한 변환 통계를 기록합니다.
자세한 정보 표시 데이터	자세한 정보 표시 초기화 추적과 함께 통합 서비스가 매핑으로 전달되는 각 행을 기록합니다. 데이터 통합 서비스가 열의 전체 자릿수에 맞춰 문자열 데이터를 자르는 위치를 기록하고 자세한 변환 통계를 제공합니다. 행 오류 로깅을 활성화하면 통합 서비스가 세션 로그와 오류 로그 모두에 오류를 기록할 수 있습니다. 추적 수준을 자세한 정보 표시 데이터로 구성하면 통합 서비스가 변환을 처리할 때 블록의 모든 행에 대해 행 데이터를 씁니다.

기본적으로 모든 변환의 추적 수준은 보통입니다. 예상대로 작동하지 않는 변환을 디버깅해야 하는 경우에만 추적 수준을 자세한 정보 표시 설정으로 변경합니다. 또한 추적 수준을 간단으로 설정하면 변환이 포함된 워크플로우를 실행할 때 최소한의 세부 정보를 세션 로그에 기록하여 성능을 약간 향상시킬 수 있습니다.

세션을 구성할 때 세션의 모든 변환에 대해 단일 추적 수준을 사용하여 개별 변환의 추적 수준을 재정의할 수 있습니다.

재사용 가능 변환

매핑은 재사용 가능 변환과 재사용 불가능 변환을 포함할 수 있습니다. 재사용 불가능 변환은 단일 매핑에만 존재합니다. 재사용 가능 변환은 여러 매핑에서 사용될 수 있습니다.

예를 들어 캐나다에서의 비즈니스 활동 비용을 분석하는 데 도움이 되도록 판매 부가 가치를 계산하는 식 변환을 작성할 수 있습니다. 동일한 작업을 매번 수행하기 보다 재사용 가능한 변환을 작성할 수 있습니다. 이러한 변환을 매핑에 통합해야 하는 경우 변환의 인스턴스를 매핑에 추가합니다. 나중에 변환의 정의를 변경하면 변환의 모든 인스턴스가 변경 내용을 상속합니다.

재사용 가능 변환을 저장할 때 디자이너는 해당 변환을 사용하는 매핑과 별개의 메타데이터로 각 변환을 저장합니다. 탐색기에서 폴더의 콘텐츠를 검토하면 해당 폴더에 있는 모든 재사용 가능 변환의 목록이 표시됩니다.

각각의 재사용 가능 변환은 디자이너에서 사용할 수 있는 변환의 범주에 속합니다. 예를 들어 재사용 가능 집계 변환을 작성하여 여러 매핑에서 동일한 집계 계산을 수행하거나, 재사용 가능 저장 프로시저 변환을 작성하여 여러 매핑에서 동일한 저장 프로시저를 호출할 수 있습니다.

대부분의 변환을 재사용 불가능 변환 또는 재사용 가능 변환으로 작성할 수 있습니다. 그러나 외부 프로시저 변환은 재사용 가능 변환으로만 작성할 수 있습니다.

재사용 가능 변환의 인스턴스를 매핑에 추가하는 경우, 변환 변경으로 인해 매핑이 무효화되거나 예상치 않은 데이터가 생성되지 않도록 주의해야 합니다.

인스턴스 및 상속된 변경 내용

재사용 가능 변환을 매핑에 추가하면 변환의 인스턴스가 추가됩니다. 변환의 정의는 계속 매핑 외부에 있는 반면 변환 인스턴스는 매핑 내부에 표시됩니다.

재사용 가능 변환의 인스턴스는 해당 변환에 대한 포인터이므로 변환 개발자에서 변환을 변경하면 해당 인스턴스에 이러한 변경 사항이 반영됩니다. 변환을 사용하는 모든 매핑에서 동일한 매핑을 업데이트하지 않고 재사용 가능 변환을 한 번 업데이트하면 변환의 모든 인스턴스에서 변경 사항을 상속합니다. 인스턴스는 속성 설정의 변경 사항을 상속하지 않고 포트, 식 및 변환 이름의 수정 사항만 상속합니다.

식의 매핑 변수

재사용 가능 변환 식에서 매핑 매개 변수 및 변수를 사용합니다. 디자이너는 매개 변수 또는 변수의 유효성을 검사할 때 정수 데이터 유형으로 처리합니다. 맵렛 또는 매핑에서 변환을 사용하면 디자이너가 다시 식의 유효성을 검사합니다. 매핑 매개 변수 또는 변수가 맵렛 또는 매핑에 없을 경우 디자이너가 오류를 기록합니다.

재사용 가능 변환 작성

다음과 같은 방법을 사용하여 재사용 가능 변환을 작성할 수 있습니다.

- **변환 개발자에서 재사용 가능 변환 디자인.** 변환 개발자에서 새로운 재사용 가능 변환을 빌드할 수 있습니다.
- **매핑 디자이너에서 재사용 불가능 변환 승격.** 변환을 매핑에 추가한 후에 재사용 가능 변환 상태로 이 매핑을 승격시킬 수 있습니다. 그런 다음 매핑에서 디자인된 변환이 리포지토리의 다른 위치에서 유지 관리되는 재사용 가능 변환의 인스턴스가 됩니다.

변환을 재사용 가능 상태로 승격시키는 경우 이 변환을 강등시킬 수 없습니다. 하지만 이 변환의 재사용 불가능 인스턴스를 작성할 수 있습니다.

참고: 시퀀스 생성기 변환은 맵렛에서 재사용 가능해야 합니다. 재사용 가능 시퀀스 생성기 변환을 맵렛에서 재사용 불가능으로 강등시킬 수 없습니다.

재사용 가능 변환을 작성하려면:

1. 디자이너에서 변환 개발자로 전환합니다.
2. 작성할 변환 유형에 해당하는 변환 도구 모음에서 단추를 클릭합니다.
3. 통합 문서 내에서 끌어서 변환을 작성합니다.
4. 변환 제목 표시줄을 두 번 클릭하여 해당 속성이 표시되는 대화 상자를 엽니다.

- 이름 바꾸기 단추를 클릭하고 변환에 대한 설명 이름을 입력하고 확인을 클릭합니다.
- 포트 탭을 클릭한 다음 이 변환에 필요한 입력 및 출력 포트를 추가합니다.
- 변환의 기타 속성을 설정하고 확인을 클릭합니다.

이러한 속성은 사용자가 작성하는 변환에 따라 다릅니다. 예를 들어 식 변환을 작성하는 경우 하나 이상의 변환 출력 포트에 대한 식을 입력해야 합니다. 저장 프로시저 변환을 작성하는 경우 호출할 저장 프로시저를 식별해야 합니다.

재사용 불가능 변환 승격

재사용 가능 변환을 작성하기 위한 다른 기법은 매핑 내의 기존 변환을 승격시키는 것입니다. 변환 편집 대화 상자에서 재사용 가능으로 지정 옵션을 선택하면 디자이너가 변환을 승격시키고 매핑에서 해당 변환의 인스턴스를 작성하게 됩니다.

재사용 불가능 변환을 승격시키려면 다음을 수행하십시오.

- 디자이너에서 매핑을 열고 승격시키려는 변환의 제목 표시줄을 두 번 클릭합니다.
- 재사용 가능으로 지정 옵션을 선택합니다.
- 변환을 승격시키려는지 확인하는 메시지가 표시되면 예를 클릭합니다.
- 확인을 클릭하여 매핑으로 돌아갑니다.

이제 작업 중인 폴더에서 재사용 가능 변환의 목록을 살펴보면 새로 승격된 변환이 이 목록에 나타납니다.

재사용 가능 변환의 재사용 불가능 인스턴스 작성

매핑 내에서 재사용 가능 변환의 재사용 불가능 인스턴스를 작성할 수 있습니다. 재사용 변환을 동일한 폴더 내에서 재사용 불가능으로 만들어야 합니다. 재사용 가능 변환의 재사용 불가능 인스턴스를 다른 폴더에 만들려면 먼저 변환의 재사용 불가능 인스턴스를 소스 폴더에서 만든 후 대상 폴더로 복사해야 합니다.

재사용 가능 변환의 재사용 불가능 인스턴스를 작성하려면

- 디자이너에서 매핑을 엽니다.
- 탐색기에서 기존 변환을 선택하고 변환을 매핑 작업 공간으로 끌어서 놓습니다. 변환을 해제하기 전에 Ctrl 키를 누른 상태로 있습니다.

상태 표시줄에 다음과 같은 메시지가 표시됩니다.

Make a non-reusable copy of this transformation and add it to this mapping.

- 변환을 해제합니다.

디자이너가 기존 재사용 가능 변환의 재사용 불가능 인스턴스를 작성합니다.

매핑에 재사용 가능 변환 추가

재사용 가능 변환을 작성한 후에 매핑에 해당 변환을 추가할 수 있습니다.

재사용 가능 변환을 추가하려면:

- 디자이너에서 매핑 디자이너로 전환합니다.
- 매핑을 열거나 작성합니다.
- 리포지토리 개체 목록에서, 원하는 재사용 가능 변환을 찾을 때까지 폴더의 변환 섹션에서 드릴다운합니다.
- 탐색기에서 매핑으로 변환을 끕니다.

재사용 가능 변환의 사본(또는 인스턴스)이 나타납니다.

- 새로운 변환을 기타 변환 또는 대상 정의에 연결합니다.

재사용 가능 변환 수정

변환 개발자를 통해 입력하는 재사용 가능 변환에 대한 변경 내용은 즉시 해당 변환의 모든 인스턴스에 반영됩니다. 이 기능은 작업을 저장하고 표준을 적용하는 강력한 방법이지만, 재사용 가능 변환을 수정하는 경우 매핑을 무효화할 위험이 있습니다.

사용자가 변환에 대해 변경한 내용이 어떤 매핑, 탭셋 또는 바로 가기에 영향을 미칠 수 있는지 보려면 작업 공간이나 탐색기에서 변환을 선택하고, 마우스 오른쪽 단추로 클릭하고, 종속성 보기를 선택합니다.

재사용 가능 변환에 대해 다음과 같이 변경할 경우 이 변환의 인스턴스를 사용하는 매핑이 무효화될 수 있습니다.

- 변환에서 하나의 포트나 여러 포트를 삭제하면 매핑을 통한 데이터 흐름의 일부 또는 전부에서 인스턴스의 연결이 끊길 수 있습니다.
- 포트 데이터 유형을 변경하는 경우 해당 포트에서 호환되지 않는 데이터 유형을 사용하는 다른 포트에 데이터를 매핑할 수 없게 됩니다.
- 포트 이름을 변경하는 경우 해당 포트를 참조하는 식이 더 이상 유효하지 않습니다.
- 재사용 가능 변환에서 유효하지 않은 식을 입력하는 경우 이 변환을 사용하는 매핑이 더 이상 유효하지 않습니다. 통합 서비스는 유효하지 않은 매핑을 기반으로 세션을 실행할 수 없습니다.

원래 상태의 재사용 가능 변환으로 되돌리기

매핑에서 재사용 가능 변환의 속성을 변경한 경우, 되돌리기 단추를 클릭하여 원래 상태의 재사용 가능 변환 속성으로 되돌릴 수 있습니다.

제 2 장

집계 변환

이 장에 포함된 항목:

- [집계 변환 개요, 48](#)
- [집계 변환의 구성 요소, 48](#)
- [집계 캐시 구성, 50](#)
- [집계 식, 50](#)
- [그룹 기준 포트, 51](#)
- [정렬된 입력 사용, 53](#)
- [집계 변환 작성, 55](#)
- [집계 변환에 대한 팁, 55](#)
- [집계 변환 문제 해결, 56](#)

집계 변환 개요

집계 변환은 평균 및 합계와 같은 집계 계산을 수행합니다. 통합 서비스는 데이터 그룹 및 행 데이터를 읽고 집계 캐시에 저장할 때 집계 계산을 수행합니다. 집계 변환은 활성 변환입니다.

집계 변환은 식 변환과 다릅니다. 즉 그룹에 대한 계산을 수행하려면 집계 변환을 사용하십시오. 식 변환을 통해 행별로 계산을 수행할 수 있습니다.

변환 언어를 사용하여 집계 식을 작성할 경우 SQL 언어보다 더 많은 유연성을 제공하는 조건부 절을 사용하여 행을 필터링할 수 있습니다.

집계 변환이 포함된 세션을 작성한 후 세션 옵션인 증분 집계를 활성화할 수 있습니다. 통합 서비스는 증분 집계를 수행할 때 매핑을 통해 소스 데이터를 전달하고 기록 캐시 데이터를 사용하여 증분 방식으로 집계 계산을 수행합니다.

집계 변환의 구성 요소

집계는 파이프라인에서 행의 수를 변경하는 활성 변환입니다. 집계 변환에는 다음과 같은 구성 요소와 옵션이 있습니다.

- **집계 캐시.** 통합 서비스는 집계 계산을 완료할 때까지 데이터를 집계 캐시에 저장합니다. 통합 서비스는 그룹 값을 인덱스 캐시에 저장하고 행 데이터를 데이터 캐시에 저장합니다.

- **집계 식.** 출력 포트에 식을 입력합니다. 식에 비집계 식과 조건절을 포함할 수 있습니다.
- **그룹 기준 포트.** 그룹을 작성하는 방법을 나타냅니다. 그룹에 대한 입력, 입력/출력, 출력 또는 변수 포트를 구성할 수 있습니다. 데이터를 그룹화하면 별도로 지정하지 않는 한 집계 변환이 각 그룹의 마지막 행을 출력합니다.
- **정렬된 입력.** 세션 성능을 향상시키려면 이 옵션을 선택합니다. 정렬된 입력을 사용하려면 그룹 기준 포트에 의해 오름차순 또는 내림차순으로 정렬된 집계 변환에 데이터를 전달해야 합니다.

속성 및 포트 탭에서 집계 변환 구성 요소 및 옵션을 구성할 수 있습니다.

집계 변환 속성 구성

속성 탭에서 집계 변환 속성을 수정합니다.

다음 옵션을 구성합니다.

집계 설정	설명
캐시 디렉터리	통합 서비스가 인덱스 및 데이터 캐시 파일을 작성하는 로컬 디렉터리입니다. 기본적으로 통합 서비스는 워크플로우 관리자에서 프로세스 변수 \$PMCacheDir에 대해 입력된 디렉터리를 사용합니다. 새 디렉터리를 입력하는 경우 해당 디렉터리가 있고 집계 캐시에 충분한 디스크 공간을 갖고 있는지 확인합니다. 충분 집계를 활성화한 경우 세션을 실행할 때마다 통합 서비스가 파일의 백업을 작성합니다. 캐시 디렉터리에 두 세트의 파일을 유지할 충분한 디스크 공간이 있어야 합니다.
추적 수준	이 변환에 대해 세션 로그에 표시되는 세부 정보의 양입니다.
정렬된 입력	입력 데이터가 그룹을 기준으로 미리 정렬되었음을 나타냅니다. 매핑에서 정렬된 데이터를 집계 변환에 전달하는 경우에만 이 옵션을 선택합니다.
집계 데이터 캐시 크기	변환의 데이터 캐시 크기입니다. 기본 캐시 크기는 2,000,000바이트입니다. 구성된 총 세션 캐시 크기가 2GB(2,147,483,648바이트) 이상일 경우 64비트 통합 서비스에서 세션을 실행해야 합니다. 캐시에 숫자 값을 사용하거나, 매개 변수 파일의 캐시 값을 사용하거나, 자동 설정을 통해 캐시 크기를 설정하도록 통합 서비스를 구성할 수 있습니다. 캐시 크기를 결정하도록 통합 서비스를 구성하는 경우 통합 서비스가 캐시에 할당하는 최대 메모리 양도 구성할 수 있습니다.
집계 인덱스 캐시 크기	변환의 인덱스 캐시 크기입니다. 기본 캐시 크기는 1,000,000바이트입니다. 구성된 총 세션 캐시 크기가 2GB(2,147,483,648바이트) 이상일 경우 64비트 통합 서비스에서 세션을 실행해야 합니다. 캐시에 숫자 값을 사용하거나, 매개 변수 파일의 캐시 값을 사용하거나, 자동 설정을 통해 캐시 크기를 설정하도록 통합 서비스를 구성할 수 있습니다. 캐시 크기를 결정하도록 통합 서비스를 구성하는 경우 통합 서비스가 캐시에 할당하는 최대 메모리 양도 구성할 수 있습니다.
변환 범위	통합 서비스가 변환 논리를 수신 데이터에 적용하는 방식을 지정합니다. <ul style="list-style-type: none"> - 트랜잭션. 트랜잭션의 모든 행에 변환 논리를 적용합니다. 데이터 행이 동일한 트랜잭션의 모든 행에 종속되지만 다른 트랜잭션의 행에는 종속되지 않는 경우 트랜잭션을 선택합니다. - 모든 입력. 변환 논리를 모든 수신 데이터에 적용합니다. 모든 입력을 선택하면 PowerCenter가 수신 트랜잭션 경계를 삭제합니다. 데이터 행이 소스의 모든 행에 종속되는 경우 모든 입력을 선택합니다.

집계 변환 포트 구성

집계 변환의 포트를 구성하려면 다음 태스크를 수행하십시오.

- 출력 포트에 조건절 또는 비집계 함수를 사용하여 포트에 식을 입력합니다.

- 여러 개의 집계 출력 포트를 작성합니다.
- 입력, 입력/출력, 출력 또는 변수 포트를 그룹 기준 포트로 구성합니다.
- 필요한 입력/출력 포트만 이후의 변환에 연결하여 데이터 캐시의 크기를 줄여 성능을 향상시킵니다.
- 변수 포트를 로컬 변수에 사용합니다.
- 식을 입력할 때 다른 변환에 대한 연결을 작성합니다.

집계 캐시 구성

집계 변환을 사용하는 세션을 실행하면 통합 서비스는 변환을 처리하기 위해 인덱스 및 데이터 캐시를 메모리에 작성합니다. 통합 서비스는 공간이 더 필요할 경우 오버플로우 값을 캐시 파일에 저장합니다.

캐시에 숫자 값을 사용하거나, 매개 변수 파일의 캐시 값을 사용하거나, 자동 설정을 통해 캐시 크기를 설정하도록 통합 서비스를 구성할 수 있습니다.

참고: 통합 서비스는 정렬된 포트가 포함된 집계 변환을 처리하기 위해 메모리를 사용합니다. 통합 서비스는 캐시 메모리를 사용하지 않습니다. 정렬된 포트를 사용하는 집계 변환의 경우 캐시 메모리를 구성하지 않아도 됩니다.

집계 식

디자이너는 집계 변환에서만 집계 식을 허용합니다. 집계 식에 조건절 및 비집계 함수를 포함할 수 있습니다. 또한 다음과 같이 다른 집계 함수 내에 하나의 집계 함수를 포함할 수 있습니다.

`MAX(COUNT(ITEM))`

집계 식의 결과는 변환의 그룹 기준 포트에 따라 다릅니다. 예를 들어 통합 서비스는 그룹 기준 포트가 정의되지 않은 다음과 같은 집계 식을 계산하는 경우 판매된 품목의 총 수량을 찾습니다.

`SUM(QUANTITY)`

하지만 동일한 식을 사용하고 `ITEM` 포트를 기준으로 그룹화할 경우 통합 서비스는 판매된 품목의 총 수량을 품목별로 반환합니다.

모든 출력 포트에 집계 식을 작성하고 여러 집계 포트를 변환에 사용할 수 있습니다.

관련 항목:

- [“식 작업” 페이지 31](#)

집계 함수

집계 변환 내에서 다음과 같은 집계 함수를 사용합니다. 다른 집계 함수 안에 하나의 집계 함수를 중첩할 수 있습니다.

변환 언어에는 다음 집계 함수가 포함됩니다.

- AVG
- COUNT
- FIRST

- LAST
- MAX
- MEDIAN
- MIN
- PERCENTILE
- STDDEV
- SUM
- VARIANCE

이러한 함수를 사용할 때는 집계 변환 내 식에 사용해야 합니다.

중첩 집계 함수

집계 변환에서는 여러 출력 포트에 여러 단일 수준 함수나 중첩 함수를 포함할 수 있습니다. 하지만 집계 변환에 단일 수준 함수와 중첩 함수를 모두 포함할 수는 없습니다. 따라서 집계 변환의 출력 포트에 단일 수준 함수가 포함되어 있으면 해당 변환의 다른 포트에서 중첩 함수를 사용할 수 없습니다. 단일 수준 함수와 중첩 함수를 동일한 집계 변환에 포함할 경우 디자이너가 매핑 또는 맷을 잘못된 것으로 표시합니다. 단일 수준 함수와 중첩된 함수를 모두 작성해야 하는 경우 개별 집계 변환을 작성해야 합니다.

조건절

집계에 사용되는 행 수를 줄이려면 집계 식에 조건절을 사용합니다. TRUE 또는 FALSE로 평가되는 모든 절을 조건절로 사용할 수 있습니다.

예를 들어 분기별 할당량을 초과한 직원의 총 보수를 계산하려면 다음 식을 사용합니다.

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

비집계 함수

또한 집계 식에서 비집계 함수를 사용할 수 있습니다.

다음 식은 각 품목에 대해 최고 판매 개수를 반환합니다(품목별로 그룹화). 판매된 품목이 없을 경우 식은 0을 반환합니다.

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

집계 함수의 Null 값

통합 서비스를 구성할 때 통합 서비스가 집계 함수에서 null 값을 처리하는 방식을 선택할 수 있습니다. 집계 함수에서 null 값을 NULL 또는 0으로 처리하도록 선택할 수 있습니다. 기본적으로 통합 서비스는 집계 함수에서 null 값을 NULL로 처리합니다.

그룹 기준 포트

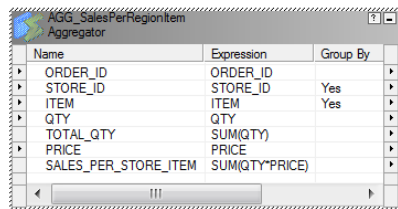
집계 변환을 사용하면 모든 입력 데이터에 집계를 수행하지 않고 집계할 그룹을 정의하여 집계를 수행할 수 있습니다. 예를 들어 회사의 전체 매출을 찾는 대신 지역별로 그룹화된 전체 매출을 찾을 수 있습니다.

집계 식의 그룹을 정의하려면 집계 변환에서 해당하는 입력, 입력/출력, 출력 및 변수 포트를 선택합니다. 여러 그룹 기준 포트를 선택하여 고유한 각 조합에 대한 새 그룹을 작성할 수 있습니다. 그러면 통합 서비스가 각 그룹에 대해 정의된 집계를 수행합니다.

값을 그룹화하면 통합 서비스가 각 그룹에 대해 행을 하나씩 생성합니다. 값을 그룹화하지 않으면 통합 서비스가 모든 입력 행에 대해 한 행을 반환합니다. 통합 서비스는 일반적으로 집계의 결과와 함께 각 그룹의 마지막 행 또는 수신한 마지막 행을 반환합니다. 하지만 **FIRST** 함수를 사용하는 등의 방식으로 특정 행을 반환하도록 지정하면 통합 서비스가 지정된 행을 반환합니다.

집계 변환에서 여러 개의 그룹 기준 포트를 선택할 경우 통합 서비스는 포트 순서를 기반으로 그룹화하는 순서를 결정합니다. 그룹 순서는 결과에 영향을 미칠 수 있으므로 적절한 그룹화가 이루어지도록 그룹 기준 포트의 순서를 지정하십시오. 예를 들어 **ITEM_ID** 다음에 **QUANTITY**를 기준으로 그룹화할 때의 결과는 수량에 대한 숫자 값이 항상 고유한 것은 아니므로 **QUANTITY** 다음에 **ITEM_ID**를 기준으로 그룹화할 때의 결과가 다를 수 있습니다.

다음 집계 변환은 먼저 **STORE_ID**를 기준으로 그룹화한 후 **ITEM**을 기준으로 그룹화합니다.



Name	Expression	Group By
ORDER_ID	ORDER_ID	
STORE_ID	STORE_ID	Yes
ITEM	ITEM	Yes
QTY	QTY	
TOTAL_QTY	SUM(QTY)	
PRICE	PRICE	
SALES_PER_STORE_ITEM	SUM(QTY*PRICE)	

이 집계 변환을 통해 다음 데이터를 전송하는 경우

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

통합 서비스는 다음과 같은 고유 그룹에 대해 집계 계산을 수행합니다.

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

그런 다음 통합 서비스는 다음과 같이 수신한 마지막 행을 집계 결과와 함께 전달합니다.

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

비집계 식

그룹 기준 포트에서 비집계 식을 사용하여 그룹을 수정하거나 바꿀 수 있습니다. 예를 들어 그룹화 전에 'AAA 건전지'를 대체하려는 경우 다음과 같은 식을 사용하여 **CORRECTED_ITEM**이라는 새로운 그룹 기준 출력 포트를 작성할 수 있습니다.

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

기본값

Null 입력 값을 대체하기 위한 그룹의 각 포트에 대한 기본값을 정의하십시오. 이렇게 하면 통합 서비스가 집계에 null 품목 그룹을 포함할 수 있습니다.

관련 항목:

- [“포트의 기본값” 페이지 38](#)

정렬된 입력 사용

정렬된 입력 옵션을 사용하여 집계 변환 성능을 향상시킬 수 있습니다. 정렬된 입력을 사용할 경우, 통합 서비스는 모든 데이터가 그룹별로 정렬된 것으로 가정하며 그룹 단위로 행을 읽으면서 집계 계산을 수행합니다. 필요에 따라 메모리에 그룹 정보를 저장합니다. 정렬된 입력 옵션을 사용하려면 정렬된 데이터를 집계 변환에 전달해야 합니다. 여러 파티션을 포함하는 세션을 구성할 경우 정렬된 포트를 통해 성능을 높일 수 있습니다.

정렬된 입력을 사용하지 않는 경우 통합 서비스는 읽으면서 집계 계산을 수행합니다. 데이터가 정렬되어 있지 않기 때문에 통합 서비스는 모든 집계 계산이 정확하도록 하기 위해 전체 소스를 읽을 때까지 각 그룹의 데이터를 저장합니다.

예를 들어 하나의 집계 변환에 정렬된 입력 옵션과 함께 **STORE_ID** 및 **ITEM** 그룹 기준 포트가 선택되어 있습니다. 집계를 통해 다음 데이터를 전달하면 통합 서비스는 새 그룹, 즉 **201/battery**를 찾는 즉시 **101/battery** 그룹의 3개 행에 대해 집계를 수행합니다.

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

STORE_ID	ITEM	QTY	PRICE
201	'battery'	1	1.99

정렬된 입력을 사용하는 경우 데이터를 올바르게 사전 분류하지 않으면 예기치 않은 결과가 발생합니다.

정렬된 입력 조건

다음 조건 중 하나에 해당되면 정렬된 입력을 사용하지 마십시오.

- 집계 식에서 중첩된 집계 함수를 사용합니다.
- 세션에서 증분 집계를 사용합니다.

정렬된 입력을 사용할 경우 데이터를 올바르게 정렬하지 않으면 세션이 실패합니다.

데이터 정렬

정렬된 입력을 사용하려면 집계를 통해 정렬된 데이터를 전달합니다.

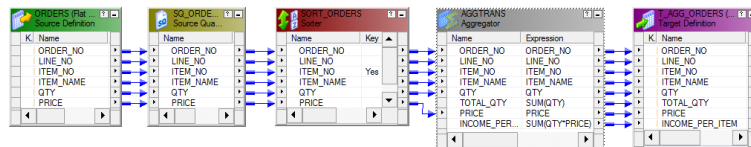
데이터는 다음과 같은 방식으로 정렬해야 합니다.

- 집계 변환에 나타나는 순서대로 집계 그룹 기준 포트별로 정렬합니다.
- 세션에 구성된 정렬 순서를 사용하여 정렬합니다. 데이터가 세션 정렬 순서를 기준으로 정확하게 오름차순이 나 내림차순으로 정렬되어 있지 않으면 통합 서비스에서 세션이 실패합니다. 예를 들어 프랑스어 정렬 순서를 사용하도록 세션을 구성한 경우 프랑스어 정렬 순서를 사용하여 집계 변환으로 전달하는 데이터를 정렬해야 합니다.

관계형 및 파일 소스의 경우 집계 변환에 전달하기 전에 분류기 변환을 사용하여 매핑의 데이터를 정렬합니다. 변환이 정렬된 데이터의 순서를 변경하지 않으면 분류기 변환을 집계 전에 매핑의 아무데나 배치할 수 있습니다. 집계 변환의 그룹 기준 열은 분류기 변환에 표시되는 것과 동일한 순서여야 합니다.

세션에서 관계형 소스를 사용하는 경우 소스 한정자 변환에서 정렬된 포트 수 옵션을 사용하여 소스 데이터베이스의 그룹 기준 열을 정렬할 수도 있습니다. 그룹 기준 열은 집계 변환과 소스 한정자 변환 모두에서 동일한 순서여야 합니다.

다음 매핑은 ITEM_NO에 따라 오름차순으로 소스 데이터를 정렬하도록 구성된 분류기 변환을 보여줍니다.



분류기 변환이 다음과 같이 데이터를 정렬합니다.

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

정렬된 입력을 사용하면 집계 변환이 다음 결과를 반환합니다.

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

집계 변환 작성

매핑에서 집계 변환을 사용하려면 집계 변환을 매핑에 추가합니다. 그런 다음 집계 식과 그룹 기준 포트를 포함하여 변환을 구성합니다.

집계 변환을 작성하려면

1. 매핑 디자이너에서 변환 > 작성을 클릭합니다. 집계 변환을 선택합니다.
2. 집계의 이름을 입력하고 작성을 클릭합니다. 그런 다음 완료를 클릭합니다.
디자이너가 집계 변환을 작성합니다.
3. 포트를 집계 변환으로 끌어서 놓습니다.
사용자가 포함하는 각 포트에 대해 디자이너가 입력/출력 포트를 작성합니다.
4. 변환의 제목 표시줄을 두 번 클릭하여 변환 편집 대화 상자를 엽니다.
5. 포트 탭을 선택합니다.
6. 집계를 그룹 작성에 사용하도록 하려는 각 열에 대해 그룹 기준 옵션을 클릭합니다.
필요한 경우 기본값을 입력하여 null 그룹을 대체합니다.
7. 추가를 클릭하여 식 포트를 추가합니다.
식 포트는 출력 포트여야 합니다. 입력(I)의 선택을 취소하여 포트를 출력 포트로 설정합니다.
8. 필요한 경우 특정 포트에 대한 기본값을 추가합니다.
대상 데이터베이스가 null 값을 처리하지 않는데 특정 포트가 null 값을 포함할 수 있을 경우 기본값을 지정합니다.
9. 속성 탭에서 속성을 구성합니다.

집계 변환에 대한 팁

정렬된 입력을 사용하여 집계 캐시의 사용을 줄입니다.

정렬된 입력은 세션 중에 캐싱되는 데이터의 양을 줄여 주고 세션 성능을 향상시킵니다. 이 옵션을 분류기 변환과 함께 사용하면 정렬된 데이터가 집계 변환으로 전달됩니다.

집계 변환이 정렬된 출력을 제공하지 않을 수도 있습니다.

집계 변환의 출력을 정렬하려면 분류기 변환을 사용합니다.

연결된 입력/출력 또는 출력 포트를 제한합니다.

연결된 입력/출력 또는 출력 포트 수를 제한하여 집계 변환이 데이터 캐시에 저장하는 데이터의 양을 줄이십시오.

데이터를 집계하기 전에 필터링합니다.

매핑에서 필터 변환을 사용하는 경우 집계 변환 앞에 변환을 배치하여 불필요한 집계를 줄이십시오.

집계 변환 문제 해결

정렬된 입력을 선택했지만 워크플로우에 소요된 시간이 이전과 동일합니다.

다음 조건 중 하나에 해당되면 정렬된 입력을 사용할 수 없습니다.

- 집계 식에 중첩된 집계 함수가 포함됩니다.
- 세션에서 증분 집계를 사용합니다.
- 소스 데이터가 데이터 기반입니다.

이러한 조건 중 하나에 해당되는 경우, 통합 서비스는 사용자가 정렬된 입력을 사용하지 않는 것처럼 변환을 처리합니다.

세션에서 집계 변환을 사용하면 속도 면에서 성능이 저하됩니다.

통합 서비스가 워크플로우 중에 디스크로 페이지징하는 것일 수 있습니다. 변환 속성에서 인덱스 및 데이터 캐시 크기를 늘려 세션 성능을 향상시킬 수 있습니다.

집계 변환에서 재정의 캐시 디렉터리를 입력했지만, 통합 서비스는 다른 곳에 세션 증분 집계 파일을 저장합니다.

변환 캐시 디렉터리는 세션 수준에서 재정의할 수 있습니다. 통합 서비스는 세션 로그에 캐시 디렉터리를 기록합니다. 사용자는 세션 속성에서 재정의 캐시 디렉터리를 확인할 수도 있습니다.

제 3 장

사용자 지정 변환

이 장에 포함된 항목:

- [사용자 지정 변환 개요, 57](#)
- [사용자 지정 변환 작성, 59](#)
- [그룹 및 포트 작업, 60](#)
- [포트 특성 작업, 61](#)
- [사용자 지정 변환 속성, 62](#)
- [트랜잭션 제어 작업, 64](#)
- [입력 데이터 차단, 66](#)
- [프로시저 속성 작업, 68](#)
- [사용자 지정 변환 프로시저 작성, 68](#)

사용자 지정 변환 개요

사용자 지정 변환은 디자이너 인터페이스 외부에서 작성하는 프로시저와 함께 작동하여 **PowerCenter** 기능을 확장합니다. 사용자 지정 변환을 작성하고 사용자 지정 변환 함수를 사용하여 개발하는 프로시저에 이러한 변환을 바인딩할 수 있습니다. 사용자 지정 변환은 활성 또는 수동 변환일 수 있습니다.

사용자 지정 변환을 사용하여 변환 응용 프로그램을 작성합니다. 예를 들어 출력 행을 출력하기 전에 모든 입력 행을 처리해야 하는 정렬 및 집계를 작성할 수 있습니다. 이러한 프로세스를 지원하기 위해 사용자 지정 변환에서는 외부 프로시저 변환과 다르게 입력 및 출력 함수가 별개로 발생합니다.

통합 서비스는 입력 함수를 사용하여 입력 데이터를 프로시저에 전달합니다. 출력 함수는 출력 데이터를 통합 서비스에 전달하기 위해 프로시저 코드에 입력해야 하는 별개의 함수입니다. 반면에 외부 프로시저 변환에서는 외부 프로시저 함수가 입력과 출력을 둘 다 처리하고 함수의 매개 변수가 변환의 모든 포트에 구성됩니다.

또한 사용자 지정 변환을 사용하여 여러 입력 그룹이나 여러 출력 그룹 또는 둘 다를 필요로 하는 변환을 작성할 수 있습니다. 그룹은 변환으로 들어오거나 나가는 데이터 행의 표현입니다. 예를 들어 하나의 입력 그룹과 여러 개의 출력 그룹을 포함하고 XML 데이터를 구문 분석하는 사용자 지정 변환을 작성할 수 있습니다. 또는 두 개의 입력 그룹 및 하나의 출력 그룹을 포함하고 두 스트림의 입력 데이터를 한 스트림의 출력 데이터로 병합하는 사용자 지정 변환을 작성할 수 있습니다.

사용자 지정 변환을 기반으로 작성되는 변환을 사용한 작업

사용자 지정 변환을 사용하여 변환을 작성할 수 있습니다. 일부 PowerCenter 변환은 사용자 지정 변환을 사용하여 작성되어 있습니다. Informatica 제품과 함께 제공되는 다음 변환은 원시 변환이며 사용자 지정 변환을 사용하여 작성되지 않습니다.

- 집계 변환
- 식 변환
- 외부 프로시저 변환
- 필터 변환
- 조이너 변환
- 조회 변환
- 노멀라이저 변환
- 순위 변환
- 라우터 변환
- 시퀀스 생성기 변환
- 분류기 변환
- 소스 한정자 변환
- 저장 프로시저 변환
- 트랜잭션 제어 변환
- 업데이트 전략 변환

다른 모든 변환은 사용자 지정 변환을 사용하여 작성됩니다. 차단 규칙과 같은 사용자 지정 변환에 적용되는 규칙은 사용자 지정 변환을 사용하여 작성된 변환에도 적용됩니다. 예를 들어 매핑에서 사용자 지정 변환을 연결하는 경우 통합 서비스에서 차단되는 소스 없이 대상 로드 순서 그룹의 모든 소스에서 대상으로 데이터가 이동할 수 있는지 확인해야 합니다. 마찬가지로, 사용자 지정 변환을 사용하여 작성된 변환에 대해서도 이를 확인해야 합니다.

코드 페이지 호환성

통합 서비스는 ASCII 모드에서 실행되는 경우 데이터를 ASCII 형식으로 사용자 지정 변환 프로시저에 전달합니다. 통합 서비스는 유니코드 모드에서 실행되는 경우 데이터를 UCS-2 형식으로 프로시저에 전달합니다.

사용자 지정 변환 프로시저 코드에서 `INFA_CTChangeStringMode()` 및 `INFA_CTSetDataCodePageID()` 함수를 사용하여 여러 가지 형식 또는 여러 가지 코드 페이지로 데이터를 요청할 수 있습니다.

사용할 수 있는 함수는 통합 서비스의 데이터 이동 모드에 따라 달라집니다.

- **ASCII 모드.** `INFA_CTChangeStringMode()` 함수를 사용하여 UCS-2의 데이터를 요청합니다. 이 함수를 사용하는 경우 UCS-2 형식의 ASCII 문자만 통합 서비스에 전달해야 합니다. 통합 서비스가 ASCII 모드에서 실행되는 경우 `INFA_CTSetDataCodePageID()` 함수를 사용하여 코드 페이지를 변경할 수 없습니다.
- **유니코드 모드.** `INFA_CTChangeStringMode()` 함수를 사용하여 MBCS(멀티바이트 문자 집합)의 데이터를 요청합니다. 프로시저가 MBCS의 데이터를 요청하면 통합 서비스가 통합 서비스 코드 페이지로 데이터를 전달합니다. `INFA_CTSetDataCodePageID()` 함수를 사용하여 통합 서비스 코드 페이지의 다른 코드 페이지로 데이터를 요청할 수 있습니다. `INFA_CTSetDataCodePageID()` 함수에서 지정하는 코드 페이지는 통합 서비스 코드 페이지와 양방향 호환되어야 합니다.

참고: 또한 `INFA_CTRebindInputDataType()` 함수를 사용하여 사용자 지정 변환에 있는 특정 포트의 형식을 변경할 수 있습니다.

사용자 지정 변환 프로시저 배포

한 리포지토리에서 다른 리포지토리로 사용자 지정 변환을 복사할 수 있습니다. 사용자 지정 변환을 리포지토리 간에 복사하는 경우 대상 리포지토리가 사용하는 통합 서비스 시스템에 사용자 지정 변환 프로시저가 있는지 확인해야 합니다.

사용자 지정 변환 작성

변환 개발자에서 재사용 가능 사용자 지정 변환을 작성하고 변환의 인스턴스를 매핑에 추가할 수 있습니다. 매핑 디자이너 또는 **Mapplet Designer**에서 재사용 불가능 사용자 지정 변환을 작성할 수 있습니다.

각 사용자 지정 변환에서는 모듈 및 프로시저 이름을 지정합니다. 프로시저를 포함하는 기존 공유 라이브러리 또는 **DLL**을 기반으로 사용자 지정 변환을 작성하거나, 프로시저를 작성하기 위한 기반으로 사용자 지정 변환을 작성할 수 있습니다. 기존 공유 라이브러리 또는 **DLL**과 함께 사용할 사용자 지정 변환을 작성하는 경우 올바른 모듈 및 프로시저 이름을 정의해야 합니다.

프로시저를 작성하기 위한 기반으로 사용자 지정 변환을 작성하는 경우 변환을 선택하고 코드를 생성합니다. 디자이너는 프로시저 코드를 생성할 때 변환 속성을 사용하며, 공통 모듈 이름을 공유하는 모든 변환에 대해 단일 디렉터리에 코드를 생성합니다.

디자이너는 다음과 같은 파일을 생성합니다.

- **m_<module_name>.c**. 모듈을 정의합니다. 이 파일은 통합 서비스가 모듈을 로드할 때 실행할 코드를 작성할 수 있는 초기화 함수 **m_<module_name>_moduleInit()**를 포함합니다. 마찬가지로 이 파일은 통합 서비스가 모듈을 언로드할 때 실행할 코드를 작성할 수 있는 초기화 취소 함수 **m_<module_name>_moduleDeinit()**를 포함합니다.
- **p_<procedure_name>.c**. 모듈의 프로시저를 정의합니다. 이 파일은 데이터 정리 또는 데이터 병합과 같은 프로시저 논리를 구현하는 코드를 포함합니다.
- **makefile.aix, makefile.aix64, makefile.hpparisc64, makefile.linux, makefile.sol 및 makefile.sol64**. zLinux를 제외한 UNIX 플랫폼용 생성 파일입니다. 64비트 AIX 플랫폼에 **makefile.aix64**를 사용하고 64비트 Solaris 플랫폼에는 **makefile.sol64**를 사용합니다.

참고: zLinux의 경우 **makefile.linux**를 업데이트해야 합니다. **-m64**를 **FLAGS** 섹션에 추가합니다. 예를 들어 **FLAGS=-Wall -fPIC -DUNIX -m64**와 같을 수 있습니다.

사용자 지정 변환에 대한 규칙 및 지침

사용자 지정 변환을 작성할 때 다음 규칙 및 지침을 따르십시오.

- 사용자 지정 변환은 연결되는 변환입니다. 사용자 지정 변환을 식에서 참조할 수는 없습니다.
- 여러 프로시저를 하나의 모듈에 포함할 수 있습니다. 예를 들어, **XML** 기록기 프로시저 및 **XML** 파서 프로시저를 동일한 모듈에 포함할 수 있습니다.
- 사용자 지정 변환 인스턴스 여러 개를 처리하기 위한 프로시저 코드를 작성하는 경우, 하나의 공유 라이브러리 또는 **DLL**을 여러 개의 사용자 지정 변환 인스턴스에 바인딩할 수 있습니다.
- 프로시저 코드를 작성할 때 기본적인 매핑 규칙이 위반되지 않도록 해야 합니다.
- 사용자 지정 변환은 많은 전체 자릿수의 10진수를 많은 전체 자릿수의 10진수로서 보내고 받습니다.
- 사용자 지정 변환 프로시저에서 다중 스레드 코드를 사용하지 마세요.

사용자 지정 변환 구성 요소

사용자 지정 변환을 구성할 때 다음과 같은 구성 요소를 정의합니다.

- **변환 탭.** 변환 탭에서 변환의 이름을 바꾸고 설명을 추가할 수 있습니다.
- **포트 탭.** 사용자 지정 변환에 대해 포트 및 그룹을 추가하고 편집할 수 있습니다. 또한 출력 포트가 종속되는 입력 포트를 정의할 수 있습니다.
- **포트 특성 정의 탭.** 사용자 지정 변환 포트에 대해 사용자 정의 포트 특성을 작성할 수 있습니다.
- **속성 탭.** 모듈 및 함수 식별자, 트랜잭션 속성 및 런타임 위치 등의 변환 속성을 정의할 수 있습니다.
- **초기화 속성 탭.** 초기화 중과 같은 런타임 시 외부 프로시저가 사용하는 속성을 정의할 수 있습니다.
- **메타데이터 확장 탭.** 초기화 중과 같은 런타임 시 프로시저가 사용하는 속성을 정의하는 메타데이터 확장을 작성할 수 있습니다.

그룹 및 포트 작업

사용자 지정 변환에는 입력 그룹 및 출력 그룹이 있습니다. 또한 입력 포트, 출력 포트 및 입력/출력 포트가 있을 수 있습니다. 그룹 및 포트는 사용자 지정 변환의 포트 탭에서 작성하고 편집합니다. 포트 탭에서는 입력 포트와 출력 포트 간의 관계를 정의할 수도 있습니다.

그룹 및 포트 작성

사용자 지정 변환에서 여러 입력 그룹과 여러 출력 그룹을 작성할 수 있습니다. 하나 이상의 입력 그룹과 하나 이상의 출력 그룹을 작성해야 합니다. 입력 그룹을 작성하려면 입력 그룹 작성 아이콘을 클릭합니다. 출력 그룹을 작성하려면 출력 그룹 작성 아이콘을 클릭합니다. 그룹 헤더에 그룹 이름을 입력하여 기존 그룹 이름을 변경할 수 있습니다. 수동 사용자 지정 변환을 작성하는 경우 하나의 입력 그룹과 하나의 출력 그룹만 작성할 수 있습니다.

포트를 작성하면 디자이너가 현재 선택된 행 또는 그룹 아래 포트를 추가합니다. 포트는 바로 상위에 표시되는 입력 그룹 및 출력 그룹에 속할 수 있습니다. 입력 그룹 아래에 표시되는 입력/출력 포트도 출력 그룹의 일부입니다. 출력 그룹 아래에 표시되는 입력/출력 포트도 입력 그룹의 일부입니다.

포트를 공유하는 그룹을 연결된 그룹이라고 합니다. 반대 유형의 인접 그룹이 포트를 공유할 수 있습니다. 한 그룹이 둘 이상의 연결된 그룹에 속할 수 있습니다. 예를 들어 [“1단계. 사용자 지정 변환 작성” 페이지 69](#)의 그림에서 InputGroup1과 OutputGroup1은 ORDER_ID1을 공유하는 연결된 그룹입니다.

변환에 포트 특성 정의 탭이 있을 경우 각 포트의 특성을 편집할 수 있습니다.

그룹 및 포트 편집

사용자 지정 변환에서 포트 및 그룹을 편집할 때 다음 규칙과 지침을 따르십시오.

- 그룹 헤더에 입력하여 그룹 이름을 변경할 수 있습니다.
- 포트 및 그룹 이름으로 ASCII 문자만 입력할 수 있습니다.
- 그룹을 작성한 후에는 그룹 유형을 변경할 수 없습니다. 그룹 유형을 변경해야 하는 경우 그룹을 삭제하고 새 그룹을 추가합니다.
- 그룹을 삭제하면 디자이너가 해당 그룹에서 동일한 유형의 모든 포트를 삭제합니다. 하지만 모든 입력/출력 포트가 변환에서 유지되고 상위 그룹에 속하며 삭제하는 그룹의 유형에 따라 입력 포트 또는 출력 포트로 변경됩니다. 예를 들어 출력 그룹에 출력 포트 및 입력/출력 포트가 있을 수 있습니다. 출력 그룹을 삭제하면 디

자이너가 출력 포트를 삭제하고 입력/출력 포트를 입력 포트로 변경합니다. 이러한 입력 포트는 바로 위 헤더의 입력 그룹에 속합니다.

- 그룹을 위 또는 아래로 이동하려면 그룹 헤더를 선택하고 포트를 위로 이동 또는 포트를 아래로 이동 단추를 클릭합니다. 그룹 헤더 위와 아래의 포트는 동일하게 유지되지만 포트가 속한 그룹이 변경될 수 있습니다.
- 입력/출력 포트를 작성하려면 변환에 입력 그룹과 출력 그룹이 있어야 합니다.

포트 관계 정의

기본적으로 사용자 지정 변환의 출력 포트는 모든 입력 포트에 종속됩니다. 하지만 사용자 지정 변환에서 입력 포트와 출력 포트 간의 관계를 정의할 수 있습니다. 관계를 정의할 때 사용자 지정 변환이 포함된 매핑에서 링크 경로를 보고 출력 포트가 종속된 입력 포트를 확인할 수 있습니다. 또한 사용자 지정 변환이 포함된 매핑에서 대상 포트에 대한 소스 열 종속성도 확인할 수 있습니다.

사용자 지정 변환에서 포트 간 관계를 정의하려면 포트 종속성을 작성합니다. 포트 종속성은 출력 또는 입력/출력 포트와 하나 이상의 입력 또는 입력/출력 포트 간의 관계입니다. 포트 종속성을 작성할 때 코드의 프로시저 논리를 기반으로 하십시오.

포트 종속성을 작성하려면 포트 탭에서 사용자 지정 변환을 클릭하고 포트 종속성을 선택합니다.

예를 들어 XML 데이터를 구문 분석하는 외부 프로시저를 작성합니다. 하나의 입력 포트가 포함된 하나의 입력 그룹과 여러 출력 포트가 포함된 여러 출력 그룹을 사용하여 사용자 지정 변환을 작성합니다. 외부 프로시저 논리에 따라 모든 출력 포트는 입력 포트에 종속됩니다. 각 출력 포트에 대해 포트 종속성을 작성하여 사용자 지정 변환에서 이러한 관계를 정의할 수 있습니다. 출력 포트가 하나의 입력 포트에 종속되도록 각 포트 종속성을 정의하십시오.

포트 종속성을 작성하려면

1. 포트 탭에서 사용자 지정 변환을 클릭하고 포트 종속성을 선택합니다.
2. 출력 포트 종속성 대화 상자의 출력 포트 필드에서 출력 또는 입력/출력 포트를 선택합니다.
3. 입력 포트 창에서 출력 포트 또는 입력/출력 포트가 종속되는 입력 또는 입력/출력 포트를 선택합니다.
4. 추가를 클릭합니다.
5. 3 ~ 4 단계를 반복하여 포트 종속성에 더 많은 입력 또는 입력/출력 포트를 포함합니다.
6. 다른 포트 종속성을 작성하려면 2 ~ 5 단계를 반복합니다.
7. 확인을 클릭합니다.

포트 특성 작업

포트에는 데이터 유형 및 전체 자릿수 같은 특성이 있습니다. 사용자 지정 변환을 작성할 때 사용자 정의 포트 특성을 작성할 수 있습니다. 사용자 정의 포트 특성은 사용자 지정 변환의 모든 포트에 적용됩니다.

예를 들어 XML 데이터를 구문 분석하는 외부 프로시저를 작성한다고 가정합니다. "XML path"라는 포트 특성을 작성하고 이 특성으로 XML 계층의 요소 위치를 정의할 수 있습니다.

포트 특성을 작성하고 사용자 지정 변환의 포트 특성 정의 탭에서 기본값을 할당합니다. 포트 탭에서 각 포트에 대한 특정 포트 특성 값을 정의할 수 있습니다.

포트 특성을 작성할 때 다음 속성을 정의하십시오.

- **이름.** 포트 특성의 이름입니다.
- **데이터 유형.** 포트 특성 값의 데이터 유형입니다. 부울, 숫자 또는 문자열을 선택할 수 있습니다.

- **값.** 포트 특성의 기본값입니다. 이 속성은 선택 사항입니다. 이 속성에 값을 입력한 경우 해당 값이 사용자 지정 변환의 모든 포트에 적용됩니다. 포트 탭에서 각 포트에 대한 포트 특성 값을 재정의할 수 있습니다.
- 각 사용자 지정 변환에 대한 포트 특성을 정의합니다. 사용자 지정 변환 간에서 포트 특성을 복사할 수 없습니다.

포트 특성 값 편집

포트 특성을 작성한 후 변환의 각 포트에 대한 포트 특성 값을 편집할 수 있습니다. 포트 특성 값을 편집하려면 포트 탭에서 사용자 지정 변환을 클릭하고 포트 특성 편집을 선택합니다.

열기 단추를 클릭하여 특정 포트의 포트 특성 값을 변경할 수 있습니다. 단추를 클릭하면 포트 특성 기본값 편집 대화 상자가 열립니다. 또는 값 열에 직접 입력하여 새 값을 입력할 수 있습니다.

그룹 선택 필드에서 그룹을 선택하여 포트 수준 특성 편집 대화 상자에서 나열되는 포트를 필터링할 수 있습니다.

사용자 지정 변환 속성

사용자 지정 변환이 프로시저 및 변환 모두에 적용하는 속성입니다. 사용자 지정 변환의 속성 탭에서 사용자 지정 변환 속성을 구성합니다.

다음 테이블에는 사용자 지정 변환 속성이 설명되어 있습니다.

옵션	설명
언어	프로시저 코드에 사용되는 언어입니다. 사용자 지정 변환을 작성할 때 언어를 정의합니다. 언어를 변경해야 하는 경우 새로운 사용자 지정 변환을 작성합니다.
모듈 식별자	모듈 이름. C 또는 C++를 사용하여 개발하는 사용자 지정 변환 프로시저에 적용됩니다. 이 필드에는 ASCII 문자만 입력합니다. 멀티바이트 문자를 입력할 수 없습니다. 이 속성은 프로시저를 포함하는 DLL 또는 공유 라이브러리의 기본 이름입니다. 사용자가 외부 프로시저 코드를 생성하면 디자이너는 이 이름을 사용하여 C 파일을 작성합니다.
함수 식별자	모듈의 프로시저 이름입니다. C를 사용하여 개발하는 사용자 지정 변환 프로시저에 적용됩니다. 이 필드에는 ASCII 문자만 입력합니다. 멀티바이트 문자를 입력할 수 없습니다. 디자이너는 이 이름을 사용하여 사용자가 프로시저 코드를 입력하는 C 파일을 작성합니다.
클래스 이름	사용자 지정 변환 프로시저의 클래스 이름입니다. C 또는 Java를 사용하여 개발하는 사용자 지정 변환 프로시저에 적용됩니다. 이 필드에는 ASCII 문자만 입력합니다. 멀티바이트 문자를 입력할 수 없습니다.
런타임 위치	DLL 또는 공유 라이브러리가 포함된 위치입니다. 기본값은 \$PMExtProcDir입니다. 사용자 지정 변환 세션을 실행하는 통합 서비스 노드의 상대 경로를 입력합니다. 이 속성이 비어 있는 경우 통합 서비스는 통합 서비스 노드에 정의된 환경 변수를 사용하여 DLL 또는 공유 라이브러리를 찾습니다. 런타임 위치 또는 통합 서비스 노드에 정의된 환경 변수에 모든 DLL이나 공유 라이브러리를 복사해야 합니다. 통합 서비스가 DLL, 공유 라이브러리 또는 참조된 파일을 찾을 수 없으면 해당 절차를 로드하지 못합니다.
추적 수준	이 변환에 대해 세션 로그에 표시되는 세부 정보의 양입니다. 기본값은 보통입니다.

옵션	설명
분할 가능 여부	<p>이 변환을 사용하는 파이프라인에서 여러 파티션을 작성할 수 있는지 여부를 나타냅니다.</p> <ul style="list-style-type: none"> - 아니요. 변환을 분할할 수 없습니다. 변환과 동일한 파이프라인에 있는 다른 변환이 하나의 파티션으로 제한됩니다. - 로컬로. 변환을 분할할 수 있지만 통합 서비스가 같은 노드의 파이프라인에서 모든 파티션을 실행해야 합니다. 사용자 지정 변환의 여러 파티션이 메모리에서 개체를 공유해야 하는 경우 로컬로 선택합니다. - 그리드에서. 변환을 분할할 수 있고 통합 서비스가 각 파티션을 여러 노드에 분산시킬 수 있습니다. <p>기본값은 아니요입니다.</p>
입력을 차단해야 함	<p>변환과 연결된 프로시저가 수신 데이터를 차단할 수 있어야 하는지 여부를 나타냅니다. 기본값은 활성화됩니다.</p>
활성 여부	<p>변환이 활성 변환인지 아니면 수동 변환인지를 나타냅니다.</p> <p>사용자 지정 변환을 작성한 후에는 이 속성을 변경할 수 없습니다. 이 속성을 변경해야 하는 경우 새로운 사용자 지정 변환을 작성하고 올바른 속성 값을 선택합니다.</p>
업데이트 전략 변환	<p>이 변환이 출력 행의 업데이트 전략을 정의하는지 여부를 나타냅니다. 기본값이 비활성화됩니다. 활성 사용자 지정 변환에 대해 이 속성을 활성화할 수 있습니다.</p>
변환 범위	<p>통합 서비스가 변환 논리를 수신 데이터에 적용하는 방식을 지정합니다.</p> <ul style="list-style-type: none"> - 행 - 트랜잭션 - 모든 입력 <p>변환이 수동일 경우 이 속성은 항상 행입니다. 변환이 활성일 경우 이 속성은 기본적으로 모든 입력입니다.</p>
트랜잭션 생성	<p>이 변환이 트랜잭션을 생성할 수 있는지 여부를 나타냅니다. 사용자 지정 변환은 트랜잭션을 생성할 때 모든 출력 그룹에 대해 생성합니다.</p> <p>기본값이 비활성화됩니다. 활성 사용자 지정 변환에 대해서만 이 속성을 활성화할 수 있습니다.</p>
출력은 반복 가능합니다.	<p>출력 데이터의 순서가 세션 실행 간에 일관적인지 여부를 나타냅니다.</p> <ul style="list-style-type: none"> - 사용 안 함 출력 데이터 순서는 세션 실행 간에 일관되지 않습니다. 활성 변환의 경우 기본값입니다. - 입력 순서 기반. 입력 데이터 순서가 세션 실행 간에 일관된 경우 입력 순서는 세션 실행 간에 일관됩니다. 수동 변환의 경우 기본값입니다. - 항상 입력 데이터 순서가 세션 실행 간에 일관되지 않더라도 출력 데이터 순서는 세션 실행 간에 일관됩니다.
파티션당 단일 스레드 필요	<p>통합 서비스가 하나의 스레드를 포함하는 프로시저에서 각 파티션을 처리하는지 여부를 나타냅니다. 이 옵션을 활성화하면 프로시저 코드에서 스레드별 작업을 사용할 수 있습니다. 기본값은 활성화됩니다.</p>
확정 출력입니다.	<p>변환이 세션 실행 간에 일치하는 출력 데이터를 생성하는지 여부를 나타냅니다. 이 변환을 사용하는 세션에 대해 복구를 수행하려면 이 속성을 활성화합니다.</p>

경고: 변환을 반복 가능 및 확정으로 구성하는 경우 데이터가 반복 가능 및 확정인지 확인하는 것은 사용자의 책임입니다. 세션과 복구 간에 동일한 데이터를 생성하지 않는 변환으로 세션을 복구하려는 경우 복구 프로세스로 인해 손상된 데이터가 발생할 수 있습니다.

업데이트 전략 설정

활성 사용자 지정 변환을 사용하여 다음 수준에서 매핑의 업데이트 전략을 설정합니다.

- **프로시저 내에서.** 출력 행에 대한 업데이트 전략을 설정하기 위해 외부 프로시저 코드를 작성할 수 있습니다. 외부 프로시저에서 삽입, 업데이트, 삭제 또는 거부 플래그를 행에 지정할 수 있습니다.
- **매핑 내에서.** 매핑에서 사용자 지정 변환을 사용하여 삽입, 업데이트, 삭제 또는 거부 플래그를 행에 지정합니다. 사용자 지정 변환의 업데이트 전략 변환 속성을 선택합니다.
- **세션 내에서.** 소스 행을 데이터 구동으로 처리하도록 세션을 구성합니다.

업데이트 전략을 정의하기 위해 사용자 지정 변환을 구성하지 않거나 세션을 데이터 구동으로 구성하지 않은 경우, 통합 서비스는 출력 행에 플래그를 지정하는 데 외부 프로시저 코드를 사용하지 않습니다. 대신, 통합 서비스는 사용자 지정 변환이 활성 상태일 때 출력 행에 삽입 플래그를 지정합니다. 사용자 지정 변환이 수동이면 통합 서비스는 행 유형을 유지합니다. 예를 들어 업데이트 플래그가 지정된 행이 수동 사용자 지정 변환으로 들어오면 통합 서비스는 행 유형을 유지하고 행을 업데이트로 출력합니다.

스레드별 프로시저 코드 작업

사용자 지정 변환 프로시저는 스레드별 연산을 포함할 수 있습니다. 스레드별 연산은 프로시저를 처리하는 스레드에 기반하여 작업을 수행하는 코드입니다.

파티션당 단일 스레드 필요 속성을 사용하면 통합 서비스에서 사용자 지정 변환을 처리할 스레드를 각 파티션당 하나씩 사용하도록 사용자 지정 변환을 구성할 수 있습니다.

파티션별로 스레드 1개를 사용하여 각 파티션을 처리하도록 사용자 지정 변환을 구성하면 통합 서비스는 각 파티션에 대해 동일한 스레드를 통해 다음 함수를 호출합니다.

- `p_<proc_name>_partitionInit()`
- `p_<proc_name>_partitionDeinit()`
- `p_<proc_name>_inputRowNotification()`
- `p_<proc_name>_dataBdryRowNotification()`
- `p_<proc_name>_eofNotification()`

통합 서비스는 각 파티션에 대해 동일한 스레드를 사용하여 이러한 함수를 처리하므로 스레드별 연산을 이러한 함수에 포함할 수 있습니다. 예를 들어 Java 가상 시스템에 스레드를 연결하고 분리할 수 있습니다.

참고: 단일 스레드가 포함된 각 파티션을 처리하도록 사용자 지정 변환을 구성한 경우 워크플로우 관리자가 매핑 구성에 따라 파티션 지점을 추가합니다.

트랜잭션 제어 작업

다음 변환 속성을 사용하여 사용자 지정 변환에 대해 트랜잭션 제어를 정의할 수 있습니다.

- **변환 범위.** 통합 서비스가 변환 논리를 수신 데이터에 어떻게 적용하는지 결정합니다.
- **트랜잭션 생성.** 프로시저가 트랜잭션 행을 생성하고 해당 행을 출력 그룹으로 출력하도록 지정합니다.

변환 범위

통합 서비스가 수신 데이터에 변환 논리를 어떻게 적용하는지 구성할 수 있습니다. 다음 값 중 하나를 선택할 수 있습니다.

- **행.** 한 번에 하나의 데이터 행에 변환 논리를 적용합니다. 데이터의 단일 행에 따라 프로시저의 결과가 결정되는 경우 행을 선택합니다. 예를 들어 XML 파일을 포함하는 행을 프로시저가 구문 분석할 경우 행을 선택할 수 있습니다.
- **트랜잭션.** 트랜잭션의 모든 행에 변환 논리를 적용합니다. 다른 트랜잭션의 행이 아닌 동일한 트랜잭션의 모든 행에 따라 프로시저의 결과가 결정되는 경우 트랜잭션을 선택합니다. 트랜잭션을 선택할 경우 모든 입력 그룹을 동일한 트랜잭션 제어점에 연결해야 합니다. 예를 들어 외부 프로시저가 단일 트랜잭션의 데이터에 대해 집계 계산을 수행할 경우 트랜잭션을 선택할 수 있습니다.
- **모든 입력.** 수신 모든 데이터에 변환 논리를 적용합니다. 모든 입력을 선택하면 통합 서비스가 트랜잭션 경계를 삭제합니다. 소스 데이터의 모든 행에 따라 프로시저의 결과가 결정되는 경우 모든 입력을 선택합니다. 예를 들어 외부 프로시저가 수신 모든 데이터에 대해 집계 계산을 수행하거나 수신 모든 데이터를 정렬하는 경우, 모든 입력을 선택할 수 있습니다.

트랜잭션 생성

커밋 및 롤백 행 등 트랜잭션을 출력하도록 외부 프로시저 코드를 작성할 수 있습니다. 외부 프로시저가 커밋 및 롤백 행을 출력하는 경우, 트랜잭션을 생성하도록 사용자 지정 변환을 구성합니다. 트랜잭션 생성 변환 속성을 선택합니다. 활성 사용자 지정 변환에 대해 이 속성을 활성화할 수 있습니다.

외부 프로시저가 커밋 또는 롤백 행을 출력할 때 모든 출력 그룹에 대해 행을 출력하거나 롤백합니다.

트랜잭션을 생성하도록 변환을 구성하는 경우 통합 서비스가 트랜잭션 제어 변환과 같이 사용자 지정 변환을 처리합니다. 또한 매핑의 트랜잭션 제어 변환에 적용되는 대부분의 규칙은 사용자 지정 변환에도 적용됩니다. 예를 들어 트랜잭션을 생성하도록 사용자 지정 변환을 구성하는 경우 해당 변환이 포함된 파이프라인 또는 파이프라인 분기를 연결할 수 없습니다.

트랜잭션을 생성하도록 구성된 사용자 지정 변환을 사용하여 세션을 편집하거나 작성하는 경우 사용자 정의 커밋용으로 이 세션을 구성합니다.

트랜잭션 경계 작업

통합 서비스는 사용자 지정 변환으로 들어오거나 사용자 지정 변환에서 나가는 트랜잭션 경계를 매핑 구성 및 사용자 지정 변환 속성에 따라 처리합니다.

다음 테이블에는 통합 서비스가 사용자 지정 변환에서 트랜잭션 경계를 처리하는 방법이 설명되어 있습니다.

변환 범위	트랜잭션 생성이 활성화됨	트랜잭션 생성이 비활성화됨
행	통합 서비스는 수신 트랜잭션 경계를 삭제하고, 데이터 경계 알림 함수를 호출하지 않습니다. 통합 서비스는 프로시저 논리에 따라 모든 출력 그룹에 걸쳐 트랜잭션 행을 출력합니다.	모든 입력 그룹의 수신 데이터가 동일한 트랜잭션 제어점에서 공급되는 경우, 통합 서비스는 수신 트랜잭션 경계를 유지하고 모든 출력 그룹에 걸쳐 트랜잭션 경계를 출력합니다. 그러나 통합 서비스가 데이터 경계 알림 함수를 호출하지는 않습니다. 입력 그룹의 수신 데이터가 서로 다른 트랜잭션 제어점에서 공급되는 경우, 통합 서비스는 수신 트랜잭션 경계를 삭제합니다. 통합 서비스는 데이터 경계 알림 함수를 호출하지 않습니다. 통합 서비스는 개방형 트랜잭션 1개의 모든 행을 출력합니다.
트랜잭션	통합 서비스는 수신 트랜잭션 경계를 유지하고, 데이터 경계 알림 함수를 호출합니다. 그러나 통합 서비스는 프로시저 논리에 따라 모든 출력 그룹에 걸쳐 트랜잭션 행을 출력합니다.	통합 서비스는 수신 트랜잭션 경계를 유지하고, 데이터 경계 알림 함수를 호출합니다. 통합 서비스는 모든 출력 그룹에 걸쳐 트랜잭션 행을 출력합니다.
모든 입력	통합 서비스는 수신 트랜잭션 경계를 삭제하고, 데이터 경계 알림 함수를 호출하지 않습니다. 통합 서비스는 프로시저 논리에 따라 모든 출력 그룹에 걸쳐 트랜잭션 행을 출력합니다.	통합 서비스는 수신 트랜잭션 경계를 삭제하고, 데이터 경계 알림 함수를 호출하지 않습니다. 통합 서비스는 개방형 트랜잭션 1개의 모든 행을 출력합니다.

입력 데이터 차단

기본적으로 통합 서비스는 대상 로드 순서 그룹의 소스를 동시에 읽습니다. 하지만 일부 입력 그룹에 대해 입력 데이터를 차단하는 외부 프로시저 코드를 작성할 수 있습니다. 차단이란 다중 입력 그룹 변환의 입력 그룹으로 데이터 흐름이 일시 중단되는 것입니다.

사용자 지정 변환을 통해 입력 데이터를 차단하려면 데이터를 차단 및 차단 해제하는 프로시저 코드를 작성해야 합니다. 또한 사용자 지정 변환의 속성 탭에서 차단을 활성화할 수도 있습니다.

데이터를 차단하는 프로시저 코드 작성

수신 데이터를 차단하거나 차단하지 않도록 프로시저를 작성할 수 있습니다. 수신 데이터를 차단하려면 `INFA_CTBlockInputFlow()` 함수를 사용합니다. 수신 데이터를 차단하지 않으려면 `INFA_CTUnblockInputFlow()` 함수를 사용합니다.

외부 프로시저가 입력 그룹에서 읽기를 대체해야 하는 경우 입력 데이터를 차단해야 합니다. 차단 기능이 없으면 프로시저 코드를 버퍼 수신 데이터에 써야 합니다. 입력 데이터를 버퍼링하는 대신 차단하면 일반적으로 세션 성능을 높일 수 있습니다.

예를 들어 두 개의 입력 그룹으로 외부 프로시저를 작성해야 한다고 가정합니다. 외부 프로시저가 첫 번째 입력 그룹에서 행을 읽은 다음 두 번째 입력 그룹에서 행을 읽습니다. 차단을 사용하는 경우 외부 프로시저 코드를 써서 다른 입력 그룹에서 데이터를 처리하는 동안 한 입력 그룹의 데이터 흐름을 차단할 수 있습니다. 데이터를 차단하기 위해 외부 프로시저 코드를 쓸 경우 프로시저는 소스 데이터를 버퍼에 복사할 필요가 없기 때문에 성능이 향상됩니다. 하지만 버퍼를 할당하고 데이터를 처리할 준비가 될 때까지 한 입력 그룹의 데이터를 버퍼에 복사하도록 외부 프로시저를 쓸 수 있습니다. 소스 데이터를 버퍼에 복사하면 성능이 저하됩니다.

관련 항목:

- [“차단 함수” 페이지 108](#)

사용자 지정 변환을 차단 변환으로 구성

사용자 지정 변환을 작성하면 디자이너가 기본적으로 입력을 차단해야 함 변환 속성을 활성화합니다. 이 속성을 매핑을 저장하거나 유효성을 검사할 때 데이터 흐름 유효성 검사에 영향을 줍니다. 이 속성을 활성화하면 사용자 지정 변환은 차단 변환이 됩니다. 이 속성의 선택을 취소하면 사용자 지정 변환은 차단 변환이 아닙니다.

외부 프로시저 코드가 반드시 입력 데이터를 차단할 수 있어야 하는 경우 사용자 지정 변환을 차단 변환으로 구성합니다.

다음과 같은 조건 중 하나에 해당되는 경우 사용자 지정 변환을 비차단 변환으로 구성할 수 있습니다.

- 프로시저 코드에 차단 함수가 포함되어 있지 않습니다.
- 프로시저 코드에 두 개의 알고리즘이 포함되어 있습니다. 한 알고리즘은 차단을 사용하고 다른 알고리즘은 데이터를 차단하는 대신 프로시저에서 할당된 버퍼에 소스 데이터를 복사합니다. 프로시저 코드는 통합 서비스가 사용자 지정 변환의 데이터 차단을 허용하는지 여부를 확인합니다. 프로시저는 차단할 수 있을 경우 차단 함수가 포함된 알고리즘을 사용하고 차단할 수 없을 경우 다른 알고리즘을 사용합니다. 여러 가지 매핑 구성에서 사용하는 사용자 지정 변환을 작성하려면 이렇게 할 수 있습니다.

참고: 프로시저가 데이터를 차단하지만 사용자 지정 변환을 비차단 변환으로 구성하는 경우 통합 서비스에서 세션을 실행할 수 없습니다.

사용자 지정 변환이 포함된 매핑의 유효성 검사

사용자 지정 변환을 매핑에 포함하면 디자이너 및 통합 서비스가 매핑의 유효성을 검사합니다. 디자이너는 사용자가 저장하거나 유효성을 검사할 때 매핑의 유효성을 검사하고, 통합 서비스는 사용자가 세션을 실행할 때 매핑의 유효성을 검사합니다.

디자인 시 유효성 검사

매핑을 저장하거나 매핑의 유효성을 검사할 때 디자이너는 데이터 흐름 유효성 검사를 수행합니다. 디자이너는 데이터 흐름 유효성 검사를 수행할 경우, 모든 소스를 차단하는 차단 변환이 없는 상태로 대상 로드 순서 그룹에 있는 모든 소스에서 대상으로 데이터가 흐를 수 있는지 확인합니다. 차단 변환을 포함하는 일부 매핑은 올바르지 않습니다.

런타임 시 유효성 검사

세션을 실행하면 통합 서비스가 런타임에 프로시저 코드를 대상으로 매핑의 유효성을 검사합니다. 통합 서비스는 유효성 검사를 수행할 때 사용자 지정 변환이 데이터를 차단하는 것을 허용할지 여부를 추적합니다.

- **사용자 지정 변환을 차단 변환으로 구성.** 통합 서비스는 사용자 지정 변환이 데이터를 차단하는 것을 항상 허용합니다.
- **사용자 지정 변환을 비차단 변환으로 구성.** 통합 서비스는 매핑 구성에 따라 사용자 지정 변환이 데이터를 차단하는 것을 허용합니다. 통합 서비스가 대상 로드 순서 그룹의 모든 소스를 동시에 차단하지 않으면서 사용자 지정 변환에서 데이터를 차단할 수 있는 경우, 통합 서비스는 사용자 지정 변환이 데이터를 차단하는 것을 허용합니다.

통합 서비스가 사용자 지정 변환의 데이터 차단을 허용하는지 여부를 확인하기 위한 프로시저 코드를 작성할 수 있습니다. `INFA_CT_TRANS_MAY_BLOCK_DATA` 속성 ID에 액세스하려면 `INFA_CT_getInternalProperty()` 함수를 사용하십시오. 사용자 지정 변환이 데이터를 차단할 수 있으면 통합 서비스가 `TRUE`를 반환하고, 사용자 지정 변환이 데이터를 차단할 수 없으면 통합 서비스가 `FALSE`를 반환합니다.

프로시저 속성 작업

통합 서비스가 프로시저를 실행할 때(예: 초기화 시) 프로시저가 사용할 수 있는 속성 이름과 값의 쌍을 사용자 지정 변환에서 정의할 수 있습니다. 사용자 지정 변환의 다음 탭에서 사용자 정의 속성을 작성할 수 있습니다.

- **메타데이터 확장.** 속성 이름, 데이터 유형, 전체 자릿수 및 값을 지정할 수 있습니다. 프로시저로 정보를 전달하는 데 메타데이터 확장을 사용합니다.
- **초기화 속성.** 속성 이름 및 값을 지정할 수 있습니다.

사용자 지정 변환의 두 가지 탭에서 속성을 정의할 수 있지만, 메타데이터 확장 탭에서는 속성에 대한 더 자세한 정보를 제공할 수 있습니다. 프로시저로 속성을 전달하는 데 메타데이터 확장을 사용합니다.

데이터를 변환한 후에 데이터를 정렬하는 사용자 지정 변환 외부 프로시저를 작성하는 경우를 예로 들어 보겠습니다. 이 경우 **Sort_Ascending**이라는 부울 메타데이터 확장을 작성할 수 있습니다. 매핑에서 사용자 지정 변환을 사용할 때 프로시저를 통해 데이터를 어떻게 정렬하려는지에 따라 메타데이터 확장에 대해 **True** 또는 **False**를 선택할 수 있습니다.

사용자 지정 변환에서 속성을 정의할 경우, **INFA_CTGetAllPropertyNamesM()** 등의 모든 속성 이름 가져오기 함수를 사용하여 초기화 속성 및 메타데이터 확장 탭에 정의된 모든 속성의 이름에 액세스합니다. 지정한 속성 ID의 속성 이름 및 값에 액세스하려면 **INFA_CT_getExternalPropertyM()** 등의 외부 속성 가져오기 함수를 사용합니다.

참고: 이름이 동일한 메타데이터 확장과 초기화 속성을 정의하면 속성 함수는 메타데이터 확장에 대한 정보만 반환합니다.

사용자 지정 변환 프로시저 작성

32비트 또는 64비트 통합 서비스 시스템에서 실행되는 사용자 지정 변환 프로시저를 작성할 수 있습니다. 다음 단계를 지침으로 참고하여 사용자 지정 변환 프로시저를 작성하십시오.

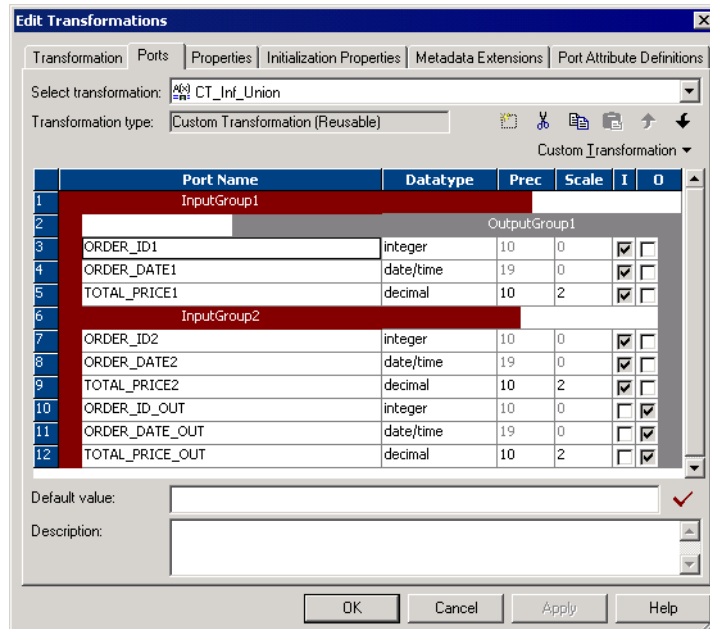
1. 변환 개발자에서 재사용 가능 사용자 지정 변환을 작성합니다. 또는 **Mapplet Designer**나 매핑 디자이너에서 재사용 불가능 사용자 지정 변환을 작성합니다.
2. 프로시저의 템플릿 코드를 생성합니다.
프로시저 코드를 생성하면 디자이너가 사용자 지정 변환의 정보를 사용하여 C 소스 코드 파일과 **makefile**을 작성합니다.
3. C 파일을 수정하여 프로시저 논리를 추가합니다.
4. C/C++ 컴파일러를 사용하여 소스 코드 파일을 컴파일하고 DLL 또는 공유 라이브러리로 연결한 후 통합 서비스 시스템에 복사합니다.
5. 사용자 지정 변환을 사용하여 매핑을 작성합니다.
6. 워크플로우에서 세션을 실행합니다.

이 섹션에는 이 프로세스를 보여 주는 예제가 포함되어 있습니다. 이 섹션의 단계에서는 두 개의 입력 그룹과 한 개의 출력 그룹이 포함된 사용자 지정 변환을 작성합니다. 사용자 지정 변환 프로시저는 사용자 지정 변환이 두 개의 입력 그룹과 한 개의 출력 그룹을 사용하는지 확인합니다. 또한 모든 그룹의 포트 수가 같고 모든 그룹의 포트 데이터 유형이 같은지 확인합니다. 프로시저는 각 입력 그룹에서 데이터 행을 받아들여 모든 행을 출력 그룹으로 출력합니다.

1단계. 사용자 지정 변환 작성

사용자 지정 변환을 작성하려면 다음을 수행하십시오.

1. 변환 개발자에서 변환 > 작성을 클릭합니다.
 2. 변환 작성 대화 상자에서 사용자 지정 변환을 선택하고 변환 이름을 입력한 후 작성을 클릭합니다.
합집합 예제에서는 CT_Inf_Union을 변환 이름으로 입력합니다.
 3. 활성 또는 수동 대화 상자에서 변환을 수동 또는 활성 변환으로 작성하고 확인을 클릭합니다.
합집합 예제에서는 활성을 선택합니다.
 4. 완료를 클릭하여 변환 작성 대화 상자를 닫습니다.
 5. 변환을 열고 포트 탭을 클릭합니다. 그룹 및 포트를 작성합니다.
그룹 및 포트는 필요에 따라 나중에 편집할 수 있습니다.
- 다음 그림은 그룹이 2개 있는 합집합 변환의 예를 보여 줍니다.



합집합 예제에서는 InputGroup1 및 InputGroup2 그룹을 작성합니다. InputGroup1에 대해 다음 포트를 작성합니다.

포트 이름	데이터 유형	전체 자릿수	배율
ORDER_ID1	정수	10	0
ORDER_DATE1	날짜/시간	19	0
TOTAL_PRICE1	10진수	10	2

InputGroup2에 대해 다음 포트를 작성합니다.

포트 이름	데이터 유형	전체 자릿수	배율	입력/출력
ORDER_ID2	정수	10	0	입력
ORDER_DATE2	날짜/시간	19	0	입력
TOTAL_PRICE2	10진수	10	2	입력
ORDER_ID_OUT	정수	10	0	출력
ORDER_DATE_OUT	날짜/시간	19	0	출력
TOTAL_PRICE_OUT	10진수	10	2	출력

- 속성 탭을 선택하고 모듈과 함수 식별자 및 런타임 위치를 입력합니다. 추적 수준, 분할 가능 여부, 입력을 블록 지정해야 함, 활성 여부, 업데이트 전략 변환, 변환 범위 및 트랜잭션 생성 값/확인란 등의 다른 변환 속성 특성을 편집합니다.

합집합 예제에서는 다음 속성을 구성합니다.

속성 이름	값
모듈 식별자	UnionDemo
함수 식별자	합집합
런타임 위치	\$PMExtProcDir
추적 수준	보통
분할 가능 여부	아니요
입력을 차단해야 함	아니요
활성 여부	예
업데이트 전략 변환	아니요
변환 범위	모든 입력
트랜잭션 생성	아니요

- 메타데이터 확장 탭을 클릭하여 초기화를 위해 외부 프로시저에서 필요할 수 있는 속성 등의 메타데이터 확장을 입력합니다.

합집합 예제에서는 메타데이터 확장을 작성하지 마십시오.

- 필요에 따라 포트 특성 정의 탭을 클릭하여 포트 특성을 작성합니다.

합집합 예제에서는 포트 특성을 작성하지 마십시오.

프로시저를 호출하는 사용자 지정 변환을 작성한 이후의 다음 단계는 C 파일을 생성하는 것입니다.

2단계. C 파일 생성

사용자 지정 변환을 작성한 후에 소스 코드 파일을 생성합니다. 디자이너에서는 소문자로 된 파일 이름을 생성합니다.

사용자 지정 변환 프로시저에 대한 코드를 생성하려면 다음을 수행하십시오.

1. 변환 개발자에서 변환을 선택하고 변환 > 코드 생성을 클릭합니다.
2. 작성한 프로시저를 선택합니다. 디자이너에서 프로시저가 <module_name>.<procedure_name>(으)로 나열됩니다.

합집합 예제에서는 UnionDemo.Union을 선택합니다.

3. 파일을 생성하려는 디렉터리를 지정하고 생성을 클릭합니다.

합집합 예제에서는 <client_installation_directory>/TX를 선택합니다.

디자이너가 하위 디렉터리 <module_name>을(를) 지정된 디렉터리에 작성합니다. 합집합 예제에서는 디자이너가 <client_installation_directory>/TX/UnionDemo를 작성합니다. 또한 다음 파일을 작성합니다.

- m_UnionDemo.c
- m_UnionDemo.h
- p_Union.c
- p_Union.h
- makefile.aix(32비트), makefile.aix64(64비트), makefile.hp(32비트), makefile.hp64(64비트), makefile.hpparisc64, makefile.linux(32비트) 및 makefile.sol(32비트).

3단계. 변환 논리를 포함하여 코드 작성

프로시저 C 파일을 코딩해야 합니다. 필요한 경우 모듈 C 파일도 코딩할 수 있습니다. 합집합 예제에서는 프로시저 C 파일만 작성합니다. 모듈 C 파일을 작성할 필요는 없습니다.

프로시저 C 파일을 코딩하려면 다음을 수행하십시오.

1. 프로시저의 p_<procedure_name>.c 파일을 엽니다.
합집합 예제에서는 p_Union.c를 엽니다.
2. 프로시저의 C 코드를 입력합니다.
3. 수정된 파일을 저장합니다.

합집합 예제에서는 다음 코드를 사용합니다.

```
/*
*****
* 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
*****/

#include <stdlib.h>
#include "p_union.h"

/*****
                                     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;
}

```

4단계. 모듈 빌드

Windows 또는 UNIX 플랫폼에서 모듈을 작성할 수 있습니다.

다음 테이블에는 모듈을 작성할 때 필요한 각 플랫폼의 라이브러리 파일 이름이 나열되어 있습니다.

플랫폼	모듈 파일 이름
Windows	<module_identifier>.dll
AIX	lib<module_identifier>.a
Linux	lib<module_identifier>.so
Solaris	lib<module_identifier>.so

Windows 모듈 작성

Windows에서 Microsoft Visual C++를 사용하여 모듈을 빌드합니다.

Windows에 모듈을 작성하려면 다음을 수행하십시오.

1. Visual C++를 시작합니다.
2. 파일 > 새로 만들기를 클릭합니다.
3. 새로 만들기 대화 상자에서 프로젝트 탭을 클릭하고 Win32 동적 연결 라이브러리 옵션을 선택합니다.
4. 위치를 입력합니다.
합집합 예제에서 <client_installation_directory>/TX/UnionDemo를 입력합니다.
5. 프로젝트 이름을 입력합니다.
사용자 지정 변환에 지정된 모듈 이름을 프로젝트 이름으로 사용해야 합니다. 합집합 예제에서 UnionDemo를 입력합니다.
6. 확인을 클릭합니다.
프로젝트 구성 요소를 정의하는 데 도움이 되도록 Visual C++가 마법사를 작성합니다.
7. 마법사에서 빈 DLL 프로젝트를 선택하고 마침을 클릭합니다. 새 프로젝트 정보 대화 상자에서 확인을 클릭합니다.
Visual C++가 지정한 디렉터리에 프로젝트 파일을 작성합니다.
8. 프로젝트 > 프로젝트에 추가 > 파일을 클릭합니다.
9. 한 수준 위의 디렉터리를 탐색합니다. 작성한 프로시저 파일이 이 디렉터리에 포함됩니다. 모든 .c 파일을 선택하고 확인을 클릭합니다.
합집합 예제에서 다음 파일을 추가합니다.
 - m_UnionDemo.c
 - p_Union.c
10. 프로젝트 > 설정을 클릭합니다.
11. C/C++ 탭을 클릭하고 범주 필드에서 전처리기를 선택합니다.
12. 추가 포함 디렉터리 필드에 다음 경로를 입력하고 확인을 클릭합니다.
...; <PowerCenter_install_dir>\extproc\include\ct
13. 빌드 > <module_name>.dll 빌드를 클릭하거나 F7을 눌러 프로젝트를 작성합니다.
Visual C++가 DLL을 작성하고 프로젝트 디렉터리 아래의 디버그 또는 릴리스 디렉터리에 배치합니다.

UNIX 모듈 작성

UNIX에서는 C 컴파일러를 사용하여 모듈을 빌드합니다.

UNIX에 모듈을 작성하려면 다음을 수행하십시오.

1. 디자이너가 생성한 makefile 및 C 파일 모두를 UNIX 시스템으로 복사합니다.
참고: 통합 서비스 시스템 이외의 시스템에서 공유 라이브러리를 빌드하는 경우 다음 디렉터리의 파일을 빌드 시스템에도 복사해야 합니다.
<PowerCenter_install_dir>\ExtProc\include\ct
합집합 예제에서 <client_installation_directory>/TX/UnionDemo의 모든 파일을 복사합니다.
2. 환경 변수 INFA_HOME을 통합 서비스 설치 디렉터리로 설정합니다.
참고: INFA_HOME 환경 변수에 대해 잘못된 디렉터리 경로를 지정하면 통합 서비스가 시작되지 않습니다.

3. 다음 테이블의 명령을 입력하여 프로젝트를 만듭니다.

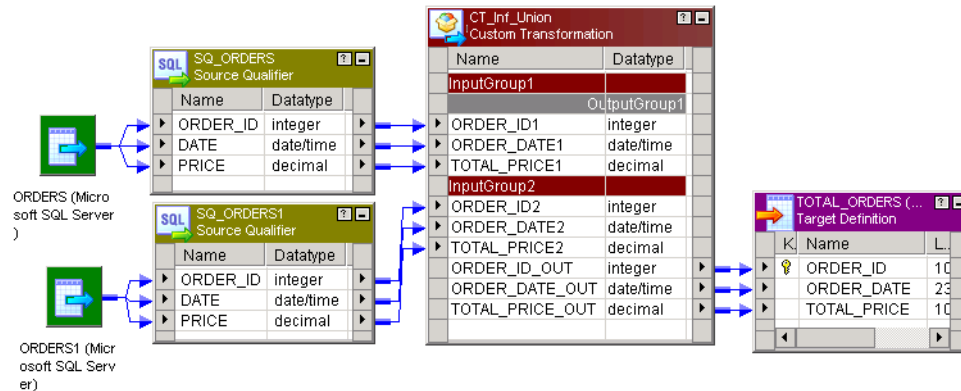
UNIX 버전	명령
AIX(32비트)	make -f makefile.aix
AIX(64비트)	make -f makefile.aix64
Linux	make -f makefile.linux
Solaris	make -f makefile.sol

5단계. 매핑 생성

매핑 디자이너에서 사용자 지정 변환을 사용하는 매핑을 작성합니다.

이 매핑에서 포트 및 데이터 유형이 동일한 두 소스가 사용자 지정 변환의 두 입력 그룹에 연결됩니다. 사용자 지정 변환은 두 소스 모두에서 행을 가져온 다음 출력 그룹 하나를 통해 모두 출력합니다. 출력 그룹은 포트 및 데이터 유형이 입력 그룹과 같습니다.

합집합 예에서는 다음 그림의 매핑과 유사한 매핑을 작성합니다.



6단계. 워크플로우의 세션 실행

세션을 실행하면 통합 서비스는 사용자 지정 변환에 지정된 런타임 위치에서 공유 라이브러리 또는 DLL을 찾습니다.

워크플로우의 세션을 실행하려면 다음을 수행하십시오.

1. 워크플로우 관리자에서 워크플로우를 작성합니다.
2. 워크플로우에서 이 매핑에 대한 세션을 작성합니다.
3. 공유 라이브러리 또는 DLL을 런타임 위치 디렉터리에 복사합니다.
4. 세션이 포함된 워크플로우를 실행합니다.

통합 서비스는 프로시저에 바인딩된 사용자 지정 변환을 로드할 때 DLL 또는 공유 라이브러리를 로드하고 사용자가 정의한 프로시저를 호출합니다.

제 4 장

사용자 지정 변환 함수

이 장에 포함된 항목:

- [사용자 지정 변환 함수 개요, 79](#)
- [함수 참조, 80](#)
- [행 작업, 83](#)
- [생성된 함수, 84](#)
- [API 함수, 89](#)
- [배열 기반 API 함수, 112](#)

사용자 지정 변환 함수 개요

사용자 지정 변환은 디자이너 외부에서 작성하는 프로시저와 함께 작동하여 **PowerCenter** 기능을 확장합니다. 사용자 지정 변환 함수를 통해 사용자 지정 변환과 연결되는 프로시저의 변환 논리를 개발할 수 있습니다.

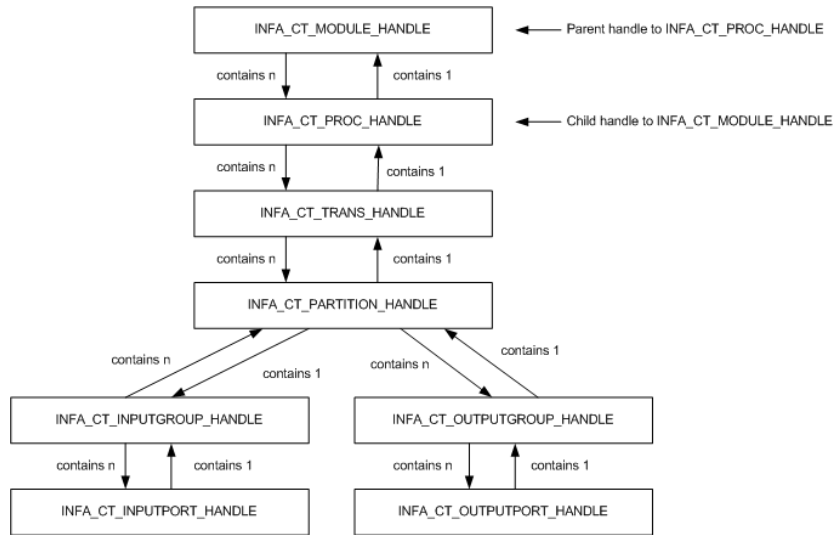
PowerCenter는 생성된 함수와 API 함수라는 두 가지 함수 집합을 제공합니다. 통합 서비스는 생성된 함수를 사용하여 프로시저와 상호 작용합니다. 사용자 지정 변환을 작성하고 소스 코드 파일을 생성하면 디자이너가 생성된 함수를 파일에 포함합니다. 프로시저 코드에 API 함수를 사용하여 변환 논리를 개발할 수 있습니다.

프로시저 코드를 작성할 때 통합 서비스에서 행 블록을 받거나 한 번에 단일 행을 받도록 구성할 수 있습니다. 행 블록을 받고 처리하는 경우 프로시저 성능을 향상시킬 수 있습니다.

핸들 작업

대부분의 함수는 **INFA_CT_PARTITION_HANDLE** 등의 핸들과 연결됩니다. 이러한 함수의 첫 번째 매개 변수는 함수의 영향을 받는 핸들입니다. 사용자 지정 변환 핸들은 서로 계층적 관계에 있습니다. 상위 핸들은 하위 핸들과 **1:n** 관계를 가집니다.

다음 그림은 사용자 지정 변환 핸들을 보여 줍니다.



다음 테이블에는 사용자 지정 변환 핸들이 설명되어 있습니다.

핸들 이름	설명
INFA_CT_MODULE_HANDLE	공유 라이브러리 또는 DLL을 나타냅니다. 외부 프로시저는 자체 공유 라이브러리 또는 DLL에 있는 모듈 핸들에만 액세스할 수 있습니다. 다른 모든 공유 라이브러리 또는 DLL의 모듈 핸들에는 액세스할 수 없습니다.
INFA_CT_PROC_HANDLE	공유 라이브러리 또는 DLL 안의 특정 프로시저를 나타냅니다. 여러 사용자 지정 변환에 의해 참조되는 프로시저에 영향을 주는 함수를 작성해야 하는 경우, 이 핸들을 사용할 수 있습니다.
INFA_CT_TRANS_HANDLE	세션에 있는 특정 사용자 지정 변환 인스턴스를 나타냅니다.
INFA_CT_PARTITION_HANDLE	특정 사용자 지정 변환 인스턴스에 있는 특정 파티션을 나타냅니다.
INFA_CT_INPUTGROUP_HANDLE	파티션에 있는 입력 그룹을 나타냅니다.
INFA_CT_INPUTPORT_HANDLE	파티션에 있는 입력 그룹의 입력 포트를 나타냅니다.
INFA_CT_OUTPUTGROUP_HANDLE	파티션에 있는 출력 그룹을 나타냅니다.
INFA_CT_OUTPUTPORT_HANDLE	파티션에 있는 출력 그룹의 출력 포트를 나타냅니다.

함수 참조

사용자 지정 변환에는 생성된 함수와 API 함수가 포함됩니다.

다음 테이블에는 사용자 지정 변환 생성된 함수가 나와 있습니다.

함수	설명
m_<module_name>_moduleInit()	모듈 초기화 함수입니다.
p_<proc_name>_procInit()	프로시저 초기화 함수입니다.
p_<proc_name>_partitionInit()	파티션 초기화 함수입니다.
p_<proc_name>_inputRowNotification()	입력 행 알림 함수입니다.
p_<proc_name>_dataBdryNotification()	데이터 경계 알림 함수입니다.
p_<proc_name>_eofNotification()	파일 끝 알림 함수입니다.
p_<proc_name>_partitionDeinit()	파티션 초기화 취소 함수입니다.
p_<proc_name>_procedureDeinit()	프로시저 초기화 취소 함수입니다.
m_<module_name>_moduleDeinit()	모듈 초기화 취소 함수입니다.

다음 테이블에는 사용자 지정 변환 API 함수가 나와 있습니다.

함수	설명
INFA_CTSetDataAccessMode()	데이터 액세스 모드 설정 함수입니다.
INFA_CTGetAncestorHandle()	상위 핸들 가져오기 함수입니다.
INFA_CTGetChildrenHandles()	하위 핸들 가져오기 함수입니다.
INFA_CTGetInputPortHandle()	입력 포트 핸들 가져오기 함수입니다.
INFA_CTGetOutputPortHandle()	출력 포트 핸들 가져오기 함수입니다.
INFA_CTGetInternalProperty<데이터 유형>()	내부 속성 가져오기 함수입니다.
INFA_CTGetAllPropertyNamesM()	MBCS 모드로 모든 속성 이름 가져오기 함수입니다.
INFA_CTGetAllPropertyNamesU()	유니코드 모드로 모든 속성 이름 가져오기 함수입니다.
INFA_CTGetExternalProperty<데이터 유형>M()	MBCS로 외부 속성 가져오기 함수입니다.
INFA_CTGetExternalProperty<데이터 유형>U()	유니코드로 외부 속성 가져오기 함수입니다.
INFA_CTRebindInputDataType()	입력 포트 데이터 유형 다시 바인딩 함수입니다.
INFA_CTRebindOutputDataType()	출력 포트 데이터 유형 다시 바인딩 함수입니다.
INFA_CTGetData<데이터 유형>()	데이터 가져오기 함수입니다.
INFA_CTSetData()	데이터 설정 함수입니다.

함수	설명
INFA_CTGetIndicator()	표시기 가져오기 함수입니다.
INFA_CTSetIndicator()	표시기 설정 함수입니다.
INFA_CTGetLength()	길이 가져오기 함수입니다.
INFA_CTSetLength()	길이 설정 함수입니다.
INFA_CTSetPassThruPort()	통과 포트 설정 함수입니다.
INFA_CTOutputNotification()	출력 알림 함수입니다.
INFA_CTDataBdryOutputNotification()	데이터 경계 출력 알림 함수입니다.
INFA_CTGetErrorMsgU()	유니코드로 오류 메시지 가져오기 함수입니다.
INFA_CTGetErrorMsgM()	MBCS로 오류 메시지 가져오기 함수입니다.
INFA_CTLogMessageU()	유니코드로 세션 로그에 메시지 로깅 함수입니다.
INFA_CTLogMessageM()	MBCS로 세션 로그에 메시지 로깅 함수입니다.
INFA_CTIncrementErrorCount()	오류 개수 증분 함수입니다.
INFA_CTIsterminateRequested()	종료 요청 여부 함수입니다.
INFA_CTBLOCKInputFlow()	입력 그룹 차단 함수입니다.
INFA_CTUnblockInputFlow()	입력 그룹 차단 해제 함수입니다.
INFA_CTSetUserDefinedPtr()	사용자 정의 포인터 설정 함수입니다.
INFA_CTGetUserDefinedPtr()	사용자 정의 포인터 가져오기 함수입니다.
INFA_CTChangeStringMode()	문자열 모드 변경 함수입니다.
INFA_CTSetDataCodePageID()	데이터 코드 페이지 ID 설정 함수입니다.
INFA_CTGetRowStrategy()	행 전략 가져오기 함수입니다.
INFA_CTSetRowStrategy()	행 전략 설정 함수입니다.
INFA_CTChangeDefaultRowStrategy()	변환의 기본 행 전략을 변경합니다.

다음 테이블에는 사용자 지정 변환 배열 기반 함수가 나와 있습니다.

함수	설명
INFA_CTAGetInputRowMax()	최대 입력 행 수 가져오기 함수입니다.
INFA_CTAGetOutputRowMax()	최대 출력 행 수 가져오기 함수입니다.

함수	설명
INFA_CTASetOutputRowMax()	최대 출력 행 수 설정 함수입니다.
INFA_CTAGetNumRows()	행 수 가져오기 함수입니다.
INFA_CTASetNumRows()	행 수 설정 함수입니다.
INFA_CTARowValid()	올바른 행임 함수입니다.
INFA_CTAGetData<데이터 유형>()	데이터 가져오기 함수입니다.
INFA_CTAGetIndicator()	표시기 가져오기 함수입니다.
INFA_CTASetData()	데이터 설정 함수입니다.
INFA_CTARowStrategy()	행 전략 가져오기 함수입니다.
INFA_CTASetRowStrategy()	행 전략 설정 함수입니다.
INFA_CTASetInputErrorRowM()	MBCS에 대해 입력 오류 행 설정 함수입니다.
INFA_CTASetInputErrorRowU()	유니코드에 대해 입력 오류 행 설정 함수입니다.

행 작업

통합 서비스는 단일 행을 사용자 지정 변환 프로시저 또는 배열의 행 블록에 전달할 수 있습니다. 프로시저 코드를 써서 프로시저에서 행 하나를 받을지 아니면 행 블록 하나를 받을지 지정할 수 있습니다. 프로시저가 행 블록을 받도록 하면 성능을 향상시킬 수 있습니다.

- 통합 서비스 및 프로시저가 수행하는 함수 호출의 수를 줄일 수 있습니다. 그러면 통합 서비스가 입력 행 알림 함수를 더 적게 호출하게 되고, 프로시저가 출력 알림 함수를 더 적게 호출하게 됩니다.
- 데이터에 대한 메모리 액세스 공간의 로컬리티를 높일 수 있습니다.
- 데이터의 각 행이 아니라 데이터 블록에 대해 알고리즘을 수행하도록 프로시저 코드를 쓸 수 있습니다.

기본적으로 프로시저는 한 번에 1개의 데이터 행을 받습니다. 행 블록을 받으려면

INFA_CTSetDataAccessMode() 함수를 포함하여 데이터 액세스 모드를 배열 기반으로 변경해야 합니다. 데이터 액세스 모드가 배열 기반인 경우, 배열 기반 데이터 처리 함수 및 행 전략 함수를 사용하여 데이터를 액세스하고 출력해야 합니다. 데이터 액세스 모드가 행 기반인 경우에는 행 기반 데이터 처리 함수 및 행 전략 함수를 사용하여 데이터를 액세스하고 출력해야 합니다.

모든 배열 기반 함수는 접두사 **INFA_CTA**를 사용합니다. 그 밖의 다른 모든 함수에서는 접두사 **INFA_CT**를 사용합니다.

행 블록에 액세스하려면 다음 단계에 따라 프로시저 코드를 작성하십시오.

1. 데이터 액세스 모드를 배열 기반으로 변경하기 위해 프로시저 초기화 중에 **INFA_CTSetDataAccessMode()**를 호출합니다.
2. 수동 사용자 지정 변환을 작성하는 경우, 입력/출력 포트에 향하는 데이터를 통과시키기 위해 프로시저 초기화 중에 **INFA_CTSetPassThruPort()**를 호출할 수도 있습니다.

데이터 블록이 사용자 지정 변환 프로시저에 도달하면 통합 서비스는 각 데이터 블록에 대해 `p_<proc_name>_inputRowNotification()`을 호출합니다. 나머지 단계를 이 함수 내에서 수행하십시오.

3. 입력 행 알림 함수에서 입력 그룹 핸들을 사용해 `INFA_CTAGetNumRows()`를 호출하여 현재 블록에 있는 행의 수를 확인합니다.
4. 입력 포트 핸들을 사용해 `INFA_CTAGetData<데이터 유형>()` 함수 중 하나를 호출하여 블록에 있는 특정 행의 데이터를 가져옵니다.
5. `INFA_CTASetData`를 호출하여 블록에 있는 행을 출력합니다.
6. `INFA_CTOutputNotification()`을 호출하기 전에 `INFA_CTASetNumRows()`를 호출하여 프로시저가 블록에서 출력하는 행의 수를 통합 서비스에 알려 줍니다.
7. `INFA_CTOutputNotification()`을 호출합니다.

행 기반 및 배열 기반 데이터 액세스 모드에 대한 규칙 및 지침

행 기반 또는 배열 기반 데이터 액세스 모드를 사용하는 프로시저 코드를 작성할 때 다음 규칙 및 지침을 사용하십시오.

- 행 기반 모드에서는 입력 행 알림 함수에서 `INFA_ROWERROR`를 반환하여 함수에서 입력의 데이터 행에 대한 오류가 발생했음을 나타낼 수 있습니다. 통합 서비스가 내부 오류 수를 증분시킵니다.
- 배열 기반 모드에서는 입력 행 알림 함수에서 `INFA_ROWERROR`를 반환하지 마십시오. 통합 서비스는 이 오류를 심각한 오류로 처리합니다. 블록의 행에 오류가 있음을 나타내야 할 경우 `INFA_CTASetInputErrorRowM()` 또는 `INFA_CTASetInputErrorRowU()` 함수를 호출합니다.
- 행 기반 모드에서 통합 서비스는 올바른 행만 프로시저로 전달합니다.
- 배열 기반 모드에서는 입력 블록에 삭제된 행, 필터링된 행 또는 오류 행 같은 올바르지 않은 행이 포함될 수 있습니다. 블록의 행이 올바른지 확인하려면 `INFA_CTIsRowValid()`를 호출하십시오.
- 배열 기반 모드에서 수동 사용자 지정 변환에 대해 `INFA_CTASetNumRows()`를 호출하지 마십시오. 활성 사용자 지정 변환에 대해서는 이 함수를 호출할 수 있습니다.
- 배열 기반 모드에서 `INFA_CTOutputNotification()`을 한 번만 호출하십시오.
- 배열 기반 모드에서 수동 사용자 지정 변환에 대해 `INFA_CTSetPassThruPort()`를 호출할 수 있습니다.
- 수동 사용자 지정 변환에 대한 배열 기반 모드에서는 출력 블록에 모든 오류 행을 포함한 모든 행을 출력해야 합니다.

생성된 함수

디자이너를 사용하여 프로시저 코드를 생성하면 디자이너가 생성된 함수라고 하는 함수 집합을 `m_<module_name>.c` 파일과 `p_<procedure_name>.c` 파일에 포함합니다. 통합 서비스는 생성된 함수를 사용하여 프로시저와 상호 작용합니다. 세션을 실행하면 통합 서비스가 매핑에서 각 대상 로드 순서 그룹에 대해 다음과 같은 순서로 이러한 생성된 함수를 호출합니다.

1. 초기화 함수
2. 알림 함수
3. 초기화 취소 함수

초기화 함수

통합 서비스는 먼저 초기화 함수를 호출합니다. 초기화 함수를 통해 통합 서비스가 사용자 지정 변환에 데이터를 전달하기 전에 실행할 프로세스를 작성할 수 있습니다. 초기화 함수에서 코드를 작성하면 통합 서비스가 이러한 프로세스를 모듈, 프로시저 또는 파티션에 대해 한 번만 실행하므로 처리 오버헤드가 줄어듭니다.

디자이너는 다음과 같은 초기화 함수를 생성합니다.

- `m_<module_name>_moduleInit()`
- `p_<proc_name>_procInit()`
- `p_<proc_name>_partitionInit()`

모듈 초기화 함수

통합 서비스는 사전 세션 태스크를 실행하기 전에 세션 초기화 중 `m_<module_name>_moduleInit()` 함수를 호출합니다. 통합 서비스는 모든 다른 함수 전에 이 함수를 모듈에 대해 한 번 호출합니다.

통합 서비스가 모듈을 로드하기 전에 특정 프로세스를 실행하도록 하려면 해당 프로세스를 이 함수에 포함해야 합니다. 예를 들어 이 모듈 내 프로시저가 액세스하는 전역 구조를 작성하는 코드를 작성할 수 있습니다.

다음 구문을 사용합니다.

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

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
모듈	INFA_CT_MODULE_HANDLE	입력	모듈 핸들입니다.

반환 값 데이터 유형은 `INFA_STATUS`입니다. 반환 값에 `INFA_SUCCESS` 및 `INFA_FAILURE`를 사용합니다. 함수가 `INFA_FAILURE`를 반환하면 통합 서비스가 세션을 중지합니다.

프로시저 초기화 함수

통합 서비스는 세션 초기화 중에 사전 세션 태스크를 실행하기 이전 시점과 모듈 초기화 함수를 실행한 이후 시점에 `p_<proc_name>_procInit()` 함수를 호출합니다. 통합 서비스는 이 함수를 모듈의 각 프로시저에 대해 한 번씩 호출합니다.

통합 서비스가 특정 프로시저에 대해 프로세스를 실행하도록 하려면 이 함수 안에 코드를 작성하십시오. 또한 탐색 및 속성 함수 등의 일부 API 함수를 프로시저 초기화 함수 안에 입력할 수 있습니다.

다음 구문을 사용합니다.

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

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
프로시저	INFA_CT_PROCEDURE_HANDLE	입력	프로시저 핸들입니다.

반환 값 데이터 유형은 `INFA_STATUS`입니다. 반환 값에 `INFA_SUCCESS` 및 `INFA_FAILURE`를 사용합니다. 함수가 `INFA_FAILURE`를 반환하면 통합 서비스가 세션을 중지합니다.

파티션 초기화 함수

통합 서비스는 사용자 지정 변환으로 데이터를 전달하기 전에 `p_<proc_name>_partitionInit()` 함수를 호출합니다. 통합 서비스는 사용자 지정 변환 인스턴스의 각 파티션에 대해 한 번씩 이 함수를 호출합니다.

통합 서비스가 사용자 지정 변환의 파티션을 통해 데이터를 전달하기 전에 특정 프로세스를 실행하도록 하려면 이 함수에 해당 프로세스를 포함해야 합니다.

다음 구문을 사용합니다.

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

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
변환	INFA_CT_PARTITION_HANDLE	입력	파티션 핸들입니다.

반환 값 데이터 유형은 `INFA_STATUS`입니다. 반환 값에 `INFA_SUCCESS` 및 `INFA_FAILURE`를 사용합니다. 함수가 `INFA_FAILURE`를 반환하면 통합 서비스가 세션을 중지합니다.

참고: 사용자 지정 변환에서 각 파티션에 대해 스레드가 1개씩 필요하면 스레드별 작업을 파티션 초기화 함수에 포함할 수 있습니다.

알림 함수

통합 서비스는 데이터 행을 사용자 지정 변환에 전달할 때 알림 함수를 호출합니다.

디자이너는 다음과 같은 알림 함수를 생성합니다.

- `p_<proc_name>_inputRowNotification()`
- `p_<proc_name>_dataBdryRowNotification()`
- `p_<proc_name>_eofNotification()`

참고: 사용자 지정 변환에서 파티션별로 하나의 스레드가 필요할 경우 스레드별 작업을 알림 함수에 포함할 수 있습니다.

입력 행 알림 함수

통합 서비스는 행 또는 행 블록을 사용자 지정 변환에 전달할 때 `p_<proc_name>_inputRowNotification()` 함수를 호출합니다. 이 함수는 입력 그룹 핸들과 파티션 핸들을 통해 데이터를 수신하는 입력 그룹과 파티션을 나타냅니다.

다음 구문을 사용합니다.

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

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
파티션	INFA_CT_PARTITION_HANDLE	입력	파티션 핸들입니다.
그룹	INFA_CT_INPUTGROUP_HANDLE	입력	입력 그룹 핸들입니다.

반환 값의 데이터 유형은 **INFA_ROWSTATUS**입니다. 다음 값을 반환 값으로 사용합니다.

- **INFA_ROWSUCCESS.** 함수가 데이터 행을 성공적으로 처리했음을 나타냅니다.
- **INFA_ROWERROR.** 함수가 데이터 행을 처리하는 동안 오류가 발생했음을 나타냅니다. 통합 서비스가 내부 오류 수를 증분시킵니다. 이 값은 데이터 액세스 모드가 행인 경우에만 반환됩니다.
입력 행 알림 함수가 배열 기반 모드에서 **INFA_ROWERROR**를 반환하면 통합 서비스가 이를 치명적 오류로 처리합니다. 블록의 행에 오류가 있음을 나타내야 할 경우 **INFA_CTASetInputErrorRowM()** 또는 **INFA_CTASetInputErrorRowU()** 함수를 호출합니다.
- **INFA_FATALERROR.** 함수가 데이터 행 또는 데이터 블록을 처리하는 동안 치명적인 오류가 발생했음을 나타냅니다. 통합 서비스가 세션에 실패했습니다.

데이터 경계 알림 함수

통합 서비스는 행을 파티션에 커밋하거나 롤백할 때 **p_<proc_name>_dataBdryNotification()** 함수를 호출합니다.

다음 구문을 사용합니다.

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

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
변환	INFA_CT_PARTITION_HANDLE	입력	파티션 핸들입니다.
dataBoundaryType	INFA_CTDataBdryType	입력	통합 서비스는 dataBoundaryType 매개 변수에 다음 값 중 하나를 사용합니다. - eBT_COMMIT - eBT_ROLLBACK

반환 값 데이터 유형은 **INFA_STATUS**입니다. 반환 값에 **INFA_SUCCESS** 및 **INFA_FAILURE**를 사용합니다. 함수가 **INFA_FAILURE**를 반환하면 통합 서비스가 세션을 중지합니다.

파일 끝 알림 함수

통합 서비스는 마지막 행을 입력 그룹의 파티션에 전달한 후 **p_<proc_name>_eofNotification()** 함수를 호출합니다.

다음 구문을 사용합니다.

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

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
변환	INFA_CT_PARTITION_HANDLE	입력	파티션 핸들입니다.
그룹	INFA_CT_INPUTGROUP_HANDLE	입력	입력 그룹 핸들입니다.

반환 값 데이터 유형은 INFA_STATUS입니다. 반환 값에 INFA_SUCCESS 및 INFA_FAILURE를 사용합니다. 함수가 INFA_FAILURE를 반환하면 통합 서비스가 세션을 중지합니다.

초기화 취소 함수

통합 서비스는 사용자 지정 변환에 대한 데이터를 처리한 후 초기화 취소 함수를 호출합니다. 초기화 취소 함수를 통해 통합 서비스가 사용자 지정 변환에 모든 데이터 행을 전달한 후 실행할 프로세스를 작성할 수 있습니다.

디자인서는 다음과 같은 초기화 취소 함수를 생성합니다.

- p_<proc_name>_partitionDeinit()
- p_<proc_name>_procDeinit()
- m_<module_name>_moduleDeinit()

참고: 사용자 지정 변환에서 파티션별로 하나의 스레드가 필요할 경우 스레드별 작업을 초기화 함수와 초기화 취소 함수에 포함할 수 있습니다.

파티션 초기화 취소 함수

통합 서비스는 p_<proc_name>_eofNotification() 또는 p_<proc_name>_abortNotification() 함수를 호출한 후에 p_<proc_name>_partitionDeinit() 함수를 호출합니다. 통합 서비스는 사용자 지정 변환의 각 파티션에 대해 한 번씩 이 함수를 호출합니다.

다음 구문을 사용합니다.

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

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
파티션	INFA_CT_PARTITION_HANDLE	입력	파티션 핸들입니다.

반환 값 데이터 유형은 INFA_STATUS입니다. 반환 값에 INFA_SUCCESS 및 INFA_FAILURE를 사용합니다. 함수가 INFA_FAILURE를 반환하면 통합 서비스가 세션을 중지합니다.

참고: 사용자 지정 변환에서 각 파티션에 대해 스레드가 1개씩 필요하면 스레드별 작업을 파티션 초기화 취소 함수에 포함할 수 있습니다.

프로시저 초기화 취소 함수

통합 서비스는 매핑에서 이 프로시저를 사용하는 각 사용자 지정 변환 인스턴스의 모든 파티션에 대해 p_<proc_name>_partitionDeinit() 함수를 호출한 후에 p_<proc_name>_procDeinit() 함수를 호출합니다.

다음 구문을 사용합니다.

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

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/ 출력	설명
프로시저	INFA_CT_PROCEDURE_HANDLE	입력	프로시저 핸들입니다.
sessionStatus	INFA_STATUS	입력	통합 서비스는 sessionStatus 매개 변수에 다음 값 중 하나를 사용합니다. - INFA_SUCCESS. 세션이 성공했음을 나타냅니다. - INFA_FAILURE. 세션이 실패했음을 나타냅니다.

반환 값 데이터 유형은 INFA_STATUS입니다. 반환 값에 INFA_SUCCESS 및 INFA_FAILURE를 사용합니다. 함수가 INFA_FAILURE를 반환하면 통합 서비스가 세션을 중지합니다.

모듈 초기화 취소 함수

통합 서비스는 사후 세션 태스크를 실행한 후 `m_<module_name>_moduleDeinit()` 함수를 호출합니다. 통합 서비스는 모든 다른 함수 후에 이 함수를 모듈에 대해 한 번 호출합니다.

다음 구문을 사용합니다.

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

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/ 출력	설명
모듈	INFA_CT_MODULE_HANDLE	입력	모듈 핸들입니다.
sessionStatus	INFA_STATUS	입력	통합 서비스는 sessionStatus 매개 변수에 다음 값 중 하나를 사용합니다. - INFA_SUCCESS. 세션이 성공했음을 나타냅니다. - INFA_FAILURE. 세션이 실패했음을 나타냅니다.

반환 값 데이터 유형은 INFA_STATUS입니다. 반환 값에 INFA_SUCCESS 및 INFA_FAILURE를 사용합니다. 함수가 INFA_FAILURE를 반환하면 통합 서비스가 세션을 중지합니다.

API 함수

PowerCenter는 변환 논리를 개발하는 데 사용하는 API 함수 집합을 제공합니다. 디자이너는 소스 코드 파일을 생성할 때 생성된 함수를 소스 코드에 포함합니다. API 함수를 코드에 추가하여 변환 논리를 구현할 수 있습니다. 프로시저는 API 함수를 사용하여 통합 서비스와 상호 작용합니다. 프로시저 C 파일에 API 함수를 코딩해야 합니다. 필요한 경우 모듈 C 파일도 코딩할 수 있습니다.

Informatica는 다음과 같은 API 함수 그룹을 제공합니다.

- 데이터 액세스 모드 설정

- 탐색
- 속성
- 데이터 유형 재바인딩
- 데이터 처리(행 기반 모드)
- 통과 포트 설정
- 출력 알림
- 데이터 경계 출력 알림
- 오류
- 세션 로그 메시지
- 오류 개수 증분
- 종료됨
- 차단
- 포인터
- 문자열 모드 변경
- 데이터 코드 페이지 설정
- 행 전략(행 기반 모드)
- 기본 행 전략 변경

Informatica는 배열 기반 API 함수도 제공합니다.

데이터 액세스 모드 설정 함수

기본적으로 통합 서비스는 한 번에 한 행씩 사용자 지정 변환 프로시저에 데이터를 전달합니다. 데이터 액세스 모드를 배열 기반으로 변경하려면 `INFA_CTSetDataAccessMode()` 함수를 사용하십시오. 데이터 액세스 모드를 배열 기반으로 설정하면 통합 서비스가 여러 행을 배열의 한 블록으로 프로시저에 전달합니다.

데이터 액세스 모드를 배열 기반으로 설정할 경우, 배열 기반 버전의 데이터 처리 함수 및 행 전략 함수를 사용해야 합니다. 행 기반 데이터 처리 함수 또는 행 전략 함수를 사용하면 배열 기반 모드로 전환할 때 예기치 않은 결과가 발생합니다. 예를 들어 DLL 또는 공유 라이브러리가 충돌할 수 있습니다.

이 함수는 프로시저 초기화 함수에서만 사용할 수 있습니다.

이 함수를 프로시저 코드에서 사용하지 않으면 데이터 액세스 모드는 행 기반입니다. 데이터 액세스 모드가 행 기반이 되도록 하려는 경우에는 이 함수를 포함하고 액세스 모드를 행 기반으로 설정합니다.

다음 구문을 사용합니다.

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

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
프로시저	INFA_CT_PROCEDURE_HANDLE	입력	프로시저 이름입니다.
모드	INFA_CT_DATA_ACCESS_MODE	입력	데이터 액세스 모드입니다. mode 매개 변수에 대해 다음 값을 사용하십시오. - eDA_ROW - eDA_ARRAY

탐색 함수

프로시저에서 핸들 계층을 탐색하려면 탐색 함수를 사용합니다.

PowerCenter는 다음과 같은 탐색 함수를 제공합니다.

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

상위 핸들 가져오기 함수

프로시저에서 지정된 핸들의 상위 핸들에 액세스하려면 INFA_CTGetAncestorHandle() 함수를 사용합니다.

다음 구문을 사용합니다.

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

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
핸들	INFA_CT_HANDLE	입력	핸들 이름입니다.
returnHandleType	INFA_CTHandleType	입력	반환 핸들 유형입니다. returnHandleType 매개 변수에 다음 값을 사용합니다. - PROCEDURETYPE - TRANSFORMATIONTYPE - PARTITIONTYPE - INPUTGROUPTYPE - OUTPUTGROUPTYPE - INPUTPORTTYPE - OUTPUTPORTTYPE

핸들 매개 변수는 프로시저에서 액세스할 상위 항목의 핸들을 지정합니다. 함수에서 올바른 핸들을 지정하면 통합 서비스가 INFA_CT_HANDLE을 반환합니다. 그렇지 않으면 null 값을 반환합니다.

컴파일 오류를 방지하려면 핸들 이름을 반환 값으로 설정하도록 프로시저를 코딩해야 합니다.

예를 들어 다음과 같은 코드를 입력할 수 있습니다.

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

하위 핸들 가져오기 함수

프로시저에서 지정된 핸들의 하위 핸들에 액세스하려면 `INFA_CTGetChildrenHandles()` 함수를 사용합니다.

다음 구문을 사용합니다.

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

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
핸들	INFA_CT_HANDLE	입력	핸들 이름입니다.
pnChildrenHandles	size_t*	출력	통합 서비스가 하위 핸들의 배열을 반환합니다. pnChildrenHandles 매개 변수는 배열에 있는 하위 핸들의 수를 나타냅니다.
returnHandleType	INFA_CT_HandleType	입력	returnHandleType 매개 변수에 다음 값을 사용합니다. - PROCEDURETYPE - TRANSFORMATIONTYPE - PARTITIONTYPE - INPUTGROUPTYPE - OUTPUTGROUPTYPE - INPUTPORTTYPE - OUTPUTPORTTYPE

핸들 매개 변수는 프로시저에서 액세스할 하위 항목의 핸들을 지정합니다. 함수에서 올바른 핸들을 지정하면 통합 서비스가 `INFA_CT_HANDLE*`를 반환합니다. 그렇지 않으면 `null` 값을 반환합니다.

컴파일 오류를 방지하려면 핸들 이름을 반환 값으로 설정하도록 프로시저를 코딩해야 합니다.

예를 들어 다음과 같은 코드를 입력할 수 있습니다.

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

포트 핸들 가져오기 함수

통합 서비스는 `INFA_CT_INPUTPORT_HANDLE`을 입력 및 입력/출력 포트와 연결하고 `INFA_CT_OUTPUTPORT_HANDLE`을 출력 및 입력/출력 포트와 연결합니다.

PowerCenter는 다음과 같은 포트 핸들 가져오기 함수를 제공합니다.

- **INFA_CTGetInputPortHandle()**. 프로시저에서 입력/출력 포트의 출력 포트 핸들을 알고 입력 포트 핸들이 필요한 경우 이 함수를 사용합니다.

다음 구문을 사용합니다.

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


다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
outputPortHandle	INFA_CT_OUTPUTPORT_HANDLE	입력	출력 포트 핸들입니다.

- **INFA_CTGetOutputPortHandle().** 프로시저에서 입력/출력 포트의 입력 포트 핸들을 알고 출력 포트 핸들이 필요한 경우 이 함수를 사용합니다.

다음 구문을 사용합니다.

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

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
inputPortHandle	INFA_CT_INPUTPORT_HANDLE	입력	입력 포트 핸들입니다.

포트 핸들 가져오기 함수를 입력 또는 출력 포트와 함께 사용하면 통합 서비스가 NULL을 반환합니다.

속성 함수

프로시저가 사용자 지정 변환 속성에 액세스하도록 하려면 속성 함수를 사용하십시오. 속성 함수는 사용자 지정 변환의 다음 탭에 있는 속성에 액세스합니다.

- 포트
- 속성
- 초기화 속성
- 메타데이터 확장
- 포트 특성 정의

초기화 함수에서 다음 속성 함수를 사용하십시오.

- INFA_CTGetInternalProperty<데이터 유형>()
- INFA_CTGetAllPropertyNamesM()
- INFA_CTGetAllPropertyNamesU()
- INFA_CTGetExternalProperty<데이터 유형>M()
- INFA_CTGetExternalProperty<데이터 유형>U()

내부 속성 가져오기 함수

PowerCenter는 포트 탭에 지정된 포트 특성에 액세스하는 함수 및 사용자 지정 변환의 속성 탭에서 특성에 대해 지정된 속성을 제공합니다.

통합 서비스는 각 포트 및 속성 특성을 속성 ID와 연결합니다. 특성에 대해 지정된 값에 액세스하려면 프로시저에서 속성 ID를 지정해야 합니다. 핸들 매개 변수에 대해 핸들 계층의 핸들 이름을 지정합니다. 핸들 이름이 잘못된 경우 통합 서비스에서 세션이 실패합니다.

프로시저가 속성에 액세스하게 하려면 다음 함수를 사용합니다.

- **INFA_CTGetInternalPropertyStringM()**. 지정된 속성 ID에 대해 MBCS로 된 문자열 유형의 값에 액세스합니다.

다음 구문을 사용합니다.

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

- **INFA_CTGetInternalPropertyStringU()**. 지정된 속성 ID에 대해 유니코드로 된 문자열 유형의 값에 액세스합니다.

다음 구문을 사용합니다.

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

- **INFA_CTGetInternalPropertyInt32()**. 지정된 속성 ID에 대해 정수 유형의 값에 액세스합니다.

다음 구문을 사용합니다.

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

- **INFA_CTGetInternalPropertyBool()**. 지정된 속성 ID에 대해 부울 유형의 값에 액세스합니다.

다음 구문을 사용합니다.

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

- **INFA_CTGetInternalPropertyINFA_PTR()**. 지정된 속성 ID에 대해 값의 포인터에 액세스합니다.

다음 구문을 사용합니다.

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

반환 값 데이터 유형은 INFA_STATUS입니다. 반환 값에 INFA_SUCCESS 및 INFA_FAILURE를 사용합니다.

포트 및 속성 특성 속성 ID

다음 테이블에는 사용자 지정 변환의 포트에 대한 속성 ID 및 속성 특성이 나열되어 있습니다. 각 테이블에는 사용자 지정 변환 핸들 및 이 핸들로 속성 함수에서 액세스할 수 있는 속성 ID가 나열되어 있습니다.

다음 테이블에는 INFA_CT_MODULE_HANDLE 속성 ID가 나열되어 있습니다.

핸들 속성 ID	데이터 유형	설명
INFA_CT_MODULE_NAME	문자열	모듈 이름을 지정합니다.
INFA_CT_SESSION_INFA_VERSION	문자열	Informatica 버전을 지정합니다.
INFA_CT_SESSION_CODE_PAGE	정수	통합 서비스 코드 페이지를 지정합니다.
INFA_CT_SESSION_DATAMOVEMENT_MODE	정수	데이터 이동 모드를 지정합니다. 통합 서비스는 다음 값 중 하나를 반환합니다. - eASM_MBCS - eASM_UNICODE
INFA_CT_SESSION_VALIDATE_CODEPAGE	부울	통합 서비스에서 코드 페이지 유효성 검사를 실시하는지 여부를 지정합니다.
INFA_CT_SESSION_PROD_INSTALL_DIR	문자열	통합 서비스 설치 디렉터리를 지정합니다.

핸들 속성 ID	데이터 유형	설명
INFA_CT_SESSION_HIGH_PRECISION_MODE	부울	많은 전체 자릿수에 맞춰 세션이 구성되는지 여부를 지정합니다.
INFA_CT_MODULE_RUNTIME_DIR	문자열	DLL 또는 공유 라이브러리의 런타임 디렉터리를 지정합니다.
INFA_CT_SESSION_IS_UPD_STR_ALLOWED	부울	업데이트 전략 변환 속성이 변환에서 선택되어 있는지 여부를 지정합니다.
INFA_CT_TRANS_OUTPUT_IS_REPEATABLE	정수	<p>사용자 지정 변환이 모든 세션 실행에서 동일한 순서로 데이터를 생성하는지 여부를 지정합니다. 통합 서비스는 다음 값 중 하나를 반환합니다.</p> <ul style="list-style-type: none"> - eOUTREPEAT_NEVER = 1 - eOUTREPEAT_ALWAYS = 2 - eOUTREPEAT_BASED_ON_INPUT_ORDER = 3
INFA_CT_TRANS_FATAL_ERROR	부울	<p>사용자 지정 변환으로 인해 심각한 오류가 발생했는지 여부를 지정합니다. 통합 서비스는 다음 값 중 하나를 반환합니다.</p> <ul style="list-style-type: none"> - INFA_TRUE - INFA_FALSE

다음 테이블에는 INFA_CT_PROC_HANDLE 속성 ID가 나열되어 있습니다.

핸들 속성 ID	데이터 유형	설명
INFA_CT_PROCEDURE_NAME	문자열	사용자 지정 변환 프로시저 이름을 지정합니다.

다음 테이블에는 INFA_CT_TRANS_HANDLE 속성 ID가 나열되어 있습니다.

핸들 속성 ID	데이터 유형	설명
INFA_CT_TRANS_INSTANCE_NAME	문자열	사용자 지정 변환 인스턴스 이름을 지정합니다.
INFA_CT_TRANS_TRACE_LEVEL	정수	<p>추적 수준을 지정합니다. 통합 서비스는 다음 값 중 하나를 반환합니다.</p> <ul style="list-style-type: none"> - eTRACE_TERSE - eTRACE_NORMAL - eTRACE_VERBOSE_INIT - eTRACE_VERBOSE_DATA
INFA_CT_TRANS_MAY_BLOCK_DATA	부울	프로시저가 현재 세션의 입력 데이터를 차단하는 것을 통합 서비스에서 허용하는지 여부를 지정합니다.

핸들 속성 ID	데이터 유형	설명
INFA_CT_TRANS_MUST_BLOCK_DATA	부울	입력을 블록 지정해야 함 사용자 지정 변환 속성이 선택되어 있는지 여부를 지정합니다.
INFA_CT_TRANS_ISACTIVE	부울	사용자 지정 변환이 활성 변환인지 또는 수동 변환인지 지정합니다.
INFA_CT_TRANS_ISPARTITIONABLE	부울	이 사용자 지정 변환을 사용하는 세션을 분할할 수 있는지 여부를 지정합니다.
INFA_CT_TRANS_IS_UPDATE_STRATEGY	부울	사용자 지정 변환이 업데이트 전략 변환처럼 동작하는지 여부를 지정합니다.
INFA_CT_TRANS_DEFAULT_UPDATE_STRATEGY	정수	기본 업데이트 전략을 지정합니다. <ul style="list-style-type: none"> - eDUS_INSERT - eDUS_UPDATE - eDUS_DELETE - eDUS_REJECT - eDUS_PASSTHROUGH
INFA_CT_TRANS_NUM_PARTITIONS	정수	이 사용자 지정 변환을 사용하는 세션에 있는 파티션 수를 지정합니다.
INFA_CT_TRANS_DATACODEPAGE	정수	통합 서비스가 사용자 지정 변환으로 데이터를 전달할 때 사용하는 코드 페이지를 지정합니다. 사용자 지정 변환이 다른 코드 페이지의 데이터에 액세스하도록 하려면 데이터 코드 페이지 설정 함수를 사용합니다.
INFA_CT_TRANS_TRANSFORM_SCOPE	정수	사용자 지정 변환의 변환 범위를 지정합니다. 통합 서비스는 다음 값 중 하나를 반환합니다. <ul style="list-style-type: none"> - eTS_ROW - eTS_TRANSACTION - eTS_ALLINPUT
INFA_CT_TRANS_GENERATE_TRANSACTION	부울	트랜잭션 생성 속성이 활성화되어 있는지 여부를 지정합니다. 통합 서비스는 다음 값 중 하나를 반환합니다. <ul style="list-style-type: none"> - INFA_TRUE - INFA_FALSE
INFA_CT_TRANS_OUTPUT_IS_REPEATABLE	정수	사용자 지정 변환이 모든 세션 실행에서 동일한 순서로 데이터를 생성하는지 여부를 지정합니다. 통합 서비스는 다음 값 중 하나를 반환합니다. <ul style="list-style-type: none"> - eOUTREPEAT_NEVER = 1 - eOUTREPEAT_ALWAYS = 2 - eOUTREPEAT_BASED_ON_INPUT_ORDER = 3
INFA_CT_TRANS_FATAL_ERROR	부울	사용자 지정 변환으로 인해 심각한 오류가 발생했는지 여부를 지정합니다. 통합 서비스는 다음 값 중 하나를 반환합니다. <ul style="list-style-type: none"> - INFA_TRUE - INFA_FALSE

다음 테이블에는 INFA_CT_INPUT_GROUP_HANDLE 및 INFA_CT_OUTPUT_GROUP_HANDLE 속성 ID가 나열되어 있습니다.

핸들 속성 ID	데이터 유형	설명
INFA_CT_GROUP_NAME	문자열	그룹 이름을 지정합니다.
INFA_CT_GROUP_NUM_PORTS	정수	그룹에 있는 포트의 수를 지정합니다.
INFA_CT_GROUP_ISCONNECTED	부울	그룹에 있는 모든 포트가 다른 변환에 연결되어 있는지 여부를 지정합니다.
INFA_CT_PORT_NAME	문자열	포트 이름을 지정합니다.
INFA_CT_PORT_CDATATYPE	정수	포트 데이터 유형을 지정합니다. 통합 서비스는 다음 값 중 하나를 반환합니다. <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	정수	포트 전체 자릿수를 지정합니다.
INFA_CT_PORT_SCALE	정수	포트 소수 자릿수를 지정합니다(해당하는 경우).
INFA_CT_PORT_IS_MAPPED	부울	포트가 매핑에서 다른 변환에 연결되어 있는지 여부를 지정합니다.
INFA_CT_PORT_STORAGE_SIZE	정수	포트에 대해 데이터의 내부 저장소 크기를 지정합니다. 저장소 크기는 포트의 데이터 유형에 따라 달라집니다.
INFA_CT_PORT_BOUND_DATATYPE	정수	포트 데이터 유형을 지정합니다. 포트를 다시 바인딩하고 기본값 이외의 데이터 유형을 지정할 경우 INFA_CT_PORT_CDATATYPE 대신 사용합니다.

다음 테이블에는 INFA_CT_INPUTPORT_HANDLE 및 INFA_CT_OUTPUT_HANDLE 속성 ID가 나열되어 있습니다.

핸들 속성 ID	데이터 유형	설명
INFA_CT_PORT_NAME	문자열	포트 이름을 지정합니다.
INFA_CT_PORT_CDATATYPE	정수	포트 데이터 유형을 지정합니다. 통합 서비스는 다음 값 중 하나를 반환합니다. <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	정수	포트 전체 자릿수를 지정합니다.
INFA_CT_PORT_SCALE	정수	해당하는 경우 포트 소수 자릿수를 지정합니다.
INFA_CT_PORT_IS_MAPPED	부울	포트가 매핑에서 다른 변환에 연결되어 있는지 여부를 지정합니다.
INFA_CT_PORT_STORAGE_SIZE	정수	포트에 대해 데이터의 내부 저장소 크기를 지정합니다. 저장소 크기는 포트의 데이터 유형에 따라 달라집니다.
INFA_CT_PORT_BOUND_DATATYPE	정수	포트 데이터 유형을 지정합니다. 포트를 다시 바인딩하고 기본값 이외의 데이터 유형을 지정할 경우 INFA_CT_PORT_CDATATYPE 대신 사용합니다.

모든 외부 속성 이름 가져오기(MBCS 또는 유니코드)

PowerCenter는 사용자 지정 변환의 메타데이터 확장 탭, 초기화 속성 탭 및 포트 특성 정의 탭에서 정의된 속성 이름에 액세스하는 두 가지 함수를 제공합니다.

프로시저에서 속성 이름에 액세스하려는 경우 다음 함수를 사용하십시오.

- **INFA_CTGetAllPropertyNamesM()**. MBCS 형식의 속성 이름에 액세스합니다.

다음 구문을 사용합니다.

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

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/ 출력	설명
핸들	INFA_CT_HANDLE	입력	핸들 이름을 지정합니다.
paPropertyNames	const char*const**	출력	속성 이름을 지정합니다. 통합 서비스가 속성 이름 배열을 MBCS 형식으로 반환합니다.
pnProperties	size_t*	출력	배열의 속성 수를 나타냅니다.

- **INFA_CTGetAllPropertyNamesU()**. 유니코드 형식의 속성 이름에 액세스합니다.

다음 구문을 사용합니다.

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

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/ 출력	설명
핸들	INFA_CT_HANDLE	입력	핸들 이름을 지정합니다.
paPropertyNames	const INFA_UNICHAR*const**	출력	속성 이름을 지정합니다. 통합 서비스가 속성 이름 배열을 유니코드 형식으로 반환합니다.
pnProperties	size_t*	출력	배열의 속성 수를 나타냅니다.

반환 값 데이터 유형은 INFA_STATUS입니다. 반환 값에 INFA_SUCCESS 및 INFA_FAILURE를 사용합니다.

외부 속성 가져오기(MBCS 또는 유니코드)

PowerCenter는 사용자 지정 변환의 메타데이터 확장 탭, 초기화 속성 탭 또는 포트 특성 정의 탭에 정의된 속성의 값에 액세스하는 함수를 제공합니다.

프로시저가 값에 액세스하게 하려면 함수에서 속성 이름을 지정해야 합니다.

INFA_CTGetAllPropertyNamesM() 또는 INFA_CTGetAllPropertyNamesU() 함수를 사용하여 속성 이름에 액세스합니다. 핸들 매개 변수에 대해 핸들 계층의 핸들 이름을 지정합니다. 핸들 이름이 잘못된 경우 통합 서비스에서 세션이 실패합니다.

참고: 메타데이터 확장과 동일한 이름으로 초기화 속성을 정의하는 경우 통합 서비스가 메타데이터 확장 값을 반환합니다.

프로시저가 속성 값에 액세스하게 하려면 다음 함수를 사용합니다.

- **INFA_CTGetExternalProperty<데이터 유형>M()**. MBCS의 속성 값에 액세스합니다.

다음 테이블에는 구문이 나와 있습니다.

구문	속성 데이터 유형
<code>INFA_STATUS INFA_CTGetExternalPropertyStringM(INFA_CT_HANDLE handle, const char* sPropName, const char** psPropValue);</code>	문자열
<code>INFA_STATUS INFA_CTGetExternalPropertyINT32M(INFA_CT_HANDLE handle, const char* sPropName, INFA_INT32* pnPropValue);</code>	정수
<code>INFA_STATUS INFA_CTGetExternalPropertyBoolM(INFA_CT_HANDLE handle, const char* sPropName, INFA_BOOLEAN* pbPropValue);</code>	부울

- **INFA_CTGetExternalProperty<데이터 유형>U()**. 유니코드의 속성 값에 액세스합니다.

다음 테이블에는 구문이 나와 있습니다.

구문	속성 데이터 유형
<code>INFA_STATUS INFA_CTGetExternalPropertyStringU(INFA_CT_HANDLE handle, INFA_UNICHAR* sPropName, INFA_UNICHAR** psPropValue);</code>	문자열
<code>INFA_STATUS INFA_CTGetExternalPropertyStringU(INFA_CT_HANDLE handle, INFA_UNICHAR* sPropName, INFA_INT32* pnPropValue);</code>	정수
<code>INFA_STATUS INFA_CTGetExternalPropertyStringU(INFA_CT_HANDLE handle, INFA_UNICHAR* sPropName, INFA_BOOLEAN* pbPropValue);</code>	부울

반환 값 데이터 유형은 `INFA_STATUS`입니다. 반환 값에 `INFA_SUCCESS` 및 `INFA_FAILURE`를 사용합니다.

데이터 유형 재바인딩 함수

PowerCenter에서 포트를 기본 데이터 유형 이외의 데이터 유형과 다시 바인딩할 수 있습니다. 프로시저가 기본 데이터 유형 이외의 데이터 유형을 갖는 데이터에 액세스하도록 하려면 데이터 유형 재바인딩 함수를 사용하십시오. 포트를 호환되는 데이터 유형과 다시 바인딩해야 합니다.

이러한 함수는 초기화 함수에서만 사용할 수 있습니다.

출력 또는 입력/출력 포트의 데이터 유형을 다시 바인딩할 때 다음 규칙을 고려하십시오.

- 데이터 처리 함수를 사용하여 해당 포트에 대해 데이터 및 표시기를 설정합니다. 행 기반 모드에서는 `INFA_CTSetData()` 및 `INFA_CTSetIndicator()` 함수를 사용하고 배열 기반 모드에서는 `INFA_CTASetData()` 함수를 사용합니다.
- 출력 포트에 대해 `INFA_CTSetPassThruPort()` 함수를 호출하지 마십시오.

다음 테이블에는 호환되는 데이터 유형이 나열되어 있습니다.

기본 데이터 유형	호환되는 데이터 유형
문자	Unichar
Unichar	문자

기본 데이터 유형	호환되는 데이터 유형
날짜	INFA_DATETIME 다음 구문을 사용합니다. <pre>struct INFA_DATETIME { int nYear; int nMonth; int nDay; int nHour; int nMinute; int nSecond; int nNanoSecond; }</pre>
Dec18	문자, Unichar
Dec28	문자, Unichar

PowerCenter에서는 다음과 같은 데이터 유형 재바인딩 함수를 제공합니다.

- **INFA_CTRebindInputDataType().** 입력 포트를 다시 바인딩합니다. 다음 구문을 사용합니다.
`INFA_STATUS INFA_CTRebindInputDataType(INFA_CT_INPUTPORT_HANDLE portHandle, INFA_CDATATYPE datatype);`
 - **INFA_CTRebindOutputDataType().** 출력 포트를 다시 바인딩합니다. 다음 구문을 사용합니다.
`INFA_STATUS INFA_CTRebindOutputDataType(INFA_CT_OUTPUTPORT_HANDLE portHandle, INFA_CDATATYPE datatype);`
- 다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
portHandle	INFA_CT_OUTPUTPORT_HANDLE	입력	출력 포트 핸들입니다.
데이터 유형	INFA_CDATATYPE	입력	<p>포트를 다시 바인딩할 데이터 유형입니다. datatype 매개 변수에 대해 다음 값을 사용하십시오.</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_CTDATEIME

반환 값 데이터 유형은 INFA_STATUS입니다. 반환 값에 INFA_SUCCESS 및 INFA_FAILURE를 사용합니다.

데이터 처리 함수(행 기반 모드)

통합 서비스는 입력 행 알림 함수를 호출할 때 프로시저가 행 또는 데이터 블록에 액세스할 수 있음을 프로시저에 알립니다. 하지만 데이터를 입력 포트에서 가져오고, 데이터를 수정하고, 출력 포트에서 데이터를 설정하려면 입력 행 알림 함수에서 데이터 처리 함수를 사용해야 합니다. 데이터 액세스 모드가 행 기반일 경우 행 기반 데이터 처리 함수를 사용하십시오.

데이터를 입력 포트에서 가져오려면 `INFA_CTGetData<데이터 유형>()` 함수를 포함하고 출력 포트에서 데이터를 설정하려면 `INFA_CTSetData()` 함수를 포함합니다. 데이터를 가져오기 전에 포트에 null 값 또는 빈 문자열이 있는지 여부를 프로시저에서 확인하려면 `INFA_CTGetIndicator()` 또는 `INFA_CTGetLength()` 함수를 사용합니다.

PowerCenter는 다음과 같은 데이터 처리 함수를 제공합니다.

- `INFA_CTGetData<데이터 유형>()`
- `INFA_CTSetData()`
- `INFA_CTGetIndicator()`
- `INFA_CTSetIndicator()`
- `INFA_CTGetLength()`
- `INFA_CTSetLength()`

데이터 가져오기 함수(행 기반 모드)

`INFA_CTGetData<데이터 유형>()` 함수를 사용하여 함수에서 지정하는 포트의 데이터를 가져옵니다.

프로시저에서 액세스하려는 포트의 데이터 유형에 따라 함수 이름을 수정해야 합니다.

다음 테이블에는 `INFA_CTGetData<데이터 유형>()` 함수 구문과 반환 값의 데이터 유형이 나와 있습니다.

구문	반환 값 데이터 유형
<code>void* INFA_CTGetDataVoid(INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	반환 값에 대한 데이터 void 포인터
<code>char* INFA_CTGetDataStringM(INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	문자열(MBCS)
<code>IUNICHAR* INFA_CTGetDataStringU(INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	문자열(유니코드)
<code>INFA_INT32 INFA_CTGetDataINT32(INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	정수
<code>double INFA_CTGetDataDouble(INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	배정밀도
<code>INFA_CT_RAWDATE INFA_CTGetDataDate(INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	원시 날짜
<code>INFA_CT_RAWDEC18 INFA_CTGetDataRawDec18(INFA_CT_INPUTPORT_HANDLE dataHandle);</code>	10진수 BLOB(전체 자릿수 18)

구문	반환 값 데이터 유형
INFA_CT_RAWDEC28 INFA_CTGetDataRawDec28(INFA_CT_INPUTPORT_HANDLE dataHandle);	10진수 BLOB(전체 자릿수 28)
INFA_CT_DATETIME INFA_CTGetDataDateTime(INFA_CT_INPUTPORT_HANDLE dataHandle);	날짜/시간

데이터 설정 함수(행 기반 모드)

프로시저가 출력 포트에 값을 전달하도록 하려면 INFA_CTSetData() 함수를 사용합니다.

다음 구문을 사용합니다.

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

반환 값 데이터 유형은 INFA_STATUS입니다. 반환 값에 INFA_SUCCESS 및 INFA_FAILURE를 사용합니다.

참고: 입력/출력 포트에 대해 INFA_CTSetPassThruPort() 함수를 사용하는 경우, 해당 포트에 대해 데이터 또는 표시기 설정을 사용하지 마십시오.

표시기 함수(행 기반 모드)

프로시저에서 입력 포트에 대한 표시기를 가져오거나 출력 포트에 대한 표시기를 설정하려면 표시기 함수를 사용합니다. 포트의 표시기는 데이터가 유효한지, null인지 또는 잘렸는지를 나타냅니다.

PowerCenter는 다음과 같은 표시기 함수를 제공합니다.

- **INFA_CTGetIndicator().** 입력 포트에 대한 표시기를 가져옵니다. 다음 구문을 사용합니다.

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

반환 값 데이터 유형은 INFA_INDICATOR입니다. INFA_INDICATOR에 다음 값을 사용하십시오.

- **INFA_DATA_VALID.** 데이터가 유효함을 나타냅니다.
- **INFA_NULL_DATA.** null 값을 나타냅니다.
- **INFA_DATA_TRUNCATED.** 데이터가 잘렸음을 나타냅니다.

- **INFA_CTSetIndicator().** 출력 포트에 대한 표시기를 설정합니다. 다음 구문을 사용합니다.

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

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
dataHandle	INFA_CT_OUTPUTPORT_HANDLE	입력	출력 포트 핸들입니다.
표시기	INFA_INDICATOR	입력	출력 포트에 대한 표시기 값입니다. 다음 값 중 하나를 사용합니다. <ul style="list-style-type: none"> - INFA_DATA_VALID. 데이터가 유효함을 나타냅니다. - INFA_NULL_DATA. null 값을 나타냅니다. - INFA_DATA_TRUNCATED. 데이터가 잘렸음을 나타냅니다.

반환 값 데이터 유형은 INFA_STATUS입니다. 반환 값에 INFA_SUCCESS 및 INFA_FAILURE를 사용합니다.

참고: INFA_CTSetPassThruPort() 함수를 입력/출력 포트에 대해 사용하는 경우 해당 포트에 대해 데이터 또는 표시기를 설정하지 마십시오.

길이 함수

프로시저에서 문자열 또는 이진 입력 포트의 길이에 액세스하거나 이진 또는 문자열 출력 포트의 길이를 설정하려면 길이 함수를 사용합니다.

다음 길이 함수를 사용하십시오.

- **INFA_CTGetLength().** 문자열 및 이진 포트에 대해서만 이 함수를 사용합니다. 통합 서비스가 후행 공백을 포함한 문자의 수로 길이를 반환합니다. 다음 구문을 사용합니다.

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

반환 값 데이터 유형은 INFA_UINT32입니다. 반환 값으로 0과 2GB 사이의 값을 사용합니다.

- **INFA_CTSetLength().** 사용자 지정 변환에 이진 또는 문자열 출력 포트가 포함된 경우 이 함수를 사용하여 후행 공백을 포함한 데이터의 길이를 설정해야 합니다. 문자열 및 이진 포트에 대해 설정하는 길이가 해당 포트의 전체 자릿수보다 크지 않은지 확인하십시오. 포트 전체 자릿수보다 크게 길이를 설정하면 예상치 않은 결과가 발생합니다. 예를 들어 세션이 실패할 수 있습니다.

다음 구문을 사용합니다.

```
INFA_STATUS INFA_CTSetLength(INFA_CT_OUTPUTPORT_HANDLE dataHandle, INFA_UINT32 length);
```

반환 값 데이터 유형은 INFA_STATUS입니다. 반환 값에 INFA_SUCCESS 및 INFA_FAILURE를 사용합니다.

통과 포트 설정 함수

통합 서비스가 데이터를 수정하지 않고 입력 포트에서 출력 포트에 데이터를 전달하도록 하려면 INFA_CTSetPassThruPort() 함수를 사용하십시오. INFA_CTSetPassThruPort() 함수를 사용할 경우, 통합 서비스는 입력 행 알림 함수를 호출할 때 데이터를 출력 포트에 전달합니다.

통과 포트 설정 함수를 사용할 때 다음 규칙 및 지침을 고려하십시오.

- 초기화 함수에서만 이 함수를 사용합니다.
- 프로시저에 이 함수가 포함되어 있으면 출력 포트에 데이터를 전달하기 위해 INFA_CTSetData(), INFA_CTSetLength, INFA_CTSetIndicator() 또는 INFA_CTASetData() 함수를 포함하지 마십시오.
- 행 기반 모드에서는 변환 범위가 행인 경우에만 이 함수를 포함할 수 있습니다. 변환 범위가 트랜잭션 또는 모든 입력이면 이 함수는 INFA_FAILURE를 반환합니다.
- 행 기반 모드에서 이 함수를 사용하여 특정 입력 행에 대해 여러 행을 출력하면 모든 출력 행에는 입력 포트로부터 전달되는 데이터가 포함됩니다.
- 배열 기반 모드에서는 수동 사용자 지정 변환에 대해서만 이 함수를 사용할 수 있습니다.

입력 포트와 출력 포트의 데이터 유형, 전체 자릿수 및 소수 자릿수가 동일해야 합니다.

INFA_CTSetPassThruPort() 함수에서 지정한 입력 포트와 출력 포트의 데이터 유형, 전체 자릿수 또는 소수 자릿수가 동일하지 않으면 통합 서비스는 세션이 실패한 것으로 처리합니다.

다음 구문을 사용합니다.

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

반환 값 데이터 유형은 INFA_STATUS입니다. 반환 값에 INFA_SUCCESS 및 INFA_FAILURE를 사용합니다.

출력 알림 함수

프로시저가 통합 서비스로 행을 출력하도록 하려면 `INFA_CTOutputNotification()` 함수를 사용합니다. 활성 사용자 지정 변환에 대해서만 이 함수를 포함하십시오. 수동 사용자 지정 변환의 경우, 프로시저는 입력 행 알림 함수가 반환 값을 제공할 때 통합 서비스로 행을 출력합니다. 프로시저가 수동 사용자 지정 변환에 대해 이 함수를 호출하면 통합 서비스는 해당 함수를 무시합니다.

참고: 변환 범위가 행인 경우, 입력 행 알림 함수에만 이 함수를 포함할 수 있습니다. 다른 곳에 이 함수를 포함하면 실패가 반환됩니다.

다음 구문을 사용합니다.

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

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
그룹	INFA_CT_OUTPUT_GROUP_HANDLE	입력	출력 그룹 핸들입니다.

반환 값 데이터 유형은 `INFA_ROWSTATUS`입니다. 다음 값을 반환 값으로 사용합니다.

- **INFA_ROWSUCCESS.** 함수가 데이터 행을 성공적으로 처리했음을 나타냅니다.
- **INFA_ROWERROR.** 함수가 데이터 행을 처리하는 동안 오류가 발생했음을 나타냅니다. 통합 서비스가 내부 오류 수를 증분시킵니다.
- **INFA_FATALERROR.** 함수가 데이터 행을 처리하는 동안 심각한 오류가 발생했음을 나타냅니다. 통합 서비스가 세션에 실패했습니다.

참고: 프로시저 코드가 `INFA_CTOutputNotification()` 함수를 호출하는 경우, 출력 포트 핸들의 모든 포인터가 올바른 데이터를 가리키는지 확인해야 합니다. 포인터가 올바른 데이터를 가리키지 않으면 통합 서비스가 예기치 않게 종료될 수 있습니다.

데이터 경계 출력 알림 함수

프로시저에서 커밋 또는 롤백 트랜잭션을 출력하려면 `INFA_CTDataBdryOutputNotification()` 함수를 포함합니다.

이 함수를 사용하는 경우 이 사용자 지정 변환에 대해 트랜잭션 생성 속성을 선택해야 합니다. 이 속성을 선택하지 않으면 통합 서비스가 세션을 실행하지 못합니다.

다음 구문을 사용합니다.

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

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
핸들	INFA_CT_PARTITION_HANDLE	입력	핸들 이름입니다.
dataBoundaryType	INFA_CTDataBdryType	입력	트랜잭션 유형입니다. dataBoundaryType 매개 변수에 다음 값을 사용합니다. - eBT_COMMIT - eBT_ROLLBACK

반환 값 데이터 유형은 INFA_STATUS입니다. 반환 값에 INFA_SUCCESS 및 INFA_FAILURE를 사용합니다.

오류 함수

오류 함수를 사용하여 프로시저 오류에 액세스합니다. 통합 서비스는 가장 최신 오류를 반환합니다.

PowerCenter는 다음과 같은 오류 함수를 제공합니다.

- **INFA_CTGetErrorMsgM()**. MBCS로 오류 메시지를 가져옵니다. 다음 구문을 사용합니다.

```
const char* INFA_CTGetErrorMsgM();
```
- **INFA_CTGetErrorMsgU()**. 유니코드로 오류 메시지를 가져옵니다. 다음 구문을 사용합니다.

```
const IUNICHAR* INFA_CTGetErrorMsgU();
```

세션 로그 메시지 함수

프로시저가 세션 로그에 유니코드 또는 MBCS로 메시지를 기록하도록 하려면 세션 로그 메시지 함수를 사용합니다.

PowerCenter에서는 다음과 같은 세션 로그 메시지 함수를 제공합니다.

- **INFA_CTLogMessageU()**. 유니코드로 메시지를 기록합니다.
 다음 구문을 사용합니다.

```
void INFA_CTLogMessageU(INFA_CT_ErrorSeverityLevel errorseverityLevel, INFA_UNICHAR* msg)
```

 다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
errorSeverityLevel	INFA_CT_ErrorSeverityLevel	입력	통합 서비스가 세션 로그에 기록하도록 할 오류 메시지의 심각도 수준입니다. errorSeverityLevel 매개 변수에 대해 다음 값을 사용하십시오. - eESL_LOG - eESL_DEBUG - eESL_ERROR
msg	INFA_UNICHAR*	입력	유니코드 메시지 텍스트를 따옴표 안에 입력합니다.

- **INFA_CTLogMessageM()**. MBCS로 메시지를 기록합니다.

다음 구문을 사용합니다.

```
void INFA_CTLogMessageM(INFA_CT_ErrorSeverityLevel errorSeverityLevel, char* msg)
```

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
errorSeverityLevel	INFA_CT_ErrorSeverityLevel	입력	통합 서비스가 세션 로그에 기록하도록 할 오류 메시지의 심각도 수준입니다. errorSeverityLevel 매개 변수에 대해 다음 값을 사용하십시오. - eESL_LOG - eESL_DEBUG - eESL_ERROR
msg	문자 *	입력	MBCS 메시지 텍스트를 따옴표 안에 입력합니다.

오류 개수 증분 함수

세션의 오류 수를 늘리려는 경우 **INFA_CTIncrementErrorCount()** 함수를 사용합니다.

다음 구문을 사용합니다.

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

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
변환	INFA_CT_PARTITION_HANDLE	입력	파티션 핸들입니다.
nErrors	size_t	입력	통합 서비스가 지정된 변환 인스턴스에 대해 nErrors만큼 오류 수를 증분시킵니다.
pStatus	INFA_STATUS*	입력	오류 수가 오류 임계값을 초과하고 세션이 실패하면 통합 서비스가 pStatus 매개 변수에 INFA_FAILURE를 사용합니다.

반환 값 데이터 유형은 **INFA_STATUS**입니다. 반환 값에 **INFA_SUCCESS** 및 **INFA_FAILURE**를 사용합니다.

종료됨 함수

PowerCenter 클라이언트가 통합 서비스에 세션을 중지하도록 요청했는지 여부를 프로시저에서 확인하려면 **INFA_CTIsTerminated()** 함수를 사용합니다. 프로시저에 시간이 많이 소요되는 프로세스가 있을 경우 이 함수를 호출할 수 있습니다.

다음 구문을 사용합니다.

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

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
핸들	INFA_CT_PARTITION_HANDLE	입력	파티션 핸들입니다.

반환 값 데이터 유형은 INFA_CTTerminateType입니다. 통합 서비스는 다음 값 중 하나를 반환합니다.

- **eTT_NOTTERMINATED.** PowerCenter 클라이언트가 세션을 중지하도록 요청하지 않았음을 나타냅니다.
- **eTT_ABORTED.** 통합 서비스가 세션을 중단했음을 나타냅니다.
- **eTT_STOPPED.** 통합 서비스가 세션을 중지했음을 나타냅니다.

차단 함수

사용자 지정 변환에 여러 입력 그룹이 있을 경우 입력 그룹의 수신 데이터를 차단하는 코드를 작성할 수 있습니다.

차단 함수를 사용할 때 다음 규칙을 고려하십시오.

- $n-1$ 개의 입력 그룹만 차단할 수 있습니다.
- 이미 차단된 입력 그룹은 차단할 수 없습니다.
- 다른 입력 그룹과 동일한 소스에서 데이터를 수신하는 입력 그룹은 차단할 수 없습니다.
- 이미 차단 해제된 입력 그룹은 차단할 수 없습니다.

PowerCenter는 다음과 같은 차단 함수를 제공합니다.

- **INFA_CTBlockInputFlow().** 프로시저에서 입력 그룹을 차단할 수 있도록 합니다.

다음 구문을 사용합니다.

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

- **INFA_CTUnblockInputFlow().** 프로시저에서 입력 그룹을 차단 해제할 수 있도록 합니다.

다음 구문을 사용합니다.

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

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
그룹	INFA_CT_INPUTGROUP_HANDLE	입력	입력 그룹 핸들입니다.

반환 값 데이터 유형은 INFA_STATUS입니다. 반환 값에 INFA_SUCCESS 및 INFA_FAILURE를 사용합니다.

차단 확인

프로시저 코드에서 INFA_CTBlockInputFlow() 및 INFA_CTUnblockInputFlow() 함수를 사용하는 경우, 사용자 지정 변환이 수신 데이터를 차단하는 것을 통합 서비스가 허용하는지 여부를 해당 프로시저에서 검사하는지 확인하십시오. 이렇게 하려면 INFA_CTGetInternalPropertyBool() 함수를 사용하여 INFA_CT_TRANS_MAY_BLOCK_DATA propID의 값을 검사합니다.

INFA_CT_TRANS_MAY_BLOCK_DATA propID의 값이 FALSE이면 프로시저는 차단 함수를 사용하지 않거나 심각한 오류를 반환하고 세션을 중지해야 합니다.

사용자 지정 변환이 데이터를 차단하는 것을 통합 서비스에서 허용하지 않을 때 프로시저 코드가 차단 함수를 사용하면 통합 서비스는 세션이 실패한 것으로 처리할 수 있습니다.

포인터 함수

통합 서비스가 개체 또는 구조에 대한 포인터를 작성하고 액세스하도록 하려면 포인터 함수를 사용합니다.

PowerCenter에서는 다음과 같은 포인터 함수를 제공합니다.

- **INFA_CTGetUserDefinedPtr()**. 프로시저가 런타임 중에 개체 또는 구조에 액세스할 수 있게 합니다.

다음 구문을 사용합니다.

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

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
핸들	INFA_CT_HANDLE	입력	핸들 이름입니다.

- **INFA_CTSetUserDefinedPtr()**. 프로시저가 개체 또는 구조를 통합 서비스에서 제공하는 핸들과 연결할 수 있게 합니다. 처리 오버헤드를 줄이려면 초기화 함수에서 이 함수를 포함합니다.

다음 구문을 사용합니다.

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

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
핸들	INFA_CT_HANDLE	입력	핸들 이름입니다.
pPtr	void*	입력	사용자 포인터입니다.

INFA_CT_HANDLE을 유효한 핸들로 대체해야 합니다.

문자열 모드 변경 함수

통합 서비스는 유니코드 모드에서 실행되는 경우 기본적으로 데이터를 UCS-2 형식으로 프로시저에 전달합니다. ASCII 모드에서 실행되는 경우에는 기본적으로 데이터를 ASCII로 전달합니다. 프로시저의 기본 문자열 모드를 변경하려면 INFA_CTChangeStringMode() 함수를 사용하십시오. 기본 문자열 모드를 MBCS로 변경하면 통합 서비스가 데이터를 통합 서비스 코드 페이지로 전달합니다. 코드 페이지를 변경하려면 INFA_CTSetDataCodePageID() 함수를 사용하십시오.

프로시저에 INFA_CTChangeStringMode() 함수가 포함되면 통합 서비스가 각 사용자 지정 변환에서 이 특정 프로시저를 사용하는 모든 포트의 문자열 모드를 변경합니다.

초기화 함수에서 문자열 모드 변경 함수를 사용합니다.

다음 구문을 사용합니다.

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

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
프로시저	INFA_CT_PROCEDURE_HANDLE	입력	프로시저 핸들 이름입니다.
stringMode	INFA_CTStringMode	입력	통합 서비스에서 사용할 문자열 모드를 지정합니다. stringMode 매개 변수에 다음 값을 사용합니다. - eASM_UNICODE. 통합 서비스가 ASCII 모드에서 실행되고 프로시저에서 유니코드 형식의 데이터에 액세스하려는 경우 이 값을 사용합니다. - eASM_MBCS. 통합 서비스가 유니코드 모드에서 실행되고 프로시저에서 MBCS 형식의 데이터에 액세스하려는 경우 이 값을 사용합니다.

반환 값 데이터 유형은 INFA_STATUS입니다. 반환 값에 INFA_SUCCESS 및 INFA_FAILURE를 사용합니다.

데이터 코드 페이지 설정 함수

통합 서비스가 통합 서비스 코드 페이지와 다른 코드 페이지로 사용자 지정 변환에 데이터를 전달하도록 하려면 INFA_CTSetDataCodePageID()를 사용합니다.

프로시저 초기화 함수에서 데이터 코드 페이지 설정 함수를 사용하십시오.

다음 구문을 사용합니다.

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

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
변환	INFA_CT_TRANSFORMATION_HANDLE	입력	변환 핸들 이름입니다.
dataCodePageID	int	입력	통합 서비스가 데이터를 전달할 때 사용할 코드 페이지를 지정합니다. dataCodePageID 매개 변수의 올바른 값은 Administrator 가이드의 “코드 페이지”를 참조하십시오.

반환 값 데이터 유형은 INFA_STATUS입니다. 반환 값에 INFA_SUCCESS 및 INFA_FAILURE를 사용합니다.

행 전략 함수(행 기반 모드)

행 전략 함수를 사용하면 각 행에 대한 업데이트 전략을 액세스하고 구성할 수 있습니다.

PowerCenter에서는 다음과 같은 행 전략 함수를 제공합니다.

- **INFA_CTGetRowStrategy()**. 프로시저가 행에 대한 업데이트 전략을 가져올 수 있게 합니다.

다음 구문을 사용합니다.

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

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/ 출력	설명
그룹	INFA_CT_INPUTGROUP_HANDLE	입력	입력 그룹 핸들입니다.
updateStrategy	INFA_CT_UPDATESTRATEGY	입력	입력 포트에 대한 업데이트 전략입니다. 통합 서비스는 다음 값을 사용합니다. - eUS_INSERT = 0 - eUS_UPDATE = 1 - eUS_DELETE = 2 - eUS_REJECT = 3

- **INFA_CTSetRowStrategy()**. 각 행에 대한 업데이트 전략을 설정합니다. 이 함수는 INFA_CTChangeDefaultRowStrategy 함수보다 우선합니다.

다음 구문을 사용합니다.

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

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/ 출력	설명
그룹	INFA_CT_OUTPUTGROUP_HANDLE	입력	출력 그룹 핸들입니다.
updateStrategy	INFA_CT_UPDATESTRATEGY	입력	출력 포트에 대해 설정하려는 업데이트 전략입니다. 다음 값 중 하나를 사용합니다. - eUS_INSERT = 0 - eUS_UPDATE = 1 - eUS_DELETE = 2 - eUS_REJECT = 3

반환 값 데이터 유형은 INFA_STATUS입니다. 반환 값에 INFA_SUCCESS 및 INFA_FAILURE를 사용합니다.

기본 행 전략 변경 함수

변환 범위가 행일 경우 기본적으로 사용자 지정 변환의 행 전략은 통과입니다. 변환 범위가 트랜잭션 또는 모든 입력일 경우 기본적으로 소스 행을 다음으로 처리 세션 속성과 동일한 값입니다.

예를 들어 매핑에 업데이트 전략 변환과 그 뒤에 행 변환 범위의 사용자 지정 변환이 있다고 가정합니다. 업데이트 전략 변환은 행에 업데이트, 삽입 또는 삭제 플래그를 지정합니다. 통합 서비스가 행을 사용자 지정 변환에 전달하면 사용자 지정 변환은 행 전략이 통과되기 때문에 플래그를 유지합니다.

하지만 PowerCenter에서 사용자 지정 변환의 행 전략을 변경할 수 있습니다.

INFA_CTChangeDefaultRowStrategy() 함수를 사용하여 변환 수준에서 기본 행 전략을 변경합니다. 예를 들어

사용자 지정 변환의 기본 행 전략을 삽입으로 변경하면 통합 서비스가 이 변환을 통해 통과하는 모든 행에 삽입 플래그를 지정합니다.

참고: 세션이 데이터 구동 모드가 아닐 경우 통합 서비스는 `INFA_FAILURE`를 반환합니다.

다음 구문을 사용합니다.

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

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
변환	INFA_CT_TRANSFORMATION_HANDLE	입력	변환 핸들입니다.
defaultUpdateStrategy	INFA_CT_DefaultUpdateStrategy	입력	통합 서비스가 사용자 지정 변환에 사용할 행 전략을 지정합니다. <ul style="list-style-type: none"> - <code>eDUS_PASSTHROUGH</code>. 행에 통과 플래그를 지정합니다. - <code>eDUS_INSERT</code>. 행에 삽입 플래그를 지정합니다. - <code>eDUS_UPDATE</code>. 행에 업데이트 플래그를 지정합니다. - <code>eDUS_DELETE</code>. 행에 삭제 플래그를 지정합니다.

반환 값 데이터 유형은 `INFA_STATUS`입니다. 반환 값에 `INFA_SUCCESS` 및 `INFA_FAILURE`를 사용합니다.

배열 기반 API 함수

배열 기반 함수는 데이터 액세스 모드를 배열 기반으로 변경할 때 사용하는 API 함수입니다.

Informatica는 다음과 같은 배열 기반 API 함수 그룹을 제공합니다.

- 최대 행 수
- 행 수
- 올바른 행입
- 데이터 처리(배열 기반 모드)
- 행 전략
- 입력 오류 행 설정

최대 행 수 함수

기본적으로 통합 서비스는 입력 블록과 출력 블록에서 최대 행 수를 허용합니다. 하지만 출력 블록에서 허용되는 최대 행 수를 변경할 수 있습니다.

INFA_CTAGetInputNumRowsMax() 함수와 INFA_CTAGetOutputNumRowsMax() 함수를 사용하여 입력 블록과 출력 블록의 최대 행 수를 확인합니다. 프로시저에 버퍼가 필요할 경우 이러한 함수에서 반환하는 값을 사용하여 버퍼 크기를 확인합니다.

INFA_CTASetOutputRowMax() 함수를 사용하여 출력 블록의 최대 행 수를 설정할 수 있습니다. 프로시저에서 좀 더 큰 버퍼 또는 작은 버퍼를 사용하려면 이 함수를 사용할 수 있습니다.

이러한 함수는 초기화 함수에서만 호출할 수 있습니다.

PowerCenter에서는 블록의 최대 행 수를 확인하고 설정하기 위한 다음과 같은 함수를 제공합니다.

- **INFA_CTAGetInputNumRowsMax()**. 이 함수를 사용하여 입력 블록에서 허용되는 최대 행 수를 확인합니다.

다음 구문을 사용합니다.

```
INT32 INFA_CTAGetInputRowMax( INFA_CT_INPUTGROUP_HANDLE inputgroup );
```

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
inputgroup	INFA_CT_INPUTGROUP_HANDLE	입력	입력 그룹 핸들입니다.

- **INFA_CTAGetOutputNumRowsMax()**. 이 함수를 사용하여 출력 블록에서 허용되는 최대 행 수를 확인합니다.

다음 구문을 사용합니다.

```
INT32 INFA_CTAGetOutputRowMax( INFA_CT_OUTPUTGROUP_HANDLE outputgroup );
```

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
outputgroup	INFA_CT_OUTPUTGROUP_HANDLE	입력	출력 그룹 핸들입니다.

- **INFA_CTASetOutputRowMax()**. 이 함수를 사용하여 출력 블록에서 허용되는 최대 행 수를 설정합니다.

다음 구문을 사용합니다.

```
INFA_STATUS INFA_CTASetOutputRowMax( INFA_CT_OUTPUTGROUP_HANDLE outputgroup, INFA_INT32 nRowMax );
```

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
outputgroup	INFA_CT_OUTPUTGROUP_HANDLE	입력	출력 그룹 핸들입니다.
nRowMax	INFA_INT32	입력	출력 블록에서 허용할 최대 행 수입니다. 양수를 입력해야 합니다. 0을 포함하여 양수가 아닌 수를 사용할 경우 함수가 심각한 오류를 반환합니다.

행 수 함수

입력 블록에 있는 행의 수를 확인하거나 지정된 입력 또는 출력 그룹에 대해 출력 블록의 행 수를 설정하려면 행 수 함수를 사용합니다.

PowerCenter에서는 다음과 같은 행 수 함수를 제공합니다.

- **INFA_CTAGetNumRows()**. 입력 블록의 행 수를 확인할 수 있습니다.

다음 구문을 사용합니다.

```
INFA_INT32 INFA_CTAGetNumRows( INFA_CT_INPUTGROUP_HANDLE inputgroup );
```

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
inputgroup	INFA_CT_INPUTGROUP_HANDLE	입력	입력 그룹 핸들입니다.

- **INFA_CTASetNumRows()**. 출력 블록의 행 수를 설정할 수 있습니다. 출력 알림 함수를 호출하기 전에 이 함수를 호출하십시오.

다음 구문을 사용합니다.

```
void INFA_CTASetNumRows( INFA_CT_OUTPUTGROUP_HANDLE outputgroup, INFA_INT32 nRows );
```

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
outputgroup	INFA_CT_OUTPUTGROUP_HANDLE	입력	출력 포트 핸들입니다.
nRows	INFA_INT32	입력	출력 블록에 정의하려는 행 수입니다. 양수를 입력해야 합니다. 양수가 아닌 숫자를 지정하면 통합 서비스는 출력 알림 함수가 실패한 것으로 처리합니다.

올바른 행임 함수

블록의 일부 행은 삭제된 행, 필터 행 또는 오류 행일 수 있습니다. INFA_CTAIsRowValid() 함수를 사용하여 블록의 행이 유효한지 확인할 수 있습니다. 행이 유효하면 이 함수는 INFA_TRUE를 반환합니다.

다음 구문을 사용합니다.

```
INFA_BOOLEAN INFA_CTAIsRowValid( INFA_CT_INPUTGROUP_HANDLE inputgroup, INFA_INT32 iRow);
```

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/ 출력	설명
inputgroup	INFA_CT_INPUTGROUP_HANDLE	입력	입력 그룹 핸들입니다.
iRow	INFA_INT32	입력	블록 내 행의 인덱스 번호입니다. 인덱스는 0부터 시작합니다. 프로시저가 데이터 블록에 존재하는 인덱스 번호만 전달하는지 확인해야 합니다. 잘못된 값을 전달하면 통합 서비스가 예기치 않게 종료됩니다.

데이터 처리 함수(배열 기반 모드)

통합 서비스는 `p_<proc_name>_inputRowNotification()` 함수를 호출할 때 프로시저가 행 또는 데이터 블록에 액세스할 수 있음을 프로시저에 알립니다. 하지만 데이터를 입력 포트에서 가져오고, 데이터를 수정하고, 배열 기반 모드의 출력 포트에서 데이터를 설정하려면 입력 행 알림 함수에서 배열 기반 데이터 처리 함수를 사용해야 합니다.

데이터를 입력 포트에서 가져오려면 `INFA_CTGetData<데이터 유형>()` 함수를 포함하고 출력 포트에서 데이터를 설정하려면 `INFA_CTSetData()` 함수를 포함합니다. 데이터를 가져오기 전에 포트에 null 값 또는 빈 문자열이 있는지 여부를 프로시저에서 확인하려면 `INFA_CTGetIndicator()` 함수를 사용합니다.

PowerCenter는 배열 기반 데이터 액세스 모드에 대해 다음과 같은 데이터 처리 함수를 제공합니다.

- `INFA_CTGetData<데이터 유형>()`
- `INFA_CTGetIndicator()`
- `INFA_CTSetData()`

데이터 가져오기 함수(배열 기반 모드)

`INFA_CTGetData<데이터 유형>()` 함수를 사용하여 함수에서 지정하는 포트의 데이터를 가져옵니다. 프로시저에서 액세스하려는 포트의 데이터 유형에 따라 함수 이름을 수정해야 합니다. 통합 서비스는 배열 기반 데이터 가져오기 함수에서 데이터의 길이를 전달합니다.

다음 테이블에는 `INFA_CTGetData<데이터 유형>()` 함수 구문과 반환 값의 데이터 유형이 나와 있습니다.

구문	반환 값 데이터 유형
<code>void* INFA_CTGetDataVoid(INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow, INFA_UINT32* pLength);</code>	반환 값에 대한 데이터 void 포인터
<code>char* INFA_CTGetDataStringM(INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow, INFA_UINT32* pLength);</code>	문자열(MBCS)
<code>IUNICHAR* INFA_CTGetDataStringU(INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow, INFA_UINT32* pLength);</code>	문자열(유니코드)
<code>INFA_INT32 INFA_CTGetDataINT32(INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow);</code>	정수
<code>double INFA_CTGetDataDouble(INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow);</code>	배정밀도

구문	반환 값 데이터 유형
<code>INFA_CT_RAWDATETIME INFA_CTGetDataRawDate(INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow);</code>	원시 날짜
<code>INFA_CT_DATETIME INFA_CTGetDataDateTime(INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow);</code>	날짜/시간
<code>INFA_CT_RAWDEC18 INFA_CTGetDataRawDec18(INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow);</code>	10진수 BLOB(전체 자릿수 18)
<code>INFA_CT_RAWDEC28 INFA_CTGetDataRawDec28(INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow);</code>	10진수 BLOB(전체 자릿수 28)

표시기 가져오기 함수(배열 기반 모드)

프로시저에서 입력 포트에 null 값이 있는지 여부를 확인하려면 표시기 가져오기 함수를 사용합니다.

다음 구문을 사용합니다.

```
INFA_INDICATOR INFA_CTGetIndicator( INFA_CT_INPUTPORT_HANDLE inputport, INFA_INT32 iRow );
```

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
inputport	INFA_CT_INPUTPORT_HANDLE	입력	입력 포트 핸들입니다.
iRow	INFA_INT32	입력	블록 내 행의 인덱스 번호입니다. 인덱스는 0부터 시작합니다. 프로시저가 데이터 블록에 존재하는 인덱스 번호만 전달하는지 확인해야 합니다. 잘못된 값을 전달하면 통합 서비스가 예기치 않게 종료됩니다.

반환 값 데이터 유형은 INFA_INDICATOR입니다. INFA_INDICATOR에 다음 값을 사용하십시오.

- **INFA_DATA_VALID.** 데이터가 유효함을 나타냅니다.
- **INFA_NULL_DATA.** null 값을 나타냅니다.
- **INFA_DATA_TRUNCATED.** 데이터가 잘렸음을 나타냅니다.

데이터 설정 함수(배열 기반 모드)

프로시저가 출력 포트에 데이터를 전달하도록 하려면 데이터 설정 함수를 사용합니다. 지정된 출력 포트에 대해 데이터, 데이터의 길이(해당하는 경우) 및 표시기를 설정할 수 있습니다. 출력 포트의 길이 또는 표시기를 설정하기 위해 서로 다른 함수를 사용하지는 않습니다.

다음 구문을 사용합니다.

```
void INFA_CTSetData( INFA_CT_OUTPUTPORT_HANDLE outputport, INFA_INT32 iRow, void* pData, INFA_UINT32 nLength, INFA_INDICATOR indicator);
```


다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
출력 포트	INFA_CT_OUTPUTPORT_HANDLE	입력	출력 포트 핸들입니다.
iRow	INFA_INT32	입력	블록 내 행의 인덱스 번호입니다. 인덱스는 0 부터 시작합니다. 프로시저가 데이터 블록에 존재하는 인덱스 번호만 전달하는지 확인해야 합니다. 잘못된 값을 전달하면 통합 서비스가 예기치 않게 종료됩니다.
pData	void*	입력	데이터에 대한 포인터입니다.
nLength	INFA_UINT32	입력	포트의 길이입니다. 문자열 및 이진 포트에 대해서만 사용됩니다. 함수가 데이터의 올바른 길이를 전달하는지 확인해야 합니다. 함수가 다른 길이를 전달하는 경우 출력 알림 함수는 이 포트에 대해 실패를 반환합니다. 문자열 및 이진 포트에 대해 설정한 길이가 해당 포트의 전체 자릿수보다 크지 않은지 확인합니다. 포트 전체 자릿수보다 크게 길이를 설정하면 예상치 않은 결과가 발생합니다. 예를 들어 세션이 실패할 수 있습니다.
표시기	INFA_INDICATOR	입력	출력 포트의 표시기 값입니다. 다음 값 중 하나를 사용합니다. - INFA_DATA_VALID. 데이터가 유효함을 나타냅니다. - INFA_NULL_DATA. null 값을 나타냅니다. - INFA_DATA_TRUNCATED. 데이터가 잘렸음을 나타냅니다.

행 전략 함수(배열 기반 모드)

배열 기반 행 전략 함수를 사용하면 블록의 각 행에 대한 업데이트 전략을 액세스하고 구성할 수 있습니다.

PowerCenter에서는 다음과 같은 행 전략 함수를 제공합니다.

- **INFA_CTGetRowStrategy()**. 프로시저가 블록의 행에 대한 업데이트 전략을 가져올 수 있게 합니다.

다음 구문을 사용합니다.

```
INFA_CT_UPDATESTRATEGY INFA_CTGetRowStrategy( INFA_CT_INPUTGROUP_HANDLE inputgroup, INFA_INT32 iRow);
```

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
inputgroup	INFA_CT_INPUTGROUP_HANDLE	입력	입력 그룹 핸들입니다.
iRow	INFA_INT32	입력	블록 내 행의 인덱스 번호입니다. 인덱스는 0부터 시작합니다. 프로시저가 데이터 블록에 존재하는 인덱스 번호만 전달하는지 확인해야 합니다. 잘못된 값을 전달하면 통합 서비스가 예기치 않게 종료됩니다.

- **INFA_CTASetRowStrategy()**. 블록의 행에 대한 업데이트 전략을 설정합니다.

다음 구문을 사용합니다.

```
void INFA_CTASetRowStrategy( INFA_CT_OUTPUTGROUP_HANDLE outputgroup, INFA_INT32 iRow,
INFA_CT_UPDATESTRATEGY updateStrategy );
```

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
outputgroup	INFA_CT_OUTPUTGROUP_HANDLE	입력	출력 그룹 핸들입니다.
iRow	INFA_INT32	입력	블록 내 행의 인덱스 번호입니다. 인덱스는 0부터 시작합니다. 프로시저가 데이터 블록에 존재하는 인덱스 번호만 전달하는지 확인해야 합니다. 잘못된 값을 전달하면 통합 서비스가 예기치 않게 종료됩니다.
updateStrategy	INFA_CT_UPDATESTRATEGY	입력	포트에 대한 업데이트 전략입니다. 다음 값 중 하나를 사용합니다. - eUS_INSERT = 0 - eUS_UPDATE = 1 - eUS_DELETE = 2 - eUS_REJECT = 3

입력 오류 행 설정 함수

배열 기반 액세스 모드를 사용할 경우, 입력 행 알림 함수에서 **INFA_ROWERROR**를 반환할 수 없습니다. 대신, 입력 오류 행 설정 함수를 사용하여 특정 입력 행에 오류가 있음을 통합 서비스에 알려 줍니다.

PowerCenter는 배열 기반 모드에서 다음과 같은 입력 행 설정 함수를 제공합니다.

- **INFA_CTASetInputErrorRowM()**. 입력 블록의 행에 오류가 있음을 통합 서비스에 알리고, MBCS 오류 메시지를 세션 로그로 출력하도록 통합 서비스에 알릴 수 있습니다.

다음 구문을 사용합니다.

```
INFA_STATUS INFA_CTASetInputErrorRowM( INFA_CT_INPUTGROUP_HANDLE inputGroup, INFA_INT32 iRow, size_t nErrors, INFA_MBCSCHAR* sErrMsg );
```

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
inputGroup	INFA_CT_INPUTGROUP_HANDLE	입력	입력 그룹 핸들입니다.
iRow	INFA_INT32	입력	블록 내 행의 인덱스 번호입니다. 인덱스는 0부터 시작합니다. 프로시저가 데이터 블록에 존재하는 인덱스 번호만 전달하는지 확인해야 합니다. 잘못된 값을 전달하면 통합 서비스가 예기치 않게 종료됩니다.
nErrors	size_t	입력	이 입력 행으로 인해 발생한 오류 수를 지정하려면 이 매개 변수를 사용합니다.
sErrMsg	INFA_MBCSCHAR*	입력	함수에서 출력하도록 하려는 오류 메시지를 포함하는 MBCS 문자열입니다. null로 끝나는 문자열을 입력해야 합니다. 이 매개 변수는 선택 사항입니다. 이 인수를 포함하면 행 오류 로깅을 활성화한 경우에도 통합 서비스가 세션 로그에 메시지를 출력합니다.

- **INFA_CTASetInputErrorRowU()**. 입력 블록의 행에 오류가 있음을 통합 서비스에 알리고, 유니코드 오류 메시지를 세션 로그로 출력하도록 통합 서비스에 알릴 수 있습니다.

다음 구문을 사용합니다.

```
INFA_STATUS INFA_CTASetInputErrorRowU( INFA_CT_INPUTGROUP_HANDLE inputGroup, INFA_INT32 iRow, size_t nErrors, INFA_UNICHAR* sErrMsg );
```

다음 테이블에는 이 함수의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
inputGroup	INFA_CT_INPUTGROUP_HANDLE	입력	입력 그룹 핸들입니다.
iRow	INFA_INT32	입력	블록 내 행의 인덱스 번호입니다. 인덱스는 0부터 시작합니다. 프로시저가 데이터 블록에 존재하는 인덱스 번호만 전달하는지 확인해야 합니다. 잘못된 값을 전달하면 통합 서비스가 예기치 않게 종료됩니다.

인수	데이터 유형	입력/ 출력	설명
nErrors	size_t	입력	이 출력 행으로 인해 발생한 오류 수를 지정하려면 이 매개 변수를 사용합니다.
sErrMsg	INFA_UNICHAR*	입력	<p>함수에서 출력하도록 하려는 오류 메시지를 포함하는 유니코드 문자열입니다. null로 끝나는 문자열을 입력해야 합니다.</p> <p>이 매개 변수는 선택 사항입니다. 이 인수를 포함하면 행 오류 로깅을 활성화한 경우에도 통합 서비스가 세션 로그에 메시지를 출력합니다.</p>

제 5 장

데이터 마스킹 변환

이 장에 포함된 항목:

- [데이터 마스킹 변환, 121](#)
- [마스킹 속성, 122](#)
- [키 마스킹, 124](#)
- [대체 마스킹, 126](#)
- [종속 마스킹, 129](#)
- [무작위 마스킹, 131](#)
- [마스킹 규칙 적용, 132](#)
- [식 마스킹, 136](#)
- [형식 유지 암호화, 138](#)
- [특수 마스크 형식, 139](#)
- [사회 보장 번호 마스킹, 139](#)
- [신용 카드 번호 마스킹, 140](#)
- [전화 번호 마스킹, 141](#)
- [전자 메일 주소 마스킹, 141](#)
- [사회 보험 번호 마스킹, 143](#)
- [IP 주소 마스킹, 143](#)
- [URL 주소 마스킹, 143](#)
- [기본값 파일, 144](#)
- [데이터 마스킹 변환 세션 속성, 144](#)
- [데이터 마스킹 변환에 대한 규칙 및 지침, 145](#)

데이터 마스킹 변환

데이터 마스킹 변환을 사용하여 중요한 프로덕션 데이터를 비프로덕션 환경의 실제와 같은 테스트 데이터로 변경합니다. 데이터 마스킹 변환은 사용자가 각 열에 구성하는 마스킹 규칙에 따라 소스 데이터를 수정합니다.

소프트웨어 개발, 테스트, 교육 및 데이터 마이닝을 위해 마스킹된 데이터를 작성할 수 있습니다. 마스킹된 데이터에서 데이터 관계를 유지하고 데이터베이스 테이블 간의 참조 무결성을 유지할 수 있습니다. 데이터 마스킹 변환은 수동 변환입니다.

데이터 마스크 변환은 사용자가 포트에 대해 구성하는 마스크 유형 및 소스 데이터 유형에 따른 마스크 규칙을 제공합니다. 문자열의 경우 문자열에서 바꿀 문자 및 마스크에서 적용할 문자를 제한할 수 있습니다. 숫자 및 날짜의 경우 마스크되는 데이터의 숫자 범위를 제공할 수 있습니다. 원래 숫자의 고정된 분산 또는 백분율 분산으로 범위를 구성할 수 있습니다. 통합 서비스는 사용자가 마스크 규칙으로 구성하는 로캘에 따라 문자를 바꿉니다.

데이터 마스크 변환을 사용하려면 해당하는 라이선스가 필요합니다.

관련 항목:

- [“포트의 기본값” 페이지 38](#)
- [“마스크 규칙 적용” 페이지 132](#)
- [“기본값 파일” 페이지 144](#)

마스크 속성

마스크 속성 탭에서 입력 포트를 정의하고 각 포트에 대한 마스크 속성을 구성합니다. 선택할 수 있는 마스크 유형은 포트 데이터 유형을 기반으로 합니다. 마스크 유형을 선택하는 경우 디자이너에 마스크 유형에 대한 마스크 규칙이 표시됩니다.

로캘

로캘은 데이터에서 문자의 언어 및 지역을 식별합니다. 목록에서 로캘을 선택합니다. 데이터 마스크 변환은 사용자가 선택하는 로캘의 문자를 사용하여 문자 데이터를 마스크합니다. 소스 데이터에는 사용자가 선택하는 로캘과 호환되는 문자가 포함되어야 합니다.

로캘이 목록에 없는 경우 유사하거나 일치하는 코드 페이지가 있는 로캘을 선택합니다. 로캘로 유니코드를 선택할 수 없습니다.

마스크 유형

마스크 유형은 선택한 열에 적용할 데이터 마스크의 유형입니다. 다음 마스크 유형 중 하나를 선택합니다.

- 키 마스크. 동일한 소스 데이터, 마스크 규칙 및 시드 값에 대한 고정 결과를 생성합니다.
- 대체 마스크. 사전에 있는 데이터 중 유사하지만 관계없는 데이터로 열의 데이터를 바꿉니다.
- 종속 마스크. 다른 소스 열의 값을 기반으로 소스 열 한 개의 값을 바꿉니다.
- 무작위 마스크. 동일한 소스 데이터 및 마스크 규칙에 대해 무작위 결과를 생성합니다.
- 식 마스크. 포트에 식을 적용하여 데이터를 변경하거나 데이터를 작성합니다.
- 특수 마스크 형식. 특수 마스크 형식을 중요 데이터의 일반 유형에 적용합니다. 사회 보장 번호, 사회 보험 번호, 신용 카드 번호, 전화 번호, URL 주소, 전자 메일 주소 또는 IP 주소를 마스크할 수 있습니다.
- 대체. 사전에 있는 데이터 중 유사하지만 관계없는 데이터로 열의 데이터를 바꿉니다.
- 마스크 없음. 데이터 마스크 변환이 소스 데이터를 변경하지 않습니다.

기본값은 마스크 없음입니다.

반복 가능 출력

반복 가능 출력은 데이터 마스킹 변환에서 반환되는 일관된 값 집합입니다.

반복 가능 출력은 특정 값을 반환합니다. 예를 들어 이름 열에 대한 반복 가능 출력을 구성한다고 가정합니다. 데이터 마스킹 변환은 워크플로우에 변환이 포함될 때마다 동일한 마스킹된 값을 반환합니다.

모든 데이터 마스킹 유형에 대해 반복 가능 마스킹을 구성할 수 있습니다. 반복 가능 마스킹을 구성하려면 **반복 가능 출력**을 클릭하고 **시드 값**을 선택합니다.

시드

시드 값은 마스킹된 값을 생성하기 위한 시작점입니다.

데이터 마스킹 변환은 1에서 1,000 사이의 난수인 기본 시드 값을 작성합니다. 시드 값을 다르게 입력하거나 매핑 매개 변수 값을 적용할 수 있습니다. 서로 다른 소스 데이터에서 동일한 마스킹된 데이터 값을 반환하려면 열에 동일한 시드 값을 적용하십시오. 예를 들어 네 개 테이블에 동일한 **Cust_ID** 열이 있으며 해당 열이 모두 동일한 마스킹된 값을 출력하게 하려면 네 개 열을 모두 동일한 시드 값으로 설정합니다.

매핑 매개 변수

매핑 매개 변수를 사용하여 시드 값을 정의할 수 있습니다. 변환에 추가할 각 시드 값에 대한 매핑 매개 변수를 작성합니다. 매핑 매개 변수 값은 1-1,000 사이 숫자입니다.

열에 대한 데이터 마스킹을 구성할 때 시드에 대한 매핑 매개 변수를 선택합니다. 디자이너에 매핑 매개 변수 목록이 표시됩니다. 목록에서 매핑 매개 변수를 선택합니다. 세션을 실행하기 전에 세션에 대한 매개 변수 파일에서 매핑 매개 변수 값을 변경할 수 있습니다.

데이터 마스킹 변환을 작성하기 전에 매핑 매개 변수를 작성합니다. 시드 값을 매개 변수화하도록 선택하고 매핑에 매핑 매개 변수가 없는 경우 오류가 나타납니다. 삭제된 매핑 매개 변수를 참조하는 마스킹 규칙이 있는 포트를 선택하면 디자이너가 이 포트에 대한 새로운 임의의 시드 값을 생성합니다. 시드 값은 매핑 매개 변수가 아닙니다. 매핑 매개 변수가 삭제되었다는 메시지가 나타나고 디자이너가 새 시드 값을 작성합니다.

다음과 같은 경우 통합 서비스가 기본 시드 값을 적용합니다.

- 매핑 매개 변수 옵션이 열에 대해 선택되었지만 세션에 매개 변수 파일이 없습니다.
- 사용자가 매핑 매개 변수를 삭제합니다.
- 매핑 매개 변수 시드 값이 1-1,000 사이가 아닙니다.

통합 서비스는 기본값 파일에서 마스킹된 값을 적용합니다. 기본값 파일을 편집하여 기본값을 변경할 수 있습니다.

기본값 파일은 XML 파일이며 다음 위치에서 찾을 수 있습니다.

```
<PowerCenter Installation Directory>\infa_shared\SrcFiles\defaultValue.xml
```

시드에 대한 이름-값 쌍은 다음과 같습니다.

```
default_seed = "500".
```

기본값 파일의 시드 값이 1-1,000 사이가 아닌 경우 통합 서비스가 시드에 값 725를 할당하고 세션 로그에 메시지를 씁니다.

관련 항목:

- [“기본값 파일” 페이지 144](#)

연결된 O/P

연결된 O/P는 입력 포트에 대한 연결된 출력 포트입니다. 데이터 마스킹 변환은 각 입력 포트에 대한 출력 포트를 작성합니다. 이름 지정 규칙은 `out_<포트 이름>`입니다. 연결된 출력 포트는 읽기 전용 포트입니다.

키 마스킹

키 마스킹이 구성된 열은 소스 값과 시드 값이 동일할 경우 항상 고정된 데이터를 마스킹된 데이터로 반환합니다. 데이터 마스킹 변환은 이러한 열에 대해 고유한 값을 반환합니다.

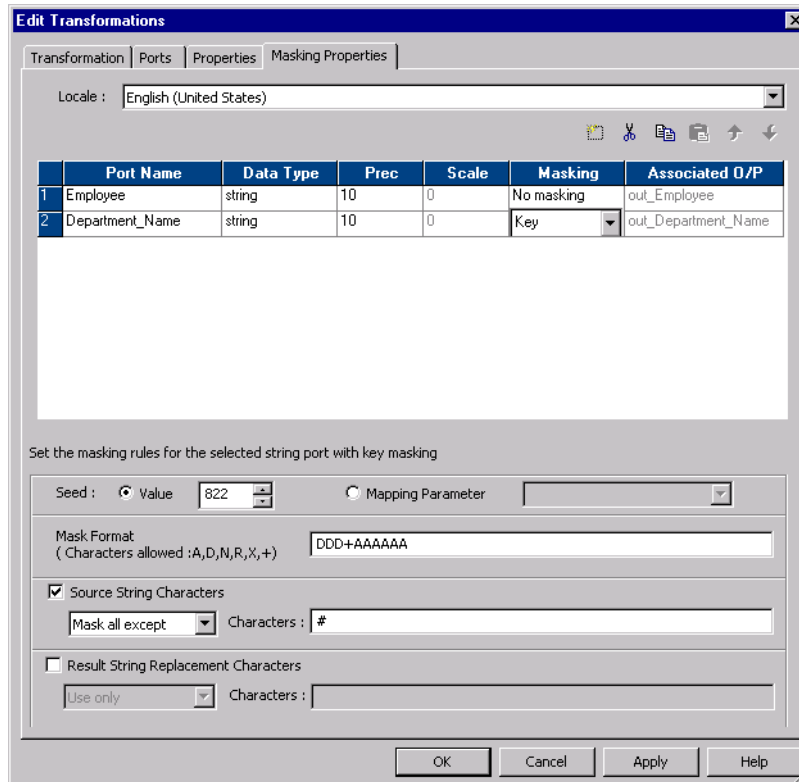
열에 키 마스킹을 구성할 경우 데이터 마스킹 변환이 해당 열에 대한 시드 값을 작성합니다. 이 시드 값을 변경하여 서로 다른 데이터 마스킹 변환 간에 반복 가능 데이터를 생성할 수 있습니다. 예를 들어 키 마스킹을 구성하여 참조 무결성을 시행하려는 경우 한 테이블의 기본 키와 다른 테이블의 외래 키를 마스킹하는 데 동일한 시드 값을 사용합니다.

마스킹 규칙을 정의하면 데이터 마스킹 변환이 반환하는 데이터 형식을 제어할 수 있습니다. 키 마스킹은 문자열 및 숫자 값을 마스킹하는 데 사용할 수 있습니다.

문자열 값 마스킹

반복 가능한 출력을 생성하도록 문자열에 대한 키 마스킹을 구성할 수 있습니다. 마스크 형식을 구성하여 출력 문자열의 각 문자에 대한 제한을 정의하고 마스크 형식을 정의합니다. 마스킹할 소스 문자 및 이러한 소스 문자를 사용하여 마스킹할 문자를 정의하는 소스 문자열 문자 및 결과 문자열 대체 문자를 구성합니다.

다음 그림에는 문자열 데이터 유형에 대한 키 마스크 속성이 나와 있습니다.



키 마스크 문자열 값에 대해 구성할 수 있는 마스크 규칙은 다음과 같습니다.

- **시드.** 열에 대해 확정 마스크된 데이터를 생성하기 위한 시드 값을 적용합니다. 다음 옵션 중 하나를 선택합니다.
 - **값.** 기본 시드 값을 수락하거나 1-1,000 사이 숫자를 입력합니다.
 - **매핑 매개 변수.** 매핑 매개 변수를 사용하여 시드 값을 정의합니다. 사용자가 매핑에 대해 작성하는 매핑 매개 변수 목록이 디자이너에 표시됩니다. 목록에서 시드 값으로 사용할 매핑 매개 변수를 선택합니다.
- **마스크 형식.** 입력 데이터의 각 문자를 대체할 문자 유형을 정의합니다. 각 문자를 알파벳, 숫자 또는 영숫자 문자 유형으로 제한할 수 있습니다.
- **소스 문자열 문자.** 마스크할 소스 문자열의 문자를 정의합니다. 예를 들어 입력 데이터에 발생하는 모든 숫자 기호(#) 문자를 마스크할 수 있습니다. 소스 문자열 문자가 비어 있는 경우 데이터 마스크 변환이 모든 입력 문자를 마스크합니다. 소스 문자열 문자의 수가 결과 문자열 문자의 수보다 작을 경우 데이터 마스크 변환이 고유하지 않은 데이터를 반환할 수 있습니다.
- **결과 문자열 문자.** 대상 문자열의 문자를 결과 문자열 문자에 정의된 문자로 대체합니다. 예를 들어 다음 문자를 입력하면 각 마스크에 모든 대문자 알파벳 문자가 포함되도록 구성합니다.

ABCDEFGHIJKLMNOPQRSTUVWXYZ

숫자 값 마스크

키 마스크를 숫자 소스 데이터에 구성하면 확정 출력이 생성됩니다. 열에 숫자 키 마스크를 구성할 경우 무작위 시드 값이 열에 할당됩니다. 데이터 마스크 변환은 시드가 필요한 마스크 알고리즘을 적용하여 소스 데이터를 마스크합니다.

동일한 소스 값이 여러 열에서 발생하는 경우 반복 가능한 결과를 생성하도록 열의 시드 값을 변경할 수 있습니다. 예를 들어 두 테이블의 기본-외래 키 관계를 유지해야 하는 경우 각 데이터 마스크 변환에서 기본 키 열의 시

드 값과 외래 키 열의 시드 값을 동일한 값으로 입력하면 데이터 마스킹 변환이 동일한 숫자 값에 대해 고정 결과를 생성하고 두 테이블 간의 참조 무결성이 유지됩니다.

날짜/시간 값 마스킹

날짜/시간 값에 대해 키 마스킹을 구성하려면 데이터 마스킹 변환에서 무작위 숫자를 시드로 입력해야 합니다. 다른 열의 시드 값과 일치하도록 시드 값을 변경하면 열 간에 반복 가능한 날짜/시간 값이 반환됩니다.

데이터 마스킹 변환은 키 마스킹을 사용하여 1753년부터 2400년까지의 날짜를 마스킹할 수 있습니다. 소스 연도가 윤년인 경우 데이터 마스킹 변환이 윤년을 반환합니다. 소스 월의 일 수가 31일인 경우 데이터 마스킹 변환이 일 수가 31일인 월을 반환합니다. 소스 월이 2월인 경우 데이터 마스킹 변환이 2월을 반환합니다.

데이터 마스킹 변환은 항상 유효한 날짜를 생성합니다.

관련 항목:

- [“날짜 범위” 페이지 135](#)

대체 마스킹

대체 마스킹은 유사하지만 관계 없는 데이터로 데이터 열을 바꿉니다. 대체 마스킹을 사용하여 프로덕션 데이터를 실제 테스트 데이터로 바꿉니다. 대체 마스킹을 구성하는 경우 대체 값이 포함된 사전을 정의합니다.

데이터 마스킹 변환이 사용자가 구성한 사전에서 조회를 수행합니다. 데이터 마스킹 변환이 소스 데이터를 사전의 데이터로 바꿉니다. 사전 파일에는 문자열 데이터, **datetime** 값, 정수 및 부동 소수점 숫자가 포함될 수 있습니다. **datetime** 값을 다음 형식으로 입력합니다.

mm/dd/yyyy

데이터를 반복 가능 또는 반복 불가능 값으로 대체할 수 있습니다. 반복 가능 값을 선택하는 경우 데이터 마스킹 변환이 동일한 소스 데이터 및 시드 값에 대해 고정 결과를 생성합니다. 데이터를 고정 결과로 대체하도록 시드 값을 구성해야 합니다. 통합 서비스가 반복 가능 마스킹을 위해 소스 및 마스킹된 값의 저장소 테이블을 유지합니다.

둘 이상의 데이터 열을 동일한 사전 행의 마스킹된 값으로 대체할 수 있습니다. 한 입력 열의 대체 마스킹을 구성합니다. 동일한 사전 행에서 마스킹된 데이터를 수신하는 다른 열의 종속 데이터 마스킹을 구성합니다.

사전

사전은 파일의 각 행에 대한 대체 데이터를 포함하는 플랫폼 파일 또는 관계형 테이블입니다. 데이터 마스킹 변환은 사전 행을 검색하기 위한 숫자를 생성합니다. 데이터 마스킹 변환에서 반복 가능한 대체 마스킹에 대한 해시 키 또는 반복 불가능한 마스킹에 대한 무작위 숫자를 생성합니다. 추가 조회 조건을 구성할 수 있습니다.

데이터 마스킹 변환에서 둘 이상의 포트를 마스킹하도록 사전을 구성할 수 있습니다.

다음 예제는 이름과 성별을 포함하는 플랫폼 파일 사전을 보여 줍니다.

```
SNO,GENDER,FIRSTNAME
1,M,Adam
2,M,Adeel
3,M,Adil
4,F,Alice
5,F,Alison
```

이 사전에서 행의 첫 번째 필드는 일련 번호이고 두 번째 필드는 성별입니다. 성별을 조회 조건으로 추가할 수 있습니다. 통합 서비스는 해시 키를 사용하여 사전에서 행을 검색하고, 소스 데이터의 성별과 일치하는 성별을 가진 행을 찾습니다.

사전을 작성할 때 다음 규칙과 지침을 따르십시오.

- 플랫폼 파일 사전의 첫 행에 각 레코드의 필드를 식별하는 열 레이블이 있어야 합니다. 필드는 쉼표로 구분됩니다. 첫 번째 행에 열 레이블이 없는 경우 통합 서비스는 첫 번째 행의 필드 값을 열 이름으로 가져옵니다.
- 플랫폼 파일 사전은 \$PMLookupFileDir 조회 파일 디렉터리에 있어야 합니다. 기본적으로 이 디렉터리는 다음 위치에 있습니다.

<PowerCenter_Installation_Directory>\server\infa_shared\LkpFiles

- Windows에서 플랫폼 파일 사전을 작성하여 UNIX 시스템으로 복사하는 경우 UNIX에서 파일 형식이 올바른지 확인하십시오. 예를 들어, Windows와 UNIX는 행 마커 끝에 다른 문자를 사용합니다.
- 두 개 이상의 포트에 대해 대체 마스킹을 구성하는 경우 모든 관계형 사전이 동일한 데이터베이스 스키마에 있어야 합니다.
- 플랫폼 파일 사전의 순차 정렬 버퍼 길이는 600자 이하여야 합니다.
- 세션 속성에서 사전 유형 또는 대체 사전 이름을 변경할 수 없습니다.

저장소 테이블

데이터 마스킹 변환은 세션 간 반복 가능 대체를 위해 저장소 테이블을 유지 관리합니다. 저장소 테이블 행에는 소스 열 및 마스킹된 값 쌍이 들어 있습니다. 데이터 마스킹 변환이 반복 가능 대체 값을 사용하여 값을 마스킹할 때마다 사전 이름, 로컬, 열 이름, 입력 값 및 시드로 저장소 테이블을 검색합니다. 행을 찾으면 저장소 테이블에서 마스킹된 값을 반환합니다. 데이터 마스킹 변환이 행을 찾지 못하면 해시 키를 사용하여 사전에서 행을 검색합니다.

플랫폼 파일 사전과 관계형 사전에 대한 저장소 테이블의 사전 이름 형식은 다릅니다. 플랫폼 파일 사전 이름은 파일 이름으로 식별됩니다. 관계형 사전 이름의 구문은 다음과 같습니다.

<Connection object>_<dictionary table name>

Informatica는 관계형 저장소 테이블을 작성하기 위해 실행할 수 있는 스크립트를 제공합니다. 스크립트는 다음 위치에 있습니다.

<PowerCenter Client installation directory>\client\bin\Extensions\DataMasking

디렉터리에는 Sybase, Microsoft SQL Server, IBM DB2 및 Oracle 데이터베이스용 스크립트가 포함되어 있습니다. 각 스크립트의 이름은 Substitution_<데이터베이스 유형>입니다. SQL 문 및 기본 키 제약 조건을 구성하면 다른 데이터베이스에서 테이블을 작성할 수 있습니다.

저장소에 암호화되지 않은 데이터가 있는 경우 대체 마스킹을 위해 저장소 테이블을 암호화하고, 동일한 시드 값 및 사전을 사용하여 동일한 열을 암호화해야 합니다.

대체 마스킹에 대한 저장소 테이블 암호화

변환 언어 인코딩 함수를 사용하여 저장소 테이블을 암호화할 수 있습니다. 저장소 암호화를 활성화한 경우 저장소 테이블을 암호화해야 합니다.

1. IDM_SUBSTITUTION_STORAGE 저장소 테이블을 소스로 사용하는 매핑을 작성합니다.
2. 데이터 마스킹 변환을 작성합니다.
3. 입력 값 및 마스킹된 값 포트에 대체 마스킹 기술을 적용합니다.
4. INPUTVALUE 포트에 다음 식을 사용합니다.

Enc_Base64(AES_Encrypt(INPUTVALUE, Key))

5. MASKEDVALUE 포트에 다음 식을 사용합니다.

Enc_Base64(AES_Encrypt(MASKEDVALUE, Key))

6. 포트를 대상에 연결합니다.

대체 마스크링 속성

대체 마스크링에 대해 다음과 같은 마스크링 규칙을 구성할 수 있습니다.

- **반복 가능 출력.** 세션 간에 확정 결과를 반환합니다. 데이터 마스크링 변환은 저장소 테이블에 마스크링된 값을 저장합니다.
- **시드.** 시드 값을 적용하면 열의 데이터가 고정 데이터로 마스크링됩니다. 다음 옵션 중 하나를 선택합니다.
 - **값.** 기본 시드 값을 수락하거나 1~1,000 사이 숫자를 입력합니다.
 - **매핑 매개 변수.** 매핑 매개 변수를 사용하여 시드 값을 정의합니다. 사용자가 매핑에 대해 작성하는 매핑 매개 변수 목록이 디자이너에 표시됩니다. 목록에서 시드 값으로 사용할 매핑 매개 변수를 선택합니다.
 - **고유한 출력.** 데이터 마스크링 변환이 고유한 입력 값에 대해 고유한 출력 값을 생성하도록 강제 적용합니다. 두 입력 값이 같은 출력 값으로 마스크링되지는 않습니다. 사전에는 고유 출력을 활성화하기에 충분한 고유 행이 있어야 합니다.
고유 출력을 비활성화하면 데이터 마스크링 변환이 입력 값을 고유 출력 값으로 마스크링할 수 없습니다. 사전에 행이 더 적게 포함될 수 있습니다.
 - **고유 키 열.** 마스크링 기술이 적용되는 소스 열입니다.
 - **사전 사용 최적화.** 반복 가능 출력 옵션을 선택하는 경우 적용됩니다. 사전에서 마스크링된 값의 사용을 늘립니다.
- **사전 정보.** 대체 데이터 값을 포함하는 플랫폼 파일 또는 관계형 테이블을 구성합니다.
 - **관계형 테이블.** 사전이 데이터베이스 테이블에 있으면 관계형 테이블을 선택합니다. 테이블 선택을 클릭하여 데이터베이스 및 테이블을 구성합니다.
 - **플랫폼 파일.** 심포로 구분된 플랫폼 파일에 사전이 있으면 플랫폼 파일을 선택합니다. 플랫폼 파일 선택을 클릭하여 파일을 찾아 선택합니다.
 - **사전 이름.** 선택한 플랫폼 파일 또는 관계형 테이블 이름을 표시합니다.
 - **출력 열.** 데이터 마스크링 변환으로 반환할 열을 선택합니다.
 - **정렬 열.** 항목을 정렬할 사전 열입니다. 사전의 항목 순서가 변경되는 경우에도 확정된 결과를 생성하려면 정렬 열을 지정합니다. 예를 들어 관계형 사전을 이동하고 항목 순서가 변경된 경우 일련 번호 열로 정렬하면 데이터를 일관되게 마스크링할 수 있습니다.
참고: 선택한 열은 고유한 값을 포함해야 합니다. 중복 값을 포함할 수 있는 열을 사용하여 데이터를 정렬할 수 없습니다.
- **조회 조건.** 대체 마스크링에 사용할 사전 행을 추가로 규정하는 조회 조건을 구성합니다. 조회 조건은 SQL 쿼리의 WHERE 절과 유사합니다. 조회 조건을 구성하는 경우 소스의 열 값을 사전의 열과 비교합니다.
이름을 마스크링하려는 경우를 예로 들어 보겠습니다. 소스 데이터와 사전에는 이름 열과 성별 열이 있습니다. 각 여성의 이름을 사전의 여성 이름으로 바꾸는 조건을 추가할 수 있습니다. 조회 조건은 소스의 성별을 사전의 성별과 비교합니다.
 - **입력 포트.** 조회에서 사용할 소스 데이터 열입니다.
 - **사전 열.** 입력 포트를 비교할 사전 열입니다.

관계형 사전

관계형 테이블을 사전으로 선택하는 경우, 해당 사전을 포함하는 데이터베이스 테이블을 구성합니다. 테이블을 정의하면 디자이너가 해당 테이블에 연결하고 테이블의 열을 표시합니다.

데이터베이스를 선택하려면 데이터 마스킹 변환의 마스킹 속성 탭에서 테이블 선택을 클릭합니다.

테이블 선택 대화 상자에서 다음 필드를 구성하십시오.

- **ODBC 데이터 소스.** 사전을 포함하는 데이터베이스에 연결하는 ODBC 데이터 소스입니다.
- **사용자 이름.** 데이터베이스 연결하기 위한 데이터베이스 사용자 이름입니다. 이 사용자 이름은 테이블을 볼 수 있는 데이터베이스 사용 권한을 갖고 있어야 합니다.
- **소유자 이름.** 사용자 이름이 사전 테이블의 소유자가 아니면 테이블 소유자를 입력합니다.
- **암호.** 데이터베이스 사용자 이름에 대한 암호입니다.
- **명명된 테이블 검색.** 선택 사항입니다. 대화 상자에 표시되는 테이블의 수를 제한합니다.

연결을 클릭하면 디자이너가 데이터 소스에 연결하고 테이블을 표시합니다. 테이블 목록을 스크롤하고 해당 사전에 대해 사용하려는 테이블을 선택합니다. 해당 테이블 이름이 마스킹 속성 탭의 사전 이름 필드에 나타납니다.

연결 요구 사항

저장소 테이블 및 관계형 사전을 대체 마스킹과 함께 사용하는 경우 이들에 대한 데이터베이스 연결을 구성해야 합니다.

워크플로우 관리자에서 데이터베이스 연결을 구성합니다. 세션 속성의 매핑 탭에서 **IDM_Dictionary** 연결 및 **IDM_Storage** 연결에 대한 관계형 연결 개체를 선택합니다.

대체 마스킹에 대한 규칙 및 지침

대체 마스킹에 대해 다음 규칙 및 지침을 사용하십시오.

- 고유한 반복 가능 대체 마스크에 대해 저장소 테이블이 존재하지 않는 경우 세션이 실패합니다.
- 사전에 행이 없는 경우 데이터 마스킹 변환이 오류 메시지를 반환합니다.
- 데이터 마스킹 변환이 저장소 테이블의 로컬, 사전 및 시드를 사용하여 입력 값을 찾는 경우 행이 사전에 더 이상 존재하지 않는 경우에도 마스킹된 값을 검색합니다.
- 연결 개체를 삭제하거나 사전을 수정하는 경우 저장소 테이블이 잠깁니다. 그렇지 않으면 예기치 않은 결과가 나타날 수 있습니다.
- 사전의 값 수가 소스 데이터의 고유한 값 수보다 적으면 데이터 마스킹 변환이 고유한 반복 가능 값을 사용하여 데이터를 마스킹할 수 없습니다. 데이터 마스킹 변환이 오류 메시지를 반환합니다.

종속 마스킹

종속 마스킹은 소스 데이터의 여러 열을 동일한 사전 행의 데이터로 대체합니다.

데이터 마스킹 변환에서 여러 열에 대해 대체 마스킹을 수행하면 마스킹된 데이터에 실제와 다른 필드 조합이 포함될 수 있습니다. 종속 마스킹을 구성하면 동일한 사전 행에서 여러 입력 열의 데이터를 대체할 수 있습니다. 마스킹된 데이터에는 "뉴욕, 뉴욕" 또는 "시카고, 일리노이"와 같이 유효한 조합이 포함됩니다.

종속 마스킹을 구성할 때는 먼저 입력 열에 대체 마스킹을 구성해야 합니다. 그런 다음 이 대체 열에 종속될 다른 입력 열을 구성합니다. 예를 들어 우편 번호 열에 대체 마스킹을 선택하고 이 우편 번호 열에 종속될 도로 및 주 열을 선택합니다. 종속 마스킹은 대체된 도로 및 주 열이 대체된 우편 번호 값에 유효한 값인지 확인합니다.

참고: 대체 마스킹을 먼저 구성하지 않은 열에는 종속 마스킹을 구성할 수 없습니다.

열에 종속 마스킹을 구성할 때 다음 마스킹 규칙을 구성해야 합니다.

종속 열

대체 마스크가 구성된 입력 열의 이름입니다. 데이터 마스크 변환은 해당 열의 마스크 규칙을 사용하여 사전에서 대체 데이터를 검색합니다. 대체 마스크가 구성된 열은 사전에서 마스크된 데이터를 검색할 때 키 열로 사용됩니다.

출력 열

종속 마스크가 구성된 열의 값을 포함하는 사전 열의 이름입니다.

종속 마스크 예제

데이터 마스크 사전에는 다음 값을 포함하는 주소 행이 포함될 수 있습니다.

```
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
```

주소 사전에 있는 도시, 주 및 우편 번호의 유효한 집합으로 소스 데이터를 마스크해야 합니다.

ZIP 포트에 대체 마스크를 구성합니다.

ZIP 포트에 다음 마스크 규칙을 입력합니다.

규칙	값
사전 FileType	플랫 파일 또는 관계형 테이블
사전 이름	Address.dic
출력 열	우편 번호

City 포트에 종속 마스크를 구성합니다.

City 포트에 다음 마스크 규칙을 입력합니다.

규칙	값
종속 열	우편 번호
출력 열	도시

State 포트에 종속 마스크를 구성합니다.

State 포트에 다음 마스크 규칙을 입력합니다.

규칙	값
종속 열	우편 번호
출력 열	상태

데이터 마스크 변환은 우편 번호를 마스크할 때 우편 번호에 해당하는 유효한 도시 및 주를 사전 행에서 반환합니다.

반복 가능 종속 마스킹

동일한 사전 행에서 마스킹된 데이터를 받도록 여러 입력 필드를 구성하는 경우, 대체 마스킹을 위해 어느 필드를 구성할지 결정해야 합니다. 반복 가능한 마스킹을 통해 소스 데이터를 고유하게 식별하는 열을 선택합니다. 데이터 마스킹 변환은 키 필드를 사용하여 반복 가능한 마스킹을 위한 저장소에 행이 이미 있는지 여부를 확인합니다.

참고: 소스 데이터에 기본 키가 있으면 대체 마스킹을 위해 기본 키 열을 구성할 수 있습니다. 종속 마스킹을 위해서는 다른 열을 구성하십시오.

예를 들어 직원 데이터를 마스킹하는 경우 대체 마스킹을 위해 **EmployeeID**를 구성하고 종속 마스킹을 위해 **FirstName** 및 **LastName**를 구성할 수 있습니다.

반복 가능한 마스킹을 구성하면 다음 행은 서로 다른 마스킹된 데이터를 받습니다.

EmployeeID	FirstName	LastName
111	John	Jones
222	Radhika	Jones

반복 가능 종속 마스킹의 키 필드로 **LastName**을 선택하면 성이 **Jones**인 각 행은 동일한 마스크 데이터를 받습니다.

무작위 마스킹

무작위 마스킹은 무작위로 확정되지 않은 마스킹된 데이터를 생성합니다. 데이터 마스킹 변환은 동일한 소스 값이 다른 행에서 발생하는 경우 다른 값을 반환합니다. 마스킹 규칙을 정의하면 데이터 마스킹 변환이 반환하는 데이터 형식을 제어할 수 있습니다. 숫자, 문자열 및 날짜 값을 무작위 마스킹으로 마스킹합니다.

숫자 값 마스킹

숫자 데이터를 마스킹할 경우 열에 대한 출력 값 범위를 구성할 수 있습니다. 데이터 마스킹 변환에서 포트 전체 자릿수에 따라 범위의 최소값 및 최대값 사이의 값을 반환합니다. 범위를 정의하려면 최소 및 최대 범위를 구성하거나 원본 소스 값의 분산을 기반으로 블러링 범위를 구성합니다.

숫자 데이터에 대해 다음 마스킹 매개 변수를 구성할 수 있습니다.

범위

출력 값 범위를 정의합니다. 데이터 마스킹 변환에서 최소값과 최대값 사이의 숫자 데이터를 반환합니다.

블러링 범위

소스 데이터의 고정된 분산 또는 백분율 분산 내에서 출력 값의 범위를 정의합니다. 데이터 마스킹 변환에서 소스 데이터 값에 가까운 숫자 데이터를 반환합니다. 범위 및 블러링 범위를 구성할 수 있습니다.

문자열 값 마스킹

문자열 열에 대한 무작위 출력을 생성하는 무작위 마스킹을 구성합니다. 출력 문자열의 각 문자에 대한 제한을 구성하려면 마스크 형식을 구성하십시오. 필터 문자를 구성하여 마스킹할 소스 문자 및 이러한 소스 문자를 사용하여 마스킹할 문자를 정의하십시오.

문자열 포트에는 다음 마스킹 규칙을 적용할 수 있습니다.

범위

최소 및 최대 문자열 길이를 구성합니다. 데이터 마스킹 변환은 최소 문자열 길이와 최대 문자열 길이 사이의 무작위 문자로 구성된 문자열을 반환합니다.

마스킹 형식

입력 데이터의 각 문자를 대체할 문자 유형을 정의합니다. 각 문자를 알파벳, 숫자 또는 영숫자 문자 유형으로 제한할 수 있습니다.

소스 문자열 문자

마스킹할 소스 문자열의 문자를 정의합니다. 예를 들어 입력 데이터에 발생하는 모든 숫자 기호(#) 문자를 마스킹할 수 있습니다. 소스 문자열 문자가 비어 있는 경우 데이터 마스킹 변환이 모든 입력 문자를 마스킹합니다.

결과 문자열 대체 문자

대상 문자열의 문자를 결과 문자열 문자에 정의된 문자로 대체합니다. 예를 들어 다음 문자를 입력하면 각 마스크에 대문자 알파벳 문자(A~Z)가 포함되도록 구성됩니다.

ABCDEFGHIJKLMNOPQRSTUVWXYZ

날짜 값 마스킹

무작위 마스킹으로 날짜 값을 마스킹하려면 출력 날짜의 범위를 구성하거나 분산을 선택합니다. 분산을 구성하는 경우 날짜에서 블러링할 부분을 선택합니다. 연도, 월, 일, 시간, 분 또는 초를 선택합니다. 데이터 마스킹 변환은 사용자가 구성한 범위의 날짜를 반환합니다.

날짜/시간 값을 마스킹하는 경우 다음 마스킹 규칙을 구성할 수 있습니다.

범위

선택한 날짜/시간 값에 대해 반환할 최소값 및 최대값을 설정합니다.

블러링

사용자가 날짜 단위에 적용한 분산을 바탕으로 날짜를 마스킹합니다. 데이터 마스킹 변환이 분산에 포함되는 날짜를 반환합니다. 연도, 월, 일, 시간, 분 또는 초를 블러링할 수 있습니다. 적용할 하한 및 상한 분산을 선택합니다.

마스킹 규칙 적용

소스 데이터 유형에 따라 마스킹 규칙을 적용합니다. 마스킹 속성 탭에서 열 속성을 클릭하면 포트의 데이터 유형에 따라 마스킹 규칙이 디자이너에 표시됩니다.

다음 테이블에는 마스킹 유형 및 소스 데이터 유형에 따라 구성할 수 있는 마스킹 규칙이 설명되어 있습니다.

마스킹 유형	소스 데이터 유형	마스킹 규칙	설명
무작위 및 키	문자열	마스킹 형식	출력 문자열의 각 문자를 알파벳, 숫자 또는 영숫자 문자로 제한하는 마스크입니다.
무작위 및 키	문자열	소스 문자열 문자	마스킹하거나 마스킹에서 제외할 소스 문자의 집합입니다.

마스킹 유형	소스 데이터 유형	마스킹 규칙	설명
무작위 및 키	문자열	결과 문자열 대체 문자	마스크에 포함하거나 제외할 문자의 집합입니다.
무작위	숫자 문자열 날짜/시간	범위	출력 값의 범위입니다. - 숫자. 데이터 마스킹 변환에서 최소값과 최대값 사이의 숫자 데이터를 반환합니다. - 문자열. 최소 문자열 길이와 최대 문자열 길이 사이의 무작위 문자로 구성된 문자열을 반환합니다. - 날짜/시간. 최소 날짜/시간과 최대 날짜/시간 사이의 날짜 및 시간을 반환합니다.
무작위	숫자 날짜/시간	블러링	소스 데이터의 고정 또는 백분율 분산을 포함하는 출력 값의 범위입니다. 데이터 마스킹 변환이 소스 데이터의 값에 근사한 데이터를 반환합니다. 날짜/시간 열에는 고정 분산이 필요합니다.

마스크 형식

마스크 형식을 구성하여 출력 열의 각 문자를 알파벳, 숫자 또는 영숫자 문자로 제한합니다. 다음 문자를 사용하여 마스크 형식을 정의하십시오.

A, D, N, X, +, R

참고: 마스크 형식에는 대문자가 포함됩니다. 소문자 마스크를 입력할 경우 데이터 마스킹 변환이 해당 문자를 대문자로 변환합니다.

다음 표에는 마스크 형식 문자가 설명되어 있습니다.

문자	설명
A	알파벳 문자. a~z 및 A~Z의 ASCII 문자를 예로 들 수 있습니다.
D	0~9의 자릿수. 데이터 마스킹 변환이 자릿수가 0~9가 아닌 문자에 대해 "X"를 반환합니다.
N	영숫자 문자. a~z, A~Z 및 0~9의 ASCII 문자를 예로 들 수 있습니다.
X	모든 문자. 영숫자 또는 기호를 예로 들 수 있습니다.
+	마스킹 없음.
R	나머지 문자. R을 지정하면 문자열의 나머지 문자가 모든 문자 유형이 될 수 있습니다. R은 마스크의 마지막 문자로 표시되어야 합니다.

예를 들어 부서 이름은 다음과 같은 형식을 가질 수 있습니다.

nnn-<department_name>

처음 3개 문자를 숫자로 바꾸고 부서 이름을 알파벳으로 바꾸고 출력에 대시를 유지하도록 마스크를 구성할 수 있습니다. 다음 마스크 형식을 구성하십시오.

DDD+AAAAAAAAAAAAAA

데이터 마스킹 변환이 첫 3개 문자를 숫자 문자로 바꿉니다. 네 번째 문자는 바뀌지 않습니다. 데이터 마스킹 변환이 나머지 문자를 알파벳 문자로 바꿉니다.

마스크 형식을 정의하지 않으면 데이터 마스크 변환이 각 소스 문자를 임의의 문자로 바꿉니다. 마스크 형식이 입력 문자열보다 긴 경우 데이터 마스크 변환이 마스크 형식에서 남는 문자를 무시합니다. 마스크 형식이 소스 문자열보다 짧은 경우 데이터 마스크 변환이 나머지 문자를 형식 R로 마스크합니다.

참고: 범위 옵션으로 마스크 형식을 구성할 수는 없습니다.

소스 문자열 문자

소스 문자열 문자는 마스크 또는 마스크하지 않기 위해 선택하는 소스 문자입니다. 소스 문자열에서 문자 위치는 중요하지 않습니다. 소스 문자는 대/소문자를 구분합니다.

원하는 수만큼 문자를 구성할 수 있습니다. 문자가 비어 있는 경우 데이터 마스크 변환이 열의 모든 소스 문자를 바꿉니다.

소스 문자열 문자에 대해 다음 옵션 중 하나를 선택합니다.

다음만 마스크

데이터 마스크 변환은 구성된 소스의 문자를 소스 문자열 문자로 마스크합니다. 예를 들어 문자 A, B 및 c를 입력하는 경우 문자가 소스 데이터에서 발생하면 데이터 마스크 변환이 A, B 또는 c를 다른 문자로 바꿉니다. A, B 또는 c가 아닌 소스 문자는 변경되지 않습니다. 마스크는 대/소문자를 구분합니다.

모두 마스크(다음 제외)

소스 문자열에서 발생하는 소스 문자열 문자를 제외하고 모든 문자를 마스크합니다. 예를 들어 필터 소스 문자 “-”를 입력하고 모두 마스크(다음 제외)를 선택하는 경우 소스 데이터에서 “-” 문자가 발생할 때 데이터 마스크 변환이 “-” 문자를 바꾸지 않습니다. 소수 문자의 나머지는 변경됩니다.

소스 문자열 예제

소스 파일에 Dependents라는 이름의 열이 있습니다. Dependents 열에는 쉼표로 구분된 이름이 2개 이상 있습니다. 사용자는 이 Dependents 열을 마스크하고 테스트 데이터에 쉼표를 유지하여 이름을 구분해야 합니다.

Dependents 열에 대해 소스 문자열 문자를 선택합니다. 마스크하지 않음을 선택하고 건너뛰 소스 문자로 “,”를 입력합니다. 따옴표는 입력하지 마십시오.

데이터 마스크 변환이 소스 문자열에서 쉼표를 제외한 모든 문자를 바꿉니다.

결과 문자열 대체 문자

결과 문자열 대체 문자는 마스크된 데이터에서 대체 문자로 선택하는 문자입니다. 결과 문자열 대체 문자를 구성할 때 데이터 마스크 변환은 소스 문자열의 문자를 결과 문자열 대체 문자로 바꿉니다. 다른 입력 값에 대해 같은 출력이 생성되지 않도록 하려면 광범위한 대체 문자를 구성하거나 소스 문자를 몇 개만 마스크하십시오. 문자열의 각 문자 위치는 중요하지 않습니다.

결과 문자열 대체 문자에 대해 다음 옵션 중 하나를 선택합니다.

다음만 사용

결과 문자열 대체 문자로 정의하는 문자만 사용하여 소스를 마스크합니다. 예를 들어 A, B, c 문자를 입력하면 데이터 마스크 변환은 소스 열의 모든 문자를 A, B, c로 바꿉니다. 예를 들어 "horse"라는 단어는 "BAcBA"로 바뀔 수 있습니다.

모두 사용(다음 제외)

결과 문자열 대체 문자로 정의하는 문자를 제외한 모든 문자를 사용하여 소스를 마스크합니다. 예를 들어 결과 문자열 대체 문자로 A, B, c를 입력하면 마스크된 데이터에는 A, B, c 문자가 포함되지 않습니다.

결과 문자열 대체 문자 예제

중속 열의 모든 쉼표를 세미콜론으로 대체하려면 다음 태스크를 수행하십시오.

1. 쉼표를 소스 문자열 문자로 구성하고 다음만 마스크를 선택합니다.
중속 열에 쉼표가 있을 경우 데이터 마스킹 변환에서 쉼표만 마스킹합니다.
2. 세미콜론을 결과 문자열 대체 문자로 구성하고 다음만 사용을 선택합니다.
데이터 마스킹 변환에서 중속 열의 모든 쉼표를 세미콜론으로 대체합니다.

범위

숫자, 날짜 또는 문자열 데이터의 범위를 정의합니다. 숫자 또는 날짜 값의 범위를 정의하면 데이터 마스킹 변환이 최소값과 최대값 사이에 있는 값을 사용하여 소스 데이터를 마스킹합니다. 문자열에 대한 범위를 구성하는 경우 문자열 길이 범위를 구성합니다.

문자열 범위

무작위 문자열 마스킹을 구성하는 경우 데이터 마스킹 변환이 소스 문자열의 길이가 다양한 문자열을 생성합니다. 필요에 따라 최소 및 최대 문자열 너비를 구성할 수 있습니다. 최소 또는 최대 너비로 입력하는 값은 양수여야 합니다. 각 너비는 포트 정밀도보다 작거나 같아야 합니다.

숫자 범위

숫자 열의 최소값과 최대값을 설정합니다. 최대값은 포트 전체 자릿수보다 작거나 같아야 합니다. 기본 범위는 1부터 포트 전체 자릿수 길이까지입니다.

날짜 범위

날짜/시간 값에 대한 최소값 및 최대값을 설정합니다. 최소값 및 최대값 필드에는 기본 최소 및 최대 날짜가 포함됩니다. 기본 날짜/시간 형식은 MM/DD/YYYY HH24:MI:SS입니다. 최대 날짜/시간은 최소 날짜/시간보다 나중이어야 합니다.

블러링

블러링은 소스 데이터 값의 고정 또는 백분율 분산에 포함되는 출력 값을 작성합니다. 블러링을 구성하면 원래 값에 근사한 무작위 값이 반환됩니다. 숫자 및 날짜 값을 블러링할 수 있습니다.

숫자 값 블러링

고정 또는 백분율 분산을 선택하여 숫자 소스 값을 블러링합니다. 낮은 블러링 값은 소스 값 미만의 분산입니다. 높은 블러링 값은 소스 값 이상의 분산입니다. 낮은 값과 높은 값은 0보다 크거나 같아야 합니다. 데이터 마스킹 변환이 마스킹된 데이터 반환할 때 사용자가 정의한 범위 내의 숫자 데이터가 반환됩니다.

다음 표에는 입력 소스 값이 66일 때 범위 값을 블러링한 경우에 대한 마스킹 결과가 설명되어 있습니다.

블러링 유형	하한	상한	결과
고정	0	10	66와 76 사이
고정	10	0	56와 66 사이

블러링 유형	하한	상한	결과
고정	10	10	56와 76 사이
백분율	0	50	66와 99 사이
백분율	50	0	33와 66 사이
백분율	50	50	33와 99 사이

날짜 값 블러링

블러링을 구성하여 소스 날짜의 분산으로 날짜를 마스킹합니다. 분산을 적용할 날짜의 단위를 선택합니다. 연도, 월, 일 또는 시간을 선택할 수 있습니다. 소스 날짜 단위의 위 및 아래 분산을 정의하는 하한 및 상한을 입력합니다. 데이터 마스킹 변환은 이 분산을 적용하고 분산 내에 포함되는 날짜를 반환합니다.

예를 들어 마스킹된 날짜를 소스 날짜에서 2년 내의 날짜로 제한하려면 연도를 단위로 선택합니다. 하한 및 상한으로 2를 입력합니다. 소스 날짜가 02/02/2006인 경우 데이터 마스킹 변환이 02/02/2004에서 02/02/2008 사이의 날짜를 반환합니다.

기본 블러링 단위는 연도입니다.

식 마스킹

식 마스킹은 포트에 식을 적용하여 데이터를 변경하거나 새 데이터를 작성합니다. 식 마스킹을 구성하는 경우 식 편집기에서 식을 작성해야 합니다. 입력 및 출력 포트, 함수, 변수 및 연산자를 선택하여 식을 빌드합니다.

여러 포트의 데이터를 연결하여 다른 포트에 대한 값을 작성할 수 있습니다. 예를 들어 로그인 이름의 경우 다음과 같이 작성할 수 있습니다. 소스에 이름과 성 열이 있습니다. 조회 파일에서 이름과 성을 마스킹합니다. 데이터 마스킹 변환에서 **Login**이라는 이름의 다른 포트를 작성합니다. **Login** 포트에 대해 이름의 첫 글자와 성을 연결하는 다음과 같은 식을 구성합니다.

```
SUBSTR(FIRSTNM,1,1)||LASTNM
```

포트에 대한 식 마스킹을 구성하는 경우 기본적으로 포트 이름이 식으로 나타납니다. 식 편집기에 액세스하려면 열기 단추를 클릭합니다. 식 편집기에는 식 마스킹에 대해 구성되지 않은 출력 포트 및 입력 포트가 표시됩니다. 한 식의 출력을 다른 식의 입력으로 사용할 수는 없습니다. 출력 포트 이름을 식에 수동으로 추가할 경우 예기치 않은 결과를 얻을 수 있습니다.

식을 작성할 때 실수를 최소화할 수 있도록 가리키고 클릭 인터페이스를 통해 함수, 포트, 변수 및 연산자를 선택하십시오.

식을 작성할 때는 해당 식이 포트의 데이터 유형과 일치하는 값을 반환하는지 확인해야 합니다. 식 포트의 데이터 유형이 숫자일 때 식의 데이터 유형이 동일하지 않으면 데이터 마스킹 변환이 0을 반환합니다. 식 포트의 데이터 유형이 문자열일 때 식의 데이터 유형이 동일하지 않으면 데이터 마스킹 변환이 Null 값을 반환합니다.

반복 가능한 식 마스킹

소스 열이 둘 이상의 테이블에서 나타나며 각 테이블의 열을 같은 값으로 마스킹해야 하는 경우 반복 가능 식 마스킹을 구성합니다.

반복 가능 식 마스킹을 구성하면 데이터 마스킹 변환은 식의 결과를 저장소 테이블에 저장합니다. 열이 다른 소스 테이블에서 나타나는 경우 데이터 마스킹 변환은 식이 아닌 저장소 테이블에서 마스킹된 값을 반환합니다.

사전 이름

반복 가능한 식 마스킹을 구성하는 경우 사전 이름을 입력해야 합니다. 사전 이름은 여러 데이터 마스킹 변환이 동일한 소스 값으로부터 동일하게 마스킹된 값을 생성할 수 있도록 하는 키입니다. 각 데이터 마스킹 변환에 동일한 사전 이름을 정의해야 합니다. 모든 텍스트를 사전 이름으로 입력할 수 있습니다.

저장소 테이블

저장소 테이블에는 세션 간 반복 가능 식 마스킹의 결과가 포함되어 있습니다. 저장소 테이블 행에는 소스 열 및 마스킹된 값 쌍이 들어 있습니다. 식 마스킹의 저장소 테이블은 대체 마스킹의 저장소 테이블과는 별개의 테이블입니다.

데이터 마스킹 변환이 반복 가능 식을 사용하여 값을 마스킹할 때마다 사전 이름, 로컬, 열 이름, 입력 값으로 저장소 테이블을 검색합니다. 저장소 테이블에서 행을 찾으면 저장소 테이블에서 마스킹된 값을 반환합니다. 데이터 마스킹 변환이 행을 찾지 못하면 열의 식에서 마스킹된 값을 생성합니다.

저장소에 암호화되지 않은 데이터가 있고 동일한 사전 이름을 키로 사용하는 경우 식 마스킹을 위해 저장소 테이블을 암호화해야 합니다.

식 마스킹에 대한 저장소 테이블 암호화

변환 언어 인코딩 함수를 사용하여 저장소 테이블을 암호화할 수 있습니다. 저장소 암호화를 활성화한 경우 저장소 테이블을 암호화해야 합니다.

1. `IDM_EXPRESSION_STORAGE` 저장소 테이블을 소스로 사용하는 매핑을 작성합니다.
2. 데이터 마스킹 변환을 작성합니다.
3. 마스킹된 값 포트에 식 마스킹 기술을 적용합니다.
4. `MASKEDVALUE` 포트에 다음 식을 사용합니다.
`Enc_Base64(AES_Encrypt(MASKEDVALUE, Key))`
5. 포트를 대상에 연결합니다.

예제

예를 들어 직원 테이블에는 다음 열이 포함됩니다.

```
FirstName
LastName
LoginID
```

데이터 마스킹 변환에서 `FirstName`과 `LastName`을 결합하는 식을 사용하여 `LoginID`를 마스킹합니다. 식 마스킹을 반복 가능하도록 구성합니다. 반복 가능한 마스킹의 키로 사용할 사전 이름을 입력합니다.

`Computer_Users` 테이블에는 `LoginID`가 포함되지만 `FirstName` 또는 `LastName` 열이 포함되지 않습니다.

```
Dept
LoginID
Password
```

`Employees` 테이블과 동일한 `LoginID`로 `Computer_Users` 테이블의 `LoginID`를 마스킹하려면 `LoginID` 열에 대해 식 마스킹을 구성합니다. 반복 가능한 마스킹을 활성화하고 `LoginID` 직원 테이블에 정의한 것과 동일한 사전 이름을 입력합니다. 통합 서비스가 저장소 테이블에서 `LoginID` 값을 검색합니다.

통합 서비스가 저장소 테이블에서 `LoginID`에 대한 행을 찾지 못할 때 사용할 기본 식을 작성합니다.

`Computer_Users` 테이블에 `FirstName` 또는 `LastName` 열이 없으므로 기본 식을 통해 작성되는 `LoginID`는 의미가 감소할 수 있습니다.

저장소 테이블 스크립트

Informatica는 저장소 테이블을 작성하기 위해 실행할 수 있는 스크립트를 제공합니다. 스크립트는 다음 위치에 있습니다.

<PowerCenter installation directory>\client\bin\Extensions\DataMasking

디렉터리에는 Sybase, Microsoft SQL Server, IBM DB2 및 Oracle 데이터베이스용 스크립트가 포함되어 있습니다. 각 스크립트의 이름은 <Expression_<데이터베이스 유형>입니다.

식 마스킹에 대한 규칙 및 지침

식 마스킹에 대해 다음 규칙 및 지침을 사용하십시오.

- 한 식의 출력을 다른 식의 입력으로 사용할 수는 없습니다. 출력 포트 이름을 식에 수동으로 추가할 경우 예기치 않은 결과를 얻을 수 있습니다.
- 가리키고 클릭 방법을 사용하여 식을 작성합니다. 식을 작성할 때 실수를 최소화할 수 있도록 가리키고 클릭 인터페이스를 통해 함수, 포트, 변수 및 연산자를 선택하십시오.
- 데이터 마스킹 변환이 반복 가능한 마스킹을 위해 구성되었지만 저장소 테이블이 존재하지 않는 경우 통합 서비스가 소스 데이터를 기본값으로 대체합니다.

관련 항목:

- [“식 편집기 사용” 페이지 33](#)

형식 유지 암호화

암호화 마스킹은 암호화 알고리즘을 적용하여 소스 데이터를 마스킹합니다.

형식 유지 암호화로 문자열 데이터 유형을 마스킹합니다.

소스 데이터의 형식 및 길이 또는 소스 데이터의 길이를 보존하도록 선택할 수 있습니다. 암호화 후 소스 데이터의 형식 및 길이를 변경하도록 선택할 수도 있습니다.

암호화하지 않을 문자를 선택할 수 있습니다.

소스 데이터를 암호화한 후 암호를 해독하여 소스 데이터를 원래 데이터로 되돌릴 수도 있습니다. 데이터를 암호 해독하려면 소스 데이터를 암호화하는 데 사용한 것과 동일한 암호 구문을 사용하여 동일한 암호화 기술을 사용하는 매핑을 생성하고 실행해야 합니다. 모드를 암호 해독으로 설정합니다.

참고: 소스 데이터에 UTF-8 4바이트 문자가 포함된 경우 암호화를 사용하여 데이터를 마스킹할 수 없습니다.

다음 암호화 기술 중 하나를 선택합니다.

형식 및 메타데이터 유지

소스 데이터의 형식 및 길이를 유지하려면 형식 및 메타데이터 유지 암호화 옵션을 사용합니다. 형식 및 메타데이터를 유지하도록 선택하면 암호화 후 모든 대문자는 대문자로, 소문자는 소문자로, 숫자는 숫자로, 특수 문자는 특수 문자로 바뀝니다. 예를 들어 `Abc123@xyz.com`이라는 전자 메일 주소는 `Mpz849#dje!kuw`가 될 수 있습니다. 이 예에서 "@" 및 "." 문자를 암호화하지 않음으로 구성하는 경우 이 전자 메일은 `Mpz849@dje.kuw`가 될 수 있습니다.

메타데이터 유지

소스 데이터의 길이를 유지하려면 메타데이터 유지 암호화 옵션을 사용합니다. 메타데이터를 보존하도록 선택하면 암호화 후에 데이터 길이가 동일하게 유지됩니다. 예를 들어 `Alexender`라는 이름은 `jl6#HB91v`가 될 수 있으며 여기서 길이는 소스 데이터와 동일하게 유지됩니다.

메타데이터 변경

암호화 후 소스 데이터의 길이를 변경하려면 메타데이터 변경 암호화 옵션을 사용합니다. 메타데이터를 변경하도록 선택하면 암호화된 데이터에서 소스 데이터의 길이와 형식이 유지되지 않습니다. 예를 들어 London이라는 도시 이름은 Xuep@8f5, fmch529 또는 6ky#ke33h*we가 될 수 있습니다.

참고: 메타데이터 변경 암호화 옵션을 사용하려면 먼저 데이터베이스에서 암호화를 적용할 열의 전체 자릿수를 변경해야 합니다.

다음 공식을 사용하여 전체 자릿수를 계산하고 다음으로 높은 정수로 값을 반올림합니다.

$$\text{Required Precision} = (1.33 * \text{Original Precision}) + 24$$

데이터베이스에서 열 전체 자릿수를 변경한 후 매핑에서 열 전체 자릿수를 업데이트해야 합니다. 열 전체 자릿수를 업데이트하려면 업데이트된 데이터베이스에서 메타데이터를 다시 가져오거나 매핑의 각 변환에서 열 전체 자릿수를 수동으로 변경할 수 있습니다.

특수 마스크 형식

다음 마스크 유형은 원래 데이터의 형식을 유지합니다.

- 사회 보장 번호
- 신용 카드 번호
- 전화 번호
- URL 주소
- 전자 메일 주소
- IP 주소
- 사회 보험 번호

데이터 마스크 변환은 중요한 데이터를 실제와 비슷한 형식의 마스크된 값으로 대체합니다. 예를 들어 SSN을 마스킹하는 경우, 데이터 마스크 변환은 올바른 형식의 유효하지 않은 SSN을 반환합니다.

소스 데이터 형식 또는 데이터 유형이 특정 마스크에 적합하지 않으면 통합 서비스가 기본 마스크를 데이터에 적용합니다. 통합 서비스는 기본값 파일에서 마스크된 값을 적용합니다. 기본값 파일을 편집하여 기본값을 변경할 수 있습니다.

관련 항목:

- [“기본값 파일” 페이지 144](#)

사회 보장 번호 마스킹

데이터 마스크 변환은 Social Security Administration에서 제공하는 최신 High Group List에 따라 유효하지 않은 사회 보장 번호를 생성합니다. 높은 그룹 목록에는 사회 보장 관리가 제출한 유효한 숫자가 들어 있습니다. 데이터 마스크 변환은 다음 위치에서 최신 High Group List에 액세스합니다.

<Installation Directory>\infa_shared\SrcFiles\highgroup.txt

데이터 마스크 변환은 높은 그룹 목록에 없는 SSN 번호를 생성합니다. 사회 보장 관리에서 매달 높은 그룹 목록을 업데이트합니다. 다음 위치에서 최신 버전의 목록을 다운로드합니다.

<http://www.socialsecurity.gov/employer/ssns/highgroup.txt>

사회 보장 번호 형식

데이터 마스킹 변환은 9개의 숫자가 포함된 SSN 형식을 허용합니다. 숫자는 문자 집합으로 구분될 수 있습니다.

예를 들어 데이터 마스킹 변환은 다음 형식을 허용합니다. +=54-*9944\$#789-,*()).

지역 번호 요구 사항

데이터 마스킹 변환은 소스와 동일한 형식에 유효하지 않은 사회 보장 번호를 반환합니다. SSN의 첫 3자리는 지역 번호를 정의합니다. 데이터 마스킹 변환은 지역 번호를 마스킹하지 않습니다. 그룹 번호와 일련 번호를 마스킹합니다. 소스 SSN에는 유효한 지역 번호가 포함되어야 합니다. 데이터 마스킹 변환은 높은 그룹 목록에서 지역 번호를 찾은 다음 사용되지 않은 번호 중에서 마스킹된 데이터로 적용할 수 있는 번호의 범위를 결정합니다. SSN이 유효하지 않은 경우 데이터 마스킹 변환이 소스 데이터를 마스킹하지 않습니다.

반복 가능한 사회 보장 번호 마스킹

데이터 마스킹 변환은 반복 가능 마스킹을 사용하여 특정 사회 보장 번호를 반환합니다. 데이터 마스킹 변환은 사회 보장 관리국에서 발급한 유효 사회 보장 번호는 반환할 수 없으므로 모든 고유 사회 보장 번호를 반환할 수 없습니다.

신용 카드 번호 마스킹

신용 카드 마스킹은 기본 제공 마스크 형식을 적용하여 신용 카드 번호를 숨깁니다. 여러 신용 카드 번호 형식을 입력할 수 있습니다. 필요에 따라 신용 카드 발급자를 바꿉니다.

PowerCenter 통합 서비스는 논리적으로 유효한 신용 카드 번호를 생성하여 유효한 신용 카드 번호를 마스킹합니다. 소스 신용 카드 번호의 길이는 13~19자리여야 합니다. 입력 신용 카드 번호에는 신용 카드 업계 규칙에 기반하는 유효한 체크섬이 포함되어야 합니다.

소스 신용 카드 번호에는 숫자, 공백 및 하이픈이 포함될 수 있습니다. 신용 카드에 잘못된 문자가 포함되거나 길이가 올바르지 않은 경우 통합 서비스가 세션 로그에 오류를 기록합니다. PowerCenter 통합 서비스는 소스 데이터가 잘못된 경우 기본 신용 카드 번호 마스크를 적용합니다.

신용 카드 번호의 처음 6자리를 유지하거나 바꾸도록 신용 카드 마스킹을 구성할 수 있습니다. 이러한 자릿수가 함께 신용 카드 발급자를 식별합니다. 신용 카드 발급자를 바꾸는 경우 다른 신용 카드 발급자를 지정할 수 있습니다. 다음과 같은 신용 카드 발급자를 지정할 수 있습니다.

- American Express
- Discover
- JCB
- MasterCard
- Visa
- 임의

임의를 선택하는 경우 데이터 마스킹 변환이 모든 신용 카드 발급자 조합을 반환합니다. 신용 카드 발급자를 마스킹하는 경우 데이터 마스킹 변환이 고유하지 않은 값을 반환할 수 있습니다.

예를 들어 CUSTOMER 테이블에 다음과 같은 신용 카드 번호가 있습니다.

```
2131 0000 0000 0008
5500 0000 0000 0004
6334 0000 0000 0004
```

단일 신용 카드 발급자를 선택하는 경우 신용 카드 번호는 마지막 자릿수를 제외하고 모두 공유됩니다. 유효한 신용 카드 번호를 생성하기 위해 데이터 마스크 변환은 각 신용 카드의 마지막 자릿수를 동일한 값으로 설정합니다.

전화 번호 마스크

데이터 마스크 변환은 원래 전화 번호의 형식을 변경하지 않고 전화 번호를 마스크합니다. 예를 들어 데이터 마스크 변환이 전화 번호 (408)382 0658을 (607)256 3106으로 마스크할 수 있습니다.

소스 데이터는 숫자, 공백, 하이픈 및 괄호를 포함할 수 있습니다. 통합 서비스는 영문자 또는 특수 문자를 마스크하지 않습니다.

데이터 마스크 변환은 마스크 문자열, 정수 및 bigint 데이터를 마스크할 수 있습니다.

전자 메일 주소 마스크

데이터 마스크 변환을 사용하여 문자열 값이 포함된 전자 메일 주소를 마스크할 수 있습니다. 데이터 마스크 변환은 전자 메일 주소를 무작위 ASCII 문자로 마스크하거나 실제 전자 메일 주소로 바꿉니다.

다음과 같은 전자 메일 주소 마스크 유형을 적용할 수 있습니다.

표준 전자 메일 마스크

데이터 마스크 변환이 무작위 ASCII 문자를 반환하여 전자 메일 주소를 마스크합니다. 예를 들어 데이터 마스크 변환은 Georgesmith@yahoo.com을 KtrlupQAPyk@vdSKh.BIC로 마스크할 수 있습니다. 기본값은 표준입니다.

고급 전자 메일 마스크

데이터 마스크 변환이 변환 출력 포트 또는 사전 열에서 파생된 실제 전자 메일 주소로 전자 메일 주소를 마스크합니다.

고급 전자 메일 마스크

고급 전자 메일 마스크를 사용하면 전자 메일 주소를 실제와 같은 다른 전자 메일 주소로 마스크할 수 있습니다. 데이터 마스크 변환은 사전 열 또는 변환 출력 포트를 바탕으로 전자 메일 주소를 작성합니다.

전자 메일 주소의 로컬 부분은 매핑 출력 포트를 바탕으로 작성할 수 있습니다. 또는 관계형 테이블 또는 플랫폼 파일 열을 바탕으로 전자 메일 주소의 로컬 부분을 작성할 수도 있습니다.

데이터 마스크 변환은 상수 값 또는 도메인 사전의 무작위 값으로부터 전자 메일 주소의 도메인 이름을 작성할 수 있습니다.

고급 전자 메일 마스크는 다음 옵션을 사용하여 작성할 수 있습니다.

매핑 기반 전자 메일 주소

데이터 마스킹 변환 출력 포트를 바탕으로 전자 메일 주소를 작성할 수 있습니다. 이름 및 성 열에 대한 변환 출력 포트를 선택합니다. 데이터 마스킹 변환이 사용자가 이름 및 성 길이에 지정한 값을 바탕으로 이름, 성 또는 이름과 성을 마스킹합니다.

사전 기반 전자 메일 주소

사전의 열을 바탕으로 전자 메일 주소를 작성할 수 있습니다. 사전은 관계형 테이블 또는 플랫폼 파일일 수 있습니다.

이름 및 성에 대한 사전 열을 선택합니다. 데이터 마스킹 변환이 사용자가 이름 및 성 길이에 지정한 값을 바탕으로 이름, 성 또는 이름과 성을 마스킹합니다. 또한 전자 메일 주소를 마스킹하도록 식을 구성할 수 있습니다. 식을 구성하는 경우 데이터 마스킹 변환은 이름이나 성 열의 길이를 기반으로 마스킹하지 않습니다.

고급 전자 메일 주소 마스킹 유형의 구성 매개 변수

고급 전자 메일 주소 마스킹을 구성하는 경우 구성 매개 변수를 지정합니다.

다음 구성 매개 변수를 지정할 수 있습니다.

구분자

점, 하이픈 또는 밑줄과 같은 구분자를 선택하여 전자 메일 주소에서 이름과 성을 구분할 수 있습니다. 전자 메일 주소에서 이름과 성을 구분하지 않으려는 경우 구분자를 비워 둡니다.

FirstName 열

전자 메일 주소의 이름을 마스킹할 데이터 마스킹 변환 출력 포트 또는 사전 열을 선택합니다.

LastName 열

전자 메일 주소의 성을 마스킹할 데이터 마스킹 변환 출력 포트 또는 사전 열을 선택합니다.

FirstName 또는 LastName 열의 길이

이름 및 성 열의 마스킹할 문자 길이를 제한합니다. 예를 들어 입력 데이터의 이름이 **Timothy**이고 성이 **Smith**인 경우가 있습니다. 이름 열의 길이로 **5**를 선택합니다. 성 열의 길이를 **1**로 선택하고 구분자로 점을 선택합니다. 데이터 마스킹 변환이 다음 전자 메일 주소를 생성합니다.

timot.s@<domain_name>

DomainName

gmail.com과 같은 상수 값을 도메인 이름에 사용할 수 있습니다. 또는 도메인 이름 목록이 포함되는 다른 사전 파일을 지정할 수 있습니다. 도메인 사전은 플랫폼 파일 또는 관계형 테이블이 될 수 있습니다.

고급 전자 메일 주소 마스킹 유형의 식

사전 열을 선택하여 전자 메일 주소를 작성하면 식 함수를 사용할 수 있습니다.

고급 전자 메일 주소 마스킹 유형의 식을 구성하려면 사전에서 전자 메일 주소를 형성할 열을 선택합니다. 그런 다음 사전 열, 변환 포트 또는 사전 열과 변환 포트 모두를 사용하여 식을 구성할 수 있습니다.

사전 포트는 식 편집기에서 다음 구문으로 나열됩니다.

<입력 문자열 매핑 포트>_Dictionary_<사전 열 이름>

사회 보험 번호 마스킹

데이터 마스킹 변환은 9개의 숫자로 이루어진 사회 보험 번호를 마스킹합니다. 숫자는 문자 집합으로 구분될 수 있습니다.

번호에 구분자가 포함되지 않은 경우 마스킹된 번호에 구분자가 포함되지 않습니다. 그렇지 않은 경우 마스킹된 번호의 형식은 다음과 같습니다.

XXX-XXX-XXX

SIN 시작 자릿수

마스킹된 SIN의 첫 번째 숫자를 정의할 수 있습니다.

시작 숫자를 활성화하고 숫자를 입력합니다. 데이터 마스킹 변환이 입력하는 숫자로 시작하는 마스킹된 SIN 번호를 작성합니다.

반복 가능 SIN 번호

반복 가능 SIN 값을 반환하도록 데이터 마스킹 변환을 구성할 수 있습니다. 반복 가능 SIN 마스킹에 대한 포트를 구성하는 경우 소스 SIN 값과 시드 값이 동일할 때마다 데이터 마스킹 변환이 확정적인 마스킹된 데이터를 반환합니다.

반복 가능 SIN 번호를 반환하려면 **반복 가능 값**을 활성화하고 시드 숫자를 입력합니다. 데이터 마스킹 변환은 각 SIN에 대해 고유한 값을 반환합니다.

IP 주소 마스킹

데이터 마스킹 변환은 IP 주소를 마침표로 구분된 4개의 번호로 분할하여 다른 IP 주소로 마스킹합니다. 첫 번째 번호는 네트워크입니다. 네트워크 번호는 네트워크 범위 내에서 마스킹됩니다.

클래스 A IP 주소는 클래스 A IP 주소로 마스킹되고 10.x.x.x 주소는 10.x.x.x 주소로 마스킹됩니다. 클래스 및 개인 네트워크 주소는 마스킹되지 않습니다. 예를 들어 데이터 마스킹 변환은 11.12.23.34를 75.32.42.52로 마스킹하고 10.23.24.32를 10.61.74.84로 마스킹할 수 있습니다.

참고: 데이터 마스킹 변환은 IP 주소의 클래스 또는 개인 네트워크를 마스킹하지 않으므로 많은 수의 IP 주소를 마스킹하는 경우 고유하지 않은 값이 반환될 수 있습니다.

URL 주소 마스킹

데이터 마스킹 변환은 “://” 문자열을 검색하고 그 오른쪽에 있는 하위 문자열을 구문 분석하여 URL을 구문 분석합니다. 소스 URL은 “://” 문자열을 포함해야 합니다. 소스 URL은 숫자 및 영문자를 포함할 수 있습니다.

데이터 마스킹 변환은 URL의 프로토콜을 마스킹하지 않습니다. 예를 들어 URL이 http://www.yahoo.com이면 데이터 마스킹 변환이 http://MgL.aHjCa.VsD/를 반환할 수 있습니다. 데이터 마스킹 변환이 유효하지 않은 URL을 생성할 수 있습니다.

참고: 데이터 마스킹 변환은 항상 URL에 대해 ASCII 문자를 반환합니다.

기본값 파일

소스 데이터 형식 또는 데이터 유형이 마스크에 유효하지 않은 경우 통합 서비스가 기본 마스크를 데이터에 적용합니다. 통합 서비스는 기본값 파일에서 마스크된 값을 적용합니다. 기본값 파일을 편집하여 기본값을 변경할 수 있습니다.

기본값 파일은 XML 파일이며 다음 위치에서 찾을 수 있습니다.

```
<PowerCenter Installation Directory>\infa_shared\SrcFiles\defaultValue.xml
```

defaultValue.xml 파일에는 다음 이름-값 쌍이 포함됩니다.

```
<?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 = "9999999999"
default_ssn = "999-99-9999"
default_cc = "9999 9999 9999 9999"
default_seed = "500"
/>
```

데이터 마스크 변환 세션 속성

데이터 마스크 변환 세션 속성을 구성하여 성능을 개선할 수 있습니다.

다음 세션 속성을 구성합니다.

캐시 크기

기본 메모리의 사전 캐시 크기입니다. 메모리 크기를 늘리면 성능이 개선됩니다. 최소 권장 크기는 32MB(100,000레코드)입니다. Default is 8 MB.

캐시 디렉터리

사전 캐시의 위치입니다. 디렉터리에 대한 쓰기 권한이 있어야 합니다. 기본값은 \$PMCacheDir입니다.

공유 저장소 테이블

데이터 마스크 변환의 인스턴스가 저장소 테이블을 공유할 수 있도록 합니다. 데이터 마스크 변환의 인스턴스 2개가 데이터베이스 연결, 시드 값 및 로캘에 대해 동일한 사전 열을 사용하는 경우 공유 저장소 테이블을 활성화하십시오. 동일한 데이터 마스크 변환의 포트 2개가 연결, 시드 및 로캘에 대해 동일한 사전 열을 사용하는 경우에도 공유 저장소 테이블을 활성화할 수 있습니다. 데이터 마스크 변환 또는 포트가 사전 열을 공유하지 않는 경우에는 공유 저장소 테이블을 비활성화합니다. 기본값이 비활성화됩니다.

저장소 커밋 간격

한 번에 저장소 테이블에 커밋할 행의 수입니다. 값을 높이면 성능이 개선됩니다. 공유 저장소 테이블을 구성하지 않는 경우 커밋 간격을 구성하십시오. 기본값은 100,000입니다.

저장소 암호화

IDM_SUBSTITUTION_STORAGE 및 IDM_EXPRESSION_STORAGE와 같은 저장소 테이블을 암호화합니다. 저장소 암호화 속성을 활성화하기 전에 저장소 테이블의 데이터가 암호화되었는지 확인하십시오. 저장소 테이블을 암호화하지 않으려는 경우 이 옵션을 선택 취소합니다. 기본값이 비활성화됩니다.

저장소 암호화 키

데이터 마스킹 변환이 저장소 암호화 키를 바탕으로 저장소를 암호화합니다. 동일한 데이터 마스킹 변환 인스턴스의 각 세션 실행에 대해서도 동일한 암호화 키를 사용합니다.

SoftHSM 사용

형식 유지 암호화에 필요합니다. 암호화 중에 **SoftHSM**을 사용할지 여부를 선택합니다. **SoftHSM**은 더욱 안전하지만 성능에 차이가 있을 수 있습니다.

암호

형식 유지 암호화에 필요합니다. 암호는 데이터를 암호화하거나 암호 해독하는 키를 생성합니다. 암호화된 암호 또는 일반 텍스트 암호를 입력하도록 선택할 수 있습니다. 사이트 키를 사용하여 암호를 암호화하고 암호화된 값을 입력합니다.

암호가 암호화됨

디자이너 도구에서 생성하는 맵핑의 경우 암호화된 암호 또는 일반 텍스트 암호를 입력하도록 선택할 수 있습니다. **Test Data Management**에서 생성하는 워크플로우의 경우 옵션은 항상 '예'입니다.

모드

형식 유지 암호화에 필요합니다. 데이터 마스킹 변환에서 데이터의 암호화 또는 암호 해독을 수행할지 여부를 결정합니다. 소스 데이터를 암호화하려면 값을 암호화로 설정합니다. 마스킹된 데이터를 해독하고 원래 소스 데이터를 반환하려면 모드를 암호 해독으로 설정하고 동일한 암호화 기술 구성 및 암호를 사용하여 맵핑을 실행합니다.

데이터 마스킹 변환에 대한 규칙 및 지침

데이터 마스킹 변환을 구성할 때 다음 규칙 및 지침을 따르십시오.

- 데이터 마스킹 변환은 **null** 값을 마스킹하지 않습니다. 소스 데이터에 **null** 값이 포함되어 있으면 데이터 마스킹 변환이 **null** 값을 반환합니다. **null** 값을 대체하려면 입력 포트에 대해 사용자 정의 기본값을 허용하는 업스트림 변환을 추가합니다.
- 소스 데이터 형식 또는 데이터 유형이 마스크에 유효하지 않은 경우 통합 서비스가 기본 마스크를 데이터에 적용합니다. 통합 서비스는 기본값 파일에 있는 마스킹된 값을 적용합니다.
- 데이터 마스킹 변환은 소스와 형식 및 지역 번호가 동일한 사회 보장 번호를 반환합니다. **Social Security Administration**에서 특정 지역의 번호 중 절반 이상을 발급한 경우, 데이터 마스킹 변환은 키 마스킹을 통해 고유하고 올바른 사회 보장 번호를 반환하지 못할 수도 있습니다.

제 6 장

데이터 마스킹 예제

이 장에 포함된 항목:

- [이름 및 주소 조회 파일, 146](#)
- [조회 변환을 사용하여 데이터 대체, 146](#)
- [식 변환을 사용하여 데이터 마스킹, 149](#)

이름 및 주소 조회 파일

데이터 마스킹 설치에는 데이터 마스킹을 위한 이름, 주소 및 회사 이름이 있는 여러 플랫폼 파일이 포함되어 있습니다. 이러한 파일에서 임의의 이름, 성 및 주소를 검색하도록 조회 변환을 구성할 수 있습니다.

데이터 마스킹 서버 구성 요소를 설치하면 설치 프로그램이 `server\infa_shared\LkpFiles` 폴더에 다음 파일을 넣습니다.

- **Address.dic.** 주소의 플랫폼 파일입니다. 각 레코드에는 일련 번호, 구/군/시, 도시, 주, 우편 번호 및 국가가 있습니다.
- **Company_names.dic.** 회사 이름의 플랫폼 파일입니다. 각 레코드에는 일련 번호 및 회사 이름이 있습니다.
- **Firstnames.dic.** 이름의 플랫폼 파일입니다. 각 레코드에는 일련 번호, 이름 및 성별 코드가 있습니다.
- **Surnames.dic.** 성의 플랫폼 파일입니다. 각 레코드에는 일련 번호 및 성이 있습니다.

조회 변환을 사용하여 데이터 대체

데이터 열을 유사하지만 관련 없는 데이터로 대체할 수 있습니다. 대체는 중요 데이터를 진짜처럼 보이는 데이터로 마스킹하는 효과적인 방법입니다.

다음 예에서는 테스트 데이터를 검색하고 소스 데이터를 테스트 데이터로 대체하도록 여러 조회 변환을 구성하는 방법을 보여 줍니다. 데이터 마스킹 매핑을 작성하여 **CUSTOMERS_PROD** 테이블의 중요 필드를 마스킹합니다.

이 예에는 다음과 같은 마스킹 유형이 포함됩니다.

- 조회 테이블에서 검색된 이름 및 주소 대체
- 키 마스킹
- 블러링
- 특수 마스크 형식

참고: 이 예는 M_CUSTOMERS_MASKING.xml 매핑이며, 이 매핑은 client\samples 폴더에서 리포지토리로 가져올 수 있습니다.

Customers_Prod라는 고객 데이터베이스 테이블에는 중요 데이터가 포함되어 있습니다. 테스트 시나리오에서 고객 데이터를 사용하면서 보안을 유지하려고 합니다. 각 열의 데이터를 마스킹하고 Customers_Test라는 대상 테이블에 테스트 데이터를 씁니다.

Customers_Prod 테이블에는 다음과 같은 열이 있습니다.

열	데이터 유형
CustID	정수
FullName	문자열
주소	문자열
전화	문자열
팩스	문자열
CreatedDate	날짜
전자 메일	문자열
SSN	문자열
CreditCard	문자열

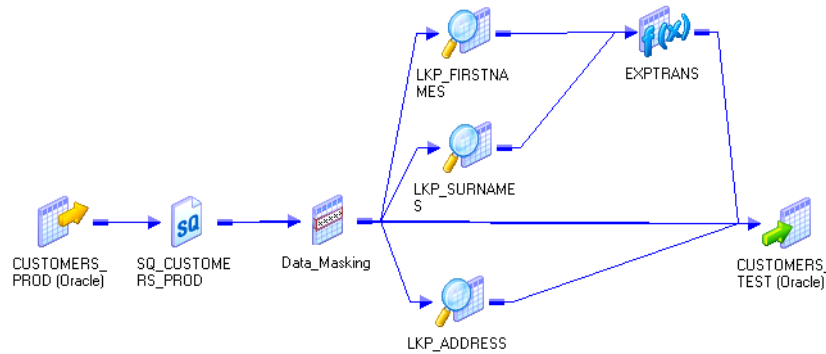
사전 파일에서 대체 값을 조회하는 매핑을 작성할 수 있습니다. 데이터 마스킹 변환은 사전 파일에 있는 값으로 고객 데이터를 마스킹합니다. 사전 파일에는 이름 파일, 성 파일 및 주소 파일이 있습니다.

다음 테이블에는 server\infa_shared\LkpFiles 폴더에 있는 파일이 나열되어 있습니다.

파일	레코드 수	필드	설명
Firstnames.dic	21,000	SNO, Gender, Firstname	이름의 사전순 목록입니다. 일련 번호는 1부터 21,000까지입니다. Gender는 이름이 남성 또는 여성인지 나타냅니다.
Surnames.dic	81,000	SNO, Surname	성의 사전순 목록입니다. 일련 번호는 1부터 81,000까지입니다.
Address.dic	13,000	SNO, Street, City, State, Zip, Country	전체 주소의 목록입니다. 일련 번호는 1부터 13,000까지입니다.

참고: Informatica는 Firstnames.dic 파일에 gender 열을 포함하여 성별에 따라 별도의 조회 소스 파일을 작성할 수 있게 합니다. 남성 이름으로 마스킹하고 여성 이름을 여성 이름으로 마스킹해야 하는 경우 조회 조건에서 gender 열을 사용할 수 있습니다.

다음 그림에는 가져올 수 있는 매핑이 나와 있습니다.



매핑에는 다음과 같은 변환이 소스 및 대상과 함께 포함되어 있습니다.

- **소스 한정자.** 고객 데이터를 데이터 마스킹 변환으로 전달합니다. CustID 열을 변환의 여러 포트에 전달합니다.
 - **CustID.** 고객 번호입니다.
 - **Randid1.** 이름 조회를 위한 난수 생성기입니다.
 - **Randid2.** 성 조회를 위한 난수 생성기입니다.
 - **Randid3.** 주소 조회를 위한 난수 생성기입니다.
- **데이터 마스킹 변환.** 대체 이름, 성 및 주소를 조회하기 위한 난수를 작성합니다. 전화 번호, 팩스, 전자 메일 주소 및 신용 카드 번호에 특수 마스크 형식을 적용합니다. 데이터 마스킹 변환은 다음 열을 마스킹합니다.

입력 포트	마스킹 유형	마스킹 규칙	설명	출력 대상
CustID	키	시드 = 934	CustID는 기본 키 열입니다. 반복 가능하고 확정적인 난수로 마스킹되어야 합니다.	Customers_Test
Randid1	무작위	범위 최소값 = 0 최대값 = 21000	LKUP_Firstnames 변환에서의 이름 조회를 위한 난수입니다.	LKUP_Firstnames
Randid2	무작위	범위 최소값 = 0 최대값 = 13000	LKUP_Surnames 변환에서의 성 조회를 위한 난수입니다.	LKUP_Surnames
Randid3	무작위	범위 최소값 = 0 최대값 = 81000	LKUP_Address 변환에서의 주소 조회를 위한 난수입니다.	LKUP_Address
전화	전화	-	전화 번호는 소스 전화 번호와 동일한 형식을 갖습니다.	Customers_Test
팩스	전화	-	전화 번호는 소스 전화 번호와 동일한 형식을 갖습니다.	Customers_Test

입력 포트	마스킹 유형	마스킹 규칙	설명	출력 대상
CreatedDate	무작위	블러링 단위 = 년 하한=1 상한 = 1	소스 연도 내의 임의 날짜입니다.	Customers_Test
전자 메일	전자 메일 주소	-	전자 메일 주소는 원본 주소와 동일한 형식을 갖습니다.	Customers_Test
SSN	SSN	-	SSN은 highgroup.txt 파일에 없습니다.	Customers_Test
CreditCard	신용 카드	-	신용 카드는 앞의 6자리가 소스와 동일하고 유효한 체크섬을 갖습니다.	Customers_Test

- **LKUP_Firstnames.** Firstnames.dic에 대한 플랫폼 파일 조회를 수행합니다. 이 변환은 난수 Randid1과 동일한 일련 번호를 갖는 레코드를 검색합니다. 조회 조건은 다음과 같습니다.

SNO = out_RANDID1

LKUP_Firstnames 변환은 마스킹된 이름을 Exptrans 식 변환으로 전달합니다.

- **LKUP_Surnames.** Surnames.dic에 대한 플랫폼 파일 조회를 수행합니다. 일련 번호가 Randid2와 동일한 레코드를 검색합니다. LKUP_Firstnames 변환은 마스킹된 성을 Exptrans 식 변환으로 전달합니다.
- **Exptrans.** 이름과 성을 결합하여 전체 이름을 반환합니다. 이 식 변환은 전체 이름을 Customers_Test 대상으로 전달합니다.

이름과 성을 결합하기 위한 식은 다음과 같습니다.

FIRSTNAME || ' ' || SURNAME

- **LKUP_Address.** Address.dic에 대한 플랫폼 파일 조회를 수행합니다. 일련 번호가 Randid3과 동일한 주소 레코드를 검색합니다. 이 조회 변환은 주소의 열을 대상으로 전달합니다.

Customer_Test 테이블은 테스트 환경에서 사용할 수 있습니다.

식 변환을 사용하여 데이터 마스킹

열 중 하나를 마스킹한 후에 두 열 간의 관계를 유지 관리하려면 데이터 마스킹 변환과 함께 식 변환을 사용합니다.

예를 들어 보험 정책에 대한 시작 날짜와 종료 날짜가 포함된 계정 정보를 마스킹하는 경우 각각의 마스킹된 레코드에서 정책 길이를 유지 관리하려고 할 수 있습니다. 데이터 마스킹 변환을 사용하여 종료 날짜를 제외한 모든 데이터를 마스킹합니다. 식 변환을 사용하여 정책 길이를 계산하고 정책 길이를 마스킹된 시작 날짜에 추가합니다.

이 예제에는 다음과 같은 유형의 마스킹이 있습니다.

- 키
- 날짜 블러링
- 숫자 블러링
- 마스크 형식 지정

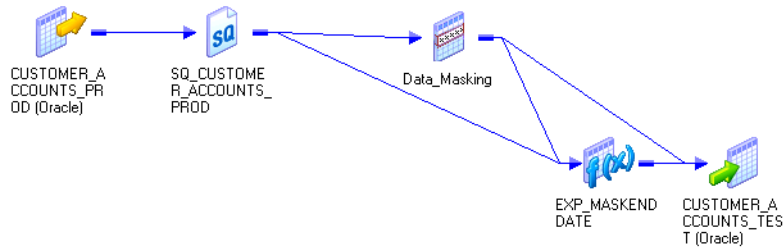
참고: 이 예제는 client\samples 폴더에서 리포지토리로 가져올 수 있는 M_CUSTOMER_ACCOUNTS_MASKING.xml 매핑입니다.

Customers_Prod라는 고객 데이터베이스 테이블에는 중요 데이터가 포함되어 있습니다. 각 열의 데이터를 마스킹하고 Customers_Test라는 대상 테이블에 테스트 데이터를 씁니다.

다음과 같은 Customer_Accounts_Prod 열을 마스킹합니다.

열	데이터 유형
AcctID	문자열
CustID	정수
Balance	배정밀도
StartDate	날짜/시간
EndDate	날짜/시간

다음 그림에는 가져올 수 있는 매핑이 나와 있습니다.



해당 매핑에는 소스 및 대상과 함께 다음과 같은 변환이 있습니다.

- **소스 한정자.** AcctID, CustID, Balance 및 Start_Date를 데이터 마스킹 변환에 전달합니다. Start_Date 및 End_Date 열을 식 변환에 전달합니다.
- **데이터 마스킹 변환.** End_Date를 제외한 모든 열을 마스킹합니다. 이 데이터 마스킹 변환은 마스킹된 열을 대상에 전달합니다. 정책 시작 날짜, 종료 날짜 및 마스킹된 시작 날짜를 식 변환에 전달합니다.

데이터 마스킹 변환이 다음 열을 마스킹합니다.

입력 포트	마스킹 유형	마스킹 규칙	설명	출력 대상
AcctID	무작위	마스킹 형식 AA+DDDDD 결과 문자열 대체 문자 ABCDEFGHIJKLMNOPQRSTUVWXYZ NOPQRSTUVWXYZ	처음 두 문자는 대문자 알파벳 문자입니다. 세 번째 문자는 대시이고 마스킹되어 있지 않습니다. 마지막 다섯 문자는 숫자입니다.	Customer_Account_Test 대상
CustID	키	시드 = 934	시드는 934입니다. CustID 마스크는 고정입니다.	Customer_Account_Test 대상

입력 포트	마스킹 유형	마스킹 규칙	설명	출력 대상
Balance	무작위	블러링 백분율 하한 = 10 상한 = 10	마스킹된 잔액은 소스 잔액의 10%에 포함됩니다.	Customer_Account_Test 대상
Start_Date	무작위	블러링 단위 = 년 하한=2 상한 = 2	마스킹된 start_date는 소스 날짜에서 2년 내에 포함됩니다.	Customer_Account_Test 대상 Exp_MaskEndDatetransformation

- **식 변환.** 마스킹된 종료 날짜를 계산합니다. 시작 날짜와 종료 날짜 사이 시간을 계산합니다. 마스킹된 시작 날짜에 해당 시간을 추가하여 마스킹된 종료 날짜를 결정합니다.

마스킹된 종료 날짜를 생성하는 식은 다음과 같습니다.

```
DIFF = DATE_DIFF(END_DATE, START_DATE, 'DD')
out_END_DATE = ADD_TO_DATE(out_START_DATE, 'DD', DIFF)
```

식 변환에서 out_END_DATE를 대상에 전달합니다.

제 7 장

식 변환

이 장에 포함된 항목:

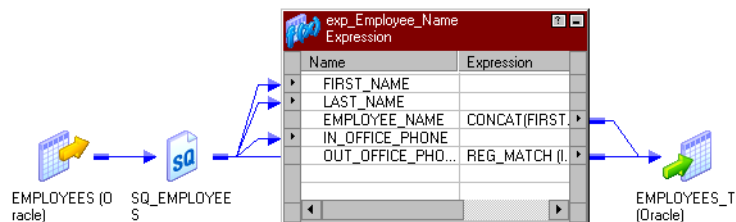
- [식 변환 개요, 152](#)
- [식 변환 구성 요소, 152](#)
- [포트 구성, 153](#)
- [식 변환 작성, 153](#)

식 변환 개요

식 변환을 사용하여 단일 행의 값을 계산합니다. 예를 들어, 직원 급여를 조정하거나, 이름과 성을 연결하거나, 문자열을 숫자로 변환해야 할 수 있습니다. 식 변환을 사용하여 결과를 대상 또는 다른 변환에 전달하기 전에 조건부 문을 테스트할 수도 있습니다. 식 변환은 수동 변환입니다.

식 변환을 사용하여 비집계 계산을 수행할 수 있습니다. 합계 또는 평균과 같이 여러 행이 연관된 계산을 수행하려면 집계 변환을 사용하십시오.

다음 그림은 EMPLOYEES 테이블의 성과 이름을 연결하는 데 사용되는 식 변환이 포함된 간단한 매핑을 보여 줍니다.



식 편집기에서 구성하는 식을 평가할 수 있습니다.

식 변환 구성 요소

변환 개발자 또는 매핑 디자이너에서 식 변환을 작성할 수 있습니다.

식 변환에는 다음과 같은 탭이 있습니다.

- **변환.** 변환 이름 및 설명을 입력합니다. 식 변환의 이름 지정 규칙은 `EXP_TransformationName`입니다. 변환을 재사용 가능하게 만들 수도 있습니다.
- **포트.** 포트를 작성하고 구성합니다.
- **속성.** 추적 수준을 구성하여 세션 로그 파일에서 보고되는 트랜잭션 세부 정보의 양을 결정합니다.
- **메타데이터 확장.** 확장 이름, 데이터 유형, 전체 자릿수 및 값을 지정합니다. 또한 재사용 가능 메타데이터 확장을 작성할 수 있습니다.

포트 구성

포트 탭에서 포트를 작성하고 수정할 수 있습니다.

포트 탭에서 다음과 같은 구성 요소를 구성합니다.

- **포트 이름.** 포트의 이름입니다.
- **데이터 유형, 전체 자릿수 및 소수 자릿수.** 데이터 유형을 구성하고 각 포트의 전체 자릿수와 소수 자릿수를 설정합니다.
- **포트 유형.** 포트는 입력, 출력, 입력/출력 또는 변수일 수 있습니다. 입력 포트는 데이터를 받고 출력 포트는 데이터를 전달합니다. 입력/출력 포트는 데이터를 변경되지 않은 상태로 전달합니다. 변수 포트는 데이터를 임시로 저장하고 행 전체에서 값을 저장할 수 있습니다.
- **식.** 식 편집기를 사용하여 식을 입력합니다. 식에서 **SQL**과 유사한 함수를 포함하는 변환 언어를 사용하여 계산을 수행합니다.
- **기본값 및 설명.** 포트의 기본값을 설정하고 설명을 추가합니다.

값 계산

식 변환을 사용하여 단일 행의 값을 계산하려면 다음과 같은 포트를 포함해야 합니다.

- **입력 또는 입력/출력 포트.** 계산에 사용되는 값을 제공합니다. 예를 들어 주문의 총 가격을 계산해야 하는 경우 두 개의 입력 또는 입력/출력 포트를 작성합니다. 한 포트는 단가를 제공하고 다른 포트는 주문 수량을 제공합니다.
- **출력 포트.** 식의 반환값을 제공합니다. 출력 포트에 대한 구성 옵션으로 식을 입력합니다. 각 포트에 대한 기본값을 구성할 수도 있습니다.

각 출력 포트마다 식을 작성하여 단일 식 변환에 여러 식을 입력할 수 있습니다. 예를 들어, 직원 급여에서 다양한 종류의 원천징수세(예: 지방/국가 소득세, 사회 보장 및 의료 보험)를 계산하고자 할 수 있습니다. 이 모든 계산에는 직원 급여와 원천징수 범주가 필요하고 해당하는 세율이 필요할 수 있으므로 급여 및 원천징수 범주에 대한 입력/출력 포트와 각 계산에 대한 별도의 출력 포트를 작성할 수 있습니다.

식 변환 작성

다음 절차에 따라 식 변환을 작성합니다.

1. 매핑 디자이너에서 매핑을 엽니다.
2. 변환 > 작성을 클릭합니다. 식 변환을 선택합니다.

3. 이름을 입력하고 완료를 클릭합니다.
4. 소스 한정자 변환 또는 다른 변환에서 포트를 선택하고 끌어서 식 변환에 추가합니다.
또한 변환을 열고 포트를 수동으로 작성할 수 있습니다.
5. 제목 표시줄을 두 번 클릭하고 포트 탭을 클릭합니다. 변환 내에서 출력 및 변수 포트를 작성할 수 있습니다.
6. 식 반환값을 일치시킬 포트 데이터 유형, 전체 자릿수 및 소수 자릿수를 할당합니다.
7. 출력 또는 변수 포트의 식 섹션에서 식 편집기를 엽니다.
8. 식을 입력합니다.
9. 유효성 검사를 클릭하여 식 구문의 유효성을 검사합니다.
10. 오른쪽 창의 식 테스트 섹션에 식 조건의 최신 변경 내용을 반영하려면 새로 고침을 클릭합니다.
11. 오른쪽 창에서 식에 사용된 입력 포트의 샘플 값을 입력합니다.
12. 식을 평가하려면 평가를 클릭합니다.
13. 확인을 클릭합니다.
14. 변환 탭에서 재사용 가능 변환을 작성합니다.
참고: 변환을 재사용 가능하게 만든 후에는 소스 한정자 변환 또는 다른 변환에서 포트를 복사할 수 없습니다. 변환 내에서 포트를 수동으로 작성할 수 있습니다.
15. 속성 탭에서 추적 수준을 구성합니다.
16. 메타데이터 확장 탭에서 메타데이터 확장을 추가합니다.
17. 확인을 클릭합니다.
18. 출력 포트를 다운스트림 변환 또는 대상에 연결합니다.

제 8 장

외부 프로시저 변환

이 장에 포함된 항목:

- [외부 프로시저 변환 개요, 155](#)
- [외부 프로시저 변환 속성 구성, 157](#)
- [COM 프로시저 개발, 159](#)
- [Informatica 외부 프로시저 개발, 166](#)
- [외부 프로시저 배포, 173](#)
- [개발 참고 사항, 175](#)
- [초기화 속성의 서비스 프로세스 변수, 181](#)
- [외부 프로시저 인터페이스, 181](#)

외부 프로시저 변환 개요

외부 프로시저 변환은 디자이너 인터페이스 외부에서 작성하는 프로시저와 함께 작동하여 **PowerCenter** 기능을 확장합니다.

표준 변환이 광범위한 옵션을 제공하지만 **PowerCenter**와 함께 제공되는 기능을 확장해야 할 수 있는 경우가 있습니다. 예를 들어 식 및 필터 변환과 같은 표준 변환의 범위가 필요한 기능을 제공하지 못할 수 있습니다. 경험이 풍부한 프로그래머는 매핑에서 필요한 식 변환을 작성하지 않고 **DLL**(동적 연결 라이브러리) 또는 **UNIX** 공유 라이브러리 내에서 복잡한 함수를 개발하려고 할 수 있습니다.

이러한 종류의 확장성을 구현하려면 **PowerCenter**에 내장된 **TX**(Transformation Exchange) 동적 호출 인터페이스를 사용합니다. **TX**를 사용하면 **Informatica** 외부 프로시저 변환을 작성하고 개발한 외부 프로시저에 바인딩할 수 있습니다. 외부 프로시저 변환을 다음 두 유형의 외부 프로시저에 바인딩할 수 있습니다.

- **COM** 외부 프로시저(**Windows**에서만 사용 가능)
- **Informatica** 외부 프로시저(**Windows**, **AIX**, **Linux** 및 **Solaris**에서 사용 가능)

TX를 사용하려면 경험이 많은 **C**, **C++** 또는 **Visual Basic** 프로그래머여야 합니다.

외부 프로시저에서 다중 스레드 코드를 사용하십시오.

코드 페이지 호환성

통합 서비스가 **ASCII** 모드로 실행되는 경우 외부 프로시저가 7비트 **ASCII** 형식의 데이터를 처리할 수 있습니다. 통합 서비스가 유니코드 모드에서 실행되는 경우에는 외부 프로시저가 통합 서비스 코드 페이지와 양방향 호환되는 데이터를 처리할 수 있습니다.

외부 프로시저 DLL 또는 공유 라이브러리에 멀티바이트 문자가 있을 경우 유니코드 모드에서 실행되도록 통합 서비스를 구성합니다. 외부 프로시저는 통합 서비스의 입력 문자열을 해석하고 멀티바이트 문자가 포함된 출력 문자열을 작성하기 위해 통합 서비스와 동일한 코드 페이지를 사용해야 합니다.

외부 프로시저 DLL 또는 공유 라이브러리에 ASCII 문자만 있을 경우 ASCII 또는 유니코드 모드에서 실행되도록 통합 서비스를 구성합니다.

외부 프로시저와 외부 프로시저 변환

TX에는 두 가지 구성 요소가 있습니다. 두 요소는 *외부 프로시저*와 *외부 프로시저 변환*입니다.

*외부 프로시저*는 통합 서비스와 별개로 존재하며 사용자가 변환을 정의하기 위해 작성한 C, C++ 또는 Visual Basic 코드로 구성됩니다. 이 코드는 컴파일된 후 DLL 또는 공유 라이브러리로 연결되며 이러한 라이브러리는 런타임 시 통합 서비스에 의해 로드됩니다. 외부 프로시저는 외부 프로시저 변환에 "바인딩"됩니다.

*외부 프로시저 변환*은 디자이너에서 작성하고, Informatica 리포지토리에 상주하는 개체이며 여러 가지 용도로 사용됩니다.

1. 다음에 나오는 외부 프로시저를 설명하는 메타데이터를 포함합니다. 통합 서비스는 이러한 메타데이터를 통해 외부 프로시저의 "서명"(매개 변수의 개수와 유형, 반환 값이 있을 경우 반환 값의 유형)을 파악합니다.
2. 외부 프로시저를 매핑에서 참조할 수 있도록 합니다. 외부 프로시저 변환의 인스턴스를 매핑에 추가함으로써 해당 변환에 바인딩된 외부 프로시저를 호출합니다.

참고: 연결되거나 연결되지 않은 외부 프로시저를 작성할 수 있습니다.

3. Informatica 외부 프로시저를 개발하는 경우 외부 프로시저 변환이 Informatica 외부 프로시저 스텝을 생성하는 데 필요한 정보를 제공합니다.

외부 프로시저 변환 속성

변환 개발자에서 재사용 가능 외부 프로시저 변환을 작성하고 변환의 인스턴스를 매핑에 추가합니다. 매핑 디자이너 또는 Maplet Designer에서는 외부 프로시저 변환을 작성할 수 없습니다.

외부 프로시저 변환은 각 입력 행에 대해 출력 행을 한 개 반환하거나 전혀 반환하지 않습니다.

외부 프로시저 변환의 속성 탭에서 모듈/프로그래밍 식별자 및 프로시저 이름 필드에 ASCII 문자만 입력합니다. 이러한 필드에 멀티바이트 문자를 입력할 수 없습니다. 외부 프로시저 변환의 포트 탭에서 포트 이름으로 ASCII 문자만 입력합니다. 외부 프로시저 변환 포트 이름에 멀티바이트 문자를 입력할 수 없습니다.

COM과 Informatica 외부 프로시저

다음 테이블에는 COM 및 Informatica 외부 프로시저 간의 차이가 설명되어 있습니다.

	COM	Informatica
기술	COM 기술 사용	Informatica 독점 기술 사용
운영 체제	Windows에서만 실행	통합 서비스를 지원하는 모든 플랫폼에서 실행: Windows, AIX, HP, Linux, Solaris
언어	C, C++, VC++, VB, Perl, VJ++	C++만 지원

BankSoft 예제

다음 섹션에서는 BankSoft 예제를 통해 COM 및 Informatica 프로시저 개발 방법을 설명합니다. BankSoft 예제에서는 외부 프로시저를 개발 및 호출하는 방법을 설명하기 위해 재무 함수 FV를 사용합니다. FV 프로시저는 정기적 납입액과 일정 이율을 기반으로 투자의 미래 가치를 계산합니다.

외부 프로시저 변환 속성 구성

속성 탭에서 변환 속성을 구성합니다.

다음 테이블에는 외부 프로시저 변환 속성이 설명되어 있습니다.

속성	설명
유형	외부 프로시저의 유형입니다. 다음과 같은 유형을 사용합니다. <ul style="list-style-type: none">- COM- Informatica 기본값은 Informatica입니다.
모듈/프로그래밍 식별자	모듈은 외부 프로시저를 포함하는 DLL(Windows) 또는 공유 개체(UNIX)의 기본 이름으로, 해당 운영 체제에서 DLL 또는 공유 개체의 이름을 결정합니다. ASCII 문자만 입력합니다. 프로그래밍 식별자 또는 ProgID는 클래스의 논리적 이름입니다. 디자이너에서는 ProgID를 통해 COM 클래스를 참조합니다. 내부적으로 클래스는 숫자 CLSID로 식별됩니다. 예를 들면 다음과 같습니다. {33B17632-1D9F-11D1-8790-0000C044ACF9} ProgID의 표준 형식은 <i>Project.Class[.Version]</i> 입니다. ASCII 문자만 입력합니다.
프로시저 이름	외부 프로시저의 이름입니다. ASCII 문자만 입력합니다.
런타임 위치	DLL 또는 공유 라이브러리가 포함된 위치입니다. 외부 프로시저 세션을 실행하는 통합 서비스 노드의 상대 경로를 입력합니다. \$PMExtProcDir를 입력하면 통합 서비스는 프로세스 변수 \$PMExtProcDir가 지정하는 디렉터리를 조회하여 라이브러리를 찾습니다. 이 속성이 비어 있을 경우 통합 서비스는 통합 서비스 노드에 정의된 환경 변수를 사용하여 DLL 또는 공유 라이브러리를 찾습니다. 경로를 런타임 위치로 하드 코딩할 수 있습니다. 경로는 단일 컴퓨터에만 국한되므로 이 방법은 권장되지 않습니다. 런타임 위치 또는 통합 서비스 노드에 정의된 환경 변수에 모든 DLL이나 공유 라이브러리를 복사해야 합니다. 통합 서비스가 DLL, 공유 라이브러리 또는 참조된 파일을 찾을 수 없으면 해당 절차를 로드하지 못합니다. 기본값은 \$PMExtProcDir입니다.
추적 수준	세션 로그 파일에서 보고되는 트랜잭션 세부 정보의 양입니다. 다음과 같은 추적 수준을 사용합니다. <ul style="list-style-type: none">- 간단- 보통- 자세한 정보 표시 초기화- 자세한 정보 표시 데이터 기본값은 보통입니다.

속성	설명
분할 가능 여부	<p>이 변환을 사용하는 파이프라인에서 여러 파티션을 작성할 수 있는지 여부를 나타냅니다. 다음 값을 사용하십시오.</p> <ul style="list-style-type: none"> - 아니요. 변환을 분할할 수 없습니다. 변환과 동일한 파이프라인에 있는 다른 변환이 하나의 파티션으로 제한됩니다. - 로컬로. 변환을 분할할 수 있지만 통합 서비스가 같은 노드의 파이프라인에서 모든 파티션을 실행해야 합니다. BAPI/RFC 변환의 여러 파티션이 메모리에서 개체를 공유해야 하는 경우 로컬로 선택합니다. - 그리드에서. 변환을 분할할 수 있고 통합 서비스가 각 파티션을 여러 노드에 분산시킬 수 있습니다. <p>기본값은 아니요입니다.</p>
출력은 반복 가능합니다.	<p>변환이 세션 실행 간에 동일한 순서로 행을 생성하는지 여부를 나타냅니다. 출력이 반복 가능하거나 확정 출력인 경우 통합 서비스가 마지막 검사점에서 세션을 다시 시작할 수 있습니다. 다음 값을 사용하십시오.</p> <ul style="list-style-type: none"> - 항상 입력 데이터 순서가 세션 실행 간에 일관되지 않더라도 출력 데이터 순서는 세션 실행 간에 일관됩니다. - 입력 순서 기반. 모든 입력 그룹의 입력 데이터 순서가 세션 실행 간에 일관적인 경우 변환에서 세션 실행 간에 반복 가능한 데이터를 생성합니다. 입력 그룹의 입력 데이터가 정렬되지 않으면 출력이 정렬되지 않습니다. - 사용 안 함 출력 데이터 순서는 세션 실행 간에 일관되지 않습니다. 변환이 반복 가능 데이터를 생성하지 않으면 마지막 검사점에서 다시 시작하도록 복구를 구성할 수 없습니다. <p>기본값은 입력 순서 기반입니다.</p>
확정 출력입니다.	<p>변환이 세션 실행 간에 일치하는 출력 데이터를 생성하는지 여부를 나타냅니다. 이 변환을 사용하는 세션에서 복구를 수행하려면 이 속성을 활성화해야 합니다.</p> <p>기본값이 비활성화됩니다.</p>

경고: 변환을 반복 가능 및 확정으로 구성하는 경우 데이터가 반복 가능 및 확정인지 확인하는 것은 사용자의 책임입니다. 세션과 복구 간에 동일한 데이터를 생성하지 않는 변환으로 세션을 복구하려는 경우 복구 프로세스로 인해 손상된 데이터가 발생할 수 있습니다.

다음 테이블에는 통합 서비스가 런타임 위치를 확인하기 위해 다양한 플랫폼에서 DLL 또는 공유 개체를 찾는 데 사용하는 환경 변수가 설명되어 있습니다.

테이블 1. 환경 변수

운영 체제	환경 변수
Windows	PATH
AIX	LIBPATH
HPUX	SHLIB_PATH
Linux	LD_LIBRARY_PATH
Solaris	LD_LIBRARY_PATH

COM 프로시저 개발

Microsoft Visual C++ 또는 Visual Basic을 사용하여 COM 외부 프로시저를 개발할 수 있습니다. 다음 섹션에는 Visual C++를 사용하여 COM 외부 프로시저를 작성하는 방법과 Visual Basic을 사용하여 COM 외부 프로시저를 작성하는 방법이 설명되어 있습니다.

COM 프로시저를 작성하기 위한 단계

COM 외부 프로시저를 작성하려면 다음 단계를 완료하십시오.

1. Microsoft Visual C++ 또는 Visual Basic을 사용하여 프로젝트를 작성합니다.
2. *IDispatch* 인터페이스 내에 클래스를 정의합니다.
3. 이 인터페이스에 메서드를 추가합니다. 이 메서드는 통합 서비스 내부에서 호출될 외부 프로시저입니다.
4. 클래스를 컴파일하고 동적 링크 라이브러리에 연결합니다.
5. 클래스를 로컬 Windows 레지스트리에 등록합니다.
6. 변환 개발자에서 COM 프로시저를 가져옵니다.
7. COM 프로시저를 포함하는 매핑을 작성합니다.
8. 매핑을 사용하여 세션을 작성합니다.

COM 외부 프로시저 서버 유형

통합 서비스는 프로세스 내 COM 서버만 지원합니다. 이러한 서버의 *서버 유형*은 *동적 연결 라이브러리*입니다. 이렇게 설계한 이유는 성능을 향상시키기 위해서입니다. 많은 양의 데이터를 처리할 때 데이터를 동일한 시스템 또는 원격 시스템의 별개 프로세스로 전달하지 않고 동일한 프로세스에서 처리하면 더욱 효율적입니다.

Visual C++를 사용하여 COM 프로시저 개발

C++ 개발자는 Visual C++ 버전 5.0 이상을 사용하여 COM 프로시저를 개발할 수 있습니다. 첫 번째 태스크는 프로젝트를 작성하는 것입니다.

관련 항목:

- [“외부 프로시저 배포” 페이지 173](#)
- [“기존 C/C++ 라이브러리 또는 VB 함수에 대한 래퍼 클래스” 페이지 176](#)

1단계. ATL COM AppWizard 프로젝트 작성

1. Visual C++를 시작하고 파일 > 새로 만들기를 클릭합니다.
2. 대화 상자가 나타나면 프로젝트 탭을 선택합니다.
3. 프로젝트 이름 및 위치를 입력합니다.
BankSoft 예에서는 프로젝트 이름으로 *COM_VC_Banksoft*를 입력하고 디렉터리로 *c:\COM_VC_Banksoft*를 입력합니다.
4. 프로젝트 목록 상자에서 *ATL COM AppWizard* 옵션을 선택하고 확인을 클릭합니다.
Visual C++에서 COM 프로젝트를 작성하는 데 사용되는 마법사가 나타납니다.
5. 서버 유형을 동적 연결 라이브러리로 설정하고 *MFC 지원* 옵션을 선택한 다음 마침을 클릭합니다.
마법사의 마지막 페이지가 나타납니다.

6. 확인을 클릭하여 Visual C++로 돌아갑니다.
7. 새 프로젝트에 클래스를 추가합니다.
8. 마법사의 다음 페이지에서 확인 단추를 클릭합니다. Developer Studio가 기본 프로젝트 파일을 작성합니다.

2단계. 프로젝트에 ATL 개체 추가

1. 작업 공간 창에서 클래스 보기 탭을 선택하고 트리 항목인 *COM_VC_BankSoft.BSoftFin* 클래스를 마우스 오른쪽 단추로 클릭한 후, 표시되는 로컬 메뉴에서 새 ATL 개체를 선택합니다.
2. 왼쪽 목록 상자에서 개체 항목을 강조 표시하고 개체 유형 목록에서 *단순 개체*를 선택합니다.
3. 다음을 클릭합니다.
4. 짧은 이름 필드에서 작성하려는 클래스의 짧은 이름을 입력합니다.
BankSoft 예제에서는 가상 기업인 BankSoft를 위해 채무 함수를 개발하는 것이므로 *BSoftFin*이라는 이름을 사용합니다. 짧은 이름 필드에 이름을 입력하면 마법사가 다른 필드에 추천 이름을 채워 넣습니다.
5. 클래스의 프로그래밍 식별자를 입력합니다.
BankSoft 예제에서는 ProgID(프로그래밍 식별자) 필드를 *COM_VC_BankSoft.BSoftFin*으로 변경합니다.
프로그래밍 식별자 또는 ProgID는 사람이 읽을 수 있는 클래스 이름입니다. 내부적으로 클래스는 숫자 CLSID로 식별됩니다. 예를 들면 다음과 같습니다.

```
{33B17632-1D9F-11D1-8790-0000C044ACF9}
```


ProgID의 표준 형식은 *프로젝트.클래스[.버전]*입니다. 디자이너에서는 ProgID를 통해 COM 클래스를 참조합니다.
6. 특성 탭을 선택하고 스레딩 모델을 *자유*로 설정하고 인터페이스를 *이중*으로 설정하며 집계 설정을 *아니요*로 지정합니다.
7. 확인을 클릭합니다.

이제 기본 클래스 정의가 있으므로 이 정의에 메서드를 추가할 수 있습니다.

3단계. 클래스에 필수 메서드 추가

1. 작업 공간 창의 클래스 보기 탭으로 돌아갑니다.
2. 트리 보기를 확장합니다.
BankSoft 예제의 경우, *COM_VC_BankSoft*를 확장합니다.
3. 새로 추가된 클래스를 마우스 오른쪽 단추로 클릭합니다.
BankSoft 예제에서는 *IBSoftFin* 트리 항목을 마우스 오른쪽 단추로 클릭합니다.
4. 메서드 추가 메뉴 항목을 클릭하고 메서드 이름을 입력합니다.
BankSoft 예제에서는 *FV*를 입력합니다.
5. 매개 변수 필드에서 메서드 서명을 입력합니다.
FV의 경우 다음을 입력합니다.

```
[in] double Rate,  
[in] long nPeriods,  
[in] double Payment,  
[in] double PresentValue,  
[in] long PaymentType,  
[out, retval] double* FV
```

이 서명은 MIDL(Microsoft Interface Description Language)로 표현됩니다. MIDL에 대한 포괄적인 설명은 MIDL 언어 참조를 참조하십시오. 다음 사항에 유의하십시오.

- **[in]**는 매개 변수가 입력 매개 변수임을 나타냅니다.
- **[out]**는 매개 변수가 출력 매개 변수임을 나타냅니다.
- **[out, retval]**는 매개 변수가 메서드의 반환 값을 나타냅니다.

또한 모든 **[out]** 매개 변수는 참조로 전달됩니다. **BankSoft** 예제에서는 매개 변수 **FV**가 **double**입니다.

6. 확인을 클릭합니다.

Developer Studio가 추가된 메서드의 스텝을 프로젝트에 추가합니다.

4단계. 구현을 포함하여 메서드 스텝 작성

1. **BankSoft** 예제에서는 작업 공간 창의 클래스 보기 탭으로 돌아가 **COM_VC_BankSoft** 클래스 항목을 확장합니다.
2. **CBSofFin** 항목을 확장합니다.
3. **CBSofFin** 항목 아래에 있는 **IBSofFin** 항목을 확장합니다.
4. **FV** 항목을 마우스 오른쪽 단추로 클릭하고 정의로 이동을 선택합니다.
5. 편집 창에서 **TODO** 주석 이후의 행에 커서를 놓고 다음 코드를 추가합니다.

```
double v = pow((1 + Rate), nPeriods);
*FV = -(
    (PresentValue * v) +
    (Payment * (1 + (Rate * PaymentType))) * ((v - 1) / Rate)
);
```

pow 함수를 참조하기 때문에 파일의 시작 부분에서 다른 모든 **include** 문 이후에 다음 전처리 문을 추가해야 합니다.

```
#include <math.h>
```

최종 단계는 **DLL**을 작성하는 것입니다. **DLL**을 작성할 때 **COM** 프로시저를 **Windows** 레지스트리에 등록합니다.

5단계. 프로젝트 빌드

1. 작성 메뉴를 펼칩니다.
2. 모두 다시 작성을 선택합니다.

Developer Studio에서 프로젝트를 작성하면서 다음과 같은 출력을 생성합니다.

```
-----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++에서 프로젝트의 파일을 컴파일하고 COM_VC_BankSoft.DLL이라는 DLL(동적 링크 라이브러리)에 파일을 링크하고 COM(ActiveX) 클래스 COM_VC_BankSoft.BSoftFin을 로컬 레지스트리에 등록합니다.

구성 요소가 등록되면 해당 호스트에서 실행되는 통합 서비스가 구성 요소에 액세스할 수 있습니다.

6단계. 리포지토리에 COM 프로시저 등록

1. 변환 개발자를 엽니다.
2. 변환 > 외부 프로시저 가져오기를 클릭합니다.
외부 COM 메서드 가져오기 대화 상자가 표시됩니다.
3. 찾아보기 단추를 클릭합니다.
4. 작성한 COM DLL을 선택하고 확인을 클릭합니다.
Banksoft 예제에서는 *COM_VC_Banksoft.DLL*을 선택합니다.
5. 메서드 선택 트리 보기 아래에서 클래스 노드(이 예제에서는 BSoftFin)를 확장합니다.
6. 메서드를 확장합니다.
7. 원하는 메서드(이 예제에서는 FV)를 선택하고 확인을 누릅니다.
디자이너가 외부 프로시저 변환을 작성합니다.
8. 외부 프로시저 변환을 열고 속성 탭을 선택합니다.
모듈/프로그래밍 식별자 및 프로시저 이름 필드에 입력합니다.

9. 포트 탭을 클릭합니다.
10. 포트 이름을 입력합니다.
11. 확인을 클릭합니다.

7단계. 매핑용 소스 및 대상 작성

다음 SQL 문을 사용하여 소스 테이블을 작성하고 이 테이블을 샘플 데이터로 채웁니다.

```
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)
```

다음 SQL 문을 사용하여 대상 테이블을 작성합니다.

```
create table FVOutputs(
    FVin_ext_proc float,
)
```

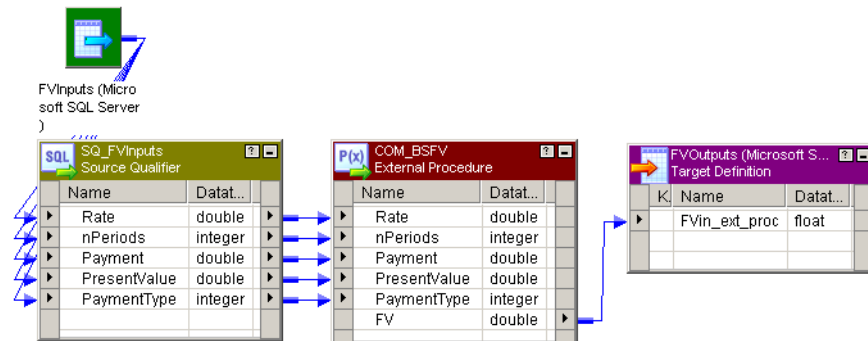
소스 분석기와 대상 디자인너를 사용하여 FVInputs 및 FVOutputs를 COM_BSFV 변환을 작성한 폴더와 동일한 폴더로 가져옵니다.

8단계. 외부 프로시저 변환을 테스트하기 위해 매핑 작성

이제 외부 프로시저 변환을 테스트하는 매핑을 작성합니다.

1. 매핑 디자인너에서 Test_BSFV라는 매핑을 작성합니다.
2. 소스 테이블 FVInputs를 매핑으로 끕니다.
3. 대상 테이블 FVOutputs를 매핑으로 끕니다.
4. 변환 COM_BSFV를 매핑으로 끕니다.
5. 소스 한정자 변환 포트를 적절한 외부 프로시저 변환 포트에 연결합니다.
6. 외부 프로시저 변환의 FV 포트를 FVin_ext_proc 대상 열에 연결합니다.
7. 매핑의 유효성을 검사하고 저장합니다.

다음 그림에서는 완성된 매핑을 보여 줍니다.



9단계. 통합 서비스 시작

통합 서비스를 시작합니다. COM 구성 요소가 등록되었던 호스트와 동일한 호스트에서 서비스가 시작되어야 합니다.

10단계. 매핑을 테스트하기 위해 워크플로우 실행

통합 서비스는 워크플로우에서 세션을 실행하면서 다음 기능을 수행합니다.

- COM 런타임 기능을 사용하여 DLL을 로드하고 클래스의 인스턴스를 작성합니다.
- COM IDispatch 인터페이스를 사용하여 매핑을 통해 전달되는 각 행에 대해 한 번씩 정의되어 있는 외부 프로시저를 호출합니다.

참고: 단일 프로젝트에서 여러 메서드를 포함하는 여러 클래스를 정의할 수 있습니다. 이러한 메서드 각각은 외부 프로시저로 호출할 수 있습니다.

워크플로우를 실행하여 매핑을 테스트하려면 다음을 수행하십시오.

1. 워크플로우 관리자에서 **Test_BSFV** 매핑에서 **s_Test_BSFV** 세션을 작성합니다.
2. **s_Test_BSFV** 세션이 포함된 워크플로우를 작성합니다.
3. 워크플로우를 실행합니다. 통합 서비스는 레지스트리에서 **COM_VC_BankSoft.BSoftFin** 클래스에 대한 항목을 검색합니다. 이 항목에는 통합 서비스가 해당 클래스가 포함된 DLL의 위치를 결정할 수 있는 정보가 들어 있습니다. 통합 서비스는 DLL을 로드하고, 클래스의 인스턴스를 작성하고, 소스 테이블의 각 행에 대해 **FV** 함수를 호출합니다.

워크플로우가 끝나면 **FVOutputs** 테이블에 다음 결과가 포함되어 있어야 합니다.

FVIn_ext_proc

2581.403374

12682.503013

82846.246372

2301.401830

Visual Basic을 사용하여 COM 프로시저 개발

Microsoft Visual Basic은 COM 프로시저를 작성하기 위한 다른 개발 환경을 제공합니다. Basic 언어의 구문과 규칙은 다르지만 개발 절차의 대체적인 윤곽은 Visual C++에서 COM 프로시저를 개발할 때와 동일합니다.

관련 항목:

- [“외부 프로시저 배포” 페이지 173](#)
- [“기존 C/C++ 라이브러리 또는 VB 함수에 대한 래퍼 클래스” 페이지 176](#)

1단계. 단일 클래스를 사용하여 Visual Basic 프로젝트 작성

1. Visual Basic을 실행하고 파일 > 새 프로젝트를 클릭합니다.
2. 대화 상자가 나타나면 프로젝트 유형으로 **ActiveX DLL**을 선택하고 확인을 클릭합니다.

Visual Basic에서 **Project1**이라는 새 프로젝트가 작성됩니다.

프로젝트 창이 표시되지 않으면 **Ctrl+R**을 입력하거나 보기 > 프로젝트 탐색기를 클릭합니다.

속성 창이 표시되지 않으면 **F4**를 누르거나 보기 > 속성을 클릭합니다.

3. 새 프로젝트의 프로젝트 탐색기 창에서 프로젝트를 마우스 오른쪽 단추로 클릭하고, 표시되는 메뉴에서 **Project1** 속성을 선택합니다.
4. 새 프로젝트 이름을 입력합니다.
프로젝트 창에서 **Project1**을 선택하고 속성 창에서 이름을 **COM_VB_BankSoft**로 변경합니다.

2단계. 프로젝트 및 클래스의 이름 변경

1. 프로젝트 탐색기 내에서 “프로젝트 – Project1” 항목을 선택합니다. 이 항목은 트리 컨트롤에서 루트 항목이어야 합니다. 속성 창에서 프로젝트 속성이 표시됩니다.
2. 속성 창에서 사전순 탭을 선택하고 이름 속성을 **COM_VB_BankSoft**로 변경합니다. 그러면 프로젝트 탐색기에서 루트 항목의 이름이 **COM_VB_BankSoft (COM_VB_BankSoft)**로 변경됩니다.
3. 프로젝트 탐색기에서 **COM_VB_BankSoft (COM_VB_BankSoft)** 항목을 확장합니다.
4. 클래스 모듈 항목을 확장합니다.
5. **Class1 (Class1)** 항목을 선택합니다. 속성 창에서 클래스 속성이 표시됩니다.
6. 속성 창에서 사전순 탭을 선택하고 이름 속성을 **BSoftFin**으로 변경합니다.

프로젝트 및 클래스의 이름을 변경하여 작성한 클래스의 프로그래밍 식별자가 “**COM_VB_BankSoft.BSoftFin**”이도록 지정합니다. 이 **ProgID**를 사용하여 디자이너 내에서 이 클래스를 참조합니다.

3단계. 클래스에 메서드 추가

코드 창 안에 포인터를 놓고 다음 텍스트를 입력합니다.

```
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
```

Visual Basic FV 함수는 “[Visual Basic을 사용하여 COM 프로시저 개발](#)” 페이지 164에서 설명된 C++ FV 함수와 동일한 작업을 수행합니다.

4단계. 프로젝트 빌드

프로젝트를 작성하려면 다음을 수행하십시오.

1. 파일 메뉴에서 **COM_VB_BankSoft.DLL** 만들기를 선택합니다. 파일 위치를 묻는 대화 상자가 표시됩니다.
2. 파일 위치를 입력하고 확인을 클릭합니다.

Visual Basic에서 소스 코드를 컴파일하고 사용자가 지정한 위치에 COM_VB_BankSoft.DLL을 작성합니다. 또한 클래스 COM_VB_BankSoft.BSoftFin을 로컬 레지스트리에 등록합니다.

구성 요소가 등록된 후 해당 호스트에서 실행되는 통합 서비스가 구성 요소에 액세스할 수 있습니다.

Informatica 외부 프로시저 개발

32비트 또는 64비트 통합 서비스 시스템에서 실행되는 외부 프로시저를 작성할 수 있습니다. Informatica 스타일 외부 프로시저를 작성하려면 다음 단계를 수행하십시오.

1. 변환 개발자에서 외부 프로시저 변환을 작성합니다.
외부 프로시저 변환은 프로시저의 서명을 정의합니다. 포트의 이름, 데이터 유형 및 포트 유형(입력 또는 출력)이 외부 프로시저의 서명과 일치해야 합니다.
2. 외부 프로시저의 템플릿 코드를 생성합니다.
이 명령을 실행하면 디자이너가 외부 프로시저 변환의 정보를 사용하여 몇 가지 C++ 소스 코드 파일과 makefile을 작성합니다. 이러한 소스 코드 파일 중 하나에는 변환에서 서명을 정의한 함수의 "스텝"이 포함됩니다.
3. 코드를 수정하여 프로시저 논리를 추가합니다. 구현을 포함하여 스텝을 작성하고 C++ 컴파일러를 사용하여 소스 코드 파일을 컴파일한 후 동적 연결 라이브러리 또는 공유 라이브러리로 연결합니다.
통합 서비스는 Informatica 프로시저에 바인딩된 외부 프로시저 변환을 발견할 경우 DLL 또는 공유 라이브러리를 로드하고 사용자가 정의한 외부 프로시저를 호출합니다.
4. 라이브러리를 빌드하고 통합 서비스 시스템에 복사합니다.
5. 외부 프로시저 변환을 사용하여 매핑을 작성합니다.
6. 워크플로우에서 세션을 실행합니다.

BankSoft 예제는 이 기능을 구현하는 방법을 보여 줍니다.

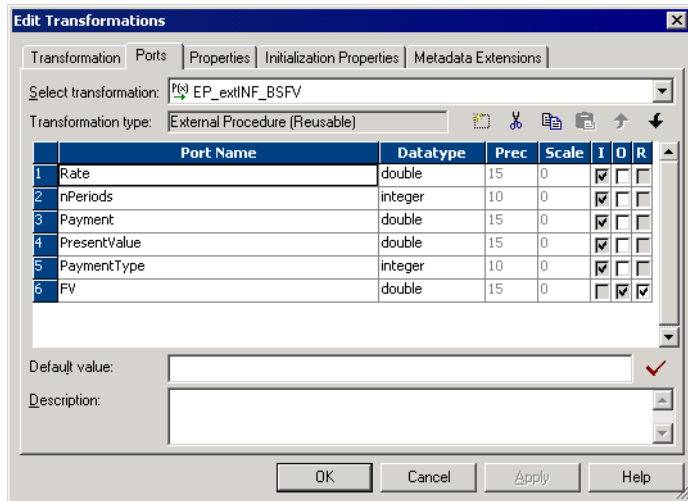
1단계. 외부 프로시저 변환 작성

1. 변환 개발자를 열고 외부 프로시저 변환을 작성합니다.
2. 변환을 열고 변환 이름을 입력합니다.
BankSoft 예제에서는 EP_extINF_BSFV를 입력합니다.
3. 포트 탭에서 정의하려는 프로시저로 전달되는 각 인수에 대한 포트를 작성합니다.
올바른 데이터 유형을 사용해야 합니다.
다음 테이블에는 포트에 대한 설명이 나와 있습니다.

포트 이름	데이터 유형	전체 자릿수	배율	입력/출력	재사용 가능
Rate	배정밀도	15	0	입력	아니요
nPeriods	정수	10	0	입력	아니요
Payment	배정밀도	15	0	입력	아니요
PresentValue	배정밀도	15	0	입력	아니요

포트 이름	데이터 유형	전체 자릿수	배율	입력/출력	재사용 가능
PaymentType	정수	10	0	입력	아니요
FV	배정밀도	15	0	출력	예

다음 BankSoft 예제에서는 외부 프로시저 변환의 예를 보여 줍니다.



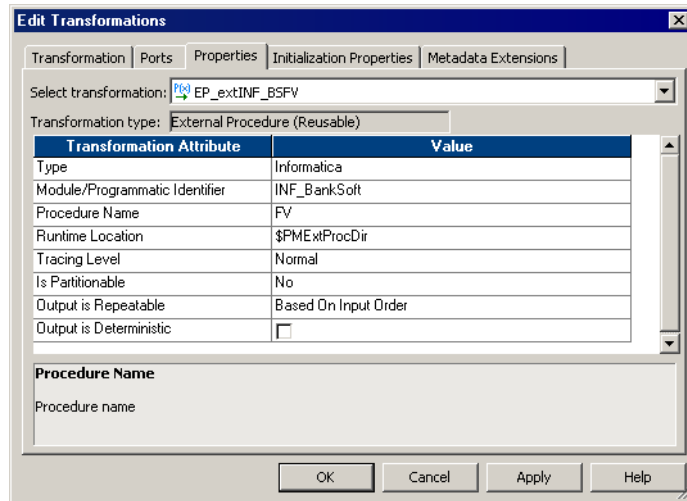
마지막 포트 **FV**는 프로시저의 반환 값을 캡처합니다.

- 속성 탭을 선택하고 프로시저를 **Informatica** 프로시저로 구성합니다.

BankSoft 예제에서는 다음 속성을 구성합니다.

변환 특성	값
유형	Informatica
모듈/프로그래밍 식별자	INF_BankSoft
프로시저 이름	FV
런타임 위치	\$PMExtProcDir
추적 수준	보통
분할 가능 여부	아니요
출력은 반복 가능합니다.	입력 순서에 따름
확정 출력입니다.	아니요

다음 BankSoft 예제에서는 Informatica 프로시저의 예를 보여 줍니다.



참고: 위에서 모듈/프로그래밍 식별자 행을 참조하십시오.

다음 테이블에는 여러 플랫폼에서 모듈 이름이 DLL 또는 공유 개체의 이름을 어떻게 결정하는지 설명되어 있습니다.

운영 체제	모듈 식별자	라이브러리 파일 이름
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. 확인을 클릭합니다.

프로시저를 호출하는 외부 프로시저 변환을 작성한 이후의 다음 단계는 C++ 파일을 생성하는 것입니다.

2단계. C++ 파일 생성

외부 프로시저 변환을 작성한 후에 코드를 생성하십시오. UNIX에 매핑된 드라이브에 작성되는 파일의 이름은 항상 소문자이기 때문에 디자이너는 소문자로 된 파일 이름을 생성합니다. 생성되는 파일에는 다음 규칙이 적용됩니다.

- **파일 이름.** TX 모듈 파일의 경우 접두사 'tx'가 사용됩니다.
- **모듈 클래스 이름.** 생성되는 코드에는 TX 프로시저를 포함하는 모듈에 대한 클래스 선언이 있습니다. TX 모듈 클래스의 경우 접두사 *Tx*가 사용됩니다. 예를 들어 외부 프로시저 변환에 **Mymod**라는 모듈 이름이 있으면 클래스 이름은 **TxMymod**입니다.

외부 프로시저에 대한 코드를 생성하려면 다음을 수행하십시오.

1. 변환을 선택하고 변환 > 코드 생성을 클릭합니다.
2. 작성한 프로시저의 이름 옆에 있는 확인란을 선택합니다.

BankSoft 예제에서는 *INF_BankSoft.FV*를 선택합니다.

3. 파일을 생성하려는 디렉터리를 지정하고 생성을 클릭합니다.

디자이너가 하위 디렉터리 `INF_BankSoft`를 지정된 디렉터리에 작성합니다.

디자이너에서 작성된 각 외부 프로시저 변환은 모듈 및 프로시저 이름을 지정해야 합니다. 디자이너는 공통 모듈 이름을 공유하는 모든 변환을 위해 단일 디렉터리에 코드를 생성합니다. 한 디렉터리에서 코드를 작성하면 단일 공유 라이브러리가 작성됩니다.

디자이너는 다음과 같은 파일을 생성합니다.

- **tx<모듈 이름>.h.** 외부 프로시저 모듈 클래스를 정의합니다. 이 클래스는 기본 클래스 `TINFExternalModule60`에서 파생됩니다. 생성되는 코드에서 이 클래스에 대해 정의된 데이터 멤버는 없습니다. 그러나 여기에 새 데이터 멤버 및 메서드를 추가할 수 있습니다.
 - **tx<모듈 이름>.cpp.** 외부 프로시저 모듈 클래스를 구현합니다. 추가한 새 데이터 멤버의 초기화를 포함하도록 `InitDerived()` 메서드를 확장할 수 있습니다. 통합 서비스는 기본 클래스의 `Init()` 메서드를 성공적으로 완료한 경우에만 파생 클래스의 `InitDerived()` 메서드를 호출합니다.
- 이 파일은 모듈에 있는 모든 외부 프로시저 변환의 서명을 정의합니다. 이러한 서명을 수정하면 디자이너에서 정의된 외부 프로시저 변환과 불일치하게 됩니다. 따라서 서명을 변경해서는 안 됩니다.
- 이 파일에는 C 함수 `CreateExternalModuleObject`가 포함되어 있고, 이 함수는 이 파일에 정의된 생성자를 사용하여 외부 프로시저 모듈 클래스의 개체를 작성합니다. 통합 서비스는 생성자를 직접 호출하는 대신 `CreateExternalModuleObject`를 호출합니다.
- **<프로시저 이름>.cpp.** 디자이너는 이 모듈에 있는 각 외부 프로시저에 대해 이 파일을 하나씩 생성합니다. 이 파일은 데이터 정리 및 필터링 같은 프로시저 논리를 구현하는 코드를 포함합니다. 데이터 정리의 경우, 입력 포트에서 값을 읽어들이 출력 포트의 값을 생성하는 코드를 작성합니다. 필터링의 경우, `INF_NO_OUTPUT_ROW`를 반환하여 출력 행의 생성을 억제하는 코드를 작성합니다.
 - **stdafx.h** UNIX 시스템에서 작성하는 데 사용되는 스탭 파일입니다. 다양한 *.cpp 파일에서 이 파일을 포함합니다. Windows 시스템에서는 Visual Studio가 `stdafx.h` 파일을 생성하며, 이 파일은 디자이너가 생성한 파일 대신 사용되어야 합니다.
 - **version.cpp** 이 구현의 버전 번호를 포함하는 작은 파일입니다. 이전 릴리스에서는 외부 프로시저 구현이 다르게 처리되었습니다. 이 파일을 통해 통합 서비스는 외부 프로시저 모듈의 버전을 확인할 수 있습니다.
 - **makefile.aix, makefile.aix64, makefile.hp, makefile.hp64, makefile.hpparisc64, makefile.linux, makefile.sol.** UNIX 플랫폼용 Make 파일입니다. 32비트 플랫폼의 경우 `makefile.aix, makefile.hp, makefile.linux` 및 `makefile.sol`을 사용합니다. 64비트 AIX 플랫폼의 경우 `makefile.aix64`를 사용하고, 64비트 HP-UX(Itanium) 플랫폼의 경우 `makefile.hp64`를 사용합니다.

예제 1

BankSoft 예제에서 디자이너는 다음과 같은 파일을 생성합니다.

- **txinf_banksoft.h.** 모듈 클래스 `TxINF_BankSoft` 및 외부 프로시저 `FV`에 대한 선언을 포함합니다.
- **txinf_banksoft.cpp.** 모듈 클래스 `TxINF_BankSoft`의 코드를 포함합니다.
- **fv.cpp.** `FV` 프로시저의 코드를 포함합니다.
- **version.cpp** TX 버전을 반환합니다.
- **stdafx.h** UNIX에서 컴파일하기 위해 필요합니다. Windows에서 `stdafx.h`는 Visual Studio에 의해 생성됩니다.
- **readme.txt** 일반 도움말 정보를 포함합니다.

예제 2

프로시저 이름이 'Myproc1'과 'Myproc2'이고 모듈 이름이 둘 다 Mymod인 두 개의 외부 프로시저 변환을 작성하는 경우 디자이너가 다음과 같은 파일을 생성합니다.

- **txmymod.h.** 모듈 클래스 TxMymod와 외부 프로시저 Myproc1 및 Myproc2에 대한 선언을 포함합니다.
- **txmymod.cpp.** 모듈 클래스 TxMymod의 코드를 포함합니다.
- **myproc1.cpp.** Myproc1 프로시저의 코드를 포함합니다.
- **myproc2.cpp.** Myproc2 프로시저의 코드를 포함합니다.
- **version.cpp**
- **stdafx.h**
- **readme.txt**

3단계. 구현을 포함하여 메서드 스텝 작성

최종 단계는 프로시저를 코딩하는 것입니다.

1. 프로시저에 대해 생성된 `<Procedure_Name>.cpp` 스텝 파일을 엽니다.
BankSoft 예제에서는 `fv.cpp`를 열어 `TxINF_BankSoft::FV` 프로시저를 코딩합니다.
2. 프로시저의 C++ 코드를 입력합니다.

다음 코드는 FV 프로시저를 구현합니다.

```
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.
    ostrstream 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;
}

```

디자이너는 인수 및 반환 값을 포함하는 함수 프로필을 생성합니다. 주석에 지시된 대로 함수 내에 실제 코드를 입력해야 합니다. POW 함수를 참조하고 `ostrstream` 변수를 정의했으므로 전처리기 문 또한 포함해야 합니다.

Windows:

```

#include <math.h>
#include <strstream> using namespace std;

```

UNIX에서는 `include` 문이 다음과 같습니다.

```

#include <math.h>
#include <strstream.h>

```

- 수정된 파일을 저장합니다.

4단계. 모듈 작성

Windows에서 Visual C++를 사용하여 DLL을 컴파일합니다.

Windows 모듈 작성

Windows에서 DLL을 빌드하려면:

- Visual C++를 시작합니다.
- 파일 > 새로 만들기를 클릭합니다.
- 새로 만들기 대화 상자에서 프로젝트 탭을 클릭하고 MFC AppWizard(DLL) 옵션을 선택합니다.
- 위치를 입력합니다.

BankSoft 예제에서 `c:\pmclient\tx\INF_BankSoft`를 입력합니다(`c:\pmclient\tx`에서 파일을 생성했다고 가정).

- 프로젝트 이름을 입력합니다.

이 이름은 외부 프로시저 변환에 대해 입력한 모듈 이름과 같아야 합니다. BankSoft 예제에서는 `INF_BankSoft`입니다.

- 확인을 클릭합니다.

이제 Visual C++에서 마법사를 통해 프로젝트의 모든 구성 요소를 정의하는 단계를 안내합니다.

- 마법사에서 MFC Extension DLL(공유 MFC DLL 사용)을 클릭합니다.
- 마침을 클릭합니다.

마법사가 여러 파일을 생성합니다.

- 프로젝트 > 프로젝트에 추가 > 파일을 클릭합니다.

- 한 수준 위의 디렉터리를 탐색합니다. 작성한 외부 프로시저 파일이 이 디렉터리에 포함됩니다. 모든 .cpp 파일을 선택합니다.

BankSoft 예제에서 다음 파일을 추가합니다.

- fv.cpp
- txinf_banksoft.cpp
- version.cpp

11. 프로젝트 > 설정을 클릭합니다.
12. C/C++ 탭을 클릭하고 범주 필드에서 전처리기를 선택합니다.
13. 추가 포함 디렉터리 필드에 `..; <pmserver install dir>\extproc\include`를 입력합니다.
14. 링크 탭을 클릭하고 범주 필드에서 일반을 선택합니다.
15. 개체/라이브러리 모듈 필드에 `<pmserver install dir>\bin\pmtx.lib`를 입력합니다.
16. 확인을 클릭합니다.
17. 빌드 > INF_BankSoft.dll 빌드를 클릭하거나 F7을 눌러 프로젝트를 빌드합니다.
이제 컴파일러가 DLL을 작성하고 프로젝트 디렉터리 아래의 디버그 또는 릴리스 디렉터리에 배치합니다.

UNIX 모듈 작성

UNIX에서 공유 라이브러리를 빌드하려면:

1. PowerCenter 클라이언트 도구에 직접 액세스할 수 없는 경우 공유 라이브러리에 필요한 모든 파일을 빌드를 수행하려고 하는 UNIX 시스템에 복사합니다.
예를 들어 BankSoft 프로시저에서 ftp 또는 다른 메커니즘을 사용하여 INF_BankSoft 디렉터리에서 UNIX 컴퓨터로 모든 파일을 복사합니다.
2. 환경 변수 INFA_HOME을 PowerCenter 설치 디렉터리로 설정합니다.
경고: INFA_HOME 환경 변수에 대해 잘못된 디렉터리 경로를 지정하면 통합 서비스가 시작되지 않습니다.
3. 프로젝트를 만들기 위해 명령을 입력합니다.
이 명령은 다음과 같이 UNIX 버전에 따라 다릅니다.

UNIX 버전	명령
AIX(32비트)	make -f makefile.aix
AIX(64비트)	make -f makefile.aix64
Linux	make -f makefile.linux
Solaris	make -f makefile.sol

5단계. 매핑 생성

매핑 디자이너에서 외부 프로시저 변환을 사용하는 매핑을 작성합니다.

6단계. 세션 실행

세션을 실행하면 통합 서비스는 런타임 위치로 지정된 디렉터리를 살펴보고 4단계에서 작성된 라이브러리(DLL)를 찾습니다. 세션 속성에서 런타임 위치 속성의 기본값은 \$PMExtProcDir입니다.

세션을 실행하려면 다음을 수행하십시오.

1. 워크플로우 관리자에서 워크플로우를 작성합니다.

2. 워크플로우에서 이 매핑에 대한 세션을 작성합니다.
팁: 또는 태스크 개발자에서 재사용 가능 세션을 작성하여 워크플로우에서 사용할 수도 있습니다.
3. 라이브러리(DLL)를 런타임 위치 디렉터리로 복사합니다.
4. 세션이 포함된 워크플로우를 실행합니다.

Windows에서 모듈의 디버그 버전으로 세션 실행

Informatica는 Windows용 PowerCenter를 외부 프로시저 변환 라이브러리의 릴리스 빌드(pmtx.dll) 및 디버그 빌드(pmtxdbg.dll)와 함께 제공합니다. 이러한 라이브러리는 서버 bin 디렉터리에 설치됩니다.

4단계에서 모듈의 릴리스 버전을 작성하는 경우, 워크플로우의 세션을 실행하여 외부 프로시저 변환 라이브러리의 릴리스 빌드(pmtx.dll)를 사용합니다. 다음 태스크를 완료할 필요는 없습니다.

4단계에서 모듈의 디버그 버전을 작성하는 경우, 아래 절차에 따라 외부 프로시저 변환 라이브러리의 디버그 빌드(pmtxdbg.dll)를 사용합니다.

모듈의 디버그 버전을 사용하여 세션을 실행하려면 다음을 수행하십시오.

1. 워크플로우 관리자에서 워크플로우를 작성합니다.
2. 워크플로우에서 이 매핑에 대한 세션을 작성합니다.
또한 태스크 개발자에서 재사용 가능 세션을 작성하여 워크플로우에서 사용할 수도 있습니다.
3. 라이브러리(DLL)를 런타임 위치 디렉터리로 복사합니다.
4. 외부 프로시저 변환 라이브러리의 디버그 빌드를 사용하려면 다음을 수행하십시오.
 - pmtx.dll 파일의 이름을 바꾸거나 서버 bin 디렉터리에서 pmtx.dll 파일을 이동하여 해당 파일을 보존합니다.
 - pmtxdbg.dll의 이름을 pmtx.dll로 바꿉니다.
5. 세션이 포함된 워크플로우를 실행합니다.
6. 외부 프로시저 변환 라이브러리의 릴리스 빌드를 기본 라이브러리로 되돌리려면 다음을 수행하십시오.
 - pmtx.dll의 이름을 pmtxdbg.dll로 다시 바꿉니다.
 - 원래 pmtx.dll 파일을 서버 bin 디렉터리로 다시 이동하거나 원래 이름으로 바꿉니다.

참고: Windows에서 모듈의 디버그 버전을 사용하여 이 세션이 포함된 워크플로우를 실행하는 경우, 원래 pmtx.dll 파일을 원래 이름 및 위치로 되돌려야 디버그 이외의 세션을 실행할 수 있습니다.

외부 프로시저 배포

일련의 외부 프로시저를 개발하고 각각 통합 서비스를 실행하고 있는 여러 서버에서 이러한 프로시저를 사용할 수 있도록 한다고 가정합니다. 이를 실현하는 방법은 외부 프로시저의 유형과 이러한 프로시저를 작성하는 운영 시스템에 따라 다릅니다.

또한 이 방법을 통해 외부 프로시저를 외부 고객에게 배포할 수도 있습니다.

COM 프로시저 배포

프로젝트를 빌드하면 Visual Basic과 Visual C++가 COM 클래스를 로컬 레지스트리에 등록합니다. 클래스가 등록되면 DLL을 컴파일한 시스템에서 실행되는 통합 서비스가 이러한 클래스에 액세스할 수 있습니다. 예를 들어 HOST1에서 프로젝트를 빌드하는 경우 프로젝트의 모든 클래스가 HOST1 레지스트리에 등록되고 HOST1에서

실행되는 통합 서비스에서 액세스할 수 있게 됩니다. 하지만 HOST2에서 실행 중인 통합 서비스도 클래스에 액세스할 수 있게 한다고 가정합니다. 이렇게 하려면 클래스를 HOST2 레지스트리에 등록해야 합니다.

Visual Basic은 Windows 시스템에서 COM 클래스를 설치하고 이러한 클래스를 해당 시스템의 레지스트리에 등록할 수 있는 설치 프로그램을 작성하기 위한 유틸리티를 제공합니다. Visual C++에서 사용할 수 있는 유틸리티는 없지만 직접 손쉽게 클래스를 등록할 수 있습니다.

COM Visual Basic 프로시저 배포

COM Visual Basic 프로시저를 배포하려면

1. DLL을 빌드한 후 Visual Basic을 종료하고 Visual Basic 응용 프로그램 설치 마법사를 실행합니다.
2. 마법사의 첫 번째 패널을 건너뛰니다.
3. 두 번째 패널에서 프로젝트의 위치를 지정하고 *설치 프로그램 작성* 옵션을 선택합니다.
4. 세 번째 패널에서 사용하려는 배포 방법을 선택합니다.
5. 다음 패널에서 설치 파일을 기록할 디렉터리를 지정합니다.

간단한 ActiveX 구성 요소의 경우 마법사의 마지막 패널로 계속할 수 있습니다. 그렇지 않으면 파일 유형 및 배포 방법에 따라 자세한 정보를 추가해야 할 수 있습니다.

6. 마지막 패널에서 마침을 클릭합니다.

그러면 Visual Basic이 DLL용 설치 프로그램을 작성합니다. 통합 서비스가 실행 중인 Windows 시스템에서 이 설치 프로그램을 실행합니다.

수동으로 COM Visual Basic 프로시저 배포

수동으로 COM Visual C++/Visual Basic 프로시저를 배포하려면

1. DLL을 저장하려는 새로운 Windows 시스템의 디렉터리에 복사합니다.
2. 이 Windows 시스템에 로그인하고 DOS 프롬프트를 엽니다.
3. DLL이 포함된 디렉터리로 이동하고 다음 명령을 실행합니다.

```
REGSVR32 project_name.DLL
```

*project_name*은 작성한 DLL의 이름입니다. BankSoft 예제에서 프로젝트 이름은 *COM_VC_BankSoft.DLL* 또는 *COM_VB_BankSoft.DLL*입니다.

이 명령줄 프로그램이 DLL과 그 안에 포함된 모든 COM 클래스를 등록합니다.

Informatica 모듈 배포

외부 프로시저를 리포지토리 간에 배포할 수 있습니다.

외부 프로시저를 리포지토리 간에 배포하려면

1. 외부 프로시저가 포함된 DLL 또는 공유 개체를 통합 서비스가 액세스할 수 있는 시스템의 디렉터리로 이동합니다.
2. 디자이너 클라이언트 도구를 사용하여 외부 프로시저 변환을 원래 리포지토리에서 대상 리포지토리로 복사합니다.

또한 외부 프로시저 변환을 XML 파일로 내보내고 대상 리포지토리로 가져올 수도 있습니다.

개발 참고 사항

이 섹션에는 COM 및 Informatica 외부 프로시저 개발과 관련된 추가 지침 및 정보가 포함되어 있습니다.

COM 데이터 유형

Visual C++ 또는 Visual Basic을 사용하여 COM 프로시저를 개발하는 경우 데이터를 읽고 변환할 때 통합 서비스가 사용하는 내부 데이터 유형과 일치하는 COM 데이터 유형을 사용해야 합니다. 이와 같은 데이터 유형 일치는 통합 서비스가 외부 프로시저 변환의 포트와 변환이 호출하는 프로시저의 인수(또는 반환 값) 간에 데이터 유형을 매핑하려 할 때 중요합니다.

다음 테이블에서는 Visual C++ 및 변환 데이터 유형을 비교합니다.

Visual C++ COM 데이터 유형	변환 데이터 유형
VT_I4	정수
VT_UI4	정수
VT_R8	배정밀도
VT_BSTR	문자열
VT_DECIMAL	10진수
VT_DATE	날짜/시간

다음 테이블에서는 Visual Basic 및 변환 데이터 유형을 비교합니다.

Visual Basic COM 데이터 유형	변환 데이터 유형
긴 정수	정수
배정밀도	배정밀도
문자열	문자열
10진수	10진수
날짜	날짜/시간

데이터 유형을 올바르게 일치시키지 않으면 통합 서비스가 변환을 시도할 수 있습니다. 예를 들어 포트에 정수 데이터 유형을 할당하지만 해당 인수의 데이터 유형이 BSTR일 경우 통합 서비스는 정수 값을 BSTR로 변환하려고 시도합니다.

행 수준 프로시저

모든 외부 프로시저 변환은 변환을 통해 전달되는 단일 행의 값을 사용하여 프로시저를 호출합니다. 단일 프로시저 호출에서 여러 행의 값을 사용할 수는 없습니다. 예를 들어 집계 함수 SUM 또는 AVG와 동등한 코드를 프로시저 호출 안에 작성할 수 없습니다. 이러한 의미에서 모든 외부 프로시저는 *상태 비저장*이어야 합니다.

프로시저의 반환 값

프로시저를 호출하면 통합 서비스는 프로시저에 코딩된 반환 값 이외의 추가적인 반환 값을 캡처합니다. 이 추가적인 값은 통합 서비스에서 프로시저를 성공적으로 호출했는지 여부를 나타냅니다.

COM 프로시저의 경우 이 반환 값은 HRESULT 유형을 사용합니다.

Informatica 프로시저는 INF_RESULT 유형을 사용합니다. 반환된 값이 S_OK/INF_SUCCESS이면 통합 서비스가 프로시저를 성공적으로 호출한 것입니다. 사용자는 외부 프로시저의 성공 또는 실패를 나타내기 위한 적절한 값을 반환해야 합니다. Informatica 프로시저는 값 4개를 반환합니다.

- **INF_SUCCESS.** 외부 프로시저가 성공적으로 행을 처리했습니다. 통합 서비스는 매핑에 있는 다음 변환으로 행을 전달합니다.
- **INF_NO_OUTPUT_ROW.** 통합 서비스가 외부 프로시저 논리로 인해 현재 행을 기록하지 못합니다. 이는 오류가 아닙니다. 행을 필터링하는 데 INF_NO_OUTPUT_ROW를 사용하는 경우, 외부 프로시저 변환은 필터 변환과 유사하게 동작합니다.

참고: 외부 프로시저에서 INF_NO_OUTPUT_ROW를 사용할 경우, 외부 프로시저 변환으로부터만 행을 받는 다른 변환에 외부 프로시저 변환을 연결해야 합니다.

- **INF_ROW_ERROR.** 변환 오류와 같습니다. 통합 서비스가 현재 행을 무시하지만, n 오류 시 중지되도록 세션을 구성하지 않았다면 다음 행을 처리할 수 있습니다.
- **INF_FATAL_ERROR.** ABORT() 함수 호출과 같습니다. 통합 서비스가 세션을 중단하고 더 이상 행을 처리하지 않습니다.

프로시저 호출의 예외

통합 서비스는 외부 프로시저 변환을 통해 COM 또는 Informatica 프로시저를 호출할 때 발생하는 대부분의 예외를 캡처합니다. 예를 들어 프로시저 호출에서 0으로 나누기 오류를 작성하는 경우 통합 서비스가 예외를 캐치합니다.

경우에 따라서는 통합 서비스가 프로시저 호출에서 생성된 오류를 캡처하지 못합니다. 통합 서비스는 프로세스 내 COM 서버만 지원하고 모든 Informatica 프로시저는 공유 라이브러리 및 DLL에 저장되기 때문에 외부 프로시저를 실행하는 코드는 통합 서비스와 동일한 메모리의 주소 공간에 존재합니다. 따라서 외부 프로시저 코드가 통합 서비스 메모리를 덮어쓰고 통합 서비스를 중지시킬 수 있습니다. COM 또는 Informatica 프로시저로 인해 이러한 중지가 발생하면 소스 코드에서 메모리 액세스 문제를 검토하십시오.

프로시저에 대한 메모리 관리

Informatica 프로시저에서 사용되는 모든 데이터 유형은 고정 길이이므로 Informatica 외부 프로시저에 대한 메모리 관리 문제는 없습니다. COM 프로시저의 경우에는 프로시저의 [out] 매개 변수가 BSTR 데이터 유형을 사용할 때만 메모리를 할당해야 합니다. 이 경우 이 프로시저에 대한 모든 호출에서 메모리를 할당해야 합니다. 세션 중에 통합 서비스는 함수 호출 후 메모리를 해제합니다.

기존 C/C++ 라이브러리 또는 VB 함수에 대한 래퍼 클래스

BankSoft에 C 또는 C++ 함수 라이브러리가 있으며 이러한 함수를 통합 서비스에 연결하려 한다고 가정합니다. 특히, 라이브러리에 BankSoft 고유의 FV 함수 구현인 PreExistingFV가 포함되어 있습니다. 이 작업을 수행하는 일반 메서드는 COM 프로시저와 Informatica 외부 프로시저 모두에서 동일합니다. Visual Basic에서 유사한 솔루션을 사용할 수 있습니다. 단, 기존 Visual Basic 함수를 호출하거나 Visual Basic에서 액세스할 수 있는 개체의 메서드를 호출해야 합니다.

오류 및 추적 메시지 생성

“4단계. 모듈 작성” 페이지 171에서 Informatica 외부 프로시저 TxINF_BankSoft::FV의 구현에는 다음과 같은 코드 줄이 포함됩니다.

```
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;
```

통합 서비스는 Tx<MODNAME> 유형의 개체를 작성하는 경우 오류 또는 디버깅 메시지를 세션 로그에 기록하는데 사용할 수 있는 콜백 함수에 대한 포인터를 개체의 생성자에 전달합니다. Tx<MODNAME> 생성자의 코드는 Tx<MODNAME>.cpp 파일에 있습니다. 이 포인터는 Tx<MODNAME> 멤버 변수 m_pfnMessageCallback에 저장됩니다. 이 포인터의 유형은 \$PMExtProcDir/include/infemmsg.h 파일의 typedef에 정의됩니다.

```
typedef void (*PFN_MESSAGE_CALLBACK)(

    enum E_MSG_TYPE eMsgType,

    unsigned long Code,

    char* Message

);
```

또한 이 파일에서는 E_MSG_TYPE 열거도 정의됩니다.

```
enum E_MSG_TYPE {

    E_MSG_TYPE_LOG = 0,

    E_MSG_TYPE_WARNING,

    E_MSG_TYPE_ERR

};
```

콜백 함수의 eMsgType을 E_MSG_TYPE_LOG로 지정하면 콜백 함수가 로그 메시지를 세션 로그에 기록합니다. E_MSG_TYPE_ERR를 지정하면 콜백 함수가 오류 메시지를 세션 로그에 기록합니다. E_MSG_TYPE_WARNING을 지정하면 콜백 함수가 경고 메시지를 세션 로그에 기록합니다. 이러한 메시지를 사용하여 Informatica 외부 프로시저에서 간단한 디버깅 기능을 제공할 수 있습니다.

COM 외부 프로시저를 디버깅하려면 Visual Basic 또는 C++ 클래스 내에서 사용할 수 있는 출력 기능을 사용합니다. 예를 들어 Visual Basic에서 MsgBox를 사용하여 각 행의 계산 결과를 출력합니다. 디버깅하는 동안 데이터의 작은 샘플에만 이 작업을 수행하고 프로덕션을 실행하기 전에 MsgBox를 제거해야 합니다.

참고: Visual Basic 또는 C++ 클래스 내에서 출력 기능을 사용하기 전에 다음 값을 레지스트리에 추가해야 합니다.

1. 다음 항목을 Windows 레지스트리에 추가합니다.

```
\HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\PowerMart\Parameters\MiscInfo
\RunInDebugMode=Yes
```

이 옵션은 통합 서비스를 서비스가 아닌 일반 응용 프로그램으로 시작합니다. 통합 서비스가 실행 중인 동안 통합 서비스에 대한 디버그 권한을 변경하지 않고 통합 서비스를 디버깅할 수 있습니다.

2. PMSERVER.EXE 명령을 사용하여 명령줄에서 통합 서비스를 시작합니다.

이제 통합 서비스가 디버그 모드에서 실행됩니다.

디버깅을 마치면 이 항목을 레지스트리에서 제거하거나 `RunInDebugMode`를 No로 설정해야 합니다. 그렇지 않으면 `PowerCenter`를 서비스로 시작하려고 할 때 `PowerCenter`가 시작되지 않습니다.

1. 통합 서비스를 중지하고 앞서 추가한 레지스트리 항목을 다음 설정으로 변경합니다.

```
\HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\PowerMart\Parameters\MiscInfo\RunInDebugMode=No
```

2. 통합 서비스를 Windows 서비스로 다시 시작합니다.

TINFPParam 클래스 및 표시기

<PROCNAME> 메서드는 매개 변수 배열 2개를 사용하여 입력 및 출력 매개 변수에 액세스하며, 각 배열 요소의 데이터 유형은 `TINFPParam`입니다. `TINFPParam` 데이터 유형은 `Informatica` 내부 데이터 유형을 저장할 수 있는 “변형” 데이터 구조 역할을 하는 C++ 클래스입니다. `TINFPParam*` 유형의 매개 변수에 있는 실제 데이터는 `Get<유형>` 및 `Set<유형>` 형태의 멤버 함수를 통해 액세스되며, 여기서 <유형>은 `Informatica` 내부 데이터 유형 중 하나입니다. 또한 `TINFPParam`에는 각 매개 변수의 표시기를 가져오고 설정하기 위한 메서드가 있습니다.

외부 프로시저에 진입할 때 표시기를 확인하고 외부 프로시저에서 나올 때 표시기를 설정하는 것은 사용자의 몫입니다. 진입 시에 모든 출력 매개 변수의 표시기는 명시적으로 `INF_SQL_DATA_NULL`로 설정되기 때문에, 외부 프로시저에서 돌아오기 전에 이러한 표시기를 재설정하지 않으면 모든 출력 매개 변수에 대해 `NULL`을 가져오게 됩니다. `TINFPParam` 클래스는 특정 매개 변수에 대한 메타데이터를 가져오기 위한 함수도 지원합니다.

`TINFPParam` 클래스의 모든 멤버 함수에 대한 전체적인 설명은 `tx/include` 디렉터리에서 `infemdef.h` 포함 파일을 참조하십시오.

COM 외부 프로시저와 비교했을 때 `Informatica` 외부 프로시저의 주요 이점 중 하나는 표시기 조작을 직접 지원한다는 점입니다. 즉, 입력 매개 변수를 점검하여 입력 매개 변수가 `NULL`인지 확인하고 출력 매개 변수를 `NULL`로 설정할 수 있습니다. COM에서는 표시기에 대한 지원을 제공하지 않습니다. 따라서, COM 스타일 외부 프로시저로 들어가는 행에 `NULL`이 있으면 해당 행은 처리될 수 없습니다. 이 단점을 극복하려면 디자이너에서 기본 값 기능을 사용하십시오. 그러나 COM 함수 외부로 `NULL`을 전달할 수는 없습니다.

연결되지 않은 외부 프로시저 변환

외부 프로시저 변환의 인스턴스를 매핑에 추가할 경우, 선택에 따라 해당 인스턴스를 파이프라인의 일부로 연결하거나 연결되지 않은 상태 그대로 둘 수 있습니다. 연결된 외부 프로시저 변환은 행이 변환을 통과할 때마다 COM 또는 `Informatica` 프로시저를 호출합니다.

연결되지 않은 외부 프로시저 변환으로부터 반환 값을 가져오려면 식에서 다음 구문을 사용하여 변환을 호출하십시오.

```
:EXT.transformation_name(arguments)
```

이 식을 포함하는 변환을 행이 통과할 때 통합 서비스는 외부 프로시저 변환과 연결된 프로시저를 호출합니다. 이 식은 외부 프로시저 변환 반환 포트를 통해 프로시저의 반환 값을 캡처하며, 반환 포트에 대해 결과(R) 옵션이 선택되어 있어야 합니다.

COM 및 Informatica 모듈 초기화

일부 외부 프로시저는 초기화 시 구성해야 합니다. 이러한 초기화는 외부 프로시저의 유형에 따라 다음 두 가지 형태 중 하나를 취합니다.

- **Informatica 스타일 외부 프로시저의 초기화.** 외부 프로시저가 포함된 Tx<MODNAME> 클래스에 초기화 함수 Tx<MODNAME>::InitDerived도 포함됩니다. 이러한 초기화 함수의 서명은 통합 서비스에 잘 알려져 있으며 세 가지 매개 변수로 구성됩니다.

- **nInitProps.** 이 매개 변수는 초기화 함수에 전달되는 초기화 속성의 수를 알려 줍니다.

- **속성.** 이 매개 변수는 초기화 속성의 이름을 나타내는 nInitProp 문자열의 배열입니다.

- **Values.** 이 매개 변수는 초기화 속성의 값을 나타내는 nInitProp 문자열의 배열입니다.

통합 서비스는 먼저 기본 클래스의 Init() 함수를 호출합니다. Init() 함수가 성공적으로 완료되면 기본 클래스는 Tx<MODNAME>::InitDerived() 함수를 호출합니다.

통합 서비스는 Tx<MODNAME> 개체를 작성한 후 초기화 함수를 호출합니다. 초기화 속성을 해석하고 이러한 속성을 사용하여 외부 프로시저를 초기화하는 Tx<MODNAME>::InitDerived() 함수의 해당 부분을 제공하는 것은 외부 프로시저 개발자의 책임입니다. 개체가 작성되고 초기화되면 통합 서비스가 각 행에 대해 개체의 외부 프로시저를 호출할 수 있습니다.

- **COM 스타일 외부 프로시저의 초기화.** 외부 프로시저 또는 EP 개체가 포함된 개체에 초기화 함수가 포함되지 않습니다. 대신 다른 개체(CF 개체)가 EP 개체의 클래스 팩터리로 작동합니다. CF 개체에는 EP 개체를 작성할 수 있는 메서드가 있습니다.

CF 개체 메서드의 서명은 유형 라이브러리에서 결정됩니다. 통합 서비스는 CF 개체를 작성한 후 EP 개체를 작성하기 위해 CF 개체의 메서드를 호출하고 필요한 모든 매개 변수를 이 메서드에 전달합니다. 이를 위해 메서드의 서명은 유형 라이브러리에서 유형을 확인할 수 있는 일련의 입력 매개 변수와 그 뒤에 단일 출력 매개 변수로 구성되어야 합니다. 이러한 단일 출력 매개 변수는 IUnknown** 또는 IDispatch**이거나 IUnknown* 또는 IDispatch*를 가리키는 VARIANT*입니다.

입력 매개 변수는 EP 개체를 초기화하는 데 필요한 값을 포함하고 출력 매개 변수는 초기화된 개체를 받습니다. 출력 매개 변수는 [out] 또는 [out, retval] 특성을 가질 수 있습니다. 즉, 초기화된 개체가 출력 매개 변수 또는 메서드의 반환 값으로 반환될 수 있습니다. 입력 매개 변수에 지원되는 데이터 유형은 다음과 같습니다.

- COM VC 유형

- VT_UI1

- VT_BOOL

- VT_I2

- VT_UI2

- VT_I4

- VT_UI4

- VT_R4

- VT_R8

- VT_BSTR

- VT_CY

- VT_DATE

디자이너에서 초기화 속성 설정

변환 편집 대화 상자의 초기화 속성 탭에 외부 프로시저 초기화 속성을 입력합니다. 이 탭에는 외부 프로시저가 COM 스타일인지 또는 Informatica 스타일인지에 따라 서로 다른 필드가 표시됩니다.

COM 스타일 외부 프로시저 변환의 경우 초기화 속성 탭에 다음 필드가 포함됩니다.

- **클래스 팩터리의 프로그래밍 식별자.** 클래스 팩터리의 프로그래밍 식별자를 입력합니다.
- **생성자.** EP 개체를 작성하는 클래스 팩터리의 메서드를 지정합니다.

COM 스타일 및 Informatica 스타일 외부 프로시저 변환 모두에서 생성자 메서드로 전달할 초기화 속성을 수직 제한 없이 입력할 수 있습니다.

새 초기화 속성을 추가하려면 추가 단추를 클릭합니다. 매개 변수 이름을 속성 열에 입력하고 매개 변수 값을 값 열에 입력하십시오. 예를 들어 다음 매개 변수를 입력할 수 있습니다.

매개 변수	값
Param1	abc
Param2	100
Param3	3.17

참고: 디자이너에서 정의한 초기화 속성과 클래스 팩터리 생성자 메서드의 입력 매개 변수 간에 일대일 관계를 작성해야 합니다. 예를 들어 생성자에 n 개의 매개 변수가 있고 마지막 매개 변수는 초기화된 개체를 받는 출력 매개 변수인 경우, 생성자 메서드의 각 입력 매개 변수에 대해 1개씩 $n - 1$ 개의 초기화 속성을 디자이너에서 정의해야 합니다.

초기화 속성에서 프로세스 변수를 사용할 수도 있습니다.

TX에서 배포 및 사용되는 다른 파일

다음은 \$PMExtProcDir/include 경로에 있는 헤더 파일이며, 이러한 파일은 외부 프로시저를 컴파일하는 데 필요합니다.

- infconfig.h
- infem60.h
- infemdef.h
- infemmsg.h
- infparam.h
- infsigtr.h

다음은 <PMInstallDir> 경로에 있는 라이브러리 파일이며, 이러한 파일은 외부 프로시저를 연결하고 세션을 실행하는 데 필요합니다.

- libpmtx.a (AIX)
- libpmtx.so (Linux)
- libpmtx.so (Solaris)
- pmtx.dll 및 pmtx.lib(Windows)

초기화 속성의 서비스 프로세스 변수

PowerCenter는 외부 프로시저 변환 초기화 속성 목록에서 기본 제공 프로세스 변수를 지원합니다. 속성 값에 기본 제공 프로세스 변수가 포함된 경우 통합 서비스는 해당 변수를 외부 프로시저 라이브러리로 전달하기 전에 먼저 확장합니다. 이 기능은 이식성이 있는 외부 프로시저 변환을 작성할 때 매우 유용하게 사용할 수 있습니다.

다음 테이블에는 변환 속성의 초기화 속성 탭에 있는 외부 프로시저 변환에 대한 초기화 속성 및 값이 설명되어 있습니다.

테이블 2. 외부 프로시저 초기화 속성

속성	값	외부 프로시저 라이브러리로 전달되는 확장된 값
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

워크플로우를 실행하면 통합 서비스가 속성 목록을 확장한 다음 외부 프로시저 초기화 함수로 확장된 목록을 전달합니다. 기본 제공 프로세스 변수 `$PMTempDir`은 `/tmp`, `$PMSourceFileDir`은 `/data/input`, `$PMTargetFileDir`은 `/data/output`이라고 가정하여, “초기화 속성의 서비스 프로세스 변수” 페이지 181의 마지막 열에는 속성 및 확장된 값 정보가 포함되어 있습니다. 통합 서비스는 마지막 속성 “`$some_other_variable`”을 확장하지 않습니다. 이는 이 속성이 기본 제공 프로세스 변수가 아니기 때문입니다.

외부 프로시저 인터페이스

통합 서비스는 외부 프로시저에서 다음과 같은 주요 함수를 사용합니다.

- 디스패치
- 외부 프로시저
- 속성 액세스
- 매개 변수 액세스
- 코드 페이지 액세스
- 변환 이름 액세스
- 프로시저 액세스
- 파티션 관련
- 추적 수준

디스패치 함수

통합 서비스는 디스패치 함수를 호출하여 각 입력 행을 외부 프로시저 모듈에 전달합니다. 그 결과 디스패치 함수는 사용자가 지정한 외부 프로시저 함수를 호출합니다.

외부 프로시저는 멤버 변수 `m_pInParamVector`(입력 포트) 및 `m_pOutParamVector`(출력 포트)를 사용하여 직접 변환의 포트에 액세스합니다.

서명

디스패치 함수에는 인덱스 매개 변수 1개를 포함하는 고정된 서명이 있습니다.

```
virtual INF_RESULT Dispatch(unsigned long ProcedureIndex) = 0
```

외부 프로시저 함수

외부 프로시저 함수는 외부 프로시저 모듈로의 기본 진입점이며 외부 프로시저 변환의 특성입니다. 디스패치 함수는 모든 입력 행에 대해 외부 프로시저 함수를 호출합니다. 외부 프로시저 변환의 경우 외부 프로시저 모듈의 입력 및 출력에 외부 프로시저 함수를 사용합니다. 이 함수는 모든 입력 행에 대해 IN 및 IN-OUT 포트 값에 액세스할 수 있으며 OUT 및 IN-OUT 포트 값을 설정할 수 있습니다. 외부 프로시저 함수에는 모든 입력 및 출력 처리 논리가 포함됩니다.

서명

외부 프로시저 함수에는 매개 변수가 없습니다. 입력 매개 변수 배열은 이미 `InitParams()` 메서드를 통해 전달되어 멤버 변수 `m_pInParamVector`에 저장되어 있습니다. 배열에 있는 각 항목은 외부 프로시저 변환의 상응하는 IN 및 IN-OUT 포트와 일치하며 순서도 동일합니다. 통합 서비스는 디스패치 함수를 호출하기 전에 이 벡터를 채웁니다.

`Dispatch()` 함수를 반환하기 전에 출력 행을 전달하려면 멤버 변수 `m_pOutParamVector`를 사용하십시오.

`MyExternal` 프로시저 변환에서 외부 프로시저 함수는 다음과 같습니다. 여기에서 입력 매개 변수는 멤버 변수 `m_pInParamVector`에 있고 출력 값은 멤버 변수 `m_pOutParamVector`에 있습니다.

```
INF_RESULT Tx<ModuleName>::MyFunc()
```

속성 액세스 함수

속성 액세스 함수는 외부 프로시저 변환과 연결된 초기화 속성에 대한 정보를 제공합니다. 초기화 속성 이름과 값은 외부 프로시저 변환을 편집할 때 초기화 속성 탭에 표시됩니다.

`Informatica`는 기본 클래스 및 `TINFCfgEntriesList` 클래스에서 속성 액세스 함수를 제공합니다. 초기화 속성 이름 및 값에 액세스하려면 `TINFCfgEntriesList` 클래스의 `GetConfigEntryName()` 및 `GetConfigEntryValue()` 함수를 각각 사용하십시오.

서명

`Informatica`의 기본 클래스에는 다음과 같은 함수가 있습니다.

```
TINFCfgEntriesList* TINFBASEExternalModule60::accessConfigEntriesList();  
const char* GetConfigEntry(const char* LHS);
```

`Informatica`의 `TINFCfgEntriesList` 클래스에는 다음과 같은 함수가 있습니다.

```
const char* TINFCfgEntriesList::GetConfigEntryValue(const char* LHS);  
const char* TINFCfgEntriesList::GetConfigEntryValue(int i);  
const char* TINFCfgEntriesList::GetConfigEntryName(int i);  
const char* TINFCfgEntriesList::GetConfigEntry(const char* LHS)
```

참고: `TINFCfgEntriesList` 클래스에서 `GetConfigEntryName()` 및 `GetConfigEntryValue()` 속성 액세스 함수를 사용하여 초기화 속성 이름 및 값에 액세스합니다.

TX 프로그램에서 이러한 함수를 호출할 수 있습니다. 그런 다음 TX 프로그램에서 `atoi`, `sscanf` 등을 사용하여 이 문자열 값을 숫자로 변환합니다. 다음 예에서 "addFactor"가 초기화 속성입니다. `accessConfigEntriesList()`는 TX 기본 클래스의 멤버 변수이며 정의할 필요가 없습니다.

```
const char* addFactorStr = accessConfigEntriesList()-> GetConfigEntryValue("addFactor");
```

매개 변수 액세스 함수

매개 변수 액세스 함수는 데이터 유형별로 다릅니다. 먼저 매개 변수 액세스 함수 `GetDataType`을 사용하여 매개 변수의 데이터 유형을 반환합니다. 그런 다음 이 데이터 유형에 상응하는 매개 변수 액세스 함수를 사용하여 해당 매개 변수에 대한 정보를 반환합니다.

외부 프로시저로 전달되는 매개 변수의 데이터 유형은 `TINFPParam*`입니다. 관련된 액세스 함수는 헤더 파일 `infparam.h`에 정의되어 있습니다. 디자이너는 매개 변수 데이터 유형을 나타내는 주석을 포함하는 스텝 코드를 생성합니다. 디자이너에 있는 상응하는 외부 프로시저 변환에서 매개 변수의 데이터 유형을 확인할 수도 있습니다.

서명

외부 프로시저로 전달된 매개 변수는 `TINFPParam` 클래스의 개체에 대한 포인터입니다. 이 고정된 서명 함수는 해당 클래스의 메서드이며 매개 변수 데이터 유형을 `enum` 값으로 반환합니다.

유효한 데이터 유형은 다음과 같습니다.

- `INF_DATATYPE_LONG`
- `INF_DATATYPE_STRING`
- `INF_DATATYPE_DOUBLE`
- `INF_DATATYPE_RAW`
- `INF_DATATYPE_TIME`

다음 테이블에는 몇 가지 매개 변수 액세스 함수가 설명되어 있습니다.

매개 변수 액세스 함수	설명
<code>INF_DATATYPE GetDataType(void);</code>	매개 변수의 데이터 유형을 가져옵니다. 이 매개 변수 데이터 유형을 사용하여 매개 변수 값에 액세스할 때 사용할 데이터 유형별 함수를 결정합니다.
<code>INF_BOOLEAN IsValid(void);</code>	입력 데이터가 유효한지 확인합니다. 매개 변수가 문자열 또는 원시 데이터 유형에 대해 잘린 데이터를 포함하는 경우 <code>FALSE</code> 를 반환합니다. 입력 데이터가 전체 자릿수를 초과하거나 입력 데이터가 <code>null</code> 값인 경우에도 <code>FALSE</code> 를 반환합니다.
<code>INF_BOOLEAN IsNULL(void);</code>	입력 데이터가 <code>NULL</code> 인지 확인합니다.
<code>INF_BOOLEAN IsInputMapped (void);</code>	이 매개 변수에 데이터를 전달하는 입력 포트가 변환에 연결되어 있는지를 확인합니다.
<code>INF_BOOLEAN IsOutput Mapped (void);</code>	이 매개 변수에서 데이터를 받는 출력 포트가 변환에 연결되어 있는지를 확인합니다.
<code>INF_BOOLEAN IsInput(void);</code>	매개 변수가 입력 포트에 해당하는지 확인합니다.
<code>INF_BOOLEAN IsOutput(void);</code>	매개 변수가 출력 포트에 해당하는지 확인합니다.

매개 변수 액세스 함수	설명
INF_BOOLEAN GetName(void);	매개 변수의 이름을 가져옵니다.
SQLIndicator GetIndicator(void);	매개 변수 표시기의 값을 가져옵니다. IsValid 및 ISNULL 함수는 이 함수의 특수한 형태입니다. 이 함수는 INF_SQL_DATA_TRUNCATED도 반환할 수 있습니다.
void SetIndicator(SQLIndicator Indicator);	출력 매개 변수 표시기를 설정합니다(예: invalid 또는 truncated).
long GetLong(void);	긴 정수 또는 정수 데이터 유형인 매개 변수의 값을 가져옵니다. 매개 변수 데이터 유형이 정수 또는 긴 정수인 경우에만 이 함수를 호출하십시오. 이 함수는 데이터를 긴 정수에서 다른 데이터 유형으로 변환하지 않습니다.
double GetDouble(void);	부동 소수점 수 또는 배정밀도 데이터 유형인 매개 변수의 값을 가져옵니다. 매개 변수 데이터 유형이 부동 소수점 수 또는 배정밀도인 경우에만 이 함수를 호출하십시오. 이 함수는 데이터를 배정밀도에서 다른 데이터 유형으로 변환하지 않습니다.
char* GetString(void);	매개 변수의 값을 Null로 끝나는 문자열로 가져옵니다. 매개 변수 데이터 유형이 문자열인 경우에만 이 함수를 호출하십시오. 이 함수는 데이터를 문자열에서 다른 데이터 유형으로 변환하지 않습니다. 다음 데이터 행을 읽는 경우 포인터의 값이 변경됩니다. 한 행의 값을 나중에 사용할 수 있도록 저장하려면 할당된 고유 버퍼에 명시적으로 이 문자열을 복사하십시오.
char* GetRaw(void);	매개 변수의 값을 Null로 끝나지 않는 바이트 배열로 가져옵니다. 매개 변수 데이터 유형이 원시인 경우에만 이 함수를 호출하십시오. 이 함수는 데이터를 원시 데이터 유형에서 다른 데이터 유형으로 변환하지 않습니다.
unsigned long GetActualDataLen(void);	GetRaw가 반환한 배열의 현재 길이를 가져옵니다.
TINFTime GetTime(void);	날짜/시간 데이터 유형인 매개 변수의 값을 가져옵니다. 매개 변수 데이터 유형이 날짜/시간인 경우에만 이 함수를 호출하십시오. 이 함수는 데이터를 날짜/시간에서 다른 데이터 유형으로 변환하지 않습니다.
void SetLong(long lVal);	출력 매개 변수의 값을 긴 정수 데이터 유형으로 설정합니다.
void SetDouble(double dblVal);	출력 매개 변수의 값을 배정밀도 데이터 유형으로 설정합니다.
void SetString(char* sVal);	출력 매개 변수의 값을 문자열 데이터 유형으로 설정합니다.
void SetRaw(char* rVal, size_t ActualDataLen);	Null로 끝나지 않는 바이트 배열을 설정합니다.
void SetTime(TINFTime timeVal);	출력 매개 변수의 값을 날짜/시간 데이터 유형으로 설정합니다.

64비트 통합 서비스에서 외부 프로시저를 실행할 경우 SetInt32 또는 GetInt32 함수만 사용하십시오. 다음 함수를 사용해서는 안 됩니다.

- GetLong
- SetLong
- GetpLong

- GetpDouble
- GetpTime

두 매개 변수 목록을 사용하여 매개 변수를 전달합니다.

다음 테이블에는 외부 프로시저 기본 클래스의 멤버 변수가 나열되어 있습니다.

변수	설명
m_nInParamCount	입력 매개 변수의 수
m_pInParamVector	실제 입력 매개 변수 배열
m_nOutParamCount	출력 매개 변수의 수
m_pOutParamVector	실제 출력 매개 변수 배열
참고: 입력/출력으로 정의된 포트는 두 매개 변수 목록 모두에 나타납니다.	

코드 페이지 액세스 함수

Informatica는 통합 서비스의 코드 페이지를 반환하는 두 개의 코드 페이지 액세스 함수와 외부 프로시저가 처리하는 데이터의 코드 페이지를 반환하는 두 개의 코드 페이지 액세스 함수를 제공합니다. 통합 서비스가 유니코드 모드에서 실행되는 경우 외부 프로시저 프로그램에 전달되는 문자열 데이터에 멀티바이트 문자가 포함될 수 있습니다. 코드 페이지는 외부 프로시저가 멀티바이트 문자열을 해석하는 방법을 결정합니다. 통합 서비스가 유니코드 모드에서 실행되는 경우 외부 프로시저 프로그램이 처리하는 데이터가 통합 서비스 코드 페이지와 양방향 호환되어야 합니다.

서명

외부 프로시저 프로그램을 통해 통합 서비스 코드 페이지를 가져오려면 다음 함수를 사용하십시오. 두 함수는 동일한 정보를 반환합니다.

```
int GetServerCodePageID() const;
const char* GetServerCodePageName() const;
```

외부 프로시저 프로그램을 통해 외부 프로시저가 처리하는 데이터의 코드 페이지를 가져오려면 다음 함수를 사용하십시오. 두 함수는 동일한 정보를 반환합니다.

```
int GetDataCodePageID(); // returns 0 in case of error
const char* GetDataCodePageName() const; // returns NULL in case of error
```

변환 이름 액세스 함수

Informatica는 외부 프로시저 변환의 이름을 반환하는 두 가지 변환 이름 액세스 함수를 제공합니다.

GetWidgetName() 함수는 변환의 이름을 반환하고, GetWidgetInstanceName() 함수는 맵렛 또는 매핑에 있는 변환 인스턴스의 이름을 반환합니다.

서명

변환 이름 액세스 함수에 의해 반환되는 char*는 통합 서비스의 코드 페이지에 있는 MBCS 문자열입니다. 데이터 코드 페이지에는 없습니다.

```
const char* GetWidgetInstanceName() const;
const char* GetWidgetName() const;
```

프로시저 액세스 함수

Informatica는 두 가지 프로시저 액세스 함수를 제공하며, 이러한 함수는 외부 프로시저 변환과 연결된 외부 프로시저에 대한 정보를 제공합니다. `GetProcedureName()` 함수는 외부 프로시저 변환의 프로시저 이름 필드에 지정된 외부 프로시저 이름을 반환합니다. `GetProcedureIndex()` 함수는 외부 프로시저의 인덱스를 반환합니다.

서명

외부 프로시저 변환과 연결된 외부 프로시저의 이름을 가져오려면 다음 함수를 사용하십시오.

```
const char* GetProcedureName() const;
```

외부 프로시저 변환과 연결된 외부 프로시저의 인덱스를 가져오려면 다음 함수를 사용하십시오.

```
inline unsigned long GetProcedureIndex() const;
```

파티션 관련 함수

여러 파티션을 포함하는 세션에서 외부 프로시저에 대해 파티션 관련 함수를 사용합니다. 외부 프로시저 변환을 포함하는 세션을 분할하면 통합 서비스가 해당 변환의 인스턴스를 각 파티션별로 작성합니다. 예를 들어 세션에 대해 5개의 파티션을 정의하는 경우, 통합 서비스는 세션 런타임에 각 외부 프로시저의 인스턴스를 5개 작성합니다.

서명

세션에 있는 파티션 수를 가져오려면 다음 함수를 사용합니다.

```
unsigned long GetNumberOfPartitions();
```

이 외부 프로시저를 호출한 파티션의 인덱스를 가져오려면 다음 함수를 사용합니다.

```
unsigned long GetPartitionIndex();
```

추적 수준 함수

추적 수준 함수는 다음 예와 같이 세션 추적 수준을 반환합니다.

```
typedef enum
{
    TRACE_UNSET = 0,
    TRACE_TERSE = 1,
    TRACE_NORMAL = 2,
    TRACE_VERBOSE_INIT = 3,
    TRACE_VERBOSE_DATA = 4
} TracingLevelType;
```

서명

다음 함수를 사용하여 세션 추적 수준을 반환합니다.

```
TracingLevelType GetSessionTraceLevel();
```

제 9 장

필터 변환

이 장에 포함된 항목:

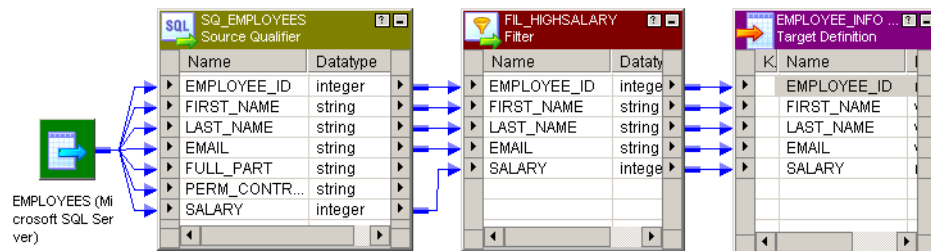
- [필터 변환 개요, 187](#)
- [필터 변환 구성 요소, 188](#)
- [필터 조건, 188](#)
- [필터 변환 작성 단계, 189](#)
- [필터 변환에 대한 팁, 189](#)

필터 변환 개요

매핑의 행을 필터링하려면 필터 변환을 사용하십시오. 활성 변환인 필터 변환은 해당 변환을 통과하는 행의 수를 변경할 수 있습니다. 필터 변환은 지정된 필터 조건을 충족하는 행의 통과를 허용합니다. 조건을 충족하지 못하는 행은 삭제됩니다. 하나 이상의 조건을 기반으로 데이터를 필터링할 수 있습니다. 필터 변환은 활성 변환입니다.

행이 지정된 조건을 충족하는지 여부에 따라 통합 서비스가 평가하는 행마다 필터 조건이 **TRUE** 또는 **FALSE**를 반환합니다. **TRUE**를 반환하는 각 행의 경우 통합 서비스가 변환을 통과합니다. **FALSE**를 반환하는 각 행의 경우 통합 서비스가 삭제하고 메시지를 세션 로그에 기록합니다.

다음 매핑은 필터 변환을 통해 직원 데이터가 포함된 인적 자원 테이블의 행을 전달합니다. 필터에서는 급여가 \$30,000 이상인 직원에 대한 행의 통과를 허용합니다.



둘 이상 변환의 포트를 필터 변환에 연결할 수 없습니다. 필터의 입력 포트는 단일 변환에서 제공되어야 합니다.

팁: 세션 성능을 극대화하기 위해 필터 변환은 매핑에서 최대한 소스에 가까이 배치합니다. 매핑을 통해 무시할 행을 전달하는 대신, 소스에서 대상까지의 데이터 흐름 초기에 원치 않는 데이터를 필터링할 수 있습니다.

필터 변환 구성 요소

변환 개발자 또는 매핑 디자이너에서 필터 변환을 작성할 수 있습니다. 필터 변환에는 다음 탭이 있습니다.

- **변환.** 변환 이름 및 설명을 입력합니다. 필터 변환의 이름 지정 규칙은 `FIL_TransformationName`입니다. 변환을 재사용 가능하게 만들 수도 있습니다.
- **포트.** 포트를 작성하고 구성합니다.
- **속성.** 행을 필터링할 필터 조건을 구성합니다. 식 편집기를 사용하여 필터 조건을 입력합니다. 또한 추적 수준을 구성하여 세션 로그 파일에 보고되는 트랜잭션 세부 정보 양을 결정할 수 있습니다.
- **메타데이터 확장.** 재사용 불가능 메타데이터 확장을 작성하여 변환의 메타데이터를 확장합니다. 확장 이름, 데이터 유형, 전체 자릿수 및 값을 구성합니다. 또한 메타데이터 확장을 모든 변환에 사용 가능하게 만들려는 경우 이 메타데이터 확장을 재사용 가능하게 승격시킬 수 있습니다.

필터 변환 포트 구성

포트 탭에서 포트를 작성하고 수정할 수 있습니다.

포트 탭의 다음 속성을 구성할 수 있습니다.

- **포트 이름.** 포트의 이름입니다.
- **데이터 유형, 전체 자릿수 및 소수 자릿수.** 데이터 유형을 구성하고 각 포트의 전체 자릿수와 소수 자릿수를 설정합니다.
- **포트 유형.** 모든 포트는 입력/출력 포트입니다. 입력 포트는 데이터를 받고 출력 포트는 데이터를 전달합니다.
- **기본값 및 설명.** 포트의 기본값을 설정하고 설명을 추가합니다.

필터 조건

필터 조건은 `TRUE` 또는 `FALSE`를 반환하는 식입니다. 속성 탭에서 사용 가능한 식 편집기를 사용하여 조건을 입력합니다.

단일 값을 반환하는 모든 식을 필터로 사용할 수 있습니다. 예를 들어, 급여가 \$30,000 미만인 직원의 행을 필터링하려면 다음 조건을 입력합니다.

```
SALARY > 30000
```

`AND` 및 `OR` 논리 연산자를 사용하여 조건의 여러 구성 요소를 지정할 수 있습니다. 급여가 \$30,000 미만인 직원과 \$100,000보다 많은 직원을 필터링하려면 다음 조건을 입력합니다.

```
SALARY > 30000 AND SALARY < 100000
```

필터 조건에 대한 상수를 입력할 수도 있습니다. `FALSE`에 해당하는 숫자는 영(0)입니다. 영(0)이 아닌 값은 `TRUE`에 해당합니다. 예를 들어, 숫자 데이터 유형을 가진 `NUMBER_OF_UNITS`라는 포트가 변환에 포함되어 있습니다. `NUMBER_OF_UNITS` 값이 영(0)일 경우 `FALSE`를 반환하도록 필터 조건을 구성하십시오. 그렇지 않을 경우 조건에서 `TRUE`를 반환합니다.

식에서 `TRUE` 또는 `FALSE`를 값으로 지정할 필요는 없습니다. `TRUE` 및 `FALSE`는 설정된 조건의 암시적인 반환 값입니다. 필터 조건이 `NULL`로 평가될 경우 행은 `FALSE`로 처리됩니다.

참고: 필터 조건은 대/소문자를 구분합니다.

Null 값을 가진 행 필터링

Null 값 또는 공백을 포함하는 행을 필터링하려면 `ISNULL` 및 `IS_SPACES` 함수를 사용하여 포트 값을 테스트하십시오. 예를 들어, `FIRST_NAME` 포트에 `NULL` 값을 포함하는 행을 필터링할 경우 다음 조건을 사용하십시오.

```
IIF(ISNULL(FIRST_NAME),FALSE,TRUE)
```

이 조건은 `FIRST_NAME` 포트가 `NULL`일 경우 반환 값이 `FALSE`이고 행이 무시됨을 기술합니다. 그렇지 않은 경우 행이 다음 변환으로 전달됩니다.

필터 변환 작성 단계

다음 절차에 따라 필터 변환을 작성합니다.

1. 매핑 디자이너에서 매핑을 엽니다.
2. 변환 > 작성을 클릭합니다. 필터 변환을 선택합니다.
3. 변환 이름을 입력합니다. 작성을 클릭한 후 완료를 클릭합니다.
4. 소스 한정자 또는 다른 변환에서 모든 포트를 선택하고 끌어 와서 필터 변환에 추가합니다.
5. 제목 표시줄을 두 번 클릭하고 포트 탭을 클릭합니다. 변환 내에 포트를 수동으로 작성할 수도 있습니다.
6. 속성 탭을 클릭하여 필터 조건 및 추적 수준을 구성합니다.
7. 필터 조건의 값 섹션에서 식 편집기를 엽니다.
8. 적용하려는 필터 조건을 입력합니다. 기본 조건은 `TRUE`를 반환합니다.
변환에 있는 입력 포트 중 하나의 값을 이 조건의 일부로 사용합니다. 다른 변환에 있는 출력 포트의 값을 사용할 수도 있습니다.
9. 식을 입력합니다. 유효성 검사를 클릭하여 입력한 조건의 구문을 확인합니다.
10. 추적 수준을 선택합니다.
11. 메타데이터 확장 탭에서 메타데이터 확장을 추가합니다.

필터 변환에 대한 팁

매핑에서 조기에 필터 변환을 사용하십시오.

세션 성능을 극대화하려면 필터 변환은 매핑에서 가능한 소스에 가까이 두십시오. 매핑을 통해 무시할 행을 전달하는 대신, 소스에서 대상까지의 데이터 흐름 초기에 원치 않는 데이터를 필터링할 수 있습니다.

소스 한정자 변환을 사용하여 필터링합니다.

소스 한정자 변환은 행을 필터링하는 다른 방법을 제공합니다. 소스 한정자 변환은 매핑 내에서 행을 필터링하는 것이 아니라 소스에서 읽을 때 행을 필터링합니다. 주요한 차이점은 소스 한정자가 소스에서 추출되는 행 집합을 제한하는 반면, 필터 변환은 대상으로 전송되는 행 집합을 제한한다는 것입니다. 소스 한정자가 매핑 전체에서 사용되는 행 수를 줄이기 때문에 더 나은 성능을 제공합니다.

하지만 소스 한정자 변환을 사용하면 관계형 소스의 행만 필터링할 수 있습니다. 이와 달리 필터 변환은 모든 유형의 소스에서 행을 필터링합니다. 또한 소스 한정자 변환은 데이터베이스에서 실행되므로 소스 한정자 변환의

필터 조건에서 표준 SQL만 사용하는지 확인해야 합니다. 필터 변환에서는 TRUE 또는 FALSE 값을 반환하는 모든 문 또는 변환 함수를 사용하여 조건을 정의할 수 있습니다.

제 10 장

HTTP 변환

이 장에 포함된 항목:

- [HTTP 변환 개요, 191](#)
- [HTTP 변환 작성, 192](#)
- [속성 탭 구성, 192](#)
- [HTTP 탭 구성, 193](#)
- [예, 199](#)

HTTP 변환 개요

HTTP 변환을 통해 HTTP 서버에 연결하여 해당 서비스 및 응용 프로그램을 사용할 수 있습니다. HTTP 변환은 수동 변환입니다. HTTP 변환을 사용하여 세션을 실행하는 경우, 통합 서비스는 HTTP 서버에 연결하고 사용자가 변환을 구성하는 방법에 따라 HTTP 서버에서 데이터를 검색하거나 업데이트하는 요청을 실행합니다.

- **HTTP 서버에서 데이터 읽기.** 통합 서비스가 HTTP 서버에서 데이터를 읽을 때, HTTP 서버에서 데이터를 검색하고 매핑의 다운스트림 변환 또는 대상에 데이터를 전달합니다. 예를 들어 HTTP 서버에 연결하여 현재 인벤토리 데이터를 읽고, PowerCenter 세션 중에 데이터에 대한 계산을 수행하고, 데이터를 대상에 전달할 수 있습니다.
- **HTTP 서버에서 데이터 업데이트.** 통합 서비스가 HTTP 서버에 기록할 때 HTTP 서버에 데이터를 게시하고 HTTP 서버 응답을 매핑의 다운스트림 변환 또는 대상에 전달합니다. 예를 들어 세션 중에 업스트림 변환에서 HTTP 서버로 예약 정보를 제공하는 데이터를 게시할 수 있습니다.

통합 서비스가 소스나 업스트림 변환의 데이터를 HTTP 변환으로 전달하고, HTTP 변환 또는 응용 프로그램 연결에서 구성된 URL을 읽고, HTTP 요청을 HTTP 서버로 보내 데이터를 읽거나 업데이트합니다.

요청에는 헤더 정보가 포함되어 있고 본문 정보가 있을 수 있습니다. 헤더에는 인증 매개 변수와 같은 정보, HTTP 서버에 상주하는 프로그램이나 웹 서비스를 활성화하기 위한 명령 및 전체 HTTP 요청에 적용되는 기타 정보 등이 포함되어 있습니다. 본문에는 통합 서비스가 HTTP 서버에 보내는 데이터가 포함되어 있습니다.

통합 서비스가 데이터를 읽기 위한 요청을 보내는 경우 HTTP 서버가 요청된 데이터와 함께 HTTP 응답을 다시 보냅니다. 통합 서비스가 요청된 데이터를 다운스트림 변환이나 대상으로 보냅니다.

통합 서비스가 데이터를 업데이트하기 위한 요청을 보내는 경우 HTTP 서버는 받은 데이터를 기록하고 업데이트가 성공했다는 HTTP 응답을 다시 보냅니다. HTTP 변환에서는 응답 코드 200, 201 및 202를 성공으로 간주합니다. HTTP 변환에서는 다른 모든 응답 코드를 실패로 간주합니다. HTTP 서버가 HTTP 변환에 대한 실패로 간주되는 응답 코드를 전달하면 세션 로그에서 오류를 표시합니다. 그런 다음 통합 서비스가 HTTP 응답을 다운스트림 변환이나 대상으로 보냅니다.

HTTP 응답의 헤더에 대한 HTTP 변환을 구성할 수 있습니다. HTTP 응답 본문 데이터가 HTTPOUT 출력 포트를 통해 전달됩니다.

인증

HTTP 변환에서는 다음 인증 형식을 사용합니다.

- **기본.** 암호화되지 않은 사용자 이름 및 암호를 기반으로 합니다.
- **다이제스트.** 암호화된 사용자 이름 및 암호를 기반으로 합니다.
- **NTLM.** 암호화된 사용자 이름, 암호 및 도메인을 기반으로 합니다.

HTTP 서버에 연결

HTTP 변환을 구성할 때 연결용 URL을 구성할 수 있습니다. 또한 워크플로우 관리자에서 HTTP 연결 개체를 작성할 수 있습니다. 다음 상황에서는 HTTP 응용 프로그램 연결을 구성하십시오.

- HTTP 서버에 인증이 필요합니다.
- 연결 제한 시간을 구성하고자 합니다.
- HTTP 변환에서 기본 URL을 재정의하고자 합니다.

HTTP 변환 작성

변환 개발자 또는 매핑 디자이너에서 HTTP 변환을 작성합니다. HTTP 변환에는 다음 탭이 있습니다.

- **변환.** 변환에 대한 이름 및 설명을 구성합니다.
- **포트.** 변환에 대한 입력 및 출력 포트를 봅니다. 포트 탭에서 포트를 추가하거나 편집할 수는 없습니다. HTTP 탭의 헤더 그룹에 포트를 추가하면 디자이너가 포트 탭에서 포트를 작성합니다.
- **속성.** 속성 탭에서 HTTP 변환에 대한 속성을 구성합니다.
- **HTTP.** HTTP 탭에서 메서드, 포트 및 URL을 구성합니다.

HTTP 변환을 작성하려면:

1. 변환 개발자 또는 매핑 디자이너에서 변환 > 작성을 클릭합니다.
2. HTTP 변환을 선택합니다.
3. 변환 이름을 입력합니다.
4. 작성을 클릭합니다.
HTTP 변환이 작업 공간에 표시됩니다.
5. 완료를 클릭합니다.
6. 변환의 탭을 구성합니다.

속성 탭 구성

HTTP 변환은 사용자 지정 변환을 사용하여 작성합니다. 일부 사용자 지정 변환 속성은 HTTP 변환에 적용되지 않거나 구성 가능하지 않습니다.

다음 테이블에는 구성할 수 있는 HTTP 변환 속성이 설명되어 있습니다.

옵션	설명
런타임 위치	DLL 또는 공유 라이브러리가 포함된 위치입니다. 기본값은 \$PMExtProcDir입니다. HTTP 변환 세션을 실행하는 통합 서비스 노드에 상대적인 경로를 입력합니다. 이 속성은 비어 있는 경우 통합 서비스가 DLL 또는 공유 라이브러리를 찾기 위해 통합 서비스에 정의된 환경 변수를 사용합니다. 런타임 위치 또는 통합 서비스 노드에 정의된 환경 변수에 모든 DLL이나 공유 라이브러리를 복사해야 합니다. 통합 서비스가 DLL, 공유 라이브러리 또는 참조된 파일을 찾을 수 없으면 해당 절차를 로드하지 못합니다.
추적 수준	이 변환에 대해 세션 로그에 표시되는 세부 정보의 양입니다. 기본값은 보통입니다.
분할 가능 여부	이 변환을 사용하는 파이프라인에서 여러 파티션을 작성할 수 있는지 여부를 나타냅니다. - 아니요. 변환을 분할할 수 없습니다. 변환과 동일한 파이프라인에 있는 다른 변환이 하나의 파티션으로 제한됩니다. - 로컬로. 변환을 분할할 수 있지만 통합 서비스가 같은 노드의 파이프라인에서 모든 파티션을 실행해야 합니다. 사용자 지정 변환의 여러 파티션이 메모리에서 개체를 공유해야 하는 경우 로컬로를 선택합니다. - 그리드에서. 변환을 분할할 수 있고 통합 서비스가 각 파티션을 여러 노드에 분산시킬 수 있습니다. 기본값은 아니요입니다.
출력은 반복 가능합니다.	출력 데이터의 순서가 세션 실행 간에 일치하는지 여부를 나타냅니다. - 사용 안 함 출력 데이터 순서는 세션 실행 간에 일관되지 않습니다. - 입력 순서 기반. 입력 데이터 순서가 세션 실행 간에 일관된 경우 입력 순서는 세션 실행 간에 일관됩니다. - 항상 입력 데이터 순서가 세션 실행 간에 일관되지 않더라도 출력 데이터 순서는 세션 실행 간에 일관됩니다. 기본값은 입력 순서 기반입니다.
파티션당 단일 스레드 필요	통합 서비스가 하나의 스레드를 포함하는 프로시저에서 각 파티션을 처리하는지 여부를 나타냅니다.
확정 출력입니다.	변환이 세션 실행 간에 일치하는 출력 데이터를 생성하는지 여부를 나타냅니다. 이 변환을 사용하는 세션에 대해 복구를 수행하려면 이 속성을 활성화합니다. 기본값은 활성화됩니다.

경고: 변환을 반복 가능 및 확정으로 구성하는 경우 데이터가 반복 가능 및 확정인지 확인하는 것은 사용자의 책임입니다. 세션과 복구 간에 동일한 데이터를 생성하지 않는 변환으로 세션을 복구하려는 경우 복구 프로세스로 인해 손상된 데이터가 발생할 수 있습니다.

HTTP 탭 구성

HTTP 탭에서는 HTTP 서버에서 데이터를 읽거나 HTTP 서버에 데이터를 쓰도록 변환을 구성할 수 있습니다. HTTP 탭에서 다음 정보를 구성합니다.

- **메서드 선택.** HTTP 서버에서 데이터를 읽거나 쓰지, 일부 업데이트를 수행하거나 전체 데이터 블록을 바꿀지 아니면 HTTP 서버에서 데이터를 삭제할지에 따라 GET, POST, SIMPLE POST, SIMPLE PATCH, SIMPLE PUT 또는 SIMPLE DELETE 메서드를 선택합니다.

- **그룹 및 포트 구성.** 입력 및 출력 포트를 구성하여 HTTP 요청/응답 본문 및 헤더 세부 정보를 관리합니다. 특수 문자로 포트 이름을 구성할 수도 있습니다.
- **기본 URL 구성.** 연결하려는 HTTP 서버의 기본 URL을 구성합니다.

메서드 선택

변환에서 정의하는 그룹 및 포트는 선택한 메서드에 따라 달라집니다. HTTP 서버에서 데이터를 읽으려면 GET 메서드를 선택합니다. HTTP 서버에 데이터를 기록하려면 POST 또는 SIMPLE POST 메서드를 선택합니다.

다음 테이블에는 여러 메서드가 설명되어 있습니다.

메서드	설명
GET	HTTP 서버에서 데이터를 읽습니다.
POST	여러 입력 포트에서 HTTP 서버로 데이터를 기록합니다.
SIMPLE POST	입력 포트 1개의 데이터를 단일 데이터 블록으로 HTTP 서버에 기록합니다.
SIMPLE PATCH	입력 포트 1개의 일부 데이터를 패치로 리소스에 업데이트합니다.
SIMPLE PUT	리소스를 바꾸거나 작성합니다.
SIMPLE DELETE	HTTP 서버에서 리소스를 삭제합니다.

그룹 및 포트 구성

HTTP 변환에 추가하는 포트는 선택하는 메서드 및 그룹에 따라 다릅니다. HTTP 변환은 다음 그룹을 사용합니다.

- **출력.** HTTP 응답에 대한 본문 데이터가 포함되어 있습니다. HTTP 서버의 응답을 다운로드 변환 또는 대상에 전달합니다. 기본적으로 하나의 출력 포트 HTTPOUT이 포함되어 있습니다. 출력 그룹에 포트를 추가할 수 없습니다. HTTPOUT 출력 포트의 전체 자릿수를 수정할 수 있습니다.
- **입력.** HTTP 요청에 대한 본문 데이터가 포함되어 있습니다. 또한 디자이너가 HTTP 서버에 연결하기 위해 최종 URL을 구성하는 데 사용하는 메타데이터가 포함되어 있습니다. HTTP 서버에 데이터를 쓰기 위해 입력 그룹이 본문 정보를 HTTP 서버에 전달합니다. 기본적으로 하나의 입력 포트가 포함되어 있습니다.
- **헤더.** 요청 및 응답에 대한 헤더 데이터가 포함되어 있습니다. HTTP 통합 서비스가 HTTP 요청을 보낼 때 HTTP 서버에 헤더 정보를 전달합니다. 헤더 그룹에 추가하는 포트가 HTTP 헤더에 대한 데이터를 전달합니다. 헤더 그룹에 포트를 추가하면 디자이너가 포트 탭의 입력 및 출력 그룹에 포트를 추가합니다. 기본적으로 포트가 포함되어 있지 않습니다. 모든 메서드에서 HTTP 요청 헤더 정보에 헤더 그룹을 사용할 수 있습니다.

참고: HTTP 변환을 통해 전달되는 데이터의 데이터 유형은 문자열이어야 합니다. 문자열 데이터에는 HTML 및 XML 등 HTTP 통신에서 일반적인 모든 마크업 언어가 포함됩니다.

GET 메서드

HTTP 서버에서 데이터를 읽습니다. HTTP 요청에 대한 메타데이터를 정의하려면 입력 그룹을 사용하여 디자이너에서 HTTP 서버에 대한 최종 URL을 구성할 때 사용하는 입력 포트를 추가합니다.

다음 테이블에는 GET 메서드의 그룹 및 포트가 설명되어 있습니다.

요청/ 응답	그룹	설명
REQUEST	입력	디자이너가 입력 포트의 이름과 값을 사용하여 최종 URL을 구성합니다.
REQUEST	머리글	HTTP 요청에 대한 입력 및 입력/출력 포트를 구성할 수 있습니다. 헤더 그룹에 추가하는 포트를 기반으로 디자이너가 입력 및 출력 그룹에 포트를 추가합니다. - 입력 그룹. 헤더 그룹의 입력 및 입력/출력 포트를 기반으로 입력 포트를 작성합니다. - 출력 그룹. 헤더 그룹의 입력/출력 포트를 기반으로 출력 포트를 작성합니다.
RESPONSE	머리글	HTTP 응답에 대한 출력 및 입력/출력 포트를 구성할 수 있습니다. 헤더 그룹에 추가하는 포트를 기반으로 디자이너가 입력 및 출력 그룹에 포트를 추가합니다. - 입력 그룹. 헤더 그룹의 입력/출력 포트를 기반으로 입력 포트를 작성합니다. - 출력 그룹. 헤더 그룹의 출력 및 입력/출력 포트를 기반으로 출력 포트를 작성합니다.
RESPONSE	출력	HTTP 응답의 모든 본문 데이터가 HTTPOUT 출력 포트를 통해 전달됩니다.

POST 메서드

여러 입력 포트에서 HTTP 서버로 데이터를 기록합니다. HTTP 요청에 대한 메타데이터를 정의하려면 HTTP 요청의 본문을 정의하는 데이터에 대한 입력 그룹을 사용합니다.

다음 테이블에는 POST 메서드의 포트가 설명되어 있습니다.

요청/ 응답	그룹	설명
REQUEST	입력	입력 그룹에 여러 포트를 추가할 수 있습니다. HTTP 요청의 본문 데이터는 헤더 그룹에 추가하는 항목에 따라 하나 이상의 입력 포트를 통해 통과될 수 있습니다.
REQUEST	머리글	HTTP 요청에 대한 입력 및 입력/출력 포트를 구성할 수 있습니다. 헤더 그룹에 추가하는 포트를 기반으로 디자이너가 입력 및 출력 그룹에 포트를 추가합니다. - 입력 그룹. 헤더 그룹의 입력 및 입력/출력 포트를 기반으로 입력 포트를 작성합니다. - 출력 그룹. 헤더 그룹의 입력/출력 포트를 기반으로 출력 포트를 작성합니다.
RESPONSE	머리글	HTTP 응답에 대한 출력 및 입력/출력 포트를 구성할 수 있습니다. 헤더 그룹에 추가하는 포트를 기반으로 디자이너가 입력 및 출력 그룹에 포트를 추가합니다. - 입력 그룹. 헤더 그룹의 입력/출력 포트를 기반으로 입력 포트를 작성합니다. - 출력 그룹. 헤더 그룹의 출력 및 입력/출력 포트를 기반으로 출력 포트를 작성합니다.
RESPONSE	출력	HTTP 응답의 모든 본문 데이터가 HTTPOUT 출력 포트를 통해 전달됩니다.

SIMPLE POST 메서드

POST 메서드의 단순화된 버전입니다. 입력 포트 1개의 데이터를 단일 데이터 블록으로 HTTP 서버에 기록합니다. HTTP 요청에 대한 메타데이터를 정의하려면 HTTP 요청의 본문을 정의하는 데이터에 대한 입력 그룹을 사용합니다.

다음 테이블에는 **SIMPLE POST** 메서드의 포트가 설명되어 있습니다.

요청/응답	그룹	설명
REQUEST	입력	하나의 입력 포트를 추가할 수 있습니다. HTTP 요청의 본문 데이터는 하나의 입력 포트를 통해 통과될 수 있습니다.
REQUEST	머리글	HTTP 요청에 대한 입력 및 입력/출력 포트를 구성할 수 있습니다. 헤더 그룹에 추가하는 포트를 기반으로 디자이너가 입력 및 출력 그룹에 포트를 추가합니다. - 입력 그룹. 헤더 그룹의 입력 및 입력/출력 포트를 기반으로 입력 포트를 작성합니다. - 출력 그룹. 헤더 그룹의 입력/출력 포트를 기반으로 출력 포트를 작성합니다.
RESPONSE	머리글	HTTP 응답에 대한 출력 및 입력/출력 포트를 구성할 수 있습니다. 헤더 그룹에 추가하는 포트를 기반으로 디자이너가 입력 및 출력 그룹에 포트를 추가합니다. - 입력 그룹. 헤더 그룹의 입력/출력 포트를 기반으로 입력 포트를 작성합니다. - 출력 그룹. 헤더 그룹의 출력 및 입력/출력 포트를 기반으로 출력 포트를 작성합니다.
RESPONSE	출력	HTTP 응답의 모든 본문 데이터가 HTTPOUT 출력 포트를 통해 전달됩니다.

SIMPLE PATCH 메서드

입력 포트 1개의 일부 데이터를 패치로 리소스에 업데이트합니다. 입력 포트 1개의 데이터를 전체 또는 일부 데이터 블록으로 **HTTP** 서버에 기록합니다. **HTTP** 요청에 대한 메타데이터를 정의하려면 **HTTP** 요청의 본문을 정의하는 데이터에 대한 입력 그룹을 사용합니다.

다음 테이블에는 **SIMPLE PATCH** 메서드의 포트가 설명되어 있습니다.

요청/응답	그룹	설명
REQUEST	입력	하나의 입력 포트를 추가할 수 있습니다. HTTP 요청의 본문 데이터는 하나의 입력 포트를 통해 통과될 수 있습니다.
REQUEST	머리글	HTTP 요청에 대한 입력 및 입력/출력 포트를 구성할 수 있습니다. 헤더 그룹에 추가하는 포트를 기반으로 디자이너가 입력 및 출력 그룹에 포트를 추가합니다. - 입력 그룹. 헤더 그룹의 입력 및 입력/출력 포트를 기반으로 입력 포트를 생성합니다. - 출력 그룹. 헤더 그룹의 입력/출력 포트를 기반으로 출력 포트를 생성합니다.
RESPONSE	머리글	HTTP 응답에 대한 출력 및 입력/출력 포트를 구성할 수 있습니다. 헤더 그룹에 추가하는 포트를 기반으로 디자이너가 입력 및 출력 그룹에 포트를 추가합니다. - 입력 그룹. 헤더 그룹의 입력/출력 포트를 기반으로 입력 포트를 생성합니다. - 출력 그룹. 헤더 그룹의 출력 및 입력/출력 포트를 기반으로 출력 포트를 생성합니다.
RESPONSE	출력	HTTP 응답의 모든 본문 데이터가 HTTPOUT 출력 포트를 통해 전달됩니다.

SIMPLE PUT 메서드

리소스를 바꾸거나 작성합니다. 입력 포트 1개의 데이터를 단일 데이터 블록으로 **HTTP** 서버에 기록합니다.

데이터가 존재하지 않는 경우에는 **SIMPLE PUT** 메서드가 데이터를 게시합니다. 데이터가 있는 경우 **SIMPLE PUT** 메서드는 입력 포트 1개의 데이터를 단일 데이터 블록으로 **HTTP** 서버에 업데이트합니다.

HTTP 요청에 대한 메타데이터를 정의하려면 **HTTP** 요청의 본문을 정의하는 데이터에 대한 입력 그룹을 사용합니다.

다음 테이블에는 **SIMPLE PUT** 메서드의 포트가 설명되어 있습니다.

요청/응답	그룹	설명
REQUEST	입력	하나의 입력 포트를 추가할 수 있습니다. HTTP 요청의 본문 데이터는 하나의 입력 포트를 통해 통과될 수 있습니다.
REQUEST	머리글	HTTP 요청에 대한 입력 및 입력/출력 포트를 구성할 수 있습니다. 헤더 그룹에 추가하는 포트를 기반으로 디자이너가 입력 및 출력 그룹에 포트를 추가합니다. <ul style="list-style-type: none"> - 입력 그룹. 헤더 그룹의 입력 및 입력/출력 포트를 기반으로 입력 포트를 생성합니다. - 출력 그룹. 헤더 그룹의 입력/출력 포트를 기반으로 출력 포트를 생성합니다.
RESPONSE	머리글	HTTP 응답에 대한 출력 및 입력/출력 포트를 구성할 수 있습니다. 헤더 그룹에 추가하는 포트를 기반으로 디자이너가 입력 및 출력 그룹에 포트를 추가합니다. <ul style="list-style-type: none"> - 입력 그룹. 헤더 그룹의 입력/출력 포트를 기반으로 입력 포트를 생성합니다. - 출력 그룹. 헤더 그룹의 출력 및 입력/출력 포트를 기반으로 출력 포트를 생성합니다.
RESPONSE	출력	HTTP 응답의 모든 본문 데이터가 HTTPOUT 출력 포트를 통해 전달됩니다.

SIMPLE DELETE 메서드

HTTP 서버에서 리소스를 삭제합니다. 요청 본문이 필요한 경우 **SIMPLE DELETE** 메서드는 입력 포트 1개의 데이터를 단일 데이터 블록으로 HTTP 서버에서 삭제합니다. HTTP 요청에 대한 메타데이터를 정의하려면 입력 그룹을 사용하여 디자이너에서 HTTP 서버에 대한 최종 URL을 구성할 때 사용하는 입력 포트를 추가합니다.

다음 테이블에는 **SIMPLE DELETE** 메서드의 포트가 설명되어 있습니다.

요청/응답	그룹	설명
REQUEST	입력	디자이너가 입력 포트의 이름과 값을 사용하여 최종 URL을 구성합니다.
REQUEST	머리글	HTTP 요청에 대한 입력 및 입력/출력 포트를 구성할 수 있습니다. 헤더 그룹에 추가하는 포트를 기반으로 디자이너가 입력 및 출력 그룹에 포트를 추가합니다. <ul style="list-style-type: none"> - 입력 그룹. 헤더 그룹의 입력 및 입력/출력 포트를 기반으로 입력 포트를 생성합니다. - 출력 그룹. 헤더 그룹의 입력/출력 포트를 기반으로 출력 포트를 생성합니다.
RESPONSE	머리글	HTTP 응답에 대한 출력 및 입력/출력 포트를 구성할 수 있습니다. 헤더 그룹에 추가하는 포트를 기반으로 디자이너가 입력 및 출력 그룹에 포트를 추가합니다. <ul style="list-style-type: none"> - 입력 그룹. 헤더 그룹의 입력/출력 포트를 기반으로 입력 포트를 생성합니다. - 출력 그룹. 헤더 그룹의 출력 및 입력/출력 포트를 기반으로 출력 포트를 생성합니다.
RESPONSE	출력	HTTP 응답의 모든 본문 데이터가 HTTPOUT 출력 포트를 통해 전달됩니다.

HTTP 이름 추가

디자이너에서는 포트 이름에 대시(-)와 같은 특수 문자를 허용하지 않습니다. 포트 이름에 특수 문자를 사용해야 하는 경우 HTTP 이름을 구성하여 포트 이름을 재정의할 수 있습니다. 예를 들어 **Content-type**이라고 명명된 입력 포트를 원하는 경우 포트에 **ContentType**으로 이름을 지정하고 HTTP 이름으로 **Content-Type**을 입력할 수 있습니다.

URL 구성

메서드를 선택하고 입력 및 출력 포트를 구성한 후 URL을 구성해야 합니다. 기본 URL을 입력합니다. 그러면 디자이너가 최종 URL을 구성합니다. GET 또는 **SIMPLE DELETE** 메서드를 선택하는 경우 최종 URL에는 입력 그룹

의 포트 이름에 기반한 매개 변수 및 기본 URL이 포함됩니다. SIMPLE PATCH, SIMPLE PUT, POST 또는 SIMPLE POST 메서드를 선택하는 경우 최종 URL은 기본 URL과 같습니다.

매핑 매개 변수나 변수를 사용하여 기본 URL을 구성할 수 있습니다. 예를 들어 매핑 매개 변수 \$ParamBaseURL을 선언하고, 기본 URL 필드에 매핑 매개 변수 \$\$ParamBaseURL을 입력한 후 매개 변수 파일에서 \$\$ParamBaseURL을 정의합니다.

또한 HTTP 응용 프로그램 연결을 구성할 때 URL을 지정할 수 있습니다. HTTP 응용 프로그램 연결에 지정된 기본 URL은 HTTP 변환에 지정된 기본 URL을 재정의합니다.

참고: HTTP 서버는 HTTP 요청을 다른 HTTP 서버로 리디렉션할 수 있습니다. 이렇게 되면 HTTP 서버는 통합 서비스로 URL을 다시 보냅니다. 그런 다음 통합 서비스가 다른 HTTP 서버에 대한 연결을 설정합니다. 통합 서비스는 추가로 최대 5개의 연결을 설정할 수 있습니다.

GET 메서드의 최종 URL 구성

디자이너가 입력 그룹의 기본 URL 및 포트 이름을 기반으로 GET 메서드의 최종 URL을 구성합니다. HTTP 인수를 기본 URL에 추가하여 HTTP 쿼리 문자열 형식으로 최종 URL을 구성합니다. 쿼리 문자열은 물음표(?) 다음에 이름/값 쌍으로 구성됩니다. 디자이너는 물음표 및 이름에 해당되는 이름/값 쌍과 사용자가 입력 그룹에 추가하는 입력 포트 값을 추가합니다.

GET 메서드를 선택하고 입력 그룹에 입력 포트를 추가하면 디자이너가 다음 그룹과 포트 정보를 기본 URL에 추가하여 최종 URL을 구성합니다.

?<input group input port 1 name> = \$<input group input port 1 value>

첫 번째 입력 그룹 입력 포트 다음에 나오는 각 입력 포트에 대해 디자이너가 다음 그룹 및 포트 정보를 추가합니다.

& <input group input port n name> = \$<input group input port n value>

여기서 *n*은 입력 포트를 나타냅니다.

예를 들어 기본 URL로 `www.company.com`을 입력하고 입력 포트 ID, EmpName 및 부서를 입력 그룹에 추가하는 경우 디자이너가 다음과 같은 최종 URL을 구성합니다.

`www.company.com?ID=$ID&EmpName=$EmpName&Department=$Department`

최종 URL을 편집하여 연산자, 변수 또는 기타 인수를 수정하거나 추가할 수 있습니다. HTTP 요청 및 쿼리 문자열에 대한 자세한 내용은 <http://www.w3c.org>를 참조하십시오.

기본 URL 매개 변수화

GET 및 SIMPLE DELETE 메서드에 대한 기본 URL을 매개 변수화할 수 있습니다.

예를 들어 기본 URL이 다음과 같다고 가정합니다.

`www.informatica.com`

디자이너는 이 기본 URL의 정보를 사용하여 다음과 같은 최종 URL을 구성할 수 있습니다.

`www.informatica.com?firstname=1&lastname=2`

디자이너에서 GET 또는 SIMPLE DELETE 메서드에 대한 기본 URL을 매개 변수화하는 확인란 옵션을 선택하는 경우 다음과 같이 매핑 매개 변수 또는 변수를 사용하도록 기본 URL을 편집할 수 있습니다.

`www.informatica.com/$firstname$lastname`

통합 서비스는 HTTP 변환의 이름 및 성 입력 포트에 소스 파일 값을 보내고 최종 URL에 지정된 HTTP 서버로 HTTP 요청을 보냅니다.

최종 URL은 다음과 같이 나타낼 수 있습니다.

`www.informatica.com/12`

URL의 특수 문자

URL을 구성하는 경우 최종 URL에 UTF-8 문자 집합으로 인코딩된 특수 문자가 포함될 수 있습니다.

다음 테이블에는 URL에 사용된 일부 특수 문자가 UTF-8 환경에서 구성되는 방법이 설명되어 있습니다.

특수 문자	인코딩된 형식(UTF-8)
'	%27
(%28
)	%29
*	%2A
공백	%20

통합 서비스 수준 또는 세션 수준에서 `UseURLAsEnteredinCURLEncoding` 사용자 지정 플래그를 사용하는 경우 기본 URL(CURL 인코딩 형식)에 입력된 URL을 정확히 전달할 수 있습니다.

예

이 섹션에는 각 메서드 유형의 예제가 포함되어 있습니다.

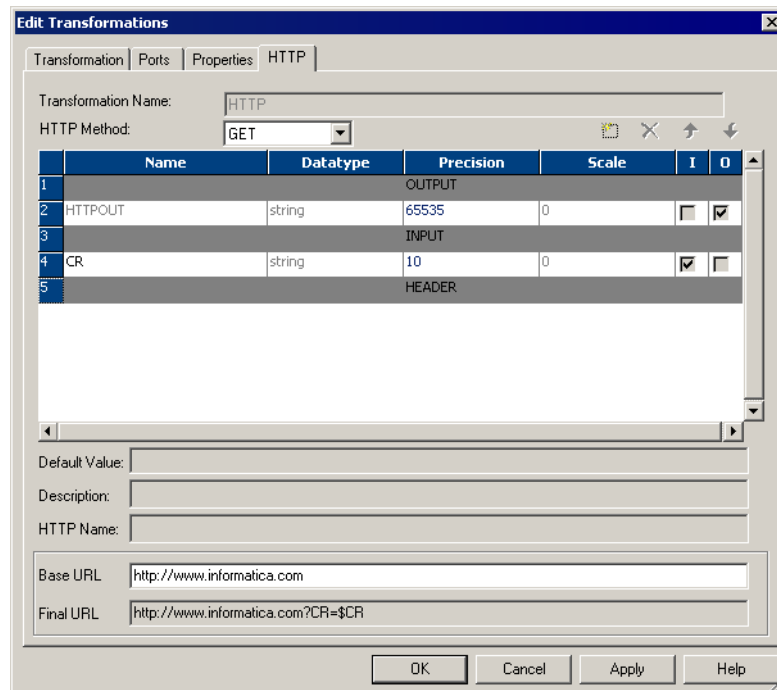
- GET
- POST
- SIMPLE POST
- SIMPLE PATCH
- SIMPLE PUT
- SIMPLE DELETE

GET 예제

이 예제에서 사용되는 소스 파일에는 다음 데이터가 포함되어 있습니다.

```
78576
78577
78578
```

다음 그림에는 GET 예제에 대한 HTTP 변환의 HTTP 탭이 나와 있습니다.



디자인너는 물음표(?), 입력 그룹 입력 포트 이름, 달러 기호(\$) 및 입력 그룹 입력 포트 이름을 기본 URL에 다시 추가하여 최종 URL을 구성합니다

`http://www.informatica.com?CR=$CR`

통합 서비스는 HTTP 변환의 CR 입력 포트에 소스 파일 값을 보내고 다음 HTTP 요청을 HTTP 서버에 보냅니다.

`http://www.informatica.com?CR=78576`
`http://www.informatica.com?CR=78577`
`http://www.informatica.com?CR=78578`

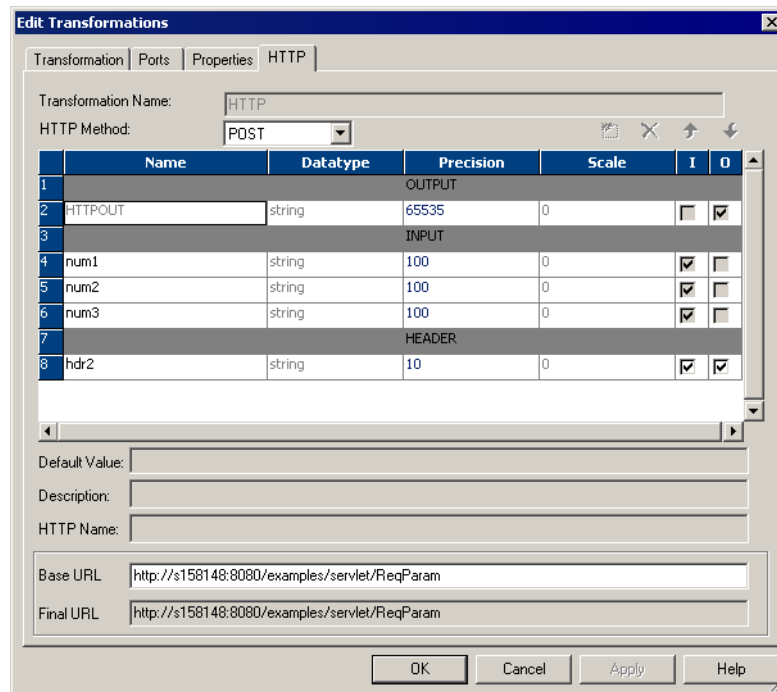
HTTP 서버는 HTTP 응답을 통합 서비스에 다시 보내고, 통합 서비스는 HTTP 변환의 출력 포트를 통해 대상에 데이터를 보냅니다.

POST 예제

이 예제에서 사용되는 소스 파일에는 다음 데이터가 포함되어 있습니다.

33,44,1
 44,55,2
 100,66,0

다음 그림은 소스 파일의 각 필드가 상응하는 입력 포트를 갖고 있음을 보여 줍니다.



통합 서비스는 HTTP 변환의 입력 포트를 통해 각 행의 3개 필드 값을 보내고 최종 URL에 지정된 HTTP 서버로 HTTP 요청을 보냅니다.

SIMPLE POST 예제

다음 텍스트는 이 예제에 사용되는 XML 파일을 보여 줍니다.

```
<?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:smplsource>
    <Metadatainfo xsi:type="xsd:string">smplsourceRequest.Metadatainfo106</Metadatainfo></cli:smplsource>
  </n4:Body>
</n4:Envelope>,capeconnect:Clienttest1services:Clienttest1#smplsource
```

다음 그림은 SIMPLE POST 예제의 HTTP 변환에 있는 HTTP 탭을 보여 줍니다.

Edit Transformations

Transformation Name: HTTP1
 HTTP Method: SIMPLE POST

	Name	Datatype	Precision	Scale	I	O
1	OUTPUT					
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 type="checkbox"/>
8	Date	string	100	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>
9	server	string	100	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Default Value:
 Description:
 HTTP Name:
 Base URL: http://scorpio:8080/axis/services/Clienttest1
 Final URL: http://scorpio:8080/axis/services/Clienttest1

OK Cancel Apply Help

통합 서비스는 입력 포트를 통해 소스 파일의 본문을 보내고, 최종 URL에 지정된 HTTP 서버로 HTTP 요청을 보냅니다.

SIMPLE PATCH 예

이 예제에서 사용되는 소스 파일에는 다음 데이터가 포함되어 있습니다.

```
{"firstname":"Raj"}
```

다음 그림은 소스 파일의 각 필드가 상응하는 입력 포트를 갖고 있음을 보여 줍니다.

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

\$\$BASEURL1 매핑 변수를 기본 URL 및 최종 URL로 포함합니다.

\$\$BASEURL1 매핑 변수는 다음 링크를 가리킵니다.

http://avengers5:8181/emp_all/1/

SIMPLE PATCH를 사용하는 경우 입력 포트 1개의 일부 데이터를 HTTP 서버에 업데이트할 수 있습니다. 통합 서비스는 HTTP 변환의 입력 포트를 통해 영향을 받는 행의 값을 보내고 최종 URL에 지정된 HTTP 서버로 HTTP 요청을 보냅니다.

SIMPLE PUT 예

이 예제에서 사용되는 소스 파일에는 다음 데이터가 포함되어 있습니다.

```
{"emp_id": 10, "firstname": "Raj", "lastname": "Kumar", "designation": "QA", "department": "RND", "age": 24}
```

다음 그림은 소스 파일의 각 필드가 상응하는 입력 포트를 갖고 있음을 보여 줍니다.

Edit Transformations

Transformation Ports Properties **HTTP**

Static

Transformation Name: HTTP

HTTP Method: SIMPLE PUT

	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 메서드는 입력 포트 1개의 데이터를 단일 데이터 블록으로 HTTP 서버에 기록합니다. 통합 서비스는 HTTP 변환의 입력 포트를 통해 소스 파일의 본문을 보내고 최종 URL에 지정된 HTTP 서버로 HTTP 요청을 보냅니다.

SIMPLE DELETE 예

이 예제에서 사용되는 소스 파일에는 다음 데이터가 포함되어 있습니다.

```
2
1
```


다음 그림에는 SIMPLE DELETE 예제에서 기본 URL 매개 변수화 옵션이 선택된 HTTP 변환의 HTTP 탭이 나와 있습니다.

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

기본 URL 매개 변수화 확인란을 선택하면 디자이너가 입력 그룹 입력 포트 이름을 기본 URL에 추가하여 최종 URL을 구성합니다.

`http://www.avengers5:8181/emp_all/$id/`

통합 서비스는 HTTP 변환의 ID 입력 포트에 소스 파일 값을 보내고 다음 HTTP 요청을 HTTP 서버에 보냅니다.

`http://www.avengers5:8181/emp_all/2`

`http://www.avengers5:8181/emp_all/1`

SIMPLE DELETE 메서드는 HTTP 서버에서 필요한 데이터를 삭제합니다. HTTP 서버는 HTTP 응답을 통합 서비스에 다시 보내고, 통합 서비스는 HTTP 변환의 출력 포트를 통해 대상에 업데이트된 데이터를 보냅니다.

제 11 장

ID 확인 변환

이 장에 포함된 항목:

- [ID 확인 변환 개요, 206](#)
- [변환 작성 및 구성, 206](#)
- [ID 확인 변환 탭, 209](#)
- [그룹 및 포트, 209](#)

ID 확인 변환 개요

ID 확인 변환은 IIR(Informatica Identity Resolution)에서 데이터를 검색하고 일치시키는 데 사용할 수 있는 활성 변환입니다. PowerCenter 통합 서비스는 사용자가 ID 확인 변환에서 지정하는 검색 정의를 사용하여 IIR 테이블에 있는 데이터를 검색하고 일치시킵니다. 시스템의 입력 및 출력 보기가 변환의 입력 및 출력 포트를 결정합니다.

ID 확인 변환에서 일치 허용한도를 구성하고 너비 매개 변수를 검색하여 일치하는 구성표 및 검색 수준을 설정합니다. 세션을 실행하면 IIR이 ID 확인 변환에 후보 레코드를 반환합니다.

IIR은 Oracle, DB2/UDB 및 Microsoft SQL Server 테이블에 저장된 모든 유형의 ID 데이터에 대한 온라인 및 일괄 검색, 일치, 스크리닝, 연결, 중복 검색을 제공합니다. IIR은 IIR 검색 서버를 사용하여 작업을 검색하고 일치시킵니다.

변환 작성 및 구성

변환 개발자에서 ID 확인 변환을 작성합니다. ID 확인 변환을 작성하는 경우 먼저 Informatica Identity Resolution 검색 서버에 연결합니다. 이 검색 서버는 IIR 테이블에 대한 액세스를 제공합니다.

연결한 후에 시스템과 검색 및 일치 매개 변수를 구성합니다. **IR 변환 구성** 창에서 다음 정보를 구성합니다.

- IR 검색 서버 연결
- 시스템 및 검색 구성
- 입력 및 출력 보기

검색 서버 연결

검색 서버에 연결하려면 호스트 이름 및 포트 번호를 제공하십시오. 그런 다음, **Rulebase** 연결 문자열을 구성합니다. **Rulebase**는 데이터 검색 및 일치를 위해 **IIR**에서 사용하는 시스템에 대한 정보를 포함합니다. ID 확인 변환에서 **Rulebase** 연결은 **Rulebase**를 포함하는 데이터베이스를 지정합니다.

참고: 변환을 작성한 후에 호스트 이름 또는 **Rulebase** 연결을 변경할 수 없습니다.

IR 호스트 이름

Informatica Identity Resolution 서버가 실행되는 시스템의 호스트 이름입니다.

포트

검색 서버의 포트 번호입니다.

Rulebase 연결 문자열

검색 서버에 대한 연결 문자열입니다. **Rulebase** 연결을 **ODBC** 또는 **SSA**로 지정할 수 있습니다. **IIR** 호스트에서 **ODBC** 및 **SSA**를 구성하십시오.

ODBC

다음 테이블에는 **ODBC** 연결 정보가 설명되어 있습니다.

연결 속성	설명
Rulebase 번호	IR Rulebase 에 대해 지정된 번호입니다.
사용자 이름	데이터베이스 사용자 이름입니다.
암호	데이터베이스 사용자 이름에 대한 암호입니다.
서비스 이름	odbc.ini 파일에 정의된 서비스 이름입니다.

SSA

SSA를 통해 연결하려면 별칭을 제공해야 합니다. 별칭은 응용 프로그램으로부터 연결 문자열을 숨깁니다. **IIR**은 **Rulebase**, 데이터베이스 및 소스 이름에 대해 별칭을 사용합니다.

시스템 및 검색 구성

검색 서버에 연결한 후, 시스템과 검색 및 일치 매개 변수를 선택할 수 있습니다.

이러한 매개 변수는 변환을 작성한 후에 **IR 설정** 탭에서 변경할 수 있습니다.

시스템

Rulebase에서 시스템을 선택합니다. 시스템은 **IIR Rulebase**에서 가장 높은 수준의 논리적 정의입니다.

검색 정의

시스템에 대한 규칙을 포함하는 검색 정의를 선택합니다. 검색 정의는 시스템에 있으며 레코드 검색, 일치 및 표시를 위한 규칙을 포함합니다.

검색 너비

수행하려는 검색의 범위를 결정하는 검색 너비를 선택합니다.

다음 테이블에는 선택할 수 있는 검색 너비가 설명되어 있습니다.

검색 너비	설명
좁음	응답 시간이 짧은 좁은 검색을 제공합니다. 일치 항목을 찾지 못할 경우의 위험이 낮거나 높은 일치 정확도가 필요하지 않거나 데이터 양이 많고 응답 시간이 중요하면 이 옵션을 사용합니다.
표준 설치	품질과 응답 시간을 균형 있게 제공합니다. 온라인 또는 일괄 트랜잭션 검색에 이 옵션을 사용합니다. 기본값은 표준입니다.
포괄적	응답 시간이 긴 상세한 검색을 제공합니다. 일치 항목을 찾지 못할 경우의 위험이 높거나 데이터 품질에 관심이 있거나 데이터 양이 적고 응답 시간이 중요하지 않으면 이 옵션을 사용합니다.

일치 허용 한도

후보 레코드 선택과 관련하여 일치 구성표가 얼마나 적극적인지 결정하는 일치 허용 한도를 선택합니다.

다음 테이블에는 선택할 수 있는 일치 허용 한도 옵션이 설명되어 있습니다.

일치 허용 한도	설명
최대	후보 일치 항목을 일련의 가능한 처리 방법과 함께 제공합니다. 처리 응답 시간은 길어집니다. 오류 및 변형을 포함할 경우에도 일치 항목을 찾아야 하는 경우 이 옵션을 사용합니다.
보수적	지나치거나 과도하지 않은 일치 항목을 제공합니다. 온라인 또는 일괄 트랜잭션 검색에 사용합니다.
표준 설치	근접한 일치 항목을 제공합니다. 정확한 일치 항목이 필요할 때 일괄 시스템에서 사용합니다.
느슨함	표준 수준보다 변형 정도가 높은 일치 항목을 제공합니다. 일치 항목을 찾지 못할 경우의 위험이 높은 시스템에서 결과에 대한 검토가 가능할 때 사용합니다.

보기 선택

시스템 및 검색 정의를 선택한 후에 입력 보기 및 출력 보기를 선택할 수 있습니다.

입력 보기

입력 보기에서 필드는 변환 입력 포트를 결정합니다. 입력 보기에는 검색 및 일치 작업에 사용되는 필드가 포함됩니다. 입력 보기를 선택하지 않은 경우, 변환은 IIR ID 테이블(IDT) 레이아웃을 입력 보기에 사용합니다.

참고: 시스템에서 여러 필드를 하나의 필드로 결합하기 위해 XFORM 절이 보기 정의에 포함되어 있으면 ID 확인 변환에서 해당 보기를 사용할 수 없습니다. 그러나 ID 확인 변환 외부에서 여러 필드를 결합하고 연결된 문자열을 ID 확인 변환의 입력 포트로 전달할 수는 있습니다.

출력 보기

출력 보기에서 필드는 변환 출력 포트를 결정합니다. 출력 보기는 검색 서버에 의해 반환된 검색 결과에 형식을 적용합니다. 출력 보기를 선택하지 않은 경우, 변환은 IIR ID 테이블(IDT) 레이아웃을 출력 보기에 사용합니다.

ID 확인 변환 탭

ID 확인 변환에는 다음과 같은 탭이 있습니다.

변환

변환 설명을 구성합니다.

포트

입력 및 출력 보기를 나타내는 그룹 및 포트를 봅니다. 그룹 및 포트는 편집할 수 없습니다.

속성

다음 테이블에는 속성 탭에서 구성할 수 있는 속성이 설명되어 있습니다.

변환 특성	설명
런타임 위치	DLL 또는 공유 라이브러리가 포함된 위치입니다. 기본값은 \$PMExtProcDir입니다.
추적 수준	이 변환에 대해 세션 로그에 표시되는 세부 정보의 양입니다. 기본값은 보통입니다.
분할 가능 여부	이 변환을 사용하는 파이프라인에서 여러 파티션을 작성할 수 있는지 여부를 나타냅니다. <ul style="list-style-type: none">- 아니요. 변환을 분할할 수 없습니다. 해당 변환 및 동일한 파이프라인의 기타 변환은 하나의 파티션으로 제한됩니다.- 로컬로. 변환을 분할할 수 있지만 통합 서비스가 같은 노드의 파이프라인에서 모든 파티션을 실행해야 합니다. 사용자 지정 변환의 여러 파티션이 메모리에서 개체를 공유해야 하는 경우 로컬로 선택합니다.- 그리드에서. 변환을 분할할 수 있고 통합 서비스가 각 파티션을 여러 노드에 분산시킬 수 있습니다. 기본값은 그리드에서입니다.

IR 설정

변환을 작성할 때 구성된 설정을 봅니다. 시스템 및 검색 정보를 변경할 수 있습니다.

그룹 및 포트

ID 확인 변환에는 입력 그룹과 출력 그룹이 포함됩니다. 입력 그룹에는 검색 정의의 입력 보기에서 필드를 나타내는 포트가 있습니다. 출력 그룹에는 검색 정의의 출력 보기에서 필드를 나타내는 포트뿐만 아니라 검색 결과를 설명하는 포트가 있습니다.

포트 탭에서 그룹 및 포트를 볼 수 있습니다. 또한 **IR 설정** 탭에서 포트를 볼 수 있습니다.

입력 그룹 및 포트

ID 확인 변환에는 입력 포트와 함께 입력 그룹이 포함됩니다. 입력 그룹의 포트 수는 선택된 입력 보기의 필드 수와 일치합니다. 검색에 필요한 모든 입력 포트를 ID 확인 변환에 연결합니다.

출력 그룹 및 포트

출력 그룹은 여러 개의 출력 포트 및 1개의 입력/출력 포트를 포함합니다. 출력 그룹은 출력 보기에 있는 포트를 포함하고, 각 ID 확인 변환을 위한 기본 포트 또한 포함합니다.

출력 보기를 기반으로 하는 출력 포트는 검색 서버에서 반환된 후보 레코드를 포함합니다.

다음 테이블에는 기본 출력 포트가 설명되어 있습니다.

기본 출력 포트	설명
IR_Search_Link	입력/출력 검색 링크 포트. 입력 측의 소스 데이터와 결과 출력 후보 간의 링크를 전달하려면 이 통과 포트를 사용합니다.
IR_Score_Out	검색 서버에 의해 수행된 검색 작업에서 얻은 점수를 포함하는 출력 포트입니다. 값은 0부터 100까지의 양수입니다. 예를 들어 "Rob"을 입력하여 이름으로 검색하는 경우, 이름 "Rob"을 포함하는 100점짜리 후보 레코드가 결과에 포함될 수 있습니다. 또한 이름 "Bob"을 포함하는 90점짜리 후보 레코드도 결과에 포함될 수 있습니다. Informatica ID 확인은 결과와 관련하여 점수를 계산합니다.
IR_Result_Count	레코드 수를 포함하는 출력 포트입니다.

제 12 장

Java 변환

이 장에 포함된 항목:

- [Java 변환 개요, 211](#)
- [Java 코드 탭 사용, 213](#)
- [포트 구성, 214](#)
- [Java 변환 속성 구성, 215](#)
- [Java 코드 개발, 218](#)
- [Java 변환 설정 구성, 221](#)
- [Java 변환 컴파일, 223](#)
- [컴파일 오류 수정, 224](#)

Java 변환 개요

Java 변환을 사용하여 PowerCenter 기능을 확장합니다. Java 변환은 Java 프로그래밍 언어가 포함된 변환 기능을 정의할 수 있는 단순한 원시 프로그래밍 인터페이스를 제공합니다. Java 변환을 사용하면 Java 프로그래밍 언어 또는 외부 Java 개발 환경에 대한 고급 지식이 없어도 간단한 변환을 빠르게 정의하거나 복잡한 변환을 적당히 정의할 수 있습니다. Java 변환은 수동 또는 활성 변환일 수 있습니다.

PowerCenter 클라이언트는 JDK(Java 개발 키트)를 사용하여 Java 코드를 컴파일하고 변환에 대한 바이트 코드를 생성합니다. PowerCenter 클라이언트는 바이트 코드를 PowerCenter 리포지토리에 저장합니다.

통합 서비스는 JRE(Java Runtime Environment)를 사용하여 생성된 바이트 코드를 런타임에 실행합니다. 통합 서비스는 Java 변환을 사용하는 세션을 실행할 때 JRE를 사용하여 바이트 코드를 실행하고 입력 행을 처리하고 출력 행을 생성합니다.

Java 변환을 작성하려면 변환 논리를 정의하는 Java 코드 조각을 작성해야 합니다. Java 변환에 대한 변환 동작은 다음과 같은 이벤트를 바탕으로 정의합니다.

- 변환이 입력 행을 수신합니다.
- 변환이 모든 입력 행을 처리했습니다.
- 변환이 커밋 또는 롤백 같은 트랜잭션 알림을 수신합니다.

관련 항목:

- [“Java 변환 API 참조” 페이지 226](#)
- [“Java 식” 페이지 235](#)

Java 변환 정의 단계

다음 단계를 완료하여 Java 변환에서 Java 코드를 작성 및 컴파일하고 컴파일 오류를 수정합니다.

1. 변환 개발자 또는 매핑 디자이너에서 변환을 작성합니다.
2. 변환의 입력 및 출력 포트와 그룹을 구성합니다. 포트 이름을 Java 코드 조각의 변수로 사용합니다.
3. 변환 속성을 구성합니다.
4. 변환의 코드 항목 탭을 사용하여 변환에 대한 Java 코드를 작성하고 컴파일합니다.
5. 변환에 대한 Java 코드에서 컴파일 오류를 찾아 수정합니다.

활성 및 수동 Java 변환

Java 변환은 변환이 활성인지 아니면 수동인지에 따라 출력 행을 다르게 생성합니다.

변환을 생성한 후에는 변환이 활성인지 수동인지 여부를 변경할 수 없습니다.

활성 Java 변환

활성 변환에서는 변환을 통과하는 행 수를 변경할 수 있습니다.

출력의 행 수를 정의하려면 코드에서 `generateRow()` 메서드를 호출하여 각 출력 행을 생성합니다. 단일 입력 행에서 여러 출력 행을 생성하거나 여러 입력 행에서 단일 출력 행을 생성하도록 선택할 수 있습니다. 예를 들어 변환에 시작 날짜 및 종료 날짜를 나타내는 입력 포트 2개가 포함되는 경우 `generateRow()` 메서드를 호출하여 시작 날짜와 종료 날짜 사이의 각 날짜에 대한 출력 행을 생성할 수 있습니다.

수동 Java 변환

수동 변환에서는 변환을 통과하는 행 수를 변경할 수 없습니다. 변환은 각 입력 행을 처리한 후 `generateRow()` 메서드를 호출하여 출력 행을 생성합니다.

데이터 유형 변환

Java 변환은 PowerCenter 데이터 유형을 Java 변환의 포트 유형에 따라 Java 데이터 유형으로 변환합니다.

Java 변환은 입력 행을 읽고 입력 포트 데이터 유형을 Java 데이터 유형으로 변환합니다.

Java 변환이 출력 행을 기록할 때는 Java 데이터 유형을 출력 포트의 데이터 유형으로 변환합니다.

예를 들어 Java 변환에서 Integer 데이터 유형을 포함하는 입력 포트는 다음과 같이 처리됩니다.

1. Java 변환이 입력 포트의 Integer 데이터 유형을 Java 기본 Integer 데이터 유형으로 변환합니다.
2. Java 변환에서 Java 변환이 입력 포트의 값을 Java 기본 Integer 데이터 유형으로 처리합니다.
3. Java 변환이 Java 기본 Integer 데이터 유형을 Integer 데이터 유형으로 변환하여 출력 행을 생성합니다.

다음 테이블에는 Java 변환이 PowerCenter 데이터 유형을 Java 기본 및 복합 데이터 유형으로 매핑하는 방식이 나와 있습니다.

PowerCenter 데이터 유형	Java 데이터 유형
문자	문자열
Binary	byte[]
Long(INT32)	int
Double	double

PowerCenter 데이터 유형	Java 데이터 유형
Decimal	double BigDecimal
BIGINT	long
Date/Time	BigDecimal long(1970년 1월 1일 00:00:00.000 GMT 이후의 시간(밀리초))

Java 변환은 기본 데이터 유형의 Null 값을 0으로 설정합니다. **입력 행에서** 탭에서 isNull 및 setNull API 메서드를 사용하여 입력 포트의 Null 값을 출력 포트의 Null 값으로 설정할 수 있습니다. 예는 [“setNull” 페이지 233](#)에서 확인하십시오.

참고: 많은 전체 자릿수가 활성화된 경우 10진수 데이터 유형은 BigDecimal로 매핑됩니다. BigDecimal은 일부 연산자(예: + 연산자)와 함께 사용할 수 없습니다. Java 코드에 10진수 포트를 사용하는 식이 포함되어 있고 포트가 이러한 연산자 중 하나와 함께 사용되면 Java 코드가 컴파일되지 않습니다.

Java 코드 탭 사용

Java 코드 탭을 사용하여 Java 코드를 정의 및 컴파일하고 컴파일 오류를 수정합니다. 코드 조각은 코드 항목 탭에서 작성합니다.

java 코드를 정의할 때 다음 태스크를 수행할 수 있습니다.

- 정적 코드 또는 정적 블록, 인스턴스 변수 및 사용자 정의 메서드를 정의합니다.
- Java 식을 정의하고 변환 논리를 정의합니다.
- Java 변환 API 메서드 및 표준 Java 언어 구조체를 사용합니다.
- 타사 Java API, 기본 제공 Java 패키지 또는 사용자 지정 Java 패키지를 가져옵니다. Java 패키지에 있는 Java 코드 조각을 사용할 수 있습니다.

코드 조각을 개발한 후에 Java 코드를 컴파일하고 출력 창에서 컴파일 결과를 보거나 전체 Java 코드를 볼 수 있습니다.

Java 코드 탭에는 다음 구성 요소가 포함되어 있습니다.

- **Navigator.** 입력 또는 출력 포트나 API를 코드 조각에 추가합니다. 탐색기에는 변환의 입력 및 출력 포트, 사용 가능한 Java 변환 API 및 포트 또는 API 함수에 대한 설명이 나열됩니다. 입력 및 출력 포트에 대한 설명에는 포트 이름, 유형, 데이터 유형, 전체 자릿수 및 소수 자릿수가 포함됩니다. API 함수에 대한 설명에는 API 함수의 구문 및 사용법이 포함됩니다.
탐색기는 코드 항목 탭에서 사용할 수 없는 포트 또는 API 함수를 비활성화합니다. 예를 들어 패키지 가져오기 코드 항목 탭에서 포트를 추가하거나 API 함수를 호출할 수 없습니다.
- **코드 창.** 변환에 대한 Java 코드를 개발합니다. 코드 창에서는 기본 Java 구문 강조 표시가 사용됩니다.
- **코드 항목 탭.** 변환 동작을 정의합니다. 각 코드 항목 탭에는 연결된 코드 창이 있습니다. 코드 항목 탭에 Java 코드를 입력하려면 해당 탭을 클릭하고 코드 창에서 Java 코드를 기록합니다.
- **식 링크를 정의합니다.** Java 식 작성에 사용하는 식 정의 대화 상자를 엽니다.

- **설정 링크.** 설정 대화 상자를 엽니다. 타사 및 사용자 지정 **Java** 패키지의 클래스 경로를 설정하고 10진수 데이터 유형에 대해 많은 전체 자릿수를 활성화하고 초 단위 이하 데이터를 처리하려면 설정 대화 상자를 사용합니다. PowerCenter 클라이언트는 **java** 코드를 컴파일할 때 클래스 경로 내에 있는 파일을 포함시킵니다.
- **컴파일 링크.** 변환에 대한 **Java** 코드를 컴파일합니다. **Java** 컴파일러에서 출력되는 오류 및 정보 메시지 등이 출력 창에 표시됩니다.
- **전체 코드 링크.** 전체 코드 창을 열어 **Java** 변환에 대한 전체 클래스 코드를 표시합니다. 변환에 대한 전체 코드에는 **Java** 변환 클래스 템플릿에 추가되는 코드 항목 탭의 **Java** 코드가 포함됩니다.
- **출력 창.** **Java** 변환 클래스의 컴파일 결과를 표시합니다. 출력 창에서 오류 메시지를 마우스 오른쪽 단추로 클릭하면 전체 코드 창에서 **Java** 변환 클래스에 대한 조각 코드 또는 전체 코드에서 오류를 찾을 수 있습니다. 또한 출력 창에서 오류를 두 번 클릭하여 오류 소스를 찾을 수도 있습니다.

포트 구성

Java 변환은 입력 포트, 출력 포트 및 입력/출력 포트를 포함할 수 있습니다. 포트 탭에서 그룹 및 포트를 작성하고 편집합니다. 포트의 기본값을 지정할 수 있습니다. 포트를 변환에 추가한 후 **Java** 코드 조각에서 포트 이름을 변수로 사용합니다.

그룹 및 포트 작성

Java 변환을 작성하면 입력 그룹 1개와 출력 그룹 1개가 **Java** 변환에 포함됩니다.

Java 변환에는 항상 1개의 입력 그룹과 1개의 출력 그룹이 포함됩니다. 입력 또는 출력 그룹이 여러 개인 **Java** 변환은 유효하지 않습니다. 그룹 헤더에 그룹 이름을 입력하여 기존 그룹 이름을 변경할 수 있습니다. 그룹을 삭제한 경우 입력 그룹 작성 또는 출력 그룹 작성 아이콘을 클릭하여 새 그룹을 추가할 수 있습니다.

포트를 작성하면 **Designer**이 현재 선택된 행 또는 그룹 아래에 포트를 추가합니다. 입력 그룹 아래에 표시되는 입력/출력 포트는 출력 그룹에도 포함됩니다. 출력 그룹 아래에 표시되는 입력/출력 포트는 입력 그룹에도 포함됩니다.

기본 포트 값 설정

Java 변환에서 포트에 대한 기본값을 정의할 수 있습니다.

Java 변환은 포트의 데이터 유형에 따라 기본 포트 값을 사용하여 포트 변수를 초기화합니다.

입력 및 출력 포트

Java 변환은 **Java** 코드 조각에서 할당된 값이 없는 연결되지 않은 입력 포트 또는 출력 포트의 값을 초기화합니다.

Java 변환은 다음 **Java** 데이터 유형에 따라 포트를 초기화합니다.

기본 데이터 유형

포트에 대해 **null**이 아닌 기본값을 정의한 경우 변환에서 포트 변수 값을 기본값으로 초기화합니다. 그렇지 않은 경우 포트 변수 값을 **0**으로 초기화합니다.

복합 데이터 유형

포트에 대한 기본값을 정의한 경우 변환에서 새 개체를 생성하고 개체를 기본값으로 초기화합니다. 그렇지 않은 경우 변환에서 포트 변수를 **Null**로 초기화합니다. 예를 들어 **string** 포트에 대한 기본값을 정의한 경우 변환이 새 **String** 개체를 생성하고 **String** 개체를 기본값으로 초기화합니다.

참고: Java 코드에서 Null 값을 가진 입력 포트 변수에 액세스할 경우 `NullPointerException`이 발생합니다.

입력 포트를 파티션 키 및 정렬 키로 활성화할 수 있으며, 정렬 방향을 할당할 수 있습니다. 데이터 통합 서비스는 데이터를 분할하고 정렬 키와 정렬 방향을 기준으로 각 파티션의 데이터를 정렬합니다. 변환 범위가 모든 입력으로 설정된 경우 파티션 키 및 정렬 키가 사용될 수 있습니다.

데이터 파티션 및 정렬을 위해 다음 속성을 사용하십시오.

파티션 키

동일한 파티션으로 그룹화할 데이터 행을 결정하는 입력 포트입니다.

하나 이상의 입력 행을 파티션 키로 활성화할 수 있습니다. 데이터 통합 서비스는 코드 실행 전에 파티션 키를 사용하여 데이터를 재분할합니다. 입력 행을 파티션 키로 선택하지 않은 경우 기본 분할 구성표를 사용하여 데이터가 처리됩니다.

정렬 키

각 파티션 내에서 정렬 조건을 결정하는 입력 포트입니다.

방향

오름차순 또는 내림차순 순서입니다. 기본값은 오름차순입니다.

입력/출력 포트

Java 변환은 입력/출력 포트를 통과 포트 처리합니다. 변환에 대한 Java 코드에서 포트 값을 설정하지 않으면 출력 값이 입력 값과 동일합니다. Java 변환은 입력 포트를 초기화하는 것과 동일하게 입력/출력 포트 값을 초기화합니다.

Java 코드에서 입력/출력 포트에 대한 포트 변수 값을 설정하면 Java 변환이 출력 행을 생성할 때 이 값을 사용합니다. 입력/출력 포트 값을 설정하지 않으면 Java 변환이 출력 행을 생성할 때 포트 변수 값을 단순 데이터 유형에 대해 0으로 설정하고 복합 데이터 유형에 대해 NULL로 설정합니다.

Java 변환 속성 구성

Java 변환에는 변환 코드 및 변환에 대한 속성이 포함됩니다. 변환 개발자에서 Java 변환을 작성하는 경우 매핑에서 변환을 사용할 때 변환 속성을 재정의할 수 있습니다.

다음 테이블에는 Java 변환 속성이 설명되어 있습니다.

속성	설명
언어	변환 코드에 사용되는 언어입니다. 이 값을 변경할 수 없습니다.
클래스 이름	변환의 Java 클래스 이름입니다. 이 값을 변경할 수 없습니다.
추적 수준	이 변환에 대해 세션 로그에 표시되는 세부 정보의 양입니다. 다음과 같은 추적 수준을 사용합니다. <ul style="list-style-type: none">- 간단- 보통- 자세한 정보 표시 초기화- 자세한 정보 표시 데이터 기본값은 보통입니다.

속성	설명
분할 가능 여부	<p>파이프라인의 여러 파티션에서 이 변환을 사용할 수 있습니다. 다음과 같은 옵션을 사용합니다.</p> <ul style="list-style-type: none"> - 아니요. 변환을 분할할 수 없습니다. 변환과 동일한 파이프라인에 있는 다른 변환이 하나의 파티션으로 제한됩니다. 변환에서 데이터 정리처럼 입력 데이터를 모두 함께 처리하는 경우 아니요를 선택할 수 있습니다. - 로컬로. 변환을 분할할 수 있지만 통합 서비스가 같은 노드의 파이프라인에서 모든 파티션을 실행해야 합니다. 변환의 여러 파티션이 메모리에서 개체를 공유해야 하는 경우 로컬로 선택합니다. - 그리드에서. 변환을 분할할 수 있고 통합 서비스가 각 파티션을 여러 노드에 분산시킬 수 있습니다. <p>기본값은 아니요입니다.</p>
입력을 차단해야 함	<p>변환과 연결된 프로시저가 수신 데이터를 차단할 수 있어야 합니다. 기본값은 활성화됩니다.</p>
활성 여부	<p>Java 변환에서 각 입력 행에 대해 2개 이상의 출력 행을 생성할 수 있습니다. 이 속성은 Java 변환을 작성한 후에 변경할 수 없습니다. 이 속성을 변경하려면 새 Java 변환을 작성해야 합니다.</p>
업데이트 전략 변환	<p>변환이 출력 행에 대해 업데이트 전략을 정의합니다. 활성 Java 변환에 대해 이 속성을 활성화할 수 있습니다. 기본값이 비활성화됩니다.</p>
변환 범위	<p>통합 서비스가 변환 논리를 수신 데이터에 적용하는 방식입니다. 다음과 같은 옵션을 사용합니다.</p> <ul style="list-style-type: none"> - 행 - 트랜잭션 - 모든 입력 <p>수동 변환의 경우 이 속성은 항상 행으로 지정됩니다. 활성 변환의 경우 기본값은 모든 입력입니다.</p>
트랜잭션 생성	<p>변환이 트랜잭션 행을 생성합니다. 활성 Java 변환에 대해 이 속성을 활성화할 수 있습니다. 기본값이 비활성화됩니다.</p>
출력은 반복 가능합니다.	<p>출력 데이터의 순서는 세션 실행 간에 일관됩니다.</p> <ul style="list-style-type: none"> - 사용 안 함 출력 데이터 순서는 세션 실행 간에 일관되지 않습니다. - 입력 순서 기반. 입력 데이터 순서가 세션 실행 간에 일관된 경우 입력 순서는 세션 실행 간에 일관됩니다. - 항상 입력 데이터 순서가 세션 실행 간에 일관되지 않더라도 출력 데이터 순서는 세션 실행 간에 일관됩니다. <p>활성 변환의 경우 기본값은 사용 안 함입니다. 수동 변환의 경우 기본값은 입력 순서 기반입니다.</p>
파티션당 단일 스레드 필요	<p>단일 스레드가 각 파티션의 데이터를 처리합니다. 이 값을 변경할 수 없습니다.</p>
확정 출력입니다.	<p>변환이 세션 실행 간에 일관된 출력 데이터를 생성합니다. 이 변환을 사용하는 세션에 대해 복구를 수행하려면 이 속성을 활성화합니다. 기본값은 활성화됩니다.</p>

경고: 변환을 반복 가능 및 확정으로 구성하는 경우 데이터가 반복 가능 및 확정인지 확인하는 것은 사용자의 책임입니다. 세션과 복구 간에 동일한 데이터를 생성하지 않는 변환으로 세션을 복구하려는 경우 복구 프로세스로 인해 손상된 데이터가 발생할 수 있습니다.

트랜잭션 제어 작업

다음 속성을 사용하여 **Java** 변환에 대한 트랜잭션 제어를 정의할 수 있습니다.

- **변환 범위.** 통합 서비스가 변환 논리를 수신 데이터에 어떻게 적용하는지 결정합니다.
- **트랜잭션 생성.** 변환에 대한 **Java** 코드가 트랜잭션 행을 생성하여 출력 그룹에 전달함을 나타냅니다.

변환 범위

통합 서비스가 수신 데이터에 변환 논리를 어떻게 적용하는지 구성할 수 있습니다. 다음 값 중 하나를 선택할 수 있습니다.

- **행.** 한 번에 데이터의 한 행에 변환 논리를 적용합니다. 데이터의 단일 행에 따라 변환의 결과가 결정되는 경우 행을 선택합니다. 수동 변환의 경우 행을 선택해야 합니다.
- **트랜잭션.** 트랜잭션의 모든 행에 변환 논리를 적용합니다. 다른 트랜잭션의 행이 아닌 동일한 트랜잭션의 모든 행에 따라 변환의 결과가 결정되는 경우 트랜잭션을 선택합니다. 예를 들어 **Java** 코드가 단일 트랜잭션의 데이터에 대한 집계 계산을 수행하는 경우 트랜잭션을 선택할 수 있습니다.
- **모든 입력.** 모든 수신 데이터에 변환 논리를 적용합니다. 모든 입력을 선택하면 통합 서비스가 트랜잭션 경계를 삭제합니다. 소스 데이터의 모든 행에 따라 변환의 결과가 결정되는 경우 모든 입력을 선택합니다. 예를 들어 변환에 대한 **Java** 코드가 모든 수신 데이터를 정렬하는 경우 모든 입력을 선택할 수 있습니다.

트랜잭션 생성

커밋 및 롤백 행 등 트랜잭션 행을 생성하도록 활성 **Java** 변환에서 **Java** 코드를 정의할 수 있습니다. 트랜잭션 행을 생성하려면 해당 변환을 활성화하여 트랜잭션 행을 생성합니다.

트랜잭션 행을 생성하도록 변환을 구성하는 경우 통합 서비스가 트랜잭션 제어 변환과 같이 **Java** 변환을 처리합니다. 또한 매핑의 트랜잭션 제어 변환에 적용되는 대부분의 규칙은 **Java** 변환에도 적용됩니다. 예를 들어 트랜잭션 행을 생성하도록 **Java** 변환을 구성하는 경우 해당 변환이 포함된 파이프라인 또는 파이프라인 분기를 연결할 수 없습니다.

트랜잭션 행을 생성하도록 구성된 **Java** 변환을 사용하여 세션을 편집하거나 작성하는 경우 사용자 정의 커밋용으로 이 세션을 구성합니다.

관련 항목:

- [“커밋” 페이지 227](#)
- [“롤백” 페이지 232](#)

업데이트 전략 설정

활성 **Java** 변환을 사용하여 매핑에 대한 업데이트 전략을 설정합니다. 다음 수준에서 업데이트 전략을 설정할 수 있습니다.

- **Java 코드 내에서.** 출력 행에 대한 업데이트 전략을 설정하기 위해 **Java** 코드를 작성할 수 있습니다. **Java** 코드에서 삽입, 업데이트, 삭제 또는 거부 플래그를 행에 지정할 수 있습니다. 업데이트 전략 설정에 대한 자세한 내용은 [“setOutRowType” 페이지 233](#)을 참조하십시오.
- **매핑 내에서.** 매핑에서 **Java** 변환을 사용하여 삽입, 업데이트, 삭제 또는 거부 플래그를 행에 지정합니다. **Java** 변환의 업데이트 전략 변환 속성을 선택합니다.

- **세션 내에서.** 소스 행을 데이터 구동으로 처리하도록 세션을 구성합니다.

업데이트 전략을 정의하기 위해 **Java** 변환을 구성하지 않거나 세션을 데이터 구동으로 구성하지 않은 경우, 통합 서비스는 출력 행에 플래그를 지정하는 데 **Java** 코드를 사용하지 않습니다. 대신 통합 서비스는 출력 행에 삽입 플래그를 지정합니다.

Java 코드 개발

코드 항목 탭을 사용하여 **Java** 변환 기능을 정의하는 **Java** 코드 조각을 입력할 수 있습니다. 코드 항목 탭에서 **Java** 패키지를 가져오고, 도우미 코드를 작성하고, **Java** 식을 정의하고, 특정 변환 이벤트에 대한 변환 동작을 정의하는 **Java** 코드를 기록할 수 있습니다. 코드 항목 탭에서 원하는 순서로 조각을 개발할 수 있습니다.

다음과 같은 코드 항목 탭에서 **Java** 코드를 입력합니다.

- **패키지 가져오기.** 타사 **Java** 패키지, 기본 제공 **Java** 패키지 또는 사용자 지정 **Java** 패키지를 가져옵니다.
- **도우미 코드.** 패키지 가져오기 탭을 제외한 모든 탭에서 사용할 수 있는 변수 및 메서드를 정의합니다.
- **입력 행에서.** 입력 행을 수신할 때의 변환 동작을 정의합니다.
- **데이터 끝에서.** 모든 입력 데이터를 처리했을 때의 변환 동작을 정의합니다.
- **수신 트랜잭션에서.** 트랜잭션 알림을 수신할 때의 변환 동작을 정의합니다. 활성 **Java** 변환에서 사용합니다.
- **Java 식.** PowerCenter 식을 호출하는 **Java** 식을 정의합니다. 도우미 코드, 입력 행에서, 데이터 끝에서 및 수신 트랜잭션에서 코드 항목 탭을 통해 **Java** 식을 사용할 수 있습니다.

입력 행에서 탭을 통해 입력 데이터에 액세스하고 출력 데이터를 설정합니다. 활성 변환의 경우 데이터 끝에서 및 수신 트랜잭션에서 탭을 통해서도 출력 데이터를 설정할 수 있습니다.

Java 코드 조각 작성

Java 코드 조각을 작성하여 변환 동작을 정의하려면 **Java 코드** 탭의 **코드**를 사용합니다.

1. 해당하는 코드 항목 탭을 클릭합니다.
2. 조각의 입력 또는 출력 열 변수에 액세스하려면 탐색기에서 포트 이름을 두 번 클릭합니다.
3. 조각의 **Java** 변환 API를 호출하려면 탐색기에서 API 이름을 두 번 클릭합니다. 필요한 경우 적절한 API 입력 값을 구성합니다..
4. 코드 조각을 바탕으로 적절한 **Java** 코드를 작성합니다.

전체 코드 창(에서 **Java** 변환의 전체 클래스 코드를 봅니다.

Java 패키지 가져오기

패키지 가져오기 탭에서 활성 또는 수동 **Java** 변환의 **Java** 패키지를 가져올 수 있습니다.

타사, 기본 제공 또는 사용자 지정 **Java** 패키지를 가져올 수 있습니다. **Java** 패키지를 가져온 후 다른 코드 항목 탭에서 가져온 패키지를 사용할 수 있습니다.

참고: **패키지 가져오기** 탭에서는 정적 변수, 인스턴스 변수 또는 사용자 메서드를 선언할 수 없습니다.

PowerCenter 클라이언트에서 **Java** 변환이 포함된 메타데이터를 내보내거나 가져오는 경우 **Java** 변환에 필요한 타사 또는 사용자 지정 패키지를 포함하는 **jar** 또는 클래스 파일은 내보내기 또는 가져오기에 포함되지 않습니다.

Java 변환이 포함된 메타데이터를 가져오는 경우에는 필요한 타사 또는 사용자 지정 패키지가 포함된 jar 또는 클래스 파일을 PowerCenter 클라이언트 및 통합 서비스 노드에 복사해야 합니다.

예를 들어 Java I/O 패키지를 가져오려면 **패키지 가져오기** 탭에서 다음 코드를 입력합니다.

```
import java.io.*;
```

비표준 Java 패키지를 가져오는 경우 패키지 또는 클래스를 Java 변환 설정의 클래스 경로에 추가합니다.

관련 항목:

- [“Java 변환 설정 구성” 페이지 221](#)

도우미 코드 정의

도우미 코드 탭에서 활성 또는 수동 Java 변환의 Java 변환 클래스에 대한 사용자 정의 변수 및 메서드를 선언할 수 있습니다.

도우미 코드 탭에서 선언한 변수 및 메서드는 **패키지 가져오기** 탭을 제외한 모든 코드 항목 탭에서 사용할 수 있습니다.

도우미 코드 탭에서 다음 유형의 코드, 변수 및 메서드를 선언할 수 있습니다.

- 정적 코드 및 정적 변수.

정적 블록 안에 정적 변수 및 정적 코드를 선언할 수 있습니다. 세션의 모든 파티션 및 매핑에서 재사용 가능한 Java 변환의 모든 인스턴스는 정적 코드 및 변수를 공유합니다. 정적 코드는 Java 변환의 다른 코드보다 먼저 실행됩니다.

예를 들어 다음 코드는 매핑에서 Java 변환의 모든 인스턴스에 대한 오류 임계값을 저장하는 정적 변수를 선언합니다.

```
static int errorThreshold;
```

변환의 오류 임계값을 저장하고 세션의 모든 파티션 및 매핑의 모든 Java 변환 인스턴스에서 오류 임계값에 액세스하려면 이 변수를 사용합니다.

참고: 여러 파티션 세션 또는 재사용 가능한 Java 변환의 정적 변수를 동기화해야 합니다.

- 인스턴스 변수.

파티션 수준 인스턴스 변수를 선언할 수 있습니다. 세션의 여러 파티션 또는 매핑에서 재사용 가능한 Java 변환의 여러 인스턴스는 인스턴스 변수를 공유하지 않습니다. 충돌을 방지하고 비기본 인스턴스 변수를 초기화하려면 접두사를 사용하여 인스턴스 변수를 선언하십시오.

예를 들어 다음 코드는 부울 변수를 사용하여 출력 행의 생성 여부를 결정합니다.

```
// boolean to decide whether to generate an output row
// based on validity of input
private boolean generateRow;
```

- 사용자 정의 정적 메서드 또는 인스턴스 메서드.

Java 변환의 기능을 확장합니다. **도우미 코드** 탭에서 선언된 Java 메서드는 출력 변수 또는 로컬로 선언된 인스턴스 변수를 사용하거나 수정할 수 있습니다. **도우미 코드** 탭에서는 Java 메서드의 입력 변수에 액세스할 수 없습니다.

예를 들어 두 정수를 더하는 함수를 선언하려면 탭에서 다음 코드를 사용합니다.

```
private int myTXAdd (int num1,int num2)
{
    return num1+num2;
}
```


입력 행에서 탭

입력 행에서의 **Java** 변환 동작을 정의하려면 입력 행에서 탭을 사용합니다. 이 탭에 있는 **Java** 코드는 각 입력 행에 대해 한 번씩 실행됩니다. 입력 행 데이터는 입력 행에서 탭에서만 액세스할 수 있습니다.

입력 행에서 탭에서 다음 입력 및 출력 포트 데이터, 변수, 메시지를 액세스하고 사용하십시오.

- **입력 및 출력 포트 변수입니다.** 포트 이름을 변수 이름으로 사용하여 입력 및 출력 포트 데이터를 변수로서 액세스합니다. 예를 들어 "in_int"가 정수 입력 포트이면 **Java** 기본 데이터 유형 **int**가 포함된 변수 "in_int"로 참조하여 이 포트의 데이터에 액세스할 수 있습니다. 입력 및 출력 포트를 변수로 선언할 필요는 없습니다.

입력 포트 변수에는 값을 할당하지 마십시오. 입력 행에서 탭에서 입력 변수에 값을 할당하는 경우, 현재 행에서 상응하는 포트에 대한 입력 데이터를 가져올 수 없습니다.

- **인스턴스 변수 및 사용자 정의 메시지.** 도우미 코드 탭에서 선언한 인스턴스 또는 정적 변수나 사용자 정의 메시지를 사용합니다.

활성 **Java** 변환에 정수 데이터 유형의 두 입력 포트 **BASE_SALARY** 및 **BONUSES**와 정수 데이터 유형의 출력 포트 **TOTAL_COMP** 하나가 포함되는 경우를 예로 들어 보겠습니다. 정수 2개를 추가하고 결과를 반환하는 **myTXAdd**라는 사용자 정의 메시지를 도우미 코드 탭에서 작성하십시오. 입력 포트의 합계 값을 출력 포트에 할당하고 출력 행을 생성하려면 입력 행에서 탭에서 다음 **Java** 코드를 사용합니다.

```
TOTAL_COMP = myTXAdd (BASE_SALARY,BONUSES);
generateRow();
```

Java 변환은 입력 행을 받으면 **BASE_SALARY** 및 **BONUSES** 입력 포트의 값을 추가하고 **TOTAL_COMP** 출력 포트에 해당 값을 할당한 다음 출력 행을 생성합니다.

- **Java 변환 API 메시지.** **Java** 변환에서 제공하는 API 메시지를 호출할 수 있습니다.

데이터 끝에서 탭

활성 또는 수동 **Java** 변환에서 데이터 끝에서 탭을 사용하여 **Java** 변환이 모든 입력 데이터를 처리한 경우 수행할 동작을 정의합니다. 데이터 끝에서 탭에서 출력 행을 생성하려면 변환의 변환 범위를 트랜잭션 또는 모든 입력으로 설정하십시오. 이 탭에서 입력 포트 변수의 값을 액세스하거나 설정할 수 없습니다.

데이터 끝에서 탭에서 다음 변수 및 메시드에 액세스하고 사용하십시오.

- **출력 포트 변수.** 출력 포트 이름을 변수로 사용하여 활성 **Java** 변환에 대한 출력 데이터를 액세스하거나 설정합니다.
- **인스턴스 변수 및 사용자 정의 메시지.** 도우미 코드 탭에서 선언한 모든 인스턴스 변수 또는 사용자 정의 메시지를 사용합니다.
- **Java 변환 API 메시지.** **Java** 변환이 제공하는 API 메시지를 호출합니다. **commit** 및 **rollBack** API 메시지를 사용하여 트랜잭션을 생성합니다.

예를 들어 데이터의 끝에 도달할 때 세션 로그에 정보를 기록하려면 다음 **Java** 코드를 사용합니다.

```
logInfo("Number of null rows for partition is: " + partCountNullRows);
```

트랜잭션 수신 시 탭

활성 **Java** 변환의 트랜잭션 수신 시 탭을 사용하여 활성 **Java** 변환이 트랜잭션 알림을 받은 경우 수행할 동작을 정의합니다. 트랜잭션 수신 시 탭의 코드 조각은 변환의 트랜잭션 범위가 트랜잭션으로 설정되어 있는 경우에만 실행됩니다. 이 탭에서 입력 포트 변수의 값을 액세스하거나 설정할 수 없습니다.

트랜잭션 수신 시 탭에서 다음 출력 데이터, 변수 및 메시지를 액세스하고 사용하십시오.

- **출력 포트 변수.** 출력 포트 이름을 변수로 사용하여 출력 데이터를 액세스하거나 설정합니다.
- **인스턴스 변수 및 사용자 정의 메시지.** 도우미 코드 탭에서 선언한 모든 인스턴스 변수 또는 사용자 정의 메시지를 사용합니다.

- **Java 변환 API 메서드.** Java 변환이 제공하는 API 메서드를 호출합니다. `commit` 및 `rollBack` API 메서드를 사용하여 트랜잭션을 생성합니다. 예를 들어 변환이 트랜잭션을 받은 후 트랜잭션을 생성하려면 다음 Java 코드를 사용합니다.

```
commit();
```

Java 코드를 사용하여 플랫 파일 구문 분석

플랫 파일을 구문 분석하기 위한 Java 코드를 개발할 수 있습니다. Java 코드를 사용하여 여러 가지 스키마의 플랫 파일 또는 JMS 메시지에서 특정 데이터 열을 추출합니다.

구분자로 분리된 플랫 파일에서 처음 2개의 데이터 열을 읽으려고 하는 경우를 예로 들어 보겠습니다. 구분자로 분리된 플랫 파일에서 데이터를 읽고 하나 이상의 출력 포트에 데이터를 전달하는 매핑을 작성하십시오.

매핑에는 다음과 같은 구성 요소가 있습니다.

- **소스 정의.** 소스는 구분자로 분리된 플랫 파일입니다. 플랫 파일의 행을 문자열로 Java 변환에 전달하도록 소스를 구성하십시오. 소스 파일에는 다음과 같은 데이터가 있습니다.

```
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 변환.** Java 변환 기능을 Java 코드 탭에서 정의합니다.

해당 Java 코드의 패키지 가져오기 탭을 사용하여 `java` 패키지를 가져옵니다. 패키지 가져오기 탭에서 다음 코드를 입력하십시오.

```
import java.util.StringTokenizer;
```

해당 Java 코드의 입력 행에서 탭을 사용하여 문자열에서 각 데이터 행을 읽고 데이터를 출력 포트에 전달합니다. 처음 2개의 데이터 열을 검색하려면 입력 행에서 탭에서 다음 코드를 입력하십시오.

```
StringTokenizer st1 = new StringTokenizer(row, ",");
f1 = st1.nextToken();
f11= st1.nextToken();
```

- **대상 정의.** Java 변환으로부터 데이터 행을 받도록 대상을 구성합니다. 이 매핑을 포함하는 워크플로우를 실행한 후에 대상은 데이터 열 2개를 포함합니다. 대상 파일에는 다음과 같은 데이터가 있습니다.

```
1a,2a
1b,2b
1c,2c
1d,2d
```

Java 변환 설정 구성

Java 변환 설정을 구성하여 타사 및 사용자 지정 Java 패키지의 클래스 경로를 설정하고 10진수 데이터 유형에 대해 많은 전체 자릿수를 활성화할 수 있습니다. 또한 초 단위 이하 데이터를 처리하도록 변환을 구성할 수 있습니다.

클래스 경로 구성

패키지 가져오기 탭에서 비표준 Java 패키지를 가져오는 경우 PowerCenter 클라이언트 및 통합 서비스의 클래스 경로를 Java 패키지와 연결된 각각의 JAR 파일 또는 클래스 파일 디렉터리로 설정합니다. UNIX의 경우 콜론을 사용하여 클래스 경로 항목을 구분합니다. Windows의 경우 세미콜론을 사용하여 클래스 경로 항목을 구분합니다. PowerCenter 클라이언트 및 통합 서비스 노드에서 JAR 파일 또는 클래스 파일에 액세스할 수 있어야 합니다.

예를 들어 패키지 가져오기 탭에서 **Java** 패키지 변환기를 가져오고 **converter.jar**에 패키지를 정의합니다. **Java** 변환에 대한 **Java** 코드를 컴파일하기 전에 **converter.jar**를 클래스 경로에 추가해야 합니다.

기본 제공 **Java** 패키지에 대한 클래스 경로는 설정하지 않아도 됩니다. 예를 들어 **java.io**는 기본 제공 **Java** 패키지입니다. **java.io**를 가져오는 경우 **java.io**의 클래스 경로를 설정할 필요가 없습니다.

통합 서비스의 클래스 경로 구성

JAR 파일 또는 클래스 파일 디렉토리를 통합 서비스 클래스 경로에 추가할 수 있습니다. 통합 서비스 노드에 대한 클래스 경로를 설정하려면 다음 태스크 중 하나를 수행하십시오.

- **Java 클래스 경로 세션 속성 구성.** Java 클래스 경로 세션 속성을 사용하여 클래스 경로를 설정합니다. 이 클래스 경로는 세션에 적용됩니다.
- **Java SDK 클래스 경로 구성.** Informatica Administrator에서 통합 서비스 속성의 프로세스 탭을 통해 Java SDK 클래스 경로를 구성합니다. 이 설정은 통합 서비스에서 실행되는 모든 세션에 적용됩니다.
- **CLASSPATH 환경 변수 구성.** 통합 서비스 노드에 대해 CLASSPATH 환경 변수를 설정합니다. 환경 변수를 설정한 후 통합 서비스를 다시 시작합니다. 이 설정은 통합 서비스에서 실행되는 모든 세션에 적용됩니다.

UNIX에서 통합 서비스의 클래스 경로 구성

UNIX에서 CLASSPATH 환경 변수를 구성하려면

- ▶ UNIX C 셸 환경에서 다음을 입력합니다.
`setenv CLASSPATH <classpath>`
- ▶ UNIX Bourne 셸 환경에서 다음을 입력합니다.
`CLASSPATH = <classpath>`
`export CLASSPATH`

Windows에서 통합 서비스의 클래스 경로 구성

Windows에서 CLASSPATH 환경 변수를 구성하려면

- ▶ CLASSPATH 환경 변수를 입력하고 값을 기본 클래스 경로로 설정합니다.

PowerCenter 클라이언트에 대한 클래스 경로 구성

jar 파일 또는 클래스 파일 디렉토리를 **PowerCenter** 클라이언트 클래스 경로에 추가할 수 있습니다.

PowerCenter 클라이언트가 실행되는 시스템에 대한 클래스 경로를 설정하려면 다음 태스크를 완료하십시오.

- CLASSPATH 환경 변수를 구성합니다. CLASSPATH 환경 변수는 **PowerCenter** 클라이언트 시스템에서 설정합니다. 이 설정은 시스템에서 실행되는 모든 **Java** 프로세스에 적용됩니다.
- Java 변환 설정에서 클래스 경로를 구성합니다. 이 설정은 이 **Java** 변환을 포함하는 세션에 적용됩니다. **PowerCenter** 클라이언트는 **java** 코드를 컴파일할 때 클래스 경로 내에 있는 파일을 포함시킵니다.

jar 또는 클래스 파일 디렉토리를 **Java** 변환의 클래스 경로에 추가하려면 다음 단계를 완료하십시오.

1. **Java 코드** 탭에서 **설정** 링크를 클릭클래스 경로 옆의 값 열에 있는 아래쪽 화살표 아이콘을 클릭합니다.
설정 대화 상자가 표시됩니다.
2. **클래스 경로 추가** 아래에서 찾아보기를 클릭하고 가져온 패키지에 대한 **jar** 파일 또는 클래스 파일 디렉토리를 선택합니다. **확인**을 클릭합니다.
3. **추가**를 클릭합니다.
jar 또는 클래스 파일 디렉터리가 변환의 jar 및 클래스 파일 디렉터리 목록에 표시됩니다.

4. jar 파일 또는 클래스 파일 디렉터를 제거하려면 jar 또는 클래스 파일 디렉터를 선택하고 **제거**를 클릭합니다.

디렉터리 목록에서 디렉터리가 사라집니다.

많은 전체 자릿수 활성화

기본적으로 Java 변환은 10진수 유형의 포트를 전체 자릿수가 15인 배정밀도 데이터 유형으로 변환합니다. 10진수 데이터 유형을 15보다 많은 전체 자릿수를 사용하여 처리하려면 Java 클래스 BigDecimal을 사용하여 10진수 포트를 처리하도록 많은 전체 자릿수를 활성화합니다.

많은 전체 자릿수를 활성화하면 전체 자릿수가 28 미만인 10진수 포트를 BigDecimal로 처리할 수 있습니다. Java 변환은 전체 자릿수가 28을 초과하는 10진수 데이터를 배정밀도 데이터 유형으로 변환합니다. Java 변환식은 이진, 정수, 배정밀도 및 문자열 데이터를 처리합니다. Java 변환식은 bigint 데이터를 처리할 수 없습니다.

예를 들어 Java 변환에 40012030304957666903의 값을 수신하는 10진수 유형의 입력 포트가 있습니다. 많은 전체 자릿수를 활성화하면 포트의 값이 표시되는 대로 처리됩니다. 많은 전체 자릿수를 활성화하지 않으면 포트의 값이 $4.00120303049577 \times 10^{19}$ 입니다.

참고: Java 변환에 대해 많은 전체 자릿수를 활성화하는 경우 Java 변환이 포함된 세션에 대해서는 많은 전체 자릿수를 활성화하십시오. 많은 전체 자릿수가 Java 변환에서 활성화되었지만 세션에서 활성화되지 않으면 세션이 다음과 같은 오류와 함께 실패합니다.

```
[ERROR]Failed to bind column with index <index number> to datatype <datatype name>.
```

초 단위 이하 처리

Java 코드에서 나노초까지 초 단위 이하 데이터를 처리할 수 있습니다. 날짜/시간 값에서 나노초를 사용하도록 설정을 구성하는 경우, 생성되는 Java 코드에서는 변환의 날짜/시간 데이터 유형을 전체 자릿수가 나노초에 이르는 Java BigDecimal 데이터 유형으로 변환합니다.

생성되는 Java 코드에서는 기본적으로 변환의 날짜/시간 데이터 유형을 전체 자릿수가 밀리초인 Java 긴 정수 데이터 유형으로 변환합니다.

Java 변환 컴파일

PowerCenter 클라이언트는 Java 컴파일러를 사용하여 Java 코드를 컴파일하고 변환의 바이트 코드를 생성합니다.

Java 컴파일러는 Java 코드를 컴파일하고 **출력** 창에 컴파일 결과를 표시합니다. Java 컴파일러는 PowerCenter 클라이언트와 함께 java/bin 디렉터리에 설치됩니다.

Java 변환의 전체 코드를 컴파일하려면 **컴파일 Java 코드** 탭 사용).

작성된 Java 변환에는 Java 변환의 기본 기능을 정의하는 Java 클래스가 포함됩니다. Java 클래스의 전체 코드에는 변환의 템플릿 클래스 코드와 사용자가 코드 항목 탭에서 정의하는 Java 코드가 포함됩니다.

Java 변환을 컴파일하면 PowerCenter 클라이언트가 코드 항목 탭의 코드를 변환의 템플릿 클래스에 추가하여 변환의 전체 클래스 코드를 생성합니다. 그런 다음 PowerCenter 클라이언트는 Java 컴파일러를 호출하여 전체 클래스 코드를 컴파일합니다. Java 컴파일러는 변환을 컴파일하여 변환의 바이트 코드를 생성합니다.

컴파일 결과는 **출력** 창에 표시됩니다. 컴파일 결과를 사용하여 Java 코드 오류를 식별하고 찾을 수 있습니다.

참고: Java 변환에서 **확인**을 클릭하여 변환을 컴파일할 수도 있습니다.

컴파일 오류 수정

출력 창에서 **Java** 변환에 대한 **Java** 코드 오류를 확인하고 **Java** 코드 오류의 원인을 찾을 수 있습니다. **Java** 변환 오류는 코드 항목 탭에서 나타나는 오류의 결과로 발생하거나 **Java** 변환 클래스에 대한 전체 코드에서 나타나는 오류의 결과로 발생할 수 있습니다.

Java 변환 문제를 해결하려면

- **오류의 원인을 찾습니다.** 변환에 대한 **Java** 조각 코드 또는 전체 클래스 코드에서 오류의 원인을 찾을 수 있습니다.
- **오류 유형을 식별합니다.** 출력 창의 컴파일 결과와 오류의 위치를 기반으로 오류의 유형을 식별합니다.

오류의 원인과 유형을 식별한 후 코드 항목 탭에서 **Java** 코드를 수정하고 변환을 다시 컴파일합니다.

컴파일 오류의 원인 찾기

Java 변환을 컴파일하면 출력 창에 컴파일의 결과가 표시됩니다. 컴파일 결과를 통해 컴파일 오류를 확인할 수 있습니다. 출력 창에서 오류의 원인을 찾을 경우 **PowerCenter** 클라이언트가 코드 항목 탭 또는 전체 코드 창에서 오류의 원인을 강조 표시합니다.

전체 코드 창에서 오류를 찾을 수 있지만 **Java** 코드를 편집할 수는 없습니다. 전체 코드 창에서 찾은 오류를 수정하려면 해당 코드 항목 탭에서 코드를 수정하십시오. 변환의 전체 클래스 코드에 사용자 코드를 추가하여 발생한 오류를 보려면 전체 코드 창을 사용해야 할 수 있습니다.

출력 창의 컴파일 결과를 기반으로 다음 위치에서 오류를 확인하십시오.

- 코드 항목 탭
- 전체 코드 창

코드 항목 탭에서 오류 찾기

코드 항목 탭에서 오류의 원인을 찾으려면 출력 창에서 오류를 마우스 오른쪽 단추로 클릭하고 조각의 오류 보기를 선택하거나 출력 창에서 오류를 두 번 클릭합니다. **PowerCenter** 클라이언트가 해당 코드 항목 탭에서 오류의 원인을 강조 표시합니다.

전체 코드 창에서 오류 찾기

전체 코드 창에서 오류의 원인을 찾으려면 출력 창에서 오류를 마우스 오른쪽 단추로 클릭하고 전체 코드의 오류 보기를 선택하거나 출력 창에서 오류를 두 번 클릭합니다. **PowerCenter** 클라이언트가 전체 코드 창에서 오류의 원인을 강조 표시합니다.

컴파일 오류의 소스 식별

사용자 코드에 오류가 있을 경우 컴파일 오류가 발생할 수 있습니다.

사용자 코드에 오류가 있을 경우 클래스에 대한 비사용자 코드에도 오류가 발생할 수 있습니다. 컴파일 오류는 **Java** 변환에 대한 사용자 및 비사용자 코드에서 발생합니다.

사용자 코드 오류

코드 입력 탭의 사용자 코드에서 오류가 발생할 수 있습니다. 사용자 코드 오류에는 표준 **Java** 구문 및 언어 오류가 포함됩니다.

PowerCenter 클라이언트가 코드 입력 탭에서 전체 클래스 코드에 사용자 코드를 추가할 때도 사용자 코드 오류가 발생할 수 있습니다.

Java 변환에 이름이 **int1**인 입력 포트와 정수 데이터 유형이 포함되는 경우를 예로 들어 보겠습니다. 클래스의 전체 코드는 다음 코드를 사용하여 입력 포트 변수를 선언합니다.

```
int int1;
```

그러나 **입력 시 행** 탭에서 같은 변수를 사용하는 경우에는 Java 컴파일러가 변수 다시 선언에 대해 오류를 표시합니다. 이 오류를 해결하려면 **입력 시 행** 탭에서 변수 이름을 바꿉니다.

비사용자 코드 오류

코드 항목 탭의 사용자 코드에서 사용자가 아닌 코드로 인해 오류가 발생할 수 있습니다.

예를 들어 Java 변환에는 정수 데이터 유형을 사용하는 입력 포트 및 출력 포트, **int1** 및 **out1**이 있습니다. 다음 코드를 **입력 행에서** 코드 항목 탭에 쓰면 입력 포트 **int1**에 대해 이자를 계산하고 출력 포트 **out1**에 할당합니다.

```
int interest;  
interest = CallInterest(int1); // calculate interest  
out1 = int1 + interest;  
}
```

변환을 컴파일하면 PowerCenter Client이 **입력 행에서** 코드 항목 탭의 코드를 변환을 위해 전체 클래스 코드에 추가합니다. Java 컴파일러가 Java 코드를 컴파일할 때 일치하지 않는 괄호가 있으면 전체 클래스 코드의 메시지가 중간에 종료되고 Java 컴파일러에서 오류가 발생합니다.

제 13 장

Java 변환 API 참조

이 장에 포함된 항목:

- [Java 변환 API 메서드 개요, 226](#)
- [커밋, 227](#)
- [failSession, 228](#)
- [generateRow, 228](#)
- [getInRowType, 229](#)
- [getMetadata, 229](#)
- [incrementErrorCount, 230](#)
- [isNull, 231](#)
- [logError, 231](#)
- [logInfo, 232](#)
- [롤백, 232](#)
- [setNull, 233](#)
- [setOutRowType, 233](#)
- [storeMetadata, 234](#)

Java 변환 API 메서드 개요

Java 변환의 **Java 코드** 탭에서 Java 코드에 API 메서드를 추가하여 변환 동작을 정의할 수 있습니다.

API 메서드를 코드에 추가하려면 코드 항목 탭의 탐색기에서 **호출 가능 API** 목록을 확장한 다음, 코드에 추가할 메서드 이름을 두 번 클릭합니다.

또는 메서드를 탐색기에서 **Java 코드 조각**으로 끌거나 **Java 코드 조각**에 API 메서드를 수동으로 입력할 수 있습니다.

Java 변환에서 다음 API 메서드를 Java 코드에 추가할 수 있습니다.

커밋

트랜잭션을 생성합니다.

failSession

오류 메시지와 함께 예외가 발생하고 세션이 실패합니다.

`generateRow`

활성 Java 변환에 대한 출력 행을 생성합니다.

`getInRowType`

변환에서 현재 행의 입력 유형을 반환합니다.

`incrementErrorCount`

세션에 대한 오류 수를 증가합니다.

`isNull`

입력 열에 Null 값이 있는지 확인합니다.

`logError`

오류 메시지를 세션 로그에 기록합니다.

`logInfo`

정보 메시지를 세션 로그에 기록합니다.

롤백

롤백 트랜잭션을 생성합니다.

`setNull`

활성 또는 수동 Java 변환의 출력 열 값을 Null로 설정합니다.

`setOutRowType`

출력 행에 대한 업데이트 전략을 설정합니다. 삽입, 업데이트 또는 삭제 플래그를 행에 지정할 수 있습니다.

커밋

트랜잭션을 생성합니다.

패키지 가져오기 탭이나 Java 식 코드 항목 탭을 제외한 모든 탭에서 커밋을 사용하십시오. 트랜잭션을 생성하도록 구성된 활성 변환에서만 커밋을 사용할 수 있습니다. 트랜잭션을 생성하도록 구성되지 않은 활성 변환에서 커밋을 사용할 경우 통합 서비스에서 오류가 발생하고 세션이 실패합니다.

다음 구문을 사용합니다.

```
commit();
```

다음 Java 코드를 사용하여 Java 변환에서 처리한 100개 행마다 트랜잭션을 생성한 다음 `rowsProcessed` 카운터를 0으로 설정합니다.

```
if (rowsProcessed==100) {  
    commit();  
    rowsProcessed=0;  
}
```

failSession

오류 메시지와 함께 예외가 발생하고 세션이 실패합니다.

다음 구문을 사용합니다.

```
failSession(String errorMessage);
```

다음 표에는 매개 변수가 설명되어 있습니다.

매개 변수	매개 변수 유형	데이터 유형	설명
errorMessage	입력	문자열	오류 메시지 문자열입니다.

세션을 종료하려면 **failSession** 메서드를 사용하십시오. 코드 항목 탭의 **try/catch** 블록에서 **failSession** 메서드를 사용하지 마십시오.

패키지 가져오기 및 **Java** 식 탭을 제외한 코드 항목 탭에서 Java 코드에 **failSession** 메서드를 추가할 수 있습니다.

다음 Java 코드에서 **input1** 입력 포트를 **Null** 값에 대해 테스트하고 **Null**일 경우 세션을 실패하는 방법을 보여줍니다.

```
if(isNull("input1")) {  
    failSession("Cannot process a null value for port input1.");  
}
```

generateRow

활성 Java 변환에 대한 출력 행을 생성합니다.

다음 구문을 사용합니다.

```
generateRow();
```

generateRow 메서드를 호출할 경우 Java 변환에서 출력 포트 변수의 현재 값을 사용하여 출력 행을 생성합니다. 입력 행에 해당하는 여러 행을 생성할 경우 각 입력 행마다 두 번 이상 **generateRow** 메서드를 호출할 수 있습니다. 활성 Java 변환에서 **generateRow** 메서드를 사용하지 않을 경우 변환에서 출력 행을 생성하지 않습니다.

패키지 가져오기 및 **Java** 식 탭을 제외한 모든 코드 항목 탭에서 **generateRow** 메서드를 Java 코드에 추가할 수 있습니다.

활성 변환에서만 **generateRow** 메서드를 호출할 수 있습니다. 수동 변환에서 **generateRow** 메서드를 호출할 경우 세션에서 오류를 생성합니다.

다음 Java 코드를 사용하여 출력 행을 1개 생성하고, 출력 포트의 값을 수정한 다음, 다른 출력 행을 생성하십시오.

```
// 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

변환에서 현재 행의 입력 유형을 반환합니다. 메서드에서 삽입, 업데이트, 삭제 또는 거부 값을 반환합니다.

다음 구문을 사용합니다.

```
rowType getInRowType();
```

다음 표에는 매개 변수가 설명되어 있습니다.

매개 변수	매개 변수 유형	데이터 유형	설명
rowType	출력	문자열	다음 값 중 하나인 업데이트 전략 유형을 반환합니다. - DELETE - INSERT - REJECT - UPDATE

입력 행에서 코드 항목 탭에서 `getInRowType` 메서드를 Java 코드에 추가할 수 있습니다.

업데이트 전략을 설정하도록 구성된 활성 변환에서 `getInRowType` 메서드를 사용할 수 있습니다. 업데이트 전략을 설정하도록 구성되지 않은 활성 변환에서 이 메서드를 호출할 경우 세션에서 오류를 생성합니다.

다음 Java 코드를 사용하여 다음 작업을 수행하십시오.

- 현재 입력 행 유형을 출력 행에 전달합니다.
- `input1` 입력 포트의 값이 100보다 큰 경우 출력 행 유형을 DELETE로 설정합니다.

```
// 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

런타임 시 Java 변환 메타데이터를 검색합니다. `getMetadata` 메서드가 `storeMetadata` 메서드와 함께 저장한 메타데이터를 검색합니다(예: 최적화 프로그램이 `pushFilter` 함수에서 Java 변환에 전달하는 필터 조건).

다음 구문을 사용합니다.

```
getMetadata (String key);
```

다음 표에는 매개 변수가 설명되어 있습니다.

매개 변수	매개 변수 유형	데이터 유형	설명
키	입력	문자열	메타데이터를 식별합니다. getMetadata 메서드는 키를 사용하여 검색할 메타데이터를 결정합니다.

다음 코드 항목 탭에서 getMetadata 메서드를 Java 코드에 추가할 수 있습니다.

- 도우미
- 입력 시
- 끝에서
- 최적화 프로그램 인터페이스
- 함수

푸시인 최적화에 대한 필터 조건을 검색하도록 getMetadata 메서드를 구성할 수 있습니다. getMetadata 메서드가 최적화 프로그램에서 사용자가 저장한 각 필터 조건을 검색할 수 있습니다.

```
// Retrieve a filter condition
String mydata = getMetadata ("FilterKey");
```

incrementErrorCount

세션에 대한 오류 수를 증가합니다. 오류 수가 세션에 대한 오류 임계값에 도달한 경우 세션이 실패합니다.

다음 구문을 사용합니다.

```
incrementErrorCount(int nErrors);
```

다음 표에는 매개 변수가 설명되어 있습니다.

매개 변수	매개 변수 유형	데이터 유형	설명
nErrors	입력	정수	세션에 대한 오류 수가 증가하는 숫자입니다.

패키지 가져오기 및 **Java 식** 탭을 제외한 코드 항목 탭에서 Java 코드에 incrementErrorCount 메서드를 추가할 수 있습니다.

다음 Java 코드는 변환의 입력 포트에 Null 값이 있을 경우 오류 수를 증가하는 방법을 보여 줍니다.

```
// 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;
}
```

isNull

입력 열 값에 Null 값이 있는지 확인합니다.

다음 구문을 사용합니다.

```
Boolean isNull(String strColName);
```

다음 표에는 매개 변수가 설명되어 있습니다.

매개 변수	매개 변수 유형	데이터 유형	설명
strColName	입력	문자열	입력 열의 이름입니다.

입력 행에서 코드 항목 탭에서 isNull 메서드를 Java 코드에 추가할 수 있습니다.

다음 Java 코드에서는 SALARY 입력 열 값을 totalSalaries 인스턴스 변수에 추가하기 전에 해당 값이 Null인지 확인하는 방법을 보여 줍니다.

```
// if value of SALARY is not null
if (!isNull("SALARY")) {
    // add to totalSalaries
    TOTAL_SALARIES += SALARY;
}
```

또는 다음 Java 코드를 사용하여 동일한 결과를 얻으십시오.

```
// if value of SALARY is not null
String strColName = "SALARY";
if (!isNull(strColName)) {
    // add to totalSalaries
    TOTAL_SALARIES += SALARY;
}
```

logError

오류 메시지를 세션 로그에 기록합니다.

다음 구문을 사용합니다.

```
logError(String msg);
```

다음 표에는 매개 변수가 설명되어 있습니다.

매개 변수	매개 변수 유형	데이터 유형	설명
msg	입력	문자열	오류 메시지 문자열입니다.

패키지 가져오기 및 **Java 식** 탭을 제외한 모든 코드 항목 탭에서 logError 메서드를 Java 코드에 추가할 수 있습니다.

다음 Java 코드에서는 입력 포트가 Null일 경우 오류를 로그하는 방법을 보여 줍니다.

```
// check BASE_SALARY
if (isNull("BASE_SALARY")) {
    logError("Cannot process a null salary field.");
}
```

코드가 실행되면 다음 메시지가 세션 로그에 표시됩니다.

```
[JTX_1013] [ERROR] Cannot process a null salary field.
```

logInfo

정보 메시지를 세션 로그에 기록합니다.

다음 구문을 사용합니다.

```
logInfo(String msg);
```

다음 표에는 매개 변수가 설명되어 있습니다.

매개 변수	매개 변수 유형	데이터 유형	설명
msg	입력	문자열	정보 메시지 문자열입니다.

패키지 가져오기 및 **Java** 식 탭을 제외한 코드 항목 탭에서 Java 코드에 logInfo 메서드를 추가할 수 있습니다.

다음 Java 코드에서는 Java 변환이 1000개 행의 메시지 임계값을 처리한 후 세션 로그에 메시지를 기록하는 방법을 보여 줍니다.

```
if (numRowsProcessed == messageThreshold) {  
    logInfo("Processed " + messageThreshold + " rows.");  
}
```

롤백

롤백 트랜잭션을 생성합니다.

패키지 가져오기 또는 Java 식 코드 항목 탭을 제외한 탭에서 롤백을 사용합니다. 트랜잭션을 생성하도록 구성된 활성 변환에서만 롤백을 사용할 수 있습니다. 트랜잭션을 생성하도록 구성되지 않은 활성 변환에서 롤백을 사용하는 경우 통합 서비스가 오류를 생성하고 세션을 실행하지 못합니다.

다음 구문을 사용합니다.

```
rollback();
```

다음 코드를 사용하여 입력 행에 잘못된 조건이 있으면 롤백 트랜잭션을 생성하고 세션을 실행시키지 않거나 처리할 행 수가 100이면 트랜잭션을 생성합니다.

```
// 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

활성 또는 수동 Java 변환에서 출력 열의 값을 null로 설정합니다.

다음 구문을 사용합니다.

```
setNull(String strColName);
```

다음 표에는 매개 변수가 설명되어 있습니다.

매개 변수	매개 변수 유형	데이터 유형	설명
strColName	입력	문자열	출력 열의 이름입니다.

setNull 메서드는 활성 또는 수동 Java 변환에서 출력 열의 값을 null로 설정합니다. 출력 열을 null로 설정한 후에는 출력 행을 생성할 때까지 값을 수정할 수 없습니다.

패키지 가져오기 및 **Java 식** 탭을 제외한 모든 코드 입력 탭에서 Java 코드에 setNull 메서드를 추가할 수 있습니다.

다음 Java 코드는 입력 열 값을 확인하고 출력 열의 해당 값을 null로 설정하는 방법을 보여 줍니다.

```
// check value of Q3RESULTS input column
if(isNull("Q3RESULTS")) {
    // set the value of output column to null
    setNull("RESULTS");
}
```

다음 Java 코드를 사용해도 같은 결과를 얻을 수 있습니다.

```
// check value of Q3RESULTS input column
String strColName = "Q3RESULTS";
if(isNull(strColName)) {
    // set the value of output column to null
    setNull(strColName);
}
```

setOutRowType

출력 행에 대한 업데이트 전략을 설정합니다. setOutRowType 메서드는 삽입, 업데이트 또는 삭제를 위해 행에 플래그를 지정할 수 있습니다.

입력 행에서 코드 항목 탭에서만 setOutRowType을 사용할 수 있습니다. 업데이트 전략을 설정하도록 구성된 활성 변환에서만 setOutRowType을 사용할 수 있습니다. 업데이트 전략을 설정하도록 구성되지 않은 활성 변환에서 setOutRowType을 사용하는 경우 세션이 오류를 생성하고 세션이 실패합니다.

다음 구문을 사용합니다.

```
setOutRowType(String rowType);
```

다음 테이블에는 이 메서드의 인수가 설명되어 있습니다.

인수	데이터 유형	입력/출력	설명
rowType	문자열	입력	업데이트 전략 유형. 값은 INSERT, UPDATE 또는 DELETE 일 수 있습니다.

다음 Java 코드를 사용하여 행 유형이 UPDATE 또는 INSERT이고 입력 포트 input1의 값이 100 미만이면 현재 행의 입력 유형을 전파하고 input1의 값이 100보다 크면 출력 유형을 DELETE로 설정합니다.

```
// 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

getMetadata 메서드를 사용하여 런타임에 검색할 수 있는 Java 변환 메타데이터를 저장합니다.

다음 구문을 사용합니다.

```
storeMetadata (String key String data);
```

다음 표에는 매개 변수가 설명되어 있습니다.

매개 변수	매개 변수 유형	데이터 유형	설명
키	입력	문자열	메타데이터를 식별합니다. storeMetadata 메서드를 사용하려면 메타데이터를 식별할 키가 필요합니다. 키를 임의의 문자열로 정의합니다.
데이터	입력	문자열	Java 변환 메타데이터로 저장할 데이터입니다.

storeMetadata 메서드를 Java 코드의 다음 코드 입력 탭에 추가할 수 있습니다.

- 도우미
- 입력 시
- 끝에서
- 최적화 프로그램 인터페이스
- 함수

푸시인 최적화를 위한 필터 조건을 허용하도록 활성 변환에서 storeMetadata 메서드를 구성할 수 있습니다. storeMetadata 메서드는 최적화 프로그램이 매핑에서 Java 변환으로 푸시하는 필터 조건을 저장합니다.

```
// Store a filter condition
storeMetadata ("FilterKey", condition);
```

제 14 장

Java 식

이 장에 포함된 항목:

- [Java 식 개요, 235](#)
- [식 정의 대화 상자를 사용하여 식 정의, 236](#)
- [단순 인터페이스 작업, 238](#)
- [고급 인터페이스 작업, 239](#)
- [JExpression 클래스 API 참조, 243](#)

Java 식 개요

Java 프로그래밍 언어를 사용하여 Java 변환에서 PowerCenter 식을 호출할 수 있습니다.

식을 사용하면 Java 변환의 기능을 확대할 수 있습니다. 예를 들어 Java 변환에서 식을 호출하여 입력 또는 출력 포트의 값을 조회하거나 Java 변환 변수의 값을 조회할 수 있습니다.

Java 변환에서 식을 호출하려면 Java 코드를 생성하거나 Java 변환 API 메서드를 사용하여 식을 호출합니다. 식을 호출한 다음 식의 결과를 적절한 코드 항목 탭에 사용합니다. 식을 호출하는 Java 코드를 생성하거나 API 메서드를 사용하여 식을 호출하는 Java 코드를 쓸 수 있습니다.

다음 표에는 Java 변환에서 식을 작성하고 호출할 때 사용할 수 있는 방법이 설명되어 있습니다.

메서드	설명
식 정의 대화 상자	식을 작성하고 식의 코드를 생성할 수 있습니다.
단순 인터페이스	단일 API 메서드를 호출하여 식을 호출하고 식의 결과를 얻을 수 있습니다.
고급 인터페이스	식을 정의하고 식을 호출하고 식의 결과를 사용할 수 있습니다. 개체 중심 프로그래밍에 익숙한 경우 식 호출에 대한 더 많은 제어 기능을 사용하려면 고급 인터페이스를 사용하십시오.

식 함수 유형

식 편집기를 사용하거나 식 정의 대화 상자에서 식을 작성하거나 단순 또는 고급 인터페이스를 사용하여 Java 변환에 대한 식을 작성할 수 있습니다.

Java 코드의 입력 또는 출력 포트 변수를 입력 매개 변수로 사용하는 식을 입력할 수 있습니다.

식 정의 대화 상자를 사용할 경우 **식 편집기**를 사용하여 **Java** 변환에서 식을 사용하기 전에 식의 유효성을 검사할 수 있습니다.

Java 변환에서 다음 유형의 식 함수를 호출할 수 있습니다.

식 함수 유형	설명
변환 언어 함수	일반 식을 처리하도록 설계된 SQL 같은 함수입니다.
사용자 정의 함수	변환 언어 함수를 기반으로 PowerCenter에서 작성하는 함수입니다.
사용자 지정 함수	사용자 지정 함수 API를 사용하여 작성한 함수입니다.

연결되지 않은 변환, 기본 제공 변수, 사용자 정의 매핑 및 워크플로우 변수, 미리 정의된 워크플로우 변수를 식에서 사용할 수도 있습니다. 예를 들어, 식에서 연결되지 않은 조회 변환을 사용할 수 있습니다.

식 정의 대화 상자를 사용하여 식 정의

Java 식을 정의하는 경우 함수를 구성하고 식을 작성한 다음 식을 호출하는 코드를 생성합니다.

식 정의 대화 상자에서 함수를 정의하고 식을 작성할 수 있습니다.

식 함수를 작성하고 Java 변환에서 식을 사용하려면 다음 고급 탭스크를 완료합니다.

1. 함수 이름, 설명 및 매개 변수를 포함하여 식을 호출하는 함수를 구성합니다. 식을 작성할 때 함수 매개 변수를 사용합니다.
2. 식 구문을 작성하고 식의 유효성을 검증합니다.
3. 식을 호출하는 Java 코드를 생성합니다.

디자이너가 변환 개발자의 **Java 식** 코드 항목 탭에 코드를 배치합니다.

Java 코드를 생성한 다음 적절한 코드 항목 탭에서 생성된 함수를 호출하여 식을 호출하거나 단순 또는 고급 인터페이스를 사용하는지 여부에 따라 **JExpression** 개체를 가져옵니다.

참고: 식을 작성할 때 식의 유효성을 검증하려면 **식 정의** 대화 상자를 사용해야 합니다.

1단계. 함수 구성

식을 호출하는 Java 함수에 대해 함수 이름, 설명 및 입력 매개 변수를 구성합니다.

함수를 구성할 때 다음 규칙과 지침을 따르십시오.

- 변환의 기존 **Java** 함수 또는 예약된 **Java** 키워드와 충돌하지 않는 고유한 함수 이름을 사용합니다.
- 매개 변수 이름, **Java** 데이터 유형, 전체 자릿수 및 소수 자릿수를 구성해야 합니다. 입력 매개 변수는 변환을 위해 **Java** 코드의 함수를 호출할 때 전달하는 값입니다.
- 날짜 데이터 유형을 식으로 전달하려면 입력 매개 변수에 문자열 데이터 유형을 사용합니다.

식이 날짜 데이터 유형을 반환하는 경우 단순 인터페이스에서는 반환 값을 문자열 데이터 유형으로 사용하고 고급 인터페이스에서는 문자열 또는 긴 정수 데이터 유형을 사용할 수 있습니다.

2단계. 식 작성 및 유효성 검사

식을 작성하는 경우 함수에 대해 구성된 매개 변수를 사용합니다.

또한 식에서 변환 언어 함수, 사용자 지정 함수 또는 다른 사용자 정의 함수를 사용할 수 있습니다. **식 정의** 대화 상자 또는 **식 편집기** 대화 상자에서 식을 작성하고 유효성을 검사할 수 있습니다.

3단계. 식에 대한 Java 코드 생성

함수 및 함수 매개 변수를 구성하여 식을 정의하고 유효성을 검사한 후 식을 호출하는 **Java** 코드를 생성할 수 있습니다.

디자이너가 생성된 **Java** 코드를 **Java 식** 코드 항목 탭에 배치합니다. 생성된 **Java** 코드를 사용하여 변환 개발자의 코드 항목 탭에서 식을 호출하는 함수를 호출합니다. 단순 또는 고급 **Java** 코드를 생성할 수 있습니다.

식을 호출하는 **Java** 코드를 생성한 다음에는 식을 편집하거나 유효성을 다시 검사할 수 없습니다. 코드를 생성한 다음 식을 수정하려면 식을 다시 작성해야 합니다.

식 정의 대화 상자를 사용한 식 작성 및 Java 코드 생성

식 정의 대화 상자에서 식을 호출하는 함수를 작성할 수 있습니다.

식을 호출하는 함수를 작성하려면 다음 단계를 완료하십시오.

1. 변환 개발자에서 **Java** 변환을 열거나 새 **Java** 변환을 작성합니다.
2. **Java 코드** 탭에서 **식 정의** 링크를 클릭합니다.
식 정의 대화 상자가 표시됩니다.
3. 함수 이름을 입력합니다.
4. 필요한 경우 식의 설명을 입력합니다.
최대 2,000자까지 입력할 수 있습니다.
5. 함수에 대한 매개 변수를 작성합니다.
매개 변수를 작성할 때 매개 변수 이름, 데이터 유형, 전체 자릿수 및 배율을 구성합니다.
6. **편집기 실행**을 클릭하여 작성한 매개 변수로 식을 작성합니다.
7. 식의 유효성을 검사하려면 **유효성 검사**를 클릭합니다.
8. 필요한 경우 **식** 상자에 식을 입력합니다. 그런 다음 **유효성 검사**를 클릭하여 식의 유효성을 검사합니다.
9. 고급 인터페이스를 사용하여 **Java** 코드를 생성하려면 **고급 코드 생성** 옵션을 선택합니다. 그런 다음 **생성**을 클릭합니다.

Designer가 식을 호출하는 함수를 **Java 식** 코드 항목 탭에 생성합니다.

Java 식 템플릿

식에 대한 단순 또는 고급 **Java** 코드를 사용하여 식에 대한 **Java** 코드를 생성할 수 있습니다.

식의 템플릿을 기반으로 식에 대한 **Java** 코드가 생성됩니다.

다음 예제에는 단순 **Java** 코드에 생성된 **Java** 식 템플릿이 표시되어 있습니다.

```
Object function_name (Java datatype x1[,  
                        Java datatype x2 ...] )  
                        throws SDK Exception  
{  
    return (Object)invokeJExpression( String expression,  
                                     new Object [] { x1[, x2, ... ]} );  
}
```

다음 예제에는 고급 인터페이스를 사용하여 생성된 **Java** 식 템플릿이 표시되어 있습니다.

```

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);
}

```

단순 인터페이스 작업

`invokeJExpression` Java API 메서드를 사용하여 단순 인터페이스에서 식을 호출할 수 있습니다.

invokeJExpression

식을 호출하고 식의 값을 반환합니다.

다음 구문을 사용합니다.

```

(datatype)invokeJExpression(
    String expression,
    Object[] paramMetadataArray);

```

`invokeJExpression` 메서드의 입력 매개 변수는 식을 나타내는 문자열 값과 식 입력 매개 변수를 포함하는 개체의 배열입니다.

다음 표에는 매개 변수가 설명되어 있습니다.

매개 변수	매개 변수 유형	데이터 유형	설명
식	입력	문자열	식을 나타내는 문자열입니다.
paramMetadataArray	입력	Object[]	식의 입력 매개 변수를 포함하는 개체의 배열입니다.

패키지 가져오기 및 **Java** 식 탭을 제외한 모든 코드 항목 탭의 **Java** 코드에 `invokeJExpression` 메서드를 추가할 수 있습니다.

다음 규칙 및 지침에 따라 `invokeJExpression` 메서드를 사용하십시오.

- **반환 데이터 유형.** `invokeJExpression` 메서드의 반환 데이터 유형은 개체입니다. 함수의 반환 값을 적절한 데이터 유형으로 캐스팅해야 합니다.

정수, 배열, 문자열 및 바이트[] 데이터 유형으로 값을 반환할 수 있습니다.

- **행 유형.** `invokeJExpression` 메서드의 반환 값에 대한 행 유형은 `INSERT`입니다.

반환 값에 서로 다른 행 유형을 사용하려면 고급 인터페이스를 사용합니다.

- Null 값. Null 값을 매개 변수로 전달하거나 `invokeJExpression` 메서드의 반환 값이 NULL인 경우 이 값은 Null 표시기로 처리됩니다.
예를 들어 식의 반환 값이 NULL이고 반환 데이터 유형이 문자열인 경우 Null 값과 함께 문자열이 반환됩니다.
 - 날짜 데이터 유형. 날짜 데이터 유형의 입력 매개 변수는 문자열 데이터 유형으로 변환해야 합니다.
식의 문자열을 날짜 데이터 유형으로 사용하려면 `to_date()` 함수를 사용하여 문자열을 날짜 데이터 유형으로 변환해야 합니다.
또한 날짜 데이터 유형을 반환하는 모든 식의 반환 유형을 문자열 데이터 유형으로 캐스팅해야 합니다.
- 참고:** 식에 전달하는 매개 변수에는 문자 `x`로 시작되는 연속 번호를 지정해야 합니다. 예를 들어 매개 변수 3개를 식에 전달하는 경우 매개 변수의 이름을 `x1`, `x2` 및 `x3`으로 지정합니다.

단순 인터페이스 예제

도우미 코드 및 입력 행에서 코드 항목 탭에서 `invokeJExpression` API 메서드를 사용하는 식을 정의하고 호출할 수 있습니다.

다음 예는 Java 변환에서 NAME 및 ADDRESS 입력 포트에 대해 조회를 완료하고 반환 값을 COMPANY_NAME 출력 포트에 할당하는 방법을 보여줍니다.

입력 행에서 코드 항목 탭에서 다음 코드를 입력합니다.

```
COMPANY_NAME = (String)invokeJExpression(":lkp.my_lookup(X1,X2)", new Object [] {str1 ,str2} );
generateRow();
```

고급 인터페이스 작업

고급 인터페이스에서는 개체 지향 API 메서드를 사용하여 식을 정의 및 호출하고 식 결과를 가져올 수 있습니다.

다음 테이블에는 고급 인터페이스에서 사용할 수 있는 클래스 및 API 메서드가 설명되어 있습니다.

클래스 또는 API 메서드	설명
EDatatype 클래스	식의 데이터 유형을 열거합니다.
JExprParamMetadata 클래스	식의 각 매개 변수에 대한 메타데이터를 포함합니다. 매개 변수 메타데이터에는 데이터 유형, 전체 자릿수 및 배열이 포함됩니다.
defineJExpression API 메서드	식을 정의합니다. PowerCenter 식 문자열 및 매개 변수를 포함합니다.
invokeJExpression API 메서드	식을 호출합니다.
JExpression 클래스	메타데이터 생성/호출/가져오기, 식 결과 가져오기, 반환 데이터 유형 확인을 위한 메서드를 포함합니다.

고급 인터페이스를 사용하여 식 호출

고급 인터페이스를 사용하여 식을 정의하고 호출하고 결과를 얻을 수 있습니다.

1. **도우미 코드** 또는 **입력 행에서** 코드 항목 탭에 식의 각 매개 변수에 대한 `JExprParamMetadata` 클래스의 인스턴스를 작성하고 메타데이터의 값을 설정합니다. 필요한 경우 `defineJExpression` 메서드의 `JExprParamMetadata` 개체를 인스턴스화할 수 있습니다.
2. `defineJExpression` 메서드를 사용하여 식의 `JExpression` 개체를 가져옵니다.
3. 적절한 코드 항목 탭에서 `invokeJExpression` 메서드를 사용하여 식을 호출합니다.
4. `isResultNull` 메서드를 사용하여 반환 값의 결과를 검사합니다.
5. `getResultDataType` 및 `getResultMetadata` 메서드를 사용하면 반환 값의 데이터 유형 또는 메타데이터를 가져올 수 있습니다.
6. 적절한 API 메서드를 사용하여 식의 결과를 가져옵니다. `getInt`, `getDouble`, `getStringBuffer` 및 `getBytes` 메서드를 사용할 수 있습니다.

고급 인터페이스 작업에 대한 규칙 및 지침

고급 인터페이스 작업을 수행할 때 규칙 및 지침을 알고 있어야 합니다.

다음 규칙 및 지침을 사용하십시오.

- **NULL** 값을 매개 변수로 전달하거나 식의 결과가 **NULL**인 경우 값이 **NULL** 표시기로 처리됩니다. 예를 들어 식의 결과가 **NULL**이고 반환 데이터 유형이 문자열인 경우 문자열이 **NULL** 값으로 반환됩니다. `isResultNull` 메서드를 사용하면 식의 결과를 확인할 수 있습니다.
- 식에서 사용할 수 있으려면 날짜 데이터 유형의 입력 매개 변수를 문자열로 변환해야 합니다. 식의 문자열을 날짜 데이터 유형으로 사용하려면 `to_date()` 함수를 사용하여 문자열을 날짜 데이터 유형으로 변환해야 합니다.

날짜 데이터 유형을 문자열 또는 긴 정수 데이터 유형으로 변환하는 식 결과를 얻을 수 있습니다.

날짜 데이터 유형을 문자열 데이터 유형으로 변환하는 식 결과를 얻으려면 `getStringBuffer` 메서드를 사용합니다. 날짜 데이터 유형을 긴 정수 데이터 유형으로 변환하는 식 결과를 얻으려면 `getLong` 메서드를 사용합니다.

EDatatype 클래스

식에 사용된 **Java** 데이터 유형을 열거합니다. 식의 반환 데이터 유형을 가져오거나 `JExprParamMetadata` 개체의 매개 변수에 대한 데이터 유형을 할당합니다. `EDatatype` 클래스는 인스턴스화가 필요하지 않습니다.

다음 표에는 식의 **Java** 데이터 유형에 대한 열거 값이 나열되어 있습니다.

데이터 유형	열거 값
INT	1
DOUBLE	2
STRING	3
BYTE_ARRAY	4
DATE_AS_LONG	5

다음 예는 `EDatatype` 클래스를 사용하여 문자열 데이터 유형을 `JExprParamMetadata` 개체에 할당하는 Java 코드를 보여 줍니다.

```
JExprParamMetadata params[] = new JExprParamMetadata[2];
params[0] = new JExprParamMetadata (
    EDatatype.STRING, // data type
    20, // precision
    0 // scale
);
...
```

JExprParamMetadata 클래스

식의 매개 변수를 나타내고 매개 변수에 대한 메타데이터를 설정하는 개체를 인스턴스화합니다.

`JExprParamMetadata` 개체의 배열을 `defineJExpression` 메서드에 대한 입력으로 사용하여 입력 매개 변수에 대한 메타데이터를 설정하십시오. `JExprParamMetadata` 개체의 인스턴스는 **Java** 식 코드 항목 탭 또는 `defineJExpression`에서 작성할 수 있습니다.

다음 구문을 사용합니다.

```
JExprParamMetadata paramMetadataArray[] = new JExprParamMetadata[numberOfParameters];
paramMetadataArray[0] = new JExprParamMetadata(datatype, precision, scale);
...
paramMetadataArray[numberOfParameters - 1] = new JExprParamMetadata(datatype, precision, scale);
```

다음 표에는 인수가 설명되어 있습니다.

인수	인수 유형	인수 데이터 유형	설명
데이터 유형	입력	<code>EDatatype</code>	매개 변수의 데이터 유형입니다.
전체 자릿수	입력	정수	매개 변수의 전체 자릿수입니다.
배율	입력	정수	매개 변수의 배율입니다.

예를 들어 데이터 유형이 문자열이고 전체 자릿수가 20이며 배율이 0인 `JExprParamMetadata` 개체 2개의 배열을 인스턴스화하려면 다음 Java 코드를 사용합니다.

```
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

식 문자열 및 입력 매개 변수를 포함하여 식을 정의합니다. `defineJExpression` 메서드의 인수에는 입력 매개 변수와 식 구문을 정의하는 문자열 값을 포함하는 `JExprParamMetadata` 개체의 배열이 포함됩니다.

다음 구문을 사용합니다.

```
defineJExpression(
    String expression,
    Object[] paramMetadataArray
);
```

다음 표에는 매개 변수가 설명되어 있습니다.

매개 변수	유형	데이터 유형	설명
식	입력	문자열	식을 나타내는 문자열입니다.
paramMetadataArray	입력	Object[]	식의 입력 매개 변수를 포함하는 JExprParamMetadata 개체의 배열입니다.

defineJExpression 메서드를 사용하려면 식의 입력 매개 변수를 나타내는 **JExprParamMetadata** 개체의 배열을 인스턴스화해야 합니다. 매개 변수에 대한 메타데이터 값을 설정하고 배열을 **defineJExpression** 메서드에 매개 변수로 전달하십시오.

예를 들어 다음 **Java** 코드는 두 문자열의 값을 조화하는 식을 작성합니다.

```
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);
```

참고: 식에 전달하는 매개 변수에는 문자 **x**로 시작되는 연속 번호를 지정해야 합니다. 예를 들어 매개 변수 3개를 식에 전달하는 경우 매개 변수의 이름을 **x1**, **x2** 및 **x3**으로 지정합니다.

JExpression 클래스

식을 작성 및 호출하고 식의 값을 반환하고 반환 데이터 유형을 검사하는 메서드가 포함됩니다.

다음 표에는 **JExpression** 클래스의 메서드가 나열되어 있습니다.

메서드 이름	설명
invoke	식을 호출합니다.
getResultDataType	식 결과의 데이터 유형을 반환합니다.
getResultMetadata	식 결과의 메타데이터를 반환합니다.
isResultNull	식 결과의 결과 값을 검사합니다.
getInt	식의 결과 값을 정수 데이터 유형으로 반환합니다.
getDouble	식의 결과 값을 배정밀도 데이터 유형으로 반환합니다.
getStringBuffer	식의 결과 값을 문자열 데이터 유형으로 반환합니다.
getBytes	식의 결과 값을 바이트[] 데이터 유형으로 반환합니다.

관련 항목:

- [“JExpression 클래스 API 참조” 페이지 243](#)

고급 인터페이스 예제

고급 인터페이스를 사용하여 Java 변환에서 조회 식을 작성하고 호출할 수 있습니다.

다음 예는 식을 호출하는 함수를 작성하고 식을 호출하여 반환 값을 얻는 Java 코드를 보여 줍니다. 이 예에서는 데이터 유형이 문자열인 두 입력 포트 **NAME** 및 **COMPANY**의 값이 **myLookup** 함수에 전달됩니다. **myLookup** 함수는 조회 식을 사용하여 **ADDRESS** 출력 포트에 대한 값을 조회합니다.

참고: 이 예에서는 **LKP_addresslookup**이라는 맵에 연결되지 않은 조회 변환이 있는 것으로 가정합니다.

도우미 코드 탭(변환 개발자)에서 다음 Java 코드를 사용합니다.

```
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;
```

입력 행에서 탭에 다음 Java 코드를 사용하면 식이 호출되고 **ADDRESS** 포트의 값이 반환됩니다.

```
...
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 클래스 API 참조

JExpression 클래스에는 식 작성 및 호출, 식의 값 반환 및 반환 데이터 유형 검사를 위한 API 메서드가 포함됩니다.

JExpression 클래스 포함되는 API 메서드는 다음과 같습니다.

- `getBytes`
- `getDouble`

- getInt
- getLong
- getResultDataType
- getResultMetadata
- getStringBuffer
- invoke
- isResultNull

getBytes

식의 결과 값을 바이트[] 데이터 유형으로 반환합니다. AES_ENCRYPT 함수로 데이터를 암호화하는 식의 결과를 가져옵니다.

다음 구문을 사용합니다.

```
objectName.getBytes();
```

다음 예는 AES_ENCRYPT 함수를 사용하여 이진 데이터를 암호화하는 식의 결과를 가져오는 Java 코드입니다. 여기에서 JExprEncryptData는 JExpression 개체입니다.

```
byte[] newBytes = JExprEncryptData.getBytes();
```

getDouble

식의 결과 값을 배정밀도 데이터 유형으로 반환합니다.

다음 구문을 사용합니다.

```
objectName.getDouble();
```

다음 예는 급여 값을 배정밀도로 반환하는 식의 결과를 가져오는 Java 코드입니다. 여기에서 JExprSalary는 JExpression 개체입니다.

```
double salary = JExprSalary.getDouble();
```

getInt

식의 결과 값을 정수 데이터 유형으로 반환합니다.

다음 구문을 사용합니다.

```
objectName.getInt();
```

예를 들어 다음 Java 코드를 사용하면 직원 ID 번호를 정수로 반환하는 식의 결과를 얻을 수 있습니다. 여기에서 findEmpID는 JExpression 개체입니다.

```
int empID = findEmpID.getInt();
```

getLong

식 결과 값을 긴 정수 데이터 유형으로 반환합니다. 날짜 데이터 유형을 사용하는 식 결과를 가져옵니다.

다음 구문을 사용합니다.

```
objectName.getLong();
```


날짜 값을 긴 정수 데이터 유형으로 반환하는 식 결과를 가져오려면 다음 예제 Java 코드를 사용하십시오. 여기서 `JExprCurrentDate`는 `JExpression` 개체입니다.

```
long currDate = JExprCurrentDate.getLong();
```

getResultDataType

식 결과의 데이터 유형을 반환합니다. `EDataType` 값을 반환합니다.

다음 구문을 사용합니다.

```
objectName.getResultDataType();
```

다음 예는 식을 호출하고 결과의 데이터 유형을 변수 데이터 유형에 할당하는 Java 코드를 보여 줍니다.

```
myObject.invoke(new Object[] { NAME,COMPANY }, ERowType INSERT);  
EDataType dataType = myObject.getResultDataType();
```

getResultMetadata

식 결과의 메타데이터를 반환합니다. `getResultMetadata`를 사용하여 식 결과의 전체 자릿수, 배열 및 데이터 유형을 가져올 수 있습니다. 식의 반환 값의 메타데이터를 `JExprParamMetadata` 개체에 할당할 수 있습니다. 결과 메타데이터를 검색하려면 `getScale`, `getPrecision` 및 `getDataType` 개체를 사용합니다.

다음 구문을 사용합니다.

```
objectName.getResultMetadata();
```

다음 예는 `myObject`의 반환 값의 배열, 전체 자릿수 및 데이터 유형을 변수에 할당하는 Java 코드를 보여 줍니다.

```
JExprParamMetadata myMetadata = myObject.getResultMetadata();  
int scale = myMetadata.getScale();  
int prec = myMetadata.getPrecision();  
int datatype = myMetadata.getDataType();
```

참고: `getDataType` 개체 메서드는 데이터 유형의 정수 값을 `EDataType`으로 열거하여 반환합니다.

getStringBuffer

식의 결과 값을 문자열 데이터 유형으로 반환합니다.

다음 구문을 사용합니다.

```
objectName.getStringBuffer();
```

두 개의 연결된 문자열을 반환하는 식 결과를 얻으려면 다음 예제 Java 코드를 사용하십시오. 여기서 `JExprConcat`는 `JExpression` 개체입니다.

```
String result = JExprConcat.getStringBuffer();
```

invoke

식을 호출합니다. `invoke`의 인수에는 입력 매개 변수 및 행 유형을 정의하는 개체가 포함됩니다. `invoke` 메서드를 사용하려면 먼저 `JExpression` 개체를 인스턴스화해야 합니다. 행 유형으로는 `ERowType.INSERT`, `ERowType.DELETE` 및 `ERowType.UPDATE`를 사용합니다.

다음 구문을 사용합니다.

```
objectName.invoke(  
    new Object[] { param1[, ... paramN ]},  
    rowType  
);
```

다음 표에는 인수가 설명되어 있습니다.

인수	데이터 유형	입력/ 출력	설명
objectName	JExpression	입력	JExpression 개체 이름입니다.
매개 변수	-	입력	식의 입력 값을 포함하는 개체 배열입니다.

예를 들어 식을 나타내는 JExpression 개체를 반환하는 address_lookup()이라는 **Java** 식 코드 항목 탭에 함수를 작성합니다. 다음 코드를 사용하면 입력 포트 NAME 및 COMPANY를 사용하는 식이 호출됩니다.

```
JExpression myObject = address_lookup();  
myObject.invoke(new Object[] { NAME,COMPANY }, ERowType INSERT);
```

isResultNull

식 결과의 값을 검사합니다.

다음 구문을 사용합니다.

```
objectName.isResultNull();
```

다음 예는 식을 호출하고 식의 반환 값이 Null이 아닐 경우 반환 값을 변수 주소에 할당하는 **Java** 코드입니다.

```
JExpression myObject = address_lookup();  
myObject.invoke(new Object[] { NAME,COMPANY }, ERowType INSERT);  
if(!myObject.isResultNull()) {  
    String address = myObject.getStringBuffer();  
}
```

제 15 장

Java 변환 예제

이 장에 포함된 항목:

- [Java 변환 예제 개요, 247](#)
- [1단계. 매핑 가져오기, 248](#)
- [2단계. 변환 작성 및 포트 구성, 248](#)
- [3단계. Java 코드 입력, 249](#)
- [패키지 가져오기 탭, 249](#)
- [도우미 코드 탭, 249](#)
- [입력 행에서 탭, 250](#)
- [4단계. Java 코드 컴파일, 251](#)
- [5단계. 세션 및 워크플로우 작성, 252](#)

Java 변환 예제 개요

이 예제의 **Java** 코드를 사용하여 활성 **Java** 변환을 작성하고 컴파일할 수 있습니다. 샘플 매핑을 가져오고 **Java** 변환을 작성하고 컴파일합니다. 그런 다음 매핑이 포함된 세션 및 워크플로우를 작성하고 실행할 수 있습니다.

Java 변환에서 가상의 회사에 대한 직원 데이터를 처리합니다. 플랫폼 파일 소스에서 입력 행을 읽고 플랫폼 파일 대상에 출력 행을 씁니다. 소스 파일에는 직원 ID 번호, 이름, 직위 및 관리자 ID 번호를 비롯한 직원 데이터가 포함되어 있습니다.

변환에서 관리자 ID 번호에 따라 지정된 직원의 관리자 이름을 찾고 직원 데이터가 포함된 출력 행을 생성합니다. 출력 데이터에는 직원 ID 번호, 이름, 직위 및 직원의 관리자 이름이 포함되어 있습니다. 소스 데이터에 관리자가 없는 직원의 경우 변환에서는 이 직원이 회사 조직도에서 최상위 계층에 있는 결로 가정합니다.

참고: 변환 논리에서는 직원 직위가 소스 파일에서 내림차순으로 정렬된 것으로 가정합니다.

샘플 매핑을 가져오고, **Java** 변환을 작성 및 컴파일하고, 매핑이 포함된 워크플로우 및 세션을 작성하려면 다음 단계를 완료합니다.

1. 샘플 매핑을 가져옵니다.
2. **Java** 변환을 작성하고 **Java** 변환 포트를 구성합니다.
3. 해당하는 코드 항목 탭에서 변환을 위한 **Java** 코드를 입력합니다.
4. **Java** 코드를 컴파일합니다.
5. 세션 및 워크플로우를 작성하고 실행합니다.

PowerCenter 클라이언트 설치에는 이 예제와 함께 사용할 수 있는 매핑, `m_jtx_hier_useCase.xml` 및 플랫폼 파일 소스, `hier_data`가 포함되어 있습니다.

관련 항목:

- [“Java 변환” 페이지 211](#)

1단계. 매핑 가져오기

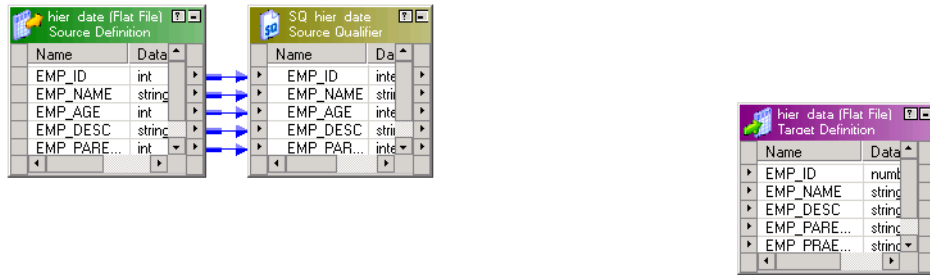
디자이너에서 샘플 매핑에 대한 메타데이터를 가져옵니다. 샘플 매핑에는 다음 구성 요소가 포함됩니다.

- **소스 정의 및 소스 한정자 변환.** 변환의 소스 데이터를 정의하는 플랫폼 파일 소스 정의(hier_input)입니다.
- **대상 정의.** 변환에서 출력 데이터를 받는 플랫폼 파일 대상 정의(hier_data)입니다.

다음 위치에서 매핑에 대한 메타데이터를 가져올 수 있습니다.

<PowerCenter Client installation directory>\client\bin\m_jtx_hier_useCase.xml

다음 그림에서는 샘플 매핑을 보여 줍니다.



2단계. 변환 작성 및 포트 구성

매핑 디자이너에서 Java 변환을 작성하고 포트를 구성하십시오. 입력 및 출력 포트 이름을 Java 코드에서 변수로 사용할 수 있습니다. Java 변환에서 입력 및 출력 포트를 입력 또는 출력 그룹에 작성합니다. Java 변환에는 입력 그룹 및 출력 그룹이 하나씩만 포함될 수 있습니다.

매핑 디자이너에서 활성 Java 변환을 작성하고 포트를 구성하십시오. 이 예제에서 변환의 이름은 jtx_hier_useCase입니다.

참고: 이 예제에서 Java 코드를 사용하려면 입력 및 출력 포트의 정확한 이름을 사용해야 합니다.

다음 테이블에는 변환의 입력 및 출력 포트가 나와 있습니다.

포트 이름	포트 유형	데이터 유형	전체 자릿수	소수 자릿수
EMP_ID_INP	입력	정수	10	0
EMP_NAME_INP	입력	문자열	100	0
EMP_AGE	입력	정수	10	0
EMP_DESC_INP	입력	문자열	100	0

포트 이름	포트 유형	데이터 유형	전체 자릿수	소수 자릿수
EMP_PARENT_EMPID	입력	정수	10	0
EMP_ID_OUT	출력	정수	10	0
EMP_NAME_OUT	출력	문자열	100	0
EMP_DESC_OUT	출력	문자열	100	0
EMP_PARENT_EMPNAME	출력	문자열	100	0

3단계. Java 코드 입력

다음 코드 항목 탭에서 변환에 대한 Java 코드를 입력합니다.

- **패키지 가져오기.** `java.util.Map` 및 `java.util.HashMap` 패키지를 가져옵니다.
- **도우미 코드.** 맵 개체, 잠금 개체 및 Java 변환에서 데이터 상태를 추적하는 데 사용되는 부울 변수를 포함합니다.
- **입력 행에서.** 입력 행에서의 Java 변환의 동작을 정의합니다.

패키지 가져오기 탭

패키지 가져오기 탭에서 타사 Java 패키지, 기본 제공 Java 패키지 또는 사용자 지정 Java 패키지를 가져옵니다. 예제 변환에서는 `Map` 및 `HashMap` 패키지를 사용합니다.

패키지 가져오기 탭에서 다음 코드를 입력합니다.

```
import java.util.Map;
import java.util.HashMap;
```

디자이너가 변환을 위한 Java 코드에 가져오기 문을 추가합니다.

관련 항목:

- [“Java 변환 설정 구성” 페이지 221](#)

도우미 코드 탭

도우미 코드 탭에서 Java 변환에 대한 사용자 정의 변수 및 메서드를 선언합니다. 도우미 코드 탭에서는 입력 행에서 탭의 Java 코드가 사용하는 다음과 같은 변수를 정의합니다.

- **empMap.** 소스의 ID 번호 및 직원 이름을 저장하는 맵 개체입니다.
- **lock.** 파티션 간에 empMap에 대한 액세스를 동기화하는 데 사용되는 잠금 개체입니다.

- **generateRow.** 현재 입력 행에 대해 출력 행이 생성되어야 하는지 여부를 확인하는 데 사용되는 부울 변수입니다.
- **isRoot.** 직원이 회사 조직도 최상위(루트)에 있는지 확인하는 데 사용되는 부울 변수입니다.

도우미 코드 탭에서 다음 코드를 입력합니다.

```
// 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;
```

입력 행에서 탭

Java 변환은 입력 행을 받으면 입력 행에서 탭에 있는 Java 코드를 실행합니다. 이 예제에서는 변환이 입력 행의 값에 따라 출력 행을 생성하거나 생성하지 않을 수 있습니다.

입력 행에서 탭에서 다음 코드를 입력하십시오.

```
// 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();

```

4단계. Java 코드 컴파일

변환 개발자에서 컴파일을 클릭하여 변환에 대한 **Java** 코드를 컴파일합니다. 출력 창에 컴파일 상태가 표시됩니다. **Java** 코드가 성공적으로 컴파일되지 않으면 코드 항목 탭에서 오류를 수정하고 **Java** 코드를 다시 컴파일합니다. 변환을 성공적으로 컴파일한 후 리포지토리에 변환을 저장합니다.

관련 항목:

- [“Java 변환 컴파일” 페이지 223](#)
- [“컴파일 오류 수정” 페이지 224](#)

5단계. 세션 및 워크플로우 작성

워크플로우 관리자에서 `m_jtx_hier_useCase` 매핑을 사용하여 해당 매핑에 대한 세션 및 워크플로우를 작성합니다.

세션을 구성할 때 다음 위치에 있는 샘플 소스 파일을 사용할 수 있습니다.

`<PowerCenter Client installation directory>\client\bin\hier_data`

샘플 데이터

다음 데이터는 샘플 소스 파일에서 발췌한 것입니다.

```
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
```

다음 데이터는 샘플 대상 파일에서 발췌한 것입니다.

```
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

제 16 장

조이너 변환

이 장에 포함된 항목:

- [조이너 변환 개요, 254](#)
- [조이너 변환 속성, 255](#)
- [조인 조건 정의, 256](#)
- [조인 유형 정의, 257](#)
- [정렬된 입력 사용, 259](#)
- [단일 소스의 데이터 조인, 261](#)
- [소스 파이프라인 차단, 263](#)
- [트랜잭션 작업, 264](#)
- [조이너 변환 작성, 265](#)
- [조이너 변환에 대한 팁, 266](#)

조이너 변환 개요

조이너 변환을 사용하여 다른 위치 또는 파일 시스템에 상주하는 다른 유형 관련 소스 2개의 소스 데이터를 조인합니다. 동일한 소스의 데이터를 조인할 수도 있습니다. 조이너 변환은 소스와 최소 1개 이상의 일치하는 열을 조인합니다. 조이너 변환은 두 소스에서 하나 이상의 열 쌍을 일치시키는 조건을 사용합니다. 조이너 변환은 활성 변환입니다.

두 입력 파이프라인에는 마스터 파이프라인 및 세부 파이프라인 또는 마스터 및 세부 분기가 포함됩니다. 마스터 파이프라인은 조이너 변환에서 종료되고 세부 파이프라인은 대상까지 계속됩니다.

매핑에서 3개 이상의 소스를 조인하려면 조이너 변환의 출력을 다른 소스 파이프라인과 조인해야 합니다. 모든 소스 파이프라인이 조인될 때까지 조이너 변환을 매핑에 추가합니다.

조이너 변환은 대다수 변환의 입력을 허용합니다. 하지만 조이너 변환에 연결하는 파이프라인에 대한 다음 제한 사항을 고려하십시오.

- 입력 파이프라인 중 하나가 업데이트 전략 변환을 포함할 때는 조이너 변환을 사용할 수 없습니다.
- 조이너 변환 전에 시퀀스 생성기 변환을 직접 연결하는 경우 조이너 변환을 사용할 수 없습니다.

조이너 변환 작업

조이너 변환 작업을 수행할 때는 변환 속성, 조인 유형 및 조인 조건을 구성해야 합니다. 정렬된 입력을 통해 통합 서비스 성능이 향상되도록 하기 위해 조이너 변환을 구성할 수 있습니다. 또한 통합 서비스에서 변환 논리를 적

용하는 방식을 제어하기 위해 변환 범위를 구성할 수 있습니다. 조이너 변환 작업을 수행하려면 다음 태스크를 완료하십시오.

- **조이너 변환 속성 구성.** 조이너 변환의 속성은 캐시 디렉터리 위치, 통합 서비스가 변환을 처리하는 방법 및 통합 서비스가 캐싱을 처리하는 방법을 식별합니다.
- **조인 조건 구성.** 조인 조건에는 통합 서비스가 두 행을 조인하기 위해 일치되어야 하는 두 입력 소스의 포트가 포함됩니다. 선택된 조인 유형에 따라 통합 서비스는 행을 결과 집합에 추가하거나 행을 삭제합니다.
- **조인 유형 구성.** 조인은 서로 다른 데이터베이스의 여러 테이블 또는 여러 플랫폼 파일에 있는 데이터를 단일 결과 집합으로 결합하는 관계형 연산자입니다. 일반, 마스터 외부, 세부 외부 또는 전체 외부 조인 유형을 사용하도록 조이너 변환을 구성할 수 있습니다.
- **정렬 또는 비정렬 입력에 대한 세션 구성.** 정렬된 입력을 사용하도록 조이너 변환을 구성하여 세션 성능을 향상시킬 수 있습니다. 정렬된 데이터를 사용하도록 매핑을 구성하려면 매핑에서 정렬 순서를 설정하고 유지하여 통합 서비스가 조이너 변환을 처리할 때 정렬된 데이터를 사용할 수 있게 하십시오.
- **트랜잭션 범위 구성.** 통합 서비스는 조이너 변환을 처리할 때 트랜잭션의 모든 데이터 또는 모든 수신 데이터에 변환 논리를 적용하거나 한 번에 데이터 행 하나에 변환 논리를 적용할 수 있습니다.

PowerCenter에서 분할 옵션이 제공되는 경우, 파이프라인에 있는 파티션 수를 늘려 세션 성능을 향상시킬 수 있습니다.

조이너 변환 속성

조이너 변환의 속성은 캐시 디렉터리 위치, 통합 서비스가 변환을 처리하는 방법 및 통합 서비스가 캐싱을 처리하는 방법을 식별합니다. 또한 해당 속성은 통합 서비스가 테이블 및 파일을 조인하는 방법을 결정합니다.

매핑을 작성할 때 각 조이너 변환에 대한 속성을 지정합니다. 세션을 작성하는 경우 각 변환에 대한 데이터 캐시 크기 및 인덱스 등 일부 속성을 재정의할 수 있습니다.

다음 테이블에는 조이너 변환 속성이 설명되어 있습니다.

옵션	설명
캐시 디렉터리	마스터 또는 세부 행과 이러한 행의 인덱스를 캐시하는 데 사용되는 디렉터리를 지정합니다. 기본적으로 캐시 파일은 프로세스 변수 \$PMCacheDir로 지정되는 디렉터리에 작성됩니다. 디렉터리를 재정의하는 경우에는 해당 디렉터리가 있으며 캐시 파일을 저장하기에 충분한 디스크 공간이 있는지 확인하십시오. 매핑되거나 마운팅된 드라이브를 디렉터리로 사용할 수 있습니다.
조인 유형	다음과 같은 조인 유형을 지정합니다. 일반, 마스터 외부, 세부 외부 또는 전체 외부.
마스터 Null 순서	이 변환 유형에는 해당되지 않습니다.
상세 Null 순서	이 변환 유형에는 해당되지 않습니다.
추적 수준	이 변환에 대해 세션 로그에 표시되는 세부 정보의 양입니다. 옵션은 간단, 보통, 자세한 정보 표시 데이터 및 자세한 정보 표시 초기화입니다.

옵션	설명
조이너 데이터 캐시 크기	변환의 데이터 캐시 크기입니다. 기본 캐시 크기는 2,000,000바이트입니다. 구성된 총 캐시 크기가 2GB 이상인 경우 64비트 통합 서비스에서 세션을 실행해야 합니다. 캐시에 숫자 값을 사용하거나, 매개 변수 파일의 캐시 값을 사용하거나, 자동 설정을 통해 캐시 크기를 설정하도록 통합 서비스를 구성할 수 있습니다. 캐시 크기를 결정하도록 통합 서비스를 구성하는 경우 통합 서비스가 캐시에 할당하는 최대 메모리 양도 구성할 수 있습니다.
조이너 인덱스 캐시 크기	변환의 인덱스 캐시 크기입니다. 기본 캐시 크기는 1,000,000바이트입니다. 구성된 총 캐시 크기가 2GB 이상인 경우 64비트 통합 서비스에서 세션을 실행해야 합니다. 캐시에 숫자 값을 사용하거나, 매개 변수 파일의 캐시 값을 사용하거나, 자동 설정을 통해 캐시 크기를 설정하도록 통합 서비스를 구성할 수 있습니다. 캐시 크기를 결정하도록 통합 서비스를 구성하는 경우 통합 서비스가 캐시에 할당하는 최대 메모리 양도 구성할 수 있습니다.
정렬된 입력	데이터가 정렬됨을 지정합니다. 정렬된 데이터를 조인하려면 정렬된 입력을 선택합니다. 정렬된 입력을 사용하면 성능이 향상될 수 있습니다.
마스터 정렬 순서	마스터 소스 데이터의 정렬 순서를 지정합니다. 마스터 소스 데이터가 오름차순인 경우 오름차순을 선택합니다. 오름차순을 선택한 경우 정렬된 입력도 활성화합니다. 기본값은 자동입니다.
변환 범위	통합 서비스에서 변환 논리를 수신 데이터에 적용하는 방법을 지정합니다. 트랜잭션, 모든 입력 또는 행을 선택할 수 있습니다.

조인 조건 정의

조인 조건에는 통합 서비스가 두 행을 조인하기 위해 일치되어야 하는 두 입력 소스의 포트가 포함됩니다. 선택된 조인 유형에 따라 통합 서비스는 행을 결과 집합에 추가하거나 행을 삭제합니다. 조이너 변환은 조인 유형, 조건 및 입력 데이터 소스를 바탕으로 결과 집합을 생성합니다.

조인 조건을 정의하기 전에 마스터 소스 및 세부 소스가 최적의 성능을 제공하도록 구성되었는지 확인하십시오. 세션 중에 통합 서비스가 마스터 소스의 각 행을 세부 소스와 비교합니다. 정렬되지 않은 조이너 변환의 성능을 개선하려면 행 수가 더 적은 소스를 마스터 소스로 사용합니다. 정렬된 조이너 변환의 성능을 개선하려면 중복 키 값이 더 적은 소스를 마스터 소스로 사용합니다.

기본적으로 조이너 변환에 포트를 추가하면 첫 번째 소스 파이프라인의 포트가 세부 소스로 표시됩니다. 두 번째 소스 파이프라인의 포트를 추가하면 이 포트가 마스터 소스로 설정됩니다. 이러한 설정을 변경하려면 마스터 소스로 설정하려는 포트에 대한 포트 탭에서 **M** 열을 클릭합니다. 그러면 이 소스의 포트가 마스터 포트로 설정되고 다른 소스의 포트가 세부 포트로 설정됩니다.

지정된 마스터와 세부 소스 간의 같음을 기반으로 하나 이상의 조건을 정의합니다. 예를 들어 **EMPLOYEE_AGE** 및 **EMPLOYEE_POSITION** 테이블이 포함된 두 소스에 모두 직원 ID 번호가 포함되어 있으면 다음 조건은 두 소스에 나열된 직원이 들어 있는 행의 일치 여부를 확인합니다.

```
EMP_ID1 = EMP_ID2
```

조인 조건에는 조이너 변환의 입력 소스의 포트를 하나 이상 사용합니다. 포트를 추가하면 두 소스를 조인하는데 필요한 시간이 늘어납니다. 조건에서 포트의 순서는 조이너 변환의 성능에 영향을 미칠 수 있습니다. 조인 조건에 여러 포트를 사용하는 경우 통합 서비스는 사용자가 지정한 순서로 포트를 비교합니다.

디자이너는 조건에서 데이터 유형의 유효성을 검사합니다. 조건에 있는 두 포트의 데이터 유형이 같아야 합니다. 데이터 유형이 일치하지 않는 두 포트를 조건에서 사용해야 하는 경우에는 데이터 유형을 일치하도록 변환하십시오.

문자와 Varchar 데이터 유형을 조인하는 경우 통합 서비스는 문자 값을 채우는 공백을 문자열 수에 포함하여 계산합니다.

```
Char(40) = "abcd"
Varchar(40) = "abcd"
```

여기에서 문자 값은 36개의 공백으로 채워진 "abcd"이고 통합 서비스는 문자 필드에 후행 공백이 포함되기 때문에 두 필드를 조인하지 않습니다.

참고: 조이너 변환은 Null 값을 일치시키지 않습니다. 예를 들어 EMP_ID1과 EMP_ID2에 Null 값이 있는 행이 포함되는 경우 이러한 행은 일치로 간주되지 않으므로 두 행이 조인되지 않습니다. Null 값이 포함된 행을 조인하려면 Null 입력을 기본값으로 바꾼 다음 이 기본값에 조인을 수행해야 합니다.

조인 유형 정의

SQL에서 조인은 여러 테이블의 데이터를 단일 결과 집합에 결합하는 관계형 연산자입니다. 조이너 변환은 데이터의 출처가 여러 유형의 소스일 수 있다는 것을 제외하고는 SQL 조인과 유사합니다.

변환의 속성 탭에서 조인 유형을 정의합니다. 조이너 변환은 다음과 같은 유형의 조인을 지원합니다.

- 보통
- 마스터 외부
- 상세 외부
- 전체 외부

참고: 일반 또는 마스터 외부 조인은 전체 외부 또는 세부 외부 조인보다 빠르게 수행됩니다.

두 소스 중 어떤 소스의 데이터도 포함하지 않는 필드가 결과 집합에 포함되는 경우 조이너 변환은 빈 필드를 Null 값으로 채웁니다. 필드에서 NULL을 반환할 것이라는 것을 알고 NULL이 대상에 삽입되는 것을 원하지 않는 경우에는 해당하는 포트에 대한 포트 탭에서 기본값을 설정할 수 있습니다.

일반 조인

일반 조인을 사용하는 경우 통합 서비스는 조건에 따라 마스터 및 세부 소스에서 일치하지 않는 모든 데이터 행을 무시합니다.

예를 들어 자동차 부품에 대한 두 데이터 소스인 PARTS_SIZE 및 PARTS_COLOR와 다음 데이터가 있을 수 있습니다.

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

PARTS_COLOR (detail source)

3	Floor Mat	Black
4	Fuzzy Dice	Yellow

두 소스에서 PART_ID 일치를 확인하여 두 테이블을 조인하려면 다음과 같이 조건을 설정합니다.

PART_ID1 = PART_ID2

일반 조인을 사용하여 이러한 테이블을 조인하면 결과 집합에는 다음 데이터가 포함됩니다.

PART_ID	DESCRIPTION	SIZE	COLOR
1	Seat Cover	Large	Blue
3	Floor Mat	Medium	Black

다음 예는 동일한 SQL 문을 보여 줍니다.

```
SELECT * FROM PARTS_SIZE, PARTS_COLOR WHERE PARTS_SIZE.PART_ID1 = PARTS_COLOR.PART_ID2
```

마스터 외부 조인

마스터 외부 조인은 세부 소스 데이터의 모든 행 및 마스터 소스의 일치하는 행을 유지합니다. 이 조인은 마스터 소스의 일치하지 않는 행을 무시합니다.

동일한 조건으로 마스터 외부 조인을 사용하여 샘플 테이블을 조인하면 결과 집합에 다음 데이터가 포함됩니다.

PART_ID	DESCRIPTION	SIZE	COLOR
1	Seat Cover	Large	Blue
3	Floor Mat	Medium	Black
4	Fuzzy Dice	NULL	Yellow

Fuzzy Dice에 크기가 지정되지 않았기 때문에 통합 서비스가 이 필드를 NULL로 채웁니다.

다음 예는 동일한 SQL 문을 보여 줍니다.

```
SELECT * FROM PARTS_SIZE RIGHT OUTER JOIN PARTS_COLOR ON (PARTS_COLOR.PART_ID2 = PARTS_SIZE.PART_ID1)
```

세부 외부 조인

세부 외부 조인은 마스터 소스 데이터의 모든 행 및 세부 소스의 일치하는 행을 유지합니다. 이 조인은 세부 소스의 일치하지 않는 행을 무시합니다.

동일한 조건으로 세부 외부 조인을 사용하여 샘플 테이블을 조인하면 결과 집합에 다음 데이터가 포함됩니다.

PART_ID	DESCRIPTION	SIZE	COLOR
1	Seat Cover	Large	Blue
2	Ash Tray	Small	NULL
3	Floor Mat	Medium	Black

Ash Tray에 색상이 지정되지 않았기 때문에 통합 서비스가 이 필드를 NULL로 채웁니다.

다음 예는 동일한 SQL 문을 보여 줍니다.

```
SELECT * FROM PARTS_SIZE LEFT OUTER JOIN PARTS_COLOR ON (PARTS_SIZE.PART_ID1 = PARTS_COLOR.PART_ID2)
```

전체 외부 조인

전체 외부 조인은 마스터 소스 데이터와 세부 소스 데이터의 모든 행을 유지합니다.

동일한 조건으로 전체 외부 조인을 사용하여 샘플 테이블을 조인하면 결과 집합에 다음이 포함됩니다.

PART_ID	DESCRIPTION	SIZE	Color
1	Seat Cover	Large	Blue
2	Ash Tray	Small	NULL
3	Floor Mat	Medium	Black
4	Fuzzy Dice	NULL	Yellow

Ash Tray에 색상이 지정되지 않았고 Fuzzy Dice에 크기가 지정되지 않았기 때문에 통합 서비스가 이러한 필드를 NULL로 채웁니다.

다음 예는 동일한 SQL 문을 보여 줍니다.

```
SELECT * FROM PARTS_SIZE FULL OUTER JOIN PARTS_COLOR ON (PARTS_SIZE.PART_ID1 = PARTS_COLOR.PART_ID2)
```

정렬된 입력 사용

정렬된 입력을 사용하도록 조이너 변환을 구성하여 세션 성능을 향상시킬 수 있습니다. 정렬된 데이터를 사용하도록 조이너 변환을 구성하면 통합 서비스가 디스크 입력과 출력을 최소화하여 성능을 개선합니다. 대규모 데이터 집합으로 작업할 때 가장 큰 성능 향상을 얻을 수 있습니다.

정렬된 데이터를 사용하도록 매핑을 구성하려면 조이너 변환을 처리할 때 통합 서비스가 정렬된 데이터를 사용할 수 있도록 매핑에서 정렬 순서를 설정하고 유지 관리합니다. 매핑을 구성하려면 다음 태스크를 완료하십시오.

- **정렬 순서를 구성합니다.** 조인하려는 데이터의 정렬 순서를 구성합니다. 정렬된 플랫폼 파일을 조인하거나 소스 한정자 변환을 사용하여 관계형 데이터를 정렬할 수 있습니다. 분류기 변환을 사용할 수도 있습니다.
- **변환을 추가합니다.** 정렬된 데이터의 순서를 유지 관리하는 변환을 사용합니다.
- **조이너 변환을 구성합니다.** 정렬된 데이터를 사용하도록 조이너 변환을 구성하고 정렬 원본 포트를 사용하도록 조인 조건을 구성합니다. 정렬 출처는 정렬된 데이터의 소스를 나타냅니다.

세션에서 정렬 순서를 구성할 때 통합 서비스 코드 페이지와 연관된 정렬 순서를 선택할 수 있습니다. 통합 서비스를 유니코드 모드에서 실행하는 경우 선택한 세션 정렬 순서를 사용하여 문자 데이터가 정렬됩니다. 통합 서비스를 ASCII 모드에서 실행하는 경우 모든 문자 데이터가 이진 정렬 순서를 사용하여 정렬됩니다. 통합 서비스의 요구에 따라 데이터가 정렬되게 하려면 데이터베이스 정렬 순서가 사용자 정의 세션 정렬 순서와 같아야 합니다.

분할된 파이프라인의 정렬된 데이터를 조인하는 경우 정렬된 데이터의 순서를 유지하도록 파티션을 구성해야 합니다.

정렬 순서 구성

통합 서비스가 정렬된 데이터를 조이너 변환에 전달할 수 있도록 정렬 순서를 구성해야 합니다.

다음 방법 중 하나를 사용하여 정렬 순서를 구성합니다.

- **정렬된 플랫폼 파일을 사용합니다.** 플랫폼 파일에 정렬된 데이터가 포함되는 경우 정렬 열의 순서가 각 소스 파일에서 일치하는지 확인합니다.
- **정렬된 관계형 데이터를 사용합니다.** 소스 한정자 변환의 정렬된 포트를 사용하여 소스 데이터베이스의 열을 정렬합니다. 정렬된 포트의 순서는 각 소스 한정자 변환에서 동일하게 구성합니다.
- **분류기 변환을 사용합니다.** 분류기 변환을 사용하여 관계형 또는 플랫폼 파일 데이터를 정렬합니다. 분류기 변환은 마스터 및 세부 파이프라인에 배치합니다. 각 분류기 변환이 동일한 순서의 정렬 키 포트 및 정렬 순서 방향을 사용하도록 구성하십시오.

정렬된 데이터를 사용하도록 구성된 조이너 변환에 정렬되지 않거나 잘못 정렬된 데이터를 전달할 경우 세션이 실패하고 통합 서비스가 세션 로그 파일에 오류를 기록합니다.

매핑에 변환 추가

정렬 출처와 조이너 변환 사이에 변환을 추가할 때 정렬된 데이터를 유지하려면 다음 지침을 사용하십시오.

- 정렬 출처와 조이너 변환 사이에는 다음 변환을 배치하지 마십시오.
 - 사용자 지정
 - 정렬되지 않은 집계
 - 노멀라이저
 - 순위
 - 합집합 변환
 - XML 파서 변환
 - XML 생성기 변환
 - 맵렛(위의 변환 중 하나가 포함된 경우)
- 정렬 출처와 조이너 변환 사이에 정렬된 집계 변환을 배치하려면 다음 지침을 사용하십시오.
 - 정렬된 입력을 사용하도록 집계 변환을 구성합니다.
 - 집계 변환의 그룹 기준 열에 정렬 출처의 포트와 동일한 포트를 사용합니다.
 - 그룹 기준 포트는 정렬 출처의 포트와 동일한 순서여야 합니다.
- 조이너 변환의 결과 집합을 다른 파이프라인과 조인하는 경우 첫 번째 조이너 변환의 데이터 출력이 정렬되었는지 확인하십시오.

팁: 정렬 출처 바로 뒤에 조이너 변환을 배치하면 정렬된 데이터를 유지할 수 있습니다.

조이너 변환 구성

조이너 변환을 구성하려면 다음 태스크를 수행하십시오.

- 속성 탭에서 정렬된 입력을 활성화합니다.
- 정렬된 데이터를 정렬 출처와 동일한 순서로 받도록 조인 조건을 정의합니다.

조인 조건 정의

다음과 같은 정렬 출처에서 설정된 정렬 순서를 유지 관리하려면 조인 조건을 구성합니다. 정렬된 플랫폼 파일, 소스 한정자 변환 또는 분류기 변환. 정렬 출처 및 조이너 변환 간에 정렬된 집계 변환을 사용할 경우 조인 조건을 정의할 때 정렬된 집계 변환을 정렬 출처로 처리하십시오. 조인 조건을 정의할 경우 다음 지침을 사용합니다.

- 조인 조건에서 사용하는 포트가 정렬 출처의 포트와 일치해야 합니다.

- 여러 조인 조건을 구성할 경우 첫 번째 조인 조건의 포트가 정렬 출처의 첫 번째 포트와 일치해야 합니다.
- 여러 조건을 구성할 경우 조건 순서가 정렬 출처의 포트 순서와 일치해야 하고 어떤 포트도 건너뛰면 안 됩니다.
- 정렬 출처의 정렬된 포트 수는 조인 조건의 포트 수보다 크거나 같을 수 있습니다.

조인 조건의 예

예를 들어 다음과 같은 정렬된 포트를 사용하여 마스터 및 세부 파이프라인에서 분류기 변환을 구성합니다.

1. ITEM_NO
2. ITEM_NAME
3. PRICE

조인 조건을 구성할 때는 다음 지침을 사용하여 정렬 순서를 유지합니다.

- ITEM_NO를 첫 번째 조인 조건에 사용해야 합니다.
- 두 번째 조인 조건을 추가하는 경우 ITEM_NAME을 사용해야 합니다.
- PRICE를 사용하려는 경우 ITEM_NAME도 두 번째 조인 조건에 사용해야 합니다.

ITEM_NAME을 건너뛰고 ITEM_NO와 PRICE에서 조인하면 정렬 순서가 손실되고 통합 서비스가 세션을 중지합니다.

조이너 변환을 사용하여 마스터 파이프라인과 세부 파이프라인을 조인하는 경우 다음 조인 조건 중 하나를 구성할 수 있습니다.

```
ITEM_NO = ITEM_NO
```

또는

```
ITEM_NO = ITEM_NO1
ITEM_NAME = ITEM_NAME1
```

또는

```
ITEM_NO = ITEM_NO1
ITEM_NAME = ITEM_NAME1
PRICE = PRICE1
```

단일 소스의 데이터 조인

일부 데이터에 대해 계산을 수행한 다음 변환된 데이터를 원래 데이터와 조인하려는 경우 동일한 소스의 데이터를 조인하려고 할 수 있습니다. 이 메서드를 사용하여 데이터를 조인하는 경우 하나의 매핑 내에서 원래 데이터를 유지 관리하고 해당 데이터의 일부를 변환할 수 있습니다. 동일한 소스의 데이터는 다음과 같이 조인할 수 있습니다.

- 동일한 파이프라인의 분기 2개를 조인합니다.
- 동일한 소스의 인스턴스 2개를 조인합니다.

동일한 파이프라인의 분기 2개 조인

동일한 소스의 데이터를 조인할 경우 파이프라인의 분기 2개를 작성할 수 있습니다. 파이프라인의 분기를 만드는 경우 최소 1개의 파이프라인 분기에서 소스 한정자와 조이너 변환 사이에 변환을 추가해야 합니다. 정렬된 데이터를 조인하고 조이너 변환을 정렬된 입력에 대해 구성해야 합니다.

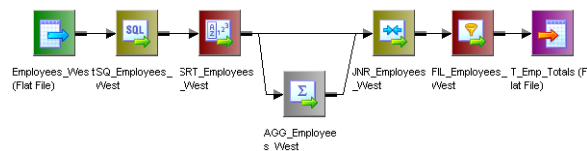
예를 들어 다음 포트를 포함하는 소스가 있습니다.

- 직원
- 부서
- 총 매출액

대상에는 부서의 평균 매출액을 초과하는 매출액을 달성한 직원을 표시하려고 합니다. 그러려면 다음 변환을 포함하는 매핑을 작성해야 합니다.

- **분류기 변환.** 데이터를 정렬합니다.
- **정렬된 집계 변환.** 매출액 데이터의 평균값을 계산하고 부서를 기준으로 그룹화합니다. 이 집계를 수행하면 개별 직원에 대한 데이터가 손실됩니다. 직원 데이터를 유지하려면 파이프라인의 분기를 집계 변환에 전달하고 동일한 데이터가 포함된 분기를 조이너 변환에 전달하여 원래 데이터를 유지해야 합니다. 파이프라인의 두 분기를 조인할 때는 집계된 데이터를 원래 데이터와 조인합니다.
- **정렬된 조이너 변환.** 정렬된 조이너 변환을 사용하여 정렬된 집계 데이터를 원래 데이터와 조인합니다.
- **필터 변환.** 평균 매출액 데이터를 각 직원의 매출액 데이터와 비교한 다음 평균 매출액 이하의 매출액을 달성한 직원을 필터링하여 제외합니다.

다음 그림에는 동일한 파이프라인의 두 분기를 조인하는 매핑이 나와 있습니다.



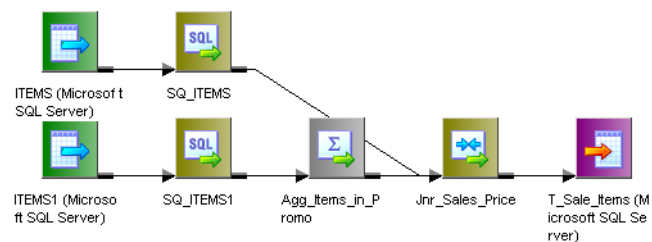
참고: 또한 사용자 지정 변환 또는 XML 소스 한정자 변환 등 동일한 변환의 출력 그룹에서 데이터를 조인할 수 있습니다. 조이너 변환 및 각 출력 그룹 사이에 분류기 변환을 배치하고 정렬된 입력을 수신하도록 조이너 변환을 구성합니다.

두 분기를 조인할 때 조이너 변환이 각 분기에서 데이터를 수신하는 간격이 길어지면 성능에 영향을 미칠 수 있습니다. 조이너 변환은 첫 번째 분기의 모든 데이터를 캐시하고 캐시가 채워지면 디스크에 해당 캐시를 기록합니다. 그런 다음 두 번째 분기의 데이터를 수신할 때 조이너 변환은 해당 데이터를 디스크에서 읽어야 합니다. 이로 인해 처리가 느려질 수 있습니다.

동일한 소스의 인스턴스 2개 조인

또한 소스의 두 번째 인스턴스를 작성하여 동일한 소스 데이터를 조인할 수 있습니다. 두 번째 소스 인스턴스를 작성한 후에 두 소스 인스턴스의 파이프라인을 조인할 수 있습니다. 정렬되지 않은 데이터를 조인하려는 경우 동일한 소스의 인스턴스 2개를 작성하고 파이프라인을 조인해야 합니다.

다음 그림은 조이너 변환을 통해 조인된 동일한 소스의 인스턴스 2개를 보여 줍니다.



참고: 이 메서드를 사용하여 데이터를 조인하면 통합 서비스가 각 소스 인스턴스에 대한 소스 데이터를 읽으므로, 파이프라인의 두 분기를 조인하는 것보다 성능이 저하될 수 있습니다.

단일 소스의 데이터 조인에 대한 지침

파이프라인의 분기를 조인할지 소스의 인스턴스 2개를 조인할지를 결정할 때 다음 지침을 사용하십시오.

- 소스가 크거나 소스 데이터를 한 번만 읽을 수 있는 경우 파이프라인의 분기 2개를 조인합니다. 예를 들어 메시징 대기열에서 소스 데이터를 한 번 읽을 수만 있습니다.
- 정렬된 데이터를 사용하는 경우 파이프라인의 분기 2개를 조인합니다. 소스 데이터가 정렬되지 않아 분류기 변환을 사용하여 데이터를 정렬하는 경우 데이터를 정렬한 후에 파이프라인의 분기를 만듭니다.
- 파이프라인에서 소스와 조이너 변환 사이에 차단 변환을 추가해야 하는 경우 소스의 인스턴스 2개를 조인합니다.
- 한 파이프라인이 다른 파이프라인보다 느리게 처리되는 경우 소스의 인스턴스 2개를 조인합니다.
- 정렬되지 않은 데이터를 조인해야 하는 경우 소스의 인스턴스 2개를 조인합니다.

소스 파이프라인 차단

조이너 변환을 포함하는 세션을 실행할 때 통합 서비스는 매핑 구성 및 정렬된 입력에 대한 조이너 변환을 구성하는지 여부에 따라 소스 데이터를 차단 및 차단 해제합니다.

정렬되지 않은 조이너 변환

통합 서비스는 정렬되지 않은 조이너 변환을 처리할 때 세부 정보 행을 읽기 전에 모든 마스터 행을 먼저 읽습니다. 세부 정보 행보다 모든 마스터 행을 먼저 읽기 위해 통합 서비스는 마스터 소스로부터 행을 캐시하는 동안 세부 정보 소스를 차단합니다. 통합 서비스는 모든 마스터 행을 읽고 캐싱한 후에 세부 정보 소스를 차단하고 세수 정보 행을 읽습니다. 정렬되지 않은 조이너 변환이 포함된 일부 매핑은 데이터 흐름 유효성 검사에 위반됩니다.

정렬된 조이너 변환

통합 서비스는 정렬된 조이너 변환을 처리할 때 매핑 구성에 따라 데이터를 차단합니다. 조이너 변환에 대한 마스터 및 세부 입력이 서로 다른 소스에서 생성되는 경우 차단 논리를 사용할 수 있습니다.

통합 서비스는 대상 로드 순서 그룹의 모든 소스를 동시에 차단하지 않으면서 차단 논리를 사용하여 조이너 변환을 처리할 수 있으면 그와 같이 조이너 변환을 처리합니다. 그렇지 않으면 차단 논리를 사용하지 않습니다. 대신 캐시에 더 많은 행을 저장합니다.

통합 서비스는 차단 논리를 사용하여 조이너 변환을 처리할 수 있으면 적은 수의 행을 캐시에 저장하며, 그에 따라 성능이 향상됩니다.

마스터 행 캐싱

통합 서비스는 조이너 변환을 처리할 때 두 소스의 행을 동시에 읽고 마스터 행을 기반으로 인덱스 및 데이터 캐시를 빌드합니다. 그런 다음 세부 소스 데이터 및 캐시 데이터를 바탕으로 조인을 수행합니다. 통합 서비스가 캐시에 저장하는 행의 수는 파티션 유형, 소스 데이터 및 정렬된 입력에 대한 조이너 변환을 구성하는지 여부에 따라 다릅니다. 정렬되지 않은 조이너 변환의 성능을 개선하려면 행 수가 더 적은 소스를 마스터 소스로 사용합니다. 정렬된 조이너 변환의 성능을 개선하려면 중복 키 값이 더 적은 소스를 마스터 소스로 사용합니다.

트랜잭션 작업

통합 서비스는 조이너 변환을 처리할 때 트랜잭션의 모든 데이터 또는 모든 수신 데이터에 변환 논리를 적용하거나 한 번에 데이터 행 하나에 변환 논리를 적용할 수 있습니다. 통합 서비스는 매핑 구성 및 변환 범위에 따라 트랜잭션 경계를 삭제하거나 유지할 수 있습니다. 통합 서비스가 변환 논리를 적용하는 방식 및 변환 범위 속성을 사용하여 트랜잭션 경계를 처리하는 방식은 사용자에 의해 구성됩니다.

변환 범위 값은 매핑 구성 및 트랜잭션 경계를 유지할지 또는 삭제할지에 따라 구성합니다.

다음 소스를 조인하는 경우 트랜잭션 경계를 유지할 수 있습니다.

- **동일한 소스 파이프라인의 두 분기를 조인합니다.** 트랜잭션 변환 범위를 사용하여 트랜잭션 경계를 유지하십시오.
- **두 소스를 조인하며, 세부 소스의 트랜잭션 경계를 유지하려고 합니다.** 행 변환 범위를 사용하여 세부 파이프라인에서 트랜잭션 경계를 유지하십시오.

다음 소스를 조인하는 경우 트랜잭션 경계를 삭제할 수 있습니다.

- **두 소스 또는 두 분기를 조인하며, 트랜잭션 경계를 삭제하려고 합니다.** 모든 입력 변환 범위를 사용하여 변환 논리를 모든 수신 데이터에 적용하고 두 파이프라인의 트랜잭션 경계를 삭제합니다.

다음 테이블에는 변환 범위를 조이너 변환과 함께 사용하여 트랜잭션 경계를 유지하는 방법이 요약되어 있습니다.

변환 범위	입력 유형	통합 서비스 동작
행	비정렬	세부 파이프라인에서 트랜잭션 경계를 유지합니다.
행	정렬됨	세션이 실패합니다.
트랜잭션	정렬됨	동일한 트랜잭션 생성기가 마스터 및 세부 출력이면 트랜잭션 경계를 유지합니다. 동일한 트랜잭션 생성기가 마스터 및 세부 출력이면 세션이 실패합니다.
트랜잭션	비정렬	세션이 실패합니다.
모든 입력	정렬, 비정렬	트랜잭션 경계를 삭제합니다.

참고: 실시간 데이터를 모든 입력 또는 트랜잭션 변환 범위와 함께 사용하면 세션이 실패합니다.

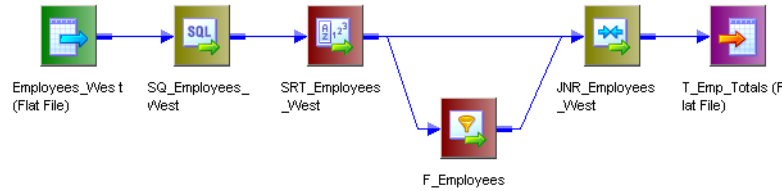
단일 파이프라인의 트랜잭션 경계 유지

동일한 소스의 데이터를 조인할 경우, 트랜잭션 변환 범위를 사용하여 단일 파이프라인의 수신 트랜잭션 경계를 유지합니다. 조이너 변환이 동일한 파이프라인의 분기 2개 또는 단일 트랜잭션 생성기의 출력 그룹 2개 같은 동일한 소스의 데이터를 조인할 경우, 트랜잭션 변환 범위를 사용하십시오. 이 변환 범위를 정렬된 데이터 및 모든 조인 유형과 함께 사용합니다.

트랜잭션 변환 범위를 사용하는 경우, 마스터 및 세부 파이프라인이 동일한 트랜잭션 제어점에서 시작되는지 확인하고 정렬된 입력이 사용되는지 확인하십시오. 예를 들어 [“단일 파이프라인의 트랜잭션 경계 유지” 페이지 264](#)에서는 분류기 변환이 트랜잭션 제어점입니다. 분류기 변환과 조이너 변환 사이에 다른 트랜잭션 제어점을 배치할 수는 없습니다. 매핑에서 마스터 및 세부 파이프라인 분기는 동일한 트랜잭션 제어점에서 시작되고, 통합 서비스는 파이프라인 분기를 조이너 변환과 조인하여 트랜잭션 경계를 유지합니다.

다음 그림은 파이프라인의 분기 2개를 조인하고 트랜잭션 경계를 유지하는 매핑을 보여 줍니다.

수치 1. 파이프라인 분기 2개를 조인할 때 트랜잭션 경계 유지



세부 파이프라인에서 트랜잭션 경계 유지

세부 파이프라인에서 트랜잭션 경계를 유지하려면 행 변환 범위를 선택합니다. 행 변환 범위를 사용하면 통합 서비스가 한 번에 한 행씩 데이터를 처리할 수 있습니다. 통합 서비스는 마스터 데이터를 캐싱하고 세부 데이터를 캐싱된 마스터 데이터와 일치시킵니다.

소스 데이터의 출처가 IBM MQ Series 등의 실시간 소스이면 통합 서비스는 세부 소스에서 메시지를 읽을 때 캐싱된 마스터 데이터를 각 메시지와 일치시킵니다.

비정렬 데이터를 사용하는 일반 및 마스터 외부 조인 유형과 함께 행 변환 범위를 사용하십시오.

두 파이프라인의 트랜잭션 경계 삭제

두 소스 또는 두 분기의 데이터를 조인하고 트랜잭션 경계를 유지할 필요가 없을 경우 모든 입력 변환 범위를 사용합니다. 모든 입력을 사용하면 통합 서비스가 두 파이프라인 모두의 수신 트랜잭션 경계를 삭제하고 변환의 모든 행을 개방형 트랜잭션으로 출력합니다. 정렬 순서를 구성하는 방식에 따라 조이너 변환에서 마스터 파이프라인의 데이터가 캐시되거나 동시에 조인될 수 있습니다. 이 변환 범위를 정렬된 데이터와 정렬되지 않은 데이터 및 모든 조인 유형에서 사용하십시오.

조이너 변환 작성

조이너 변환을 사용하려면 매핑에 조이너 변환을 추가하고, 입력 소스를 설정하고, 조건 및 조인 유형과 정렬 유형을 사용하여 변환을 구성합니다.

조이너 변환을 작성하려면:

1. 매핑 디자이너에서 변환 > 작성을 클릭합니다. 조이너 변환을 선택합니다. 이름을 입력하고 확인을 클릭합니다.
조이너 변환의 이름 지정 규칙은 *JNR_TransformationName*입니다. 변환의 설명을 입력합니다.
디자이너가 조이너 변환을 작성합니다.
2. 첫 번째 소스에서 모든 입력/출력 포트를 조이너 변환으로 끕니다.
디자이너가 기본적으로 세부 필드로 조이너 변환의 소스 필드에 대한 입력/출력 포트를 작성합니다. 나중에 이 속성을 편집할 수 있습니다.
3. 두 번째 소스에서 모든 입력/출력 포트를 선택하여 조이너 변환으로 끕니다.
디자이너는 기본적으로 소스 필드와 마스터 필드의 두 번째 집합을 구성합니다.

4. 조이너 변환의 제목 표시줄을 두 번 클릭하여 변환을 엽니다.
5. 포트 탭을 클릭합니다.
6. M 열의 아무 상자나 클릭하여 소스에 대한 마스터/세부 관계를 전환합니다.
팁: 정렬되지 않은 조이너 변환의 성능을 개선하려면 행 수가 더 적은 소스를 마스터 소스로 사용합니다. 정렬된 조이너 변환의 성능을 개선하려면 중복 키 값이 더 적은 소스를 마스터 소스로 사용합니다.
7. 특정 포트에 대한 기본값을 추가합니다.
소스 중 하나의 필드가 비어 있을 수 있기 때문에 일부 포트에는 Null 값이 포함될 수 있습니다. 대상 데이터베이스가 NULL을 처리하지 않는 경우 기본값을 지정할 수 있습니다.
8. 조건 탭을 클릭하여 조인 조건을 설정합니다.
9. 추가 단추를 클릭하여 조건을 추가합니다. 여러 조건을 추가할 수 있습니다.
마스터 및 세부 포트에는 일치하는 데이터 유형이 있어야 합니다. 조이너 변환은 같음(=) 조인만을 지원합니다.
10. 속성 탭을 클릭하고 변환에 대한 속성을 구성합니다.
참고: 조건 탭에서 조인 조건을 편집할 수 있습니다. 키워드 AND는 여러 조건을 구분합니다.
11. 확인을 클릭합니다.
12. 메타데이터 확장 탭을 클릭하여 메타데이터 확장을 구성합니다.

조이너 변환에 대한 팁

가능한 경우 데이터베이스에서 조인을 수행합니다.

데이터베이스에서 조인을 수행하는 것이 세션에서 조인을 수행하는 것보다 빠릅니다. 하지만 서로 다른 두 데이터베이스 또는 플랫폼 파일 시스템에서 테이블을 조인하는 경우처럼 데이터베이스에서 조인하는 것이 불가능한 경우도 있습니다. 데이터베이스에서 조인을 수행하려면 다음 옵션을 사용하십시오.

- 데이터베이스에서 테이블을 조인하기 위한 세션 이전 저장 프로시저를 작성합니다.
- 조인 수행을 위해 소스 한정자 변환을 사용합니다.

가능한 경우 정렬된 데이터를 조인하십시오.

정렬된 입력을 사용하도록 조이너 변환을 구성하여 세션 성능을 향상시킬 수 있습니다. 정렬된 데이터를 사용하도록 조이너 변환을 구성하면 통합 서비스가 디스크 입력과 출력을 최소화하여 성능을 개선합니다. 대규모 데이터 집합으로 작업할 때 가장 큰 성능 향상을 얻을 수 있습니다.

정렬되지 않은 조이너 변환의 경우 행 수가 더 적은 소스를 마스터 소스로 지정하십시오.

성능 및 디스크 저장소를 최적화하려면 행 수가 더 적은 소스를 마스터 소스로 지정하십시오. 세션 중에 조이너 변환이 마스터 소스의 각 행을 세부 소스와 비교합니다. 마스터의 고유 행 수가 적을수록 조인 비교 횟수가 줄어들고 조인 프로세스가 빨라집니다.

정렬된 조이너 변환의 경우에는 중복 키 값이 더 적은 소스를 마스터 소스로 지정하십시오.

성능 및 디스크 저장소를 최적화하려면 중복 키 값이 더 적은 소스를 마스터 소스로 지정하십시오. 통합 서비스가 정렬된 조이너 변환을 처리할 때 한 번에 백 개의 키에 대한 행을 캐싱합니다. 마스터 소스에 키 값이 같은 행이 많이 포함된 경우 통합 서비스가 더 많은 행을 캐싱해야 하므로 성능이 저하될 수 있습니다.

제 17 장

조회 변환

이 장에 포함된 항목:

- [조회 변환 개요, 267](#)
- [조회 소스 유형, 268](#)
- [연결된 조회 및 연결되지 않은 조회, 271](#)
- [조회 구성 요소, 272](#)
- [조회 속성, 274](#)
- [조회 쿼리, 280](#)
- [조회 조건, 283](#)
- [조회 캐시, 285](#)
- [여러 행 반환, 286](#)
- [연결되지 않은 조회 변환 구성, 286](#)
- [데이터베이스 교착 상태 복원력, 288](#)
- [조회 변환 작성, 289](#)
- [조회 변환에 대한 팁, 290](#)

조회 변환 개요

매핑의 조회 변환을 사용하여 플랫폼 파일의 데이터, 관계형 테이블, 보기 또는 동의어를 조회합니다.

PowerCenter 클라이언트 및 통합 서비스 모두가 연결할 수 있는 관계형 데이터베이스 또는 플랫폼 파일에서 조회 정의를 가져올 수 있습니다. 또한 소스 한정자에서 조회 정의를 작성할 수 있습니다. 매핑에서 여러 조회 변환을 사용할 수 있습니다. 조회 변환은 활성 또는 수동 변환일 수 있습니다. 연결되거나 연결되지 않은 조회 변환을 구성할 수 있습니다.

통합 서비스는 조회 조건 및 변환의 조회 포트를 기반으로 조회 소스를 쿼리합니다. 조회 변환에서 조회 결과를 대상 또는 다른 변환에 반환합니다. 단일 행 또는 여러 행을 반환하도록 조회 변환을 구성할 수 있습니다.

조회 변환과 함께 다음 태스크를 수행합니다.

- **관련 값 가져오기.** 소스의 값을 기반으로 조회 테이블에서 값을 검색합니다. 예를 들어, 소스에 직원 ID가 있습니다. 조회 테이블에서 직원 이름 검색
- **여러 값 가져오기.** 조회 테이블에서 여러 행을 검색합니다. 예를 들어 부서의 모든 직원을 반환합니다.
- **계산 수행.** 조회 테이블에서 값을 검색하고 그 값을 계산에 사용합니다. 예를 들어, 판매세를 검색하고, 세금을 계산한 다음 대상에 세금을 반환합니다.
- **느린 변경 차원 테이블 업데이트.** 행이 대상에 존재하는지 확인합니다.

다음 유형의 조회를 수행하도록 조회 변환을 구성합니다.

- **관계형 또는 플랫 파일 조회.** 플랫 파일 또는 관계형 테이블에서 조회를 수행합니다. 조회 소스로 관계형 테이블을 사용하여 조회 변환을 작성하는 경우 ODBC를 사용하여 조회 소스에 연결하고 조회 변환에 대한 구조로 테이블 정의를 가져올 수 있습니다. 조회 소스로 플랫 파일을 사용하여 조회 변환을 작성하는 경우 디자이너에서 플랫 파일 마법사를 호출합니다.
- **파이프라인 조회.** JMS 또는 MSMQ 같은 응용 프로그램 소스에 대해 조회를 수행합니다. 소스를 매핑으로 끌어서 조회 변환을 소스 한정자와 연결합니다. 통합 서비스가 조회 캐시에 대한 소스 데이터를 검색할 때 성능이 개선되도록 파티션을 구성합니다.
- **연결되거나 연결되지 않은 조회.** 연결되지 않은 조회 변환에서는 소스 데이터를 검색하고, 조회를 수행하고, 파이프라인에 데이터를 반환합니다. 연결되지 않은 조회 변환은 소스나 대상에 연결되어 있지 않습니다. 파이프라인의 변환은 :LKP 식을 사용하여 조회 변환을 호출합니다. 연결되지 않은 조회 변환은 호출 변환에 하나의 열을 반환합니다.
- **캐시되거나 캐시되지 않은 조회.** 조회 소스를 캐시하여 성능을 개선합니다. 조회 소스를 캐시하는 경우 동적 또는 정적 캐시를 사용할 수 있습니다. 기본적으로 조회 캐시는 정적으로 유지되며 세션 중에 변경되지 않습니다. 동적 캐시를 통해 통합 서비스는 캐시에서 행을 삽입하거나 업데이트합니다. 조회 소스로 대상 테이블을 캐시하는 경우 캐시에서 값을 조회하여 이 값이 대상에 존재하는지 확인할 수 있습니다. 조회 변환은 대상을 삽입하거나 업데이트하기 위해 행을 표시합니다.

조회 소스 유형

조회 변환을 작성하는 경우 관계형 테이블, 플랫 파일 또는 소스 한정자를 조회 소스로 선택할 수 있습니다.

관계형 조회

관계형 테이블을 조회 소스로 사용하여 조회 변환을 작성하는 경우, ODBC를 사용하여 조회 소스에 연결하고 테이블 정의를 조회 변환의 구조로 가져올 수 있습니다.

다음 옵션을 관계형 조회와 함께 사용하십시오.

- 기본 SQL 문을 재정의하여 WHERE 절을 추가하거나 여러 테이블을 쿼리합니다.
- 데이터베이스 지원에 따라 null 데이터를 높음 또는 낮음으로 정렬합니다.
- 데이터베이스 지원에 따라 대/소문자를 구분하는 비교를 수행합니다.

플랫 파일 조회

플랫 파일을 조회 소스로 사용하여 조회 변환을 작성하는 경우 리포지토리에서 플랫 파일 정의를 선택하거나 변환을 작성할 때 소스를 가져옵니다. 플랫 파일 조회 소스를 가져오는 경우 디자이너에서 플랫 파일 마법사를 호출합니다.

플랫 파일 조회와 함께 다음 옵션을 사용합니다.

- 파일 목록을 조회 파일 이름으로 구성하여 간접 파일을 조회 소스로 사용합니다.
- 정렬된 입력을 조회에 사용합니다.
- Null 데이터를 높게 또는 낮게 정렬합니다.
- 플랫 파일 조회와 함께 대/소문자 구분 문자열 비교를 사용합니다.

정렬된 입력 사용

정렬된 입력을 위해 플랫폼 파일 조회 변환을 구성할 경우, 조건 열이 그룹화되어야 합니다. 조건 열이 그룹화되지 않으면 조회 변환에서 올바르게 않은 결과를 반환합니다. 캐싱 성능을 최적화하려면 조건 열을 정렬하십시오.

예를 들어 조회 변환에 다음과 같은 조건이 있는 것으로 가정합니다.

```
OrderID = OrderID1
CustID = CustID1
```

다음 플랫폼 파일 조회 소스에서는 키가 그룹화되어 있지만 정렬되어 있지는 않습니다. 통합 서비스가 데이터를 캐싱할 수 있지만 성능은 최적화되지 않을 수 있습니다.

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	

다음 플랫폼 파일 조회 소스에서는 키가 그룹화되지 않았습니다. 조회 변환에서 올바르게 않은 결과를 반환합니다.

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.

간접 파일에 대해 정렬된 입력을 선택하면 데이터 범위가 파일에서 겹치지 않아야 합니다.

파이프라인 조회

관계형 테이블 또는 플랫폼 파일이 아닌 응용 프로그램 소스에 대해 조회를 수행하려면 파이프라인 조회 변환을 작성합니다. 파이프라인 조회 변환에서는 소스 한정자가 조회 소스입니다. 응용 프로그램 다중 그룹 소스 한정자 변환을 제외한 모든 데이터 소스에 대해 파이프라인 조회를 수행할 수 있습니다.

파이프라인 조회 변환을 구성하는 경우 조회 소스와 소스 한정자는 조회 변환과 다른 파이프라인에 있습니다. 소스 및 소스 한정자는 대상을 포함하지 않는 부분 파이프라인에 있습니다. 통합 서비스는 이 파이프라인에서 소스 데이터를 읽고 조회 변환으로 해당 데이터를 전달하여 캐시를 작성합니다. 성능 향상을 위해 부분 파이프라인에 여러 파티션을 작성할 수 있습니다.

관계형 또는 플랫폼 파일 조회 소스를 처리할 때 성능을 향상시키려면 관계형 또는 플랫폼 파일 조회 변환 대신 파이프라인 조회 변환을 작성합니다. 파티션을 작성하여 조회 소스를 처리하고 조회 변환으로 조회 소스를 전달할 수 있습니다.

연결된 또는 연결되지 않은 파이프라인 조회 변환을 작성하십시오.

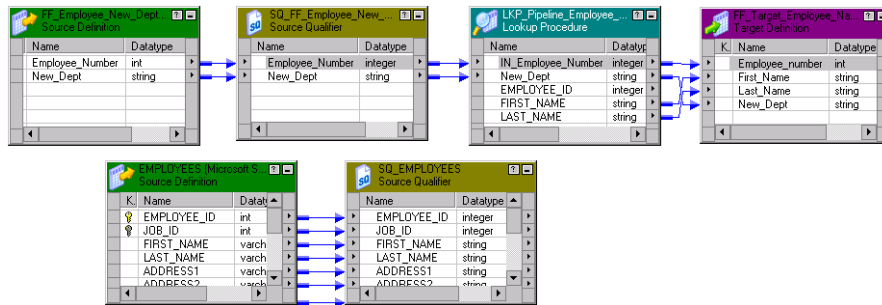
참고: 파이프라인 조회를 위한 실시간 소스가 있는 세션에 대해 HA 복구를 활성화하지 마십시오. 이 경우 예기치 않은 결과가 나타날 수 있습니다.

매핑에서 파이프라인 조회 변환 구성

파이프라인 조회 변환이 있는 매핑은 조회 소스 및 소스 한정자가 있는 부분 파이프라인을 포함합니다. 부분 파이프라인은 대상을 포함하지 않습니다. 통합 서비스는 이러한 파이프라인에서 조회 소스 데이터를 검색하여 데이터를 조회 캐시에 전달합니다.

부분 파이프라인은 세션 속성에서 별개의 대상 로드 순서 그룹에 있습니다. 파이프라인에서 여러 파티션을 작성하여 성능을 향상시킬 수 있습니다. 부분 파이프라인을 포함하여 대상 로드 순서를 구성할 수는 없습니다.

다음 매핑은 조회 소스를 처리하는 부분 파이프라인과 파이프라인 조회 변환이 포함된 매핑을 보여 줍니다.



이 매핑에는 다음 개체가 포함됩니다.

- 조회 소스 정의와 소스 한정자는 별개의 파이프라인에 있습니다. 통합 서비스는 파이프라인에서 조회 소스 데이터를 처리한 후 조회 캐시를 작성합니다.
- 플랫폼 파일 소스에 직원 번호별 새 부서 이름이 있습니다.
- 파이프라인 조회 변환은 소스 파일에서 Employee_Number 및 New_Dept를 받습니다. 파이프라인 조회는 조회 캐시에서 Employee_ID를 조회합니다. 조회 캐시에서 먼저 직원을 검색하고 마지막으로 이름을 검색합니다.
- 플랫폼 파일 대상은 조회 변환에서 Employee_ID, First_Name, Last_Name 및 New_Dept를 받습니다.

연결된 조회 및 연결되지 않은 조회

연결된 조회 변환 또는 연결되지 않은 조회 변환을 구성할 수 있습니다. 연결된 조회 변환은 매핑의 다른 변환에 연결되는 입력 및 출력 포트가 있는 변환입니다. 연결되지 않은 조회 변환은 매핑에 표시되지만 다른 변환에 연결되지 않습니다.

연결되지 않은 조회 변환은 식 변환 또는 집계 변환 같은 변환에서 :LKP 식 결과의 입력을 수신합니다. :LKP 식은 조회 변환에 인수를 전달하고 조회 변환으로부터 결과를 다시 수신합니다. :LKP 식은 조회 결과를 식 변환 또는 집계 변환의 다른 식에 전달하여 결과를 필터링할 수 있습니다.

다음 표에는 연결된 조회와 연결되지 않은 조회 간 차이점이 나열되어 있습니다.

연결된 조회	연결되지 않은 조회
파이프라인에서 직접 입력 값을 받습니다.	다른 변환의 :LKP 식 결과에서 입력 값을 받습니다.
동적 또는 정적 캐시를 사용합니다.	정적 캐시를 사용합니다.
캐시에는 조회 조건의 조회 소스 열과 출력 포트인 조회 소스 열이 포함됩니다.	캐시에는 조회 조건의 모든 조회 및 출력 포트와 조회/반환 포트가 포함됩니다.
동일한 행 또는 삽입의 여러 열을 동적 조회 캐시에 반환합니다.	각 행의 열 1개를 반환 포트에 반환합니다.
조회 조건에 일치하는 값이 없으면 통합 서비스는 모든 출력 포트에 대해 기본값을 반환합니다. 동적 캐싱을 구성한 경우 통합 서비스는 행을 캐시에 삽입하거나 변경하지 않은 상태로 그대로 둡니다.	조회 조건에 일치하는 값이 없으면 통합 서비스는 NULL을 반환합니다.
조회 조건에 일치하는 값이 있으면 통합 서비스는 모든 조회/출력 포트에 대해 조회 조건 결과를 반환합니다. 동적 캐싱을 구성한 경우 통합 서비스는 캐시의 행을 업데이트하거나 행을 변경되지 않은 상태로 그대로 둡니다.	조회 조건에 대한 일치가 발생할 경우 통합 서비스는 조회 조건 결과를 반환 포트에 반환합니다.
여러 출력 값을 다른 변환에 전달합니다. 조회/출력 포트를 다른 변환에 연결합니다.	출력 값 1개를 다른 변환에 반환합니다. 조회 변환 반환 포트에서 다른 변환의 :LKP 식을 포함하는 포트에 값을 전달합니다.
사용자 정의 기본값을 지원합니다.	사용자 정의 기본값을 지원하지 않습니다.

연결된 조회

연결된 조회 변환은 매핑의 소스 또는 대상에 연결된 조회 변환입니다.

연결된 조회 변환이 포함된 매핑을 실행할 경우 통합 서비스에서 다음 단계를 수행합니다.

1. 통합 서비스에서 다른 변환의 값을 조회 변환의 입력 포트에 전달합니다.
2. 각 입력 행마다 통합 서비스에서 변환의 조회 포트 및 조회 조건을 기반으로 조회 소스 또는 캐시를 쿼리합니다.
3. 변환이 캐시되지 않거나 변환에서 정적 캐시를 사용할 경우 통합 서비스가 조회 쿼리의 값을 반환합니다.
변환에서 동적 캐시를 사용할 경우 통합 서비스가 캐시에서 행을 찾지 못하면 캐시에 행을 삽입합니다. 통합 서비스가 캐시에서 행을 찾은 경우 캐시의 행을 업데이트하거나 변경되지 않은 상태로 그대로 둡니다. 행을 삽입, 업데이트 또는 변경 없음으로 플래그 지정합니다.

4. 통합 서비스가 쿼리의 데이터를 반환하여 이를 매핑의 다음 변환에 전달합니다.

변환에서 동적 캐시를 사용할 경우 행을 필터 또는 라우터 변환에 전달하여 새 행을 대상으로 필터링할 수 있습니다.

참고: 이 장에서는 달리 지정하지 않는 한 연결된 조회 변환을 설명합니다.

연결되지 않은 조회

연결되지 않은 조회 변환은 매핑의 소스 또는 대상에 연결되지 않은 조회 변환입니다. 식을 허용하는 변환에서 **:LKP** 식을 사용하여 조회를 호출합니다.

연결되지 않은 조회 변환은 일반적으로 느린 변경 차원 테이블을 업데이트할 때 사용됩니다. 느린 변경 차원 테이블에 대한 자세한 내용은 Informatica 기술 자료(<http://mysupport.informatica.com>)를 참조하십시오.

조회 식에 대한 구문은 **:LKP lookup_transformation_name(argument, argument, ...)**입니다.

각 인수를 나열하는 순서는 조회 변환의 조회 조건 순서와 일치해야 합니다. 조회 변환은 조회 변환 반환 포트를 통해 쿼리 결과를 반환합니다. 조회를 호출하는 변환은 **:LKP** 식이 포함된 포트에서 조회 결과 값을 수신합니다. 조회 쿼리가 값을 반환하지 못하는 경우 포트가 **Null** 값을 수신합니다.

연결되지 않은 조회를 수행하는 경우 동일한 조회를 매핑에서 여러 번 수행할 수 있습니다. 다른 식에서 조회 결과를 테스트하고 결과에 따라 행을 필터링할 수 있습니다.

연결되지 않은 조회 변환이 포함된 매핑을 실행하는 경우 통합 서비스가 다음 단계를 수행합니다.

1. 연결되지 않은 조회 변환은 집계 변환, 식 변환 또는 업데이트 전략 변환과 같은 다른 변환의 **:LKP** 식 결과에서 입력 값을 수신합니다.
2. 통합 서비스가 조회 변환의 조건 및 조회 포트에 따라 조회 소스 또는 캐시를 쿼리합니다.
3. 통합 서비스가 조회 변환의 반환 포트를 통해 값을 반환합니다.
4. 통합 서비스가 반환 값을 **:LKP** 식이 포함된 포트에 전달합니다.

조회 구성 요소

매핑에서 조회 변환을 구성하는 경우 다음 구성 요소를 정의합니다.

- 조회 소스
- 포트
- 속성
- 조건

조회 소스

조회 소스에 대한 플랫폼 파일, 관계형 테이블 또는 소스 한정자를 사용합니다. 조회 변환을 작성하는 경우 다음 위치에서 조회 소스를 작성할 수 있습니다.

- 리포지토리의 관계형 소스 또는 대상 정의
- 리포지토리의 플랫폼 파일 소스 또는 대상 정의
- 통합 서비스 및 PowerCenter 클라이언트 시스템이 연결할 수 있는 테이블 또는 파일
- 매핑의 소스 한정자 정의

조회 테이블은 단일 테이블일 수 있습니다. 또는 조회 SQL 재정의의 사용하여 동일한 데이터베이스에서 여러 테이블을 조인할 수 있습니다. 통합 서비스는 조회 변환에 들어오는 모든 행에 대한 테이블의 메모리 내 캐시 또는 조회 테이블을 쿼리합니다.

통합 서비스는 ODBC 또는 원시 드라이버를 사용하여 조회 테이블에 연결할 수 있습니다. 최적의 성능을 위해 원시 드라이버를 구성합니다.

인덱스 및 조회 테이블

조회 테이블이 포함된 데이터베이스를 수정할 권한이 있는 경우 조회 테이블에 인덱스를 추가하여 조회 초기화 시간을 개선할 수 있습니다. 매우 큰 조회 테이블의 성능을 개선할 수 있습니다. 통합 서비스는 조회 열에서 값을 쿼리, 정렬 및 비교하므로, 인덱스가 조회 조건의 모든 열을 포함해야 합니다.

다음과 같은 유형의 조회를 인덱싱하여 성능을 향상시킬 수 있습니다.

- **캐싱된 조회.** 조회 ORDER BY에서 열을 인덱싱하여 성능을 향상시킬 수 있습니다. 세션 로그에는 ORDER BY 절이 포함됩니다.
- **캐싱되지 않은 조회.** 통합 서비스가 조회 변환에 전달되는 각 행에 대해 SELECT 문을 실행하기 때문에 조회 조건의 열을 인덱싱하여 성능을 향상시킬 수 있습니다.

조회 포트

포트 탭에는 입력 및 출력 포트가 포함되어 있습니다. 또한 포트 탭에는 조회 소스에서 반환할 데이터 열을 나타내는 조회 포트가 포함되어 있습니다. 연결되지 않은 조회 변환은 이 포트에서 호출 변환에 하나의 데이터 열을 반환합니다. 연결되지 않은 조회 변환에는 하나의 반환 포트가 있습니다.

다음 테이블에는 조회 변환의 포트 유형이 설명되어 있습니다.

포트	조회 유형	설명
I	연결됨 연결되지 않음	입력 포트. 조회 조건에서 사용할 각 조회 포트에 대한 입력 포트를 작성합니다. 각 조회 변환에서 하나 이상의 입력 또는 입력/출력 포트가 있어야 합니다.
O	연결됨 연결되지 않음	출력 포트. 다른 변환에 연결할 각 조회 포트의 출력 포트를 작성합니다. 입력 및 조회 포트 모두를 출력 포트 지정할 수 있습니다. 연결된 조회의 경우 하나 이상의 출력 포트가 있어야 합니다. 연결되지 않은 조회의 경우 조회 포트를 반환 포트(R)로 선택하여 반환 값을 전달합니다.
L	연결됨 연결되지 않음	조회 포트. 디자이너는 조회 소스의 각 열을 조회(L) 및 출력 포트(O)로 지정합니다.
R	연결되지 않음	반환 포트. 연결되지 않은 조회 변환에서만 사용합니다. 조회 조건을 기준으로 반환할 데이터 열을 지정합니다. 하나의 조회 포트를 반환 포트 지정할 수 있습니다.

또한 조회 변환은 사용자가 동적 캐시를 사용할 때 구성하는 연결된 식 속성을 활성화합니다. 연결된 식 속성은 조회 캐시를 업데이트하기 위한 데이터가 포함되어 있습니다. 이 속성은 동적 캐시를 업데이트하기 위한 식을 포함하거나 입력 포트 이름을 포함할 수 있습니다.

다음 지침을 사용하여 조회 포트를 구성합니다.

- 플랫폼 파일 조회에서 조회 포트를 삭제하는 경우 세션이 실패합니다.
- 매핑이 조회 포트를 사용하지 않는 경우 관계형 조회에서 조회 포트를 삭제할 수 있습니다. 그러면 통합 서비스가 세션을 실행하는 데 필요한 메모리 양이 감소됩니다.

조회 속성

속성 탭에서 관계형 조회에 대한 SQL 재정의, 조회 소스 이름 캐싱 속성 등의 속성을 구성합니다.

관련 항목:

- [“조회 속성” 페이지 274](#)

조회 조건

조건 탭에서 통합 서비스가 조회 소스의 데이터를 찾는 데 사용하게 할 조건을 입력합니다.

관련 항목:

- [“조회 조건” 페이지 283](#)

조회 속성

조회 속성 탭에서 캐싱 및 여러 개의 일치 항목 등 조회 속성을 구성합니다. 조회 테이블을 쿼리하기 위한 조회 조건 또는 SQL 문을 구성합니다. 조회 테이블 이름을 구성할 수도 있습니다.

매핑을 작성하는 경우 각 조회 변환에 대한 속성을 구성합니다. 세션을 작성하는 경우 각 변환에 대한 데이터 캐시 크기 및 인덱스 등 속성을 재정의할 수 있습니다.

다음 테이블에는 조회 변환 속성이 설명되어 있습니다.

옵션	조회 유형	설명
조회 SQL 재정의	관계형	조회 테이블을 쿼리하기 위해 기본 SQL 문을 재정의합니다. 통합 서비스가 조회 값 쿼리에 사용하게 할 SQL 문을 지정합니다. 조회 캐시가 활성화된 상태로 사용됩니다.
조회 테이블 이름	파이프라인 관계형	변환이 값을 조회하고 캐시하는 소스 한정자 또는 테이블의 이름입니다. 조회 변환을 생성하는 경우 소스, 대상 또는 소스 한정자를 조회 소스로 선택합니다. 조회 변환을 생성하는 경우 다른 데이터베이스에서 테이블, 보기 또는 동의어를 가져올 수도 있습니다. 조회 SQL 재정의의 경우 조회 테이블 이름을 입력할 필요가 없습니다.
조회 소스 필터	관계형	조회 변환의 임의 포트에 있는 데이터 값을 기반으로 통합 서비스가 수행하는 조회를 제한합니다. 조회 캐시가 활성화된 상태로 사용됩니다.
조회 캐싱 설정	플랫 파일 파이프라인 관계형	통합 서비스가 세션 중에 조회 값을 캐시하는지 여부를 나타냅니다. 조회 캐싱을 활성화하는 경우 통합 서비스는 세션 중에 조회 소스를 한 번 쿼리하고 값을 캐시하고 캐시의 값을 조회합니다. 조회 값을 캐시하면 세션 성능을 개선할 수 있습니다. 캐싱을 비활성화하는 경우 행이 변환에 전달될 때마다 통합 서비스가 조회 값의 조회 소스에 대해 select 문을 실행합니다. 참고: 통합 서비스는 항상 플랫 파일 조회 및 파이프라인 조회를 캐시합니다.

옵션	조회 유형	설명
일치 항목이 여러 개인 경우의 조회 정책	플랫 파일 파이프라인 관계형	<p>조회 변환이 조회 조건에 일치하는 여러 행을 찾았을 때 반환할 행을 결정합니다. 다음 값 중 하나를 선택합니다.</p> <ul style="list-style-type: none"> - 첫 번째 값 사용. 조회 조건에 일치하는 첫 번째 행을 반환합니다. - 마지막 값 사용. 조회 조건에 일치하는 마지막 행을 반환합니다. - 모든 값 사용. 일치하는 모든 행을 반환합니다. - 임의 값 사용. 통합 서비스가 조회 조건과 일치하는 첫 번째 값을 반환합니다. 모든 조회 변환 포트 대신 키 포트를 기반으로 인덱스를 작성합니다. - 오류 보고. 통합 서비스에서 오류를 보고하며 행은 반환하지 않습니다. 업데이트 시 이전 값 출력 옵션을 활성화하지 않는 경우 일치 항목이 여러 개인 경우의 조회 정책 옵션이 동적 조회에 대해 오류 보고로 설정됩니다.
조회 조건	플랫 파일 파이프라인 관계형	조건 탭에서 설정하는 조회 조건이 표시됩니다.
연결 정보	관계형	<p>조회 테이블을 포함하는 데이터베이스를 지정합니다. 매핑, 세션 또는 매개 변수 파일에서 데이터베이스를 정의할 수 있습니다.</p> <ul style="list-style-type: none"> - 매핑. 연결 개체를 선택합니다. 데이터베이스 연결 유형을 지정할 수도 있습니다. 관계형 연결인 경우 연결 이름 앞에 Relational:을 입력합니다. 응용 프로그램 연결인 경우 연결 이름 앞에 Application:을 입력합니다. - 세션. \$Source 또는 \$Target 연결 변수를 사용합니다. 이러한 변수 중 하나를 사용하는 경우 조회 테이블이 소스 또는 대상 데이터베이스에 상주해야 합니다. 각 변수에 대한 세션 속성에서 데이터베이스 연결을 지정합니다. - 매개 변수 파일. 세션 매개 변수 \$DBConnectionName 또는 \$AppConnectionName을 사용하고 매개 변수 파일에서 해당 매개 변수를 정의합니다. <p>기본적으로, 사용자가 조회 변환을 생성할 때 소스 테이블을 선택하는 경우에는 \$Source를, 대상 테이블을 선택하는 경우에는 \$Target을 디자이너가 지정합니다. 세션 속성에서 해당 값을 재정의할 수 있습니다. 통합 서비스에서 데이터베이스 연결 유형을 확인할 수 없으면 세션이 실패합니다.</p>
소스 유형	플랫 파일 파이프라인 관계형	조회 변환이 관계형 테이블, 플랫 파일 또는 소스 한정자에서 값을 읽는다는 것을 나타냅니다.
추적 수준	플랫 파일 파이프라인 관계형	세션 로그에 포함된 세부 정보의 양을 설정합니다.
조회 캐시 디렉터리 이름	플랫 파일 파이프라인 관계형	<p>조회 소스를 캐시하도록 조회 변환을 구성하는 경우 조회 캐시 파일을 작성하는 데 사용되는 디렉터리를 지정합니다. 또한 조회 지속형 옵션을 선택하면 지속형 조회 캐시 파일을 저장합니다.</p> <p>기본적으로 통합 서비스는 통합 서비스에 대해 구성된 \$PMCacheDir 디렉터리를 사용합니다.</p>

옵션	조회 유형	설명
영구 조회 캐시	플랫 파일 파이프라인 관계형	통합 서비스가 2개 이상의 캐시 파일로 구성된 지속형 조회 캐시를 사용하는지 여부를 나타냅니다. 조회 변환이 지속형 조회 캐시에 대해 구성되고 지속형 조회 캐시 파일이 없는 경우, 통합 서비스에서 세션 중에 파일을 생성합니다. 조회 캐시가 활성화된 상태로 사용됩니다.
조회 데이터 캐시 크기 조회 인덱스 캐시 크기	플랫 파일 파이프라인 관계형	기본값은 자동입니다. 통합 서비스가 메모리에서 데이터 캐시 및 인덱스에 할당하는 최대 크기를 나타냅니다. 캐시에 숫자 값을 사용하거나, 매개 변수 파일의 캐시 값을 사용하거나, 자동 설정을 통해 캐시 크기를 설정하도록 통합 서비스를 구성할 수 있습니다. 캐시 크기를 결정하도록 통합 서비스를 구성하는 경우 통합 서비스가 캐시에 할당하는 최대 메모리 양도 구성할 수 있습니다. 통합 서비스가 세션을 초기화할 때 구성된 양의 메모리를 할당할 수 없으면 해당 세션이 실패합니다. 통합 서비스가 메모리에 일부 데이터 캐시 데이터를 저장할 수 없는 경우 디스크에 페이징합니다. 조회 캐시가 활성화된 상태로 사용됩니다.
동적 조회 캐시	플랫 파일 파이프라인 관계형	동적 조회 캐시를 사용하도록 나타냅니다. 대상 테이블에 행을 전달하면서 조회 캐시에서 행을 삽입하거나 업데이트합니다. 조회 캐시가 활성화된 상태로 사용됩니다.
업데이트 시 이전 값 출력	플랫 파일 파이프라인 관계형	동적 캐시가 활성화된 상태로 사용됩니다. 이 속성을 활성화하는 경우 통합 서비스가 조회/출력 포트에서 이전 값을 출력합니다. 통합 서비스는 캐시의 행을 업데이트할 때 조회 캐시에 존재한 값을 출력한 다음 입력 데이터를 바탕으로 행을 업데이트합니다. 통합 서비스가 캐시에 행을 삽입할 때는 null 값이 출력됩니다. 이 속성을 비활성화하는 경우 통합 서비스가 조회/출력 및 입력/출력 포트에서 동일한 값을 출력합니다. 이 속성은 기본적으로 활성화됩니다.
동적 캐시 조건 업데이트	플랫 파일 파이프라인 관계형	동적 캐시를 업데이트할지 여부를 표시하는 식입니다. 조회 포트 또는 입력 포트를 사용하여 식을 생성합니다. 조건 식에는 입력 값 또는 조회 캐시의 값이 포함될 수 있습니다. 조건이 true이고 캐시에 데이터가 있는 경우 통합 서비스가 캐시를 업데이트합니다. 동적 캐시가 활성화된 상태로 사용됩니다. 기본값은 true입니다.
캐시 파일 이름 접두사	플랫 파일 파이프라인 관계형	지속형 조회 캐시와 함께 사용합니다. 지속형 조회 캐시 파일과 함께 사용할 파일 이름 접두사를 지정합니다. 통합 서비스는 디스크에 저장하는 지속형 캐시 파일에 대한 파일 이름으로 파일 이름 접두사를 사용합니다. 접두사를 입력합니다. .idx 또는 .dat를 입력하지 마십시오. 파일 이름 접두사에 대한 매개 변수 또는 변수를 입력할 수 있습니다. 매개 변수 파일에 정의할 수 있는 모든 매개 변수 또는 변수 유형을 사용할 수 있습니다. 지속형 명명된 캐시 파일이 있는 경우 통합 서비스가 파일에서 메모리 캐시를 작성합니다. 지속형 명명된 캐시 파일이 없는 경우 통합 서비스가 지속형 캐시 파일을 다시 작성합니다.
조회 소스에서 다시 캐시	플랫 파일 파이프라인 관계형	지속형 조회 캐시와 함께 사용합니다. 지속형 조회 캐시를 사용하고 이 옵션을 활성화하면 통합 서비스가 조회 변환 인스턴스를 처음 호출할 때 조회 소스에서 지속형 조회 캐시를 다시 작성합니다. 이 옵션을 활성화하지 않으면 통합 서비스가 지속형 캐시 파일을 재사용합니다.

옵션	조회 유형	설명
다른 업데이트 삽입	플랫 파일 파이프라인 관계형	동적 캐싱이 활성화된 상태로 사용합니다. 조회 변환에 들어가는 삽입 유형의 행에 적용됩니다. 활성화한 경우 통합 서비스가 캐시에서 행을 삽입하고 기존 행을 업데이트합니다. 비활성화한 경우 통합 서비스가 기존 행을 업데이트하지 않습니다.
업데이트 기타 항목 삽입	플랫 파일 파이프라인 관계형	동적 캐싱이 활성화된 상태로 사용합니다. 조회 변환에 들어가는 업데이트 유형의 행에 적용됩니다. 활성화한 경우 통합 서비스가 기존 행을 업데이트하고, 새로운 행이면 이 행을 삽입합니다. 비활성화한 경우 통합 서비스가 새 행을 삽입하지 않습니다.
날짜/시간 형식	플랫 파일	열기 단추를 클릭하여 날짜/시간 형식을 선택합니다. 형식 및 필드 너비를 정의합니다. 밀리초, 마이크로초 또는 나노초 형식의 필드 너비는 29입니다. 포트에 대한 날짜/시간 형식을 선택하지 않은 경우 날짜/시간 형식을 입력할 수 있습니다. 기본값은 MM/DD/YYYY HH24:MI:SS입니다. 날짜/시간 형식에서 포트 크기를 변경하지 않습니다.
1000 단위 구분 기호	플랫 파일	포트에 대한 1000 단위 구분 기호를 정의하지 않는 경우 통합 서비스가 여기에 정의된 속성을 사용합니다. 구분 기호 없음, 쉼표 또는 마침표를 선택할 수 있습니다. 기본값은 구분 기호 없음입니다.
소수 구분 기호	플랫 파일	포트 탭 또는 조회 정의의 특정 필드에 대한 소수 구분 기호를 정의하지 않는 경우 통합 서비스가 여기에 정의된 속성을 사용합니다. 쉼표 또는 마침표 소수 구분 기호를 선택할 수 있습니다. 기본값은 마침표입니다.
대/소문자 구분 문자열 비교	플랫 파일 파이프라인	문자열 열에서 조회를 수행할 경우 통합 서비스가 대/소문자 구분 문자열 비교를 사용합니다. 관계형 조회의 경우 대/소문자 구분 비교는 데이터베이스 지원에 따라 다릅니다.
Null 순서	플랫 파일 파이프라인	통합 서비스에서 Null 값 순서를 지정하는 방법을 결정합니다. Null 값을 높게 또는 낮게 정렬하도록 선택할 수 있습니다. 기본적으로 통합 서비스는 Null 값을 높게 정렬합니다. 이 경우 통합 서비스 구성을 재정의하여 비교 연산자의 Null을 높게, 낮게 또는 Null로 처리합니다. 관계형 조회의 경우 Null 순서는 데이터베이스 기본값에 따라 다릅니다.
정렬된 입력	플랫 파일 파이프라인	조회 파일 데이터가 정렬된 순서로 되어 있는지 여부를 나타냅니다. 이 옵션은 파일 조회의 조회 성능을 높입니다. 정렬된 입력을 활성화하고 조건 열이 그룹화되지 않은 경우 통합 서비스에서 세션이 실패합니다. 조건 열이 그룹화되었지만 정렬되지 않은 경우 통합 서비스는 정렬된 입력이 구성되지 않은 것처럼 조회를 처리합니다.
조회 소스가 정적임	플랫 파일 파이프라인 관계형	조회 소스가 세션에서 변경되지 않습니다.

옵션	조회 유형	설명
사전 빌드 조회 캐시	플랫 파일 파이프라인 관계형	<p>조회 변환이 데이터를 수신하기 전에 통합 서비스가 조회 캐시를 작성하도록 할 수 있습니다. 통합 서비스는 여러 조회 캐시 파일을 동시에 작성하여 성능을 개선할 수 있습니다.</p> <p>이 옵션을 매핑 또는 세션에서 구성할 수 있습니다. 조회 변환 옵션을 자동으로 구성하면 통합 서비스가 세션 수준 설정을 사용합니다.</p> <p>다음 옵션 중 하나를 구성합니다.</p> <ul style="list-style-type: none"> - 자동. 통합 서비스가 세션에 구성된 값을 사용합니다. - 항상 허용. 통합 서비스가 조회 변환에서 첫 번째 소스 행을 수신하기 전에 조회 캐시를 작성할 수 있습니다. 통합 서비스가 추가 파이프라인을 작성하여 캐시를 작성합니다. - 항상 허용 안 함. 조회 변환이 첫 번째 행을 수신하기 전에 통합 서비스가 조회 캐시를 작성할 수 없습니다. <p>통합 서비스가 동시에 작성할 수 있는 파이프라인의 수를 구성해야 합니다. 조회 캐시 생성을 위한 추가 동시 파이프라인 세션 속성을 구성하십시오. 이 속성이 0보다 클 경우 통합 서비스가 조회 캐시를 사전 빌드할 수 있습니다.</p>
초 단위 이하 전체 자릿수	관계형	<p>날짜/시간 포트에 대한 초 단위 이하 전체 자릿수를 지정합니다.</p> <p>관계형 조회의 경우 날짜/시간 데이터에 대한 편집 가능한 소수 자릿수가 있는 데이터베이스의 전체 자릿수를 변경할 수 있습니다. Oracle 타임스탬프, Informix 날짜/시간 및 Teradata 타임스탬프 데이터 유형에 대한 초 단위 이하 전체 자릿수를 변경할 수 있습니다.</p> <p>0 ~ 9의 양의 정수 값을 입력합니다. 기본값은 6마이크로초입니다. 무시다운 최적화를 활성화하는 경우 초 단위 이하 전체 자릿수 설정에 관계없이 데이터베이스가 전체 날짜/시간 값을 반환합니다.</p>

세션에서 조회 속성 구성

세션을 구성하는 경우 세션에 고유한 조회 속성을 구성할 수 있습니다.

- **플랫 파일 조회.** 소스 파일 디렉터리, 파일 이름 및 파일 유형 등 조회 위치 정보를 구성합니다.
- **관계형 조회.** 세션 속성에서 `$Source` 및 `$Target` 변수를 정의할 수 있습니다. 또한 `$DBConnectionName` 또는 `$AppConnectionName` 세션 매개 변수를 사용하도록 연결 정보를 재정의할 수 있습니다.
- **파이프라인 조회.** 소스 파일 디렉터리, 파일 이름 및 파일 유형 등 조회 소스 파일 속성을 구성합니다. 소스가 관계형 테이블 또는 응용 프로그램 소스인 경우 연결 정보를 구성합니다.

세션에서 플랫 파일 조회 구성

세션에서 플랫 파일 조회를 구성하는 경우 매핑 탭의 변환 보기에서 조회 소스 파일 속성을 구성합니다. 조회 변환을 선택하고 변환에 대한 세션 속성에서 플랫 파일 속성을 구성합니다.

다음 테이블에는 플랫폼 파일 조회에 대해 구성하는 세션 속성이 설명되어 있습니다.

속성	설명
조회 소스 파일 디렉터리	<p>디렉터리 이름을 입력합니다. 기본적으로 통합 서비스는 조회 파일에 대한 프로세스 변수 디렉터리, <code>\$PMLookupFileDir</code>에서 찾습니다.</p> <p>전체 경로 및 파일 이름을 입력할 수 있습니다. 조회 소스 파일 이름 필드에서 디렉터리 및 파일 이름 모두를 지정하는 경우 이 필드를 지웁니다. 통합 서비스가 세션을 실행할 때 이 필드를 조회 소스 파일 이름 필드와 연결합니다.</p> <p><code>\$InputFileName</code> 세션 매개 변수를 입력하여 파일 이름을 구성할 수도 있습니다.</p>
조회 소스 파일 이름	<p>조회 파일의 이름입니다. 간접 파일을 사용하는 경우 통합 서비스가 읽도록 하려는 간접 파일 이름을 입력합니다.</p> <p>조회 파일 매개 변수 <code>\$LookupFileName</code>을 입력하여 세션에 대한 조회 파일 이름을 변경할 수도 있습니다.</p> <p>소스 파일 디렉터리 필드에서 디렉터리 및 파일 이름 모두를 구성하는 경우 조회 소스 파일 이름을 지웁니다. 통합 서비스가 조회 소스 파일 이름을 세션에 대한 조회 소스 파일 디렉터리 필드와 연결합니다. 예를 들어 조회 소스 파일 디렉터리 필드에 <code>"C:\lookup_data\"</code>가 포함되어 있고 조회 소스 파일 이름 필드에 <code>"filename.txt"</code>가 포함되어 있습니다. 통합 서비스가 세션을 시작할 때 <code>"C:\lookup_data\filename.txt"</code>를 찾습니다.</p>
조회 소스 파일 유형	<p>조회 소스 파일에 소스 데이터 또는 동일한 파일 속성이 있는 파일 목록이 포함되는지 여부를 나타냅니다. 조회 소스 파일에 소스 데이터가 포함된 경우 직접을 선택합니다. 조회 소스 파일에 파일 목록이 포함된 경우 간접을 선택합니다.</p> <p>간접을 선택하면 통합 서비스가 모든 파일에 대해 하나의 캐시를 작성합니다. 정렬된 입력을 간접 파일과 함께 사용하는 경우 해당 파일의 데이터 범위가 겹치지 않는지 확인합니다. 데이터 범위가 겹치는 경우 통합 서비스가 해당 조회를 정렬되지 않은 조회 소스로 처리합니다.</p>

세션에서 관계형 조회 구성

세션에서 관계형 조회를 구성하는 경우 매핑 탭의 변환 보기에서 조회 데이터베이스에 대한 연결을 구성합니다. 조회 변환을 선택하고 변환에 대한 세션 속성에서 연결을 구성합니다.

다음 옵션에서 선택하여 관계형 조회 변환에 대한 연결을 구성합니다.

- 관계형 또는 응용 프로그램 연결을 선택합니다.
- `$Source` 또는 `$Target` 연결 변수를 사용하여 데이터베이스 연결을 구성합니다.
- 세션 매개 변수 `$DBConnectionName` 또는 `$AppConnectionName`을 구성하고 매개 변수 파일에서 세션 매개 변수를 정의합니다.

세션에서 파이프라인 조회 구성

세션에서 파이프라인 조회를 구성하는 경우 매핑 탭의 소스 노트에서 조회 소스 파일의 위치 또는 조회 테이블의 연결을 구성합니다. 조회 소스를 나타내는 소스 한정자를 선택합니다.

조회 쿼리

통합 서비스는 조회 변환에서 구성된 포트 및 속성을 기반으로 조회를 쿼리합니다. 통합 서비스는 첫 번째 행이 조회 변환에 들어올 때 기본 조회 쿼리를 실행합니다.

관계형 테이블에 대해 관계형 조회 또는 파이프라인 조회를 사용할 경우 조회 쿼리를 재정의할 수 있습니다. 재정의의 사용하여 **ORDER BY** 절을 변경하거나, **WHERE** 절을 추가하거나, 캐시되기 전에 조회 데이터를 변환할 수 있습니다.

SQL 재정의의를 구성하고 조회 쿼리에서 필터링할 경우 통합 서비스가 필터를 무시합니다.

기본 조회 쿼리

기본 조회 쿼리에 다음 문이 포함되어 있습니다.

SELECT

SELECT 문에는 매핑의 모든 조회 포트가 포함되어 있습니다. 조회 쿼리에 대한 **SELECT** 문을 보려면 조회 SQL 재정의의 속성을 선택합니다.

ORDER BY

ORDER BY 절은 조회 변환에 표시되는 것과 동일한 순서로 열이 표시되도록 합니다. 통합 서비스에서 **ORDER BY** 절을 생성합니다. 기본 SQL을 생성할 경우 이 절이 표시되지 않습니다.

조회 쿼리 재정의

조회 SQL 재정의의는 소스 한정자 변환에서 사용자 지정 쿼리를 입력하는 것과 유사합니다. 관계형 조회에 대해 조회 쿼리를 재정의할 수 있습니다. 전체 재정의의를 입력하거나, 기본 SQL 문을 생성한 후 편집할 수 있습니다. 디자인어는 조회 SQL 재정의의를 위한 기본 SQL 문을 생성할 때 조회 조건의 조회/출력 포트 및 조회/반환 포트를 포함합니다.

다른 조회 쿼리를 입력하거나 기존 조회 쿼리를 편집할 수 있습니다. 조회 쿼리에는 조회 포트, 출력 포트 및 반환 포트가 포함됩니다.

ORDER BY 절 재정의

기본적으로 통합 서비스는 캐싱되는 조회에 대해 **ORDER BY** 절을 생성합니다. **ORDER BY** 절에는 모든 조회 조건 포트가 포함됩니다. 성능을 향상시키기 위해 기본 **ORDER BY** 절을 억제하고 열 수가 더 적은 재정의의 **ORDER BY**를 입력할 수 있습니다.

참고: SQL 재정의의는 조회 키를 기준으로 정렬된 데이터를 반환해야 합니다. 조회 변환이 조회의 모든 행을 검색하는 경우 통합 서비스가 정렬된 순서의 키를 사용하여 데이터 캐시를 작성합니다. 행이 정렬되지 않은 경우 통합 서비스가 캐시의 모든 행을 검색할 수 없습니다. 데이터가 키로 정렬되지 않은 경우 예기치 않은 결과가 나올 수 있습니다.

푸시다운 최적화를 사용하는 경우, **ORDER BY** 절을 재정의하거나 생성된 **ORDER BY** 절을 주석 기호로 억제할 수 있습니다.

사용자가 재정의의에 **ORDER BY** 절을 입력할 경우에도 통합 서비스는 항상 **ORDER BY** 절을 생성합니다. 생성되는 **ORDER BY** 절을 억제하려면 **ORDER BY** 재정의의 뒤에 대시 두 개('--')를 배치하십시오. 예를 들어 조회 변환에서 다음과 같은 조회 조건을 사용한다고 가정합니다.

ITEM_ID = IN_ITEM_ID

PRICE <= IN_PRICE

이 조회 변환에는 매핑에 사용되는 조회 조건 포트 2개(**ITEM_ID**, 및 **PRICE**)가 포함됩니다. 하나 이상의 열을 포함하는 **ORDER BY** 절을 입력할 경우 조회 조건의 포트와 같은 순서로 열을 입력합니다. 또한, 모든 데이터베이스 예약어를 따옴표로 묶어야 합니다. 조회 **SQL** 재정의에 다음과 같은 조회 쿼리를 입력합니다.

```
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 --
```

관계형 조회의 기본 **ORDER BY** 절을 재정의하려면 다음 단계를 완료하십시오.

1. 조회 변환에서 조회 쿼리를 생성합니다.
2. 조회 조건에 나타나는 순서와 동일하게 조건 포트를 포함하는 **ORDER BY** 절을 입력합니다.
3. **ORDER BY** 절 뒤에 대시 2개('--')를 주석 기호로 배치하여 통합 서비스가 생성하는 **ORDER BY** 절을 억제합니다.

주석 기호를 추가하지 않은 채 **ORDER BY** 절을 포함하는 조회 쿼리를 재정의하면 조회가 실패합니다.

참고: Sybase의 경우 **ORDER BY** 절에서 열이 16개로 제한됩니다. 조회 조건의 포트를 포함하여 16개를 넘는 조회/출력 포트가 조회 변환에 있는 경우, **ORDER BY** 절을 재정의하거나 여러 개의 조회 변환을 사용하여 조회 테이블을 쿼리하십시오.

예약어

조회 이름 또는 열 이름에 데이터베이스 **MONTH** 또는 **YEAR** 같은 예약어가 포함되어 있는 경우, 통합 서비스가 데이터베이스에 대해 **SQL**을 실행하면 데이터베이스 오류로 인해 세션이 실패합니다. 통합 서비스 설치 디렉터리에서 예약어 파일 **reswords.txt**를 생성하고 유지 관리해야 합니다.

통합 서비스 설치 디렉터리에서 예약어 파일 **reswords.txt**를 생성하고 유지 관리해야 합니다. 통합 서비스는 세션을 초기화할 때 **reswords.txt** 파일을 검색하고 예약어를 따옴표로 묶은 다음 소스, 대상 및 조회 데이터베이스에 대해 **SQL**을 실행합니다.

Microsoft SQL Server 및 Sybase와 같은 일부 데이터베이스가 따옴표로 묶은 식별자에 대해 **SQL-92** 표준을 사용하도록 설정해야 할 수 있습니다. 연결 환경 **SQL**을 사용하여 명령을 실행합니다. 예를 들어 Microsoft SQL Server의 경우 다음 명령을 사용합니다.

```
SET QUOTED_IDENTIFIER ON
```

조회 쿼리 재정의에 대한 지침

조회 쿼리를 재정의할 경우 특정한 규칙 및 지침이 적용됩니다.

조회 **SQL** 쿼리를 재정의하는 경우 다음 지침을 고려하십시오.

- 관계형 조회에 대한 조회 **SQL** 쿼리를 재정의할 수 있습니다.
- 기본 쿼리를 생성한 다음 재정의의 구성합니다. 이렇게 해야 모든 조회/출력 포트가 쿼리에 포함됩니다. **SELECT** 문의 포트를 추가하거나 뺄 경우 세션이 실패합니다.
- 소스 조회 필터를 추가하여 조회 캐시에 추가되는 행을 필터링합니다. 이렇게 하면 통합 서비스가 **WHERE** 절에 일치하는 동적 캐시 및 대상 테이블에 행을 삽입합니다.
- 여러 조회 변환이 조회 캐시를 공유하는 경우 동일한 조회 **SQL** 재정의의 각 조회 변환에 사용합니다.
- 모든 행을 반환하는 조회 변환을 구성한 경우 통합 서비스는 정렬된 키를 사용하여 조회 캐시를 작성합니다. 조회 변환이 조회의 모든 행을 검색하는 경우 통합 서비스가 정렬된 순서의 키를 사용하여 데이터 캐시를 작성합니다. 행이 정렬되지 않은 경우 통합 서비스가 캐시의 모든 행을 검색할 수 없습니다. 데이터가 키로 정렬되지 않은 경우 예기치 않은 결과가 나올 수 있습니다.
- **ORDER BY** 절에는 조회 조건에 표시되는 동일한 순서로 조건 포트가 포함되어야 합니다.
- **ORDER BY** 절을 재정의하는 경우 설명 표기법을 사용하여 조회 변환이 생성하는 **ORDER BY** 절을 억제하십시오.

- 푸시다운 최적화를 사용하는 경우 **ORDER BY** 절을 재정의하거나 생성된 **ORDER BY** 절을 설명 표기법으로 억제할 수 없습니다.
- 조회 쿼리의 테이블 이름 또는 열 이름에 예약어가 포함되는 경우 예약어를 따옴표로 묶어야 합니다.
- 캐시되지 않은 조회에 대한 조회 쿼리를 재정의하려면 통합 서비스가 여러 일치를 찾을 때 값을 반환하도록 선택하십시오.
- 기본 SQL 문의 열은 추가하거나 삭제할 수 없습니다.

조회 쿼리 재정의 단계

다음 단계를 사용하여 기본 조회 SQL 쿼리를 재정의합니다.

1. 속성 탭의 조회 SQL 재정의 필드에서 SQL 편집기를 엽니다.
2. SQL 생성을 클릭하여 기본 **SELECT** 문을 생성합니다. 조회 SQL 재정의에 입력합니다.
3. 데이터베이스에 연결하고 유효성 검사를 클릭하여 조회 SQL 재정의에 테스트합니다.
4. 확인을 클릭하여 속성 탭으로 돌아옵니다.

캐시되지 않은 조회에 대한 SQL 재정의

캐시되지 않은 조회에 대한 SQL 재정의의 정의를 할 수 있습니다. 통합 서비스는 캐시되지 않은 조회에 대한 재정의의 문으로부터 캐시를 작성하지 않습니다. 재정의의 **SELECT** 문에서 SQL 함수를 사용할 수 있습니다. **WHERE** 및 **ORDER BY** 절을 비롯하여 모든 SQL 쿼리를 재정의할 수 있습니다.

기본 **SELECT** 문을 생성할 경우, 디자이너는 조회 조건에 기반하여 조회 및 출력 포트와 **WHERE** 절을 포함하는 **SELECT** 문을 생성합니다. 조회 변환이 연결되지 않은 조회인 경우 **SELECT** 문에는 조회 포트와 반환 포트가 포함됩니다. 통합 서비스는 조회 변환의 조건 탭에서 구성한 조건으로부터 **WHERE** 절을 생성하지는 않습니다.

SELECT 쿼리의 각 열은 별칭을 사용하여 출력 열을 정의합니다. SQL 문에서 이 구문을 변경하지 마십시오. 변경할 경우 쿼리가 실패하게 됩니다. **WHERE** 절에서 입력 포트를 참조하려면 매개 변수 바인딩을 구성합니다. 다음 예제에서는 **Name** 포트를 참조하는 **WHERE** 문을 포함합니다.

```
SELECT EMPLOYEE.NAME as NAME, max(EMPLOYEE.ID) as ID from EMPLOYEE WHERE EMPLOYEE.NAME=?NAME1?
```

캐시되지 않은 조회에 대한 SQL 편집기에는 포트 탭에 있는 입력 포트 및 조회 포트가 표시됩니다.

함수를 SQL 문에 추가하는 경우, 반환 데이터 유형은 **ALIAS** 열의 데이터 유형과 일치해야 합니다. 예를 들어 **ID**의 데이터 유형은 **MAX** 함수의 반환 유형과 일치합니다.

```
SELECT EMPLOYEE.NAME as NAME, MAX(EMPLOYEE.ID) as ID FROM EMPLOYEE
```

참고: 캐시되지 않은 조회에 대해서는 SQL 재정의에서 하위 쿼리를 사용할 수 없습니다.

조회 소스 필터

캐싱이 활성화된 관계형 조회 변환에 대한 조회 소스 필터를 구성할 수 있습니다. 통합 서비스가 조회 소스 테이블에서 수행하는 조회 수를 제한하려면 조회 소스 필터를 추가합니다.

조회 소스 필터를 구성한 경우 통합 서비스가 필터 문의 결과를 기반으로 조회를 수행합니다. 예를 들어, **ID**가 510보다 큰 모든 직원을 성을 검색해야 할 경우가 있습니다.

EmployeeID 열에서 다음 조회 소스 필터를 구성하십시오.

```
EmployeeID >= 510
```

EmployeeID는 조회 변환의 입력 포트입니다. 통합 서비스가 소스 행을 읽을 때 **EmployeeID** 값이 510보다 클 경우 캐시에서 조회를 수행합니다. **EmployeeID**가 510보다 작거나 같을 경우 조회 변환에서 성을 검색하지 않습니다.

푸시다운 최적화용으로 구성된 세션의 조회 쿼리에 조회 소스 필터를 추가할 경우 통합 서비스에서 SQL 재정의의 결과를 나타내는 보기를 작성합니다. 통합 서비스가 이 보기에 대해 SQL 쿼리를 실행하여 데이터베이스에 변환 논리를 푸시합니다.

조회 소스 행 필터링

통합 서비스가 관계형 조회 소스에서 수행하는 조회 수를 제한하려면 조회 소스 필터를 추가합니다.

1. 매핑 디자이너 또는 변환 개발자에서 조회 변환을 엽니다.
2. **속성** 탭을 선택합니다.
3. 캐싱이 활성화되었는지 확인합니다.
4. **조회 소스 필터** 필드에서 **열기** 단추를 클릭합니다.
5. SQL 편집기에서 입력 포트를 선택하거나 필터링할 조회 변환 포트를 입력합니다.
6. 필터 조건을 입력합니다.
필터 조건에 **WHERE** 키워드를 포함하지 않습니다. 문자열 매핑 매개 변수 및 변수를 문자열 식별자로 묶습니다.
7. 조건의 유효성을 검사하려면 쿼리에 포함된 소스가 있는 ODBC 데이터 소스를 선택합니다.
8. 이 데이터베이스에 연결할 사용자 이름과 암호를 입력합니다.
9. **유효성 검사**를 클릭합니다.
디자이너가 쿼리를 실행하고 해당 구문이 올바른지 여부를 보고합니다.

조회 조건

통합 서비스가 조회 조건을 사용하여 조회 소스에서 데이터를 찾습니다. 조회 조건은 SQL 쿼리의 **WHERE** 절과 유사합니다. 조회 변환에 조회 조건을 구성한 경우 소스 데이터에 있는 하나 이상의 열 값을 조회 소스 또는 캐시의 값과 비교하십시오.

예를 들어 소스 데이터는 **employee_number**를 포함합니다. 조회 소스 테이블은 **employee_ID**, **first_name** 및 **last_name**을 포함합니다. 다음 조회 조건을 구성합니다.

```
employee_ID = employee_number
```

employee_number마다 통합 서비스가 조회 소스의 **employee_ID**, **last_name** 및 **first_name** 열을 반환합니다.

통합 서비스가 조회 소스에서 둘 이상의 행을 반환할 수 있습니다. 다음 조회 조건을 구성합니다.

```
employee_ID > employee_number
```

통합 서비스가 소스 직원 번호보다 큰 모든 **employee_ID** 번호의 행을 반환합니다.

조회 변환에 대해 조건을 입력할 때 다음 지침을 사용합니다.

- 조회 조건의 열 데이터 유형이 일치해야 합니다.
- 모든 조회 변환에서 조회 조건을 입력해야 합니다.
- 조회 조건의 각 조회 포트에 대해 입력 포트를 하나씩 사용합니다. 변환에서 둘 이상의 조건에 같은 입력 포트를 사용합니다.
- 여러 조건을 입력하는 경우 통합 서비스가 각 조건을 **OR**가 아닌 **AND**로 평가합니다. 사용자가 구성하는 모든 조건과 일치하는 행을 통합 서비스에서 반환합니다.

- 여러 조건을 포함하는 경우 다음 순서로 조건을 입력하여 조회 성능을 최적화합니다.
 - 같음(=)
 - 보다 작음(<), 보다 큼(>), 작거나 같음(<=), 크거나 같음(>=)
 - 같지 않음(!=)
- 통합 서비스는 null 값의 일치 여부를 확인합니다. 예를 들어 입력 조회 조건 열이 NULL인 경우 통합 서비스가 조회에서 NULL과 같은 NULL을 평가합니다.
- 정렬된 입력에 대한 플랫폼 파일 조회를 구성하는 경우 조건 열이 그룹화되지 않으면 통합 서비스에서 세션이 실패합니다. 열이 그룹화되었지만 정렬되지 않은 경우 통합 서비스는 정렬된 입력이 구성되지 않은 것처럼 조회를 처리합니다.

사용자가 동적 캐시 또는 캐시되지 않거나 정적 캐시에 대한 변환을 구성하는지 여부에 따라 통합 서비스에서 조회 일치 항목을 다르게 처리합니다.

캐시되지 않거나 정적 캐시

정적 조회 캐시 또는 캐시되지 않는 조회 소스가 있는 조회 변환을 구성할 때 다음 지침을 따르십시오.

- 조회 조건을 작성할 경우 다음 연산자를 사용합니다.
 - =, >, <, >=, <=, !=
 둘 이상의 조회 조건을 포함하는 경우 다음과 같은 순서로 조건을 배치하여 조회 성능을 최적화합니다.
 - 같음(=)
 - 보다 작음(<), 보다 큼(>), 작거나 같음(<=), 크거나 같음(>=)
 - 같지 않음(!=)
 예를 들어 다음 조회 조건을 작성하십시오.

```
ITEM_ID = IN_ITEM_ID
```

```
PRICE <= IN_PRICE
```

- 조회에서 값을 반환하려면 입력 값이 모든 조건을 충족해야 합니다.

조건은 동등한 값과 일치하거나 임계값 조건을 제공할 수 있습니다. 예를 들어 캘리포니아에 거주하지 않는 고객 또는 급여가 \$30,000 이하인 직원을 찾을 수 있습니다. 소스 및 조건의 특성에 따라 조회에서 여러 값이 반환될 수도 있습니다.

동적 캐시

동적 캐시를 사용하도록 조회 변환을 구성하는 경우 조회 조건에서 같음 연산자(=)만 사용할 수 있습니다.

여러 일치 항목 처리

조회 변환은 사용자가 변환에서 구성하는 조건에 따라 값을 찾습니다. 조회 조건이 고유한 키를 기반으로 하지 않거나 조회 소스가 비정규화된 경우 통합 서비스가 조회 소스나 조회 캐시에서 여러 일치 항목을 찾을 수 있습니다.

다음과 같은 방식으로 여러 일치 항목을 처리하게 조회 변환을 구성할 수 있습니다.

- **첫 번째 일치하는 값을 사용하거나 마지막 일치하는 값을 사용합니다.** 첫 번째 일치하는 값이나 마지막 일치하는 값을 반환하도록 변환을 구성할 수 있습니다. 첫 번째 값과 마지막 값은 조회 조건과 일치하는 조회 캐시에서 찾은 첫 번째 값과 마지막 값입니다. 조회 소스를 캐시할 경우 통합 서비스는 조회 캐시의 각 열에 대한 ORDER BY 절을 생성하여 캐시의 첫 번째 행과 마지막 행을 결정합니다. 그런 다음 오름차순으로 각 조회 소스 열을 정렬합니다.

통합 서비스는 0 ~ 10과 같이 숫자 오름차순으로 숫자 열을 정렬합니다. 1월 ~ 12월 그리고 첫 번째 월 ~ 마지막 월로 날짜/시간 열을 정렬합니다. 세션에 대해 구성된 정렬 순서에 따라 데이터 통합 서비스가 문자열 열을 정렬합니다.

- **일치하는 임의 값 사용.** 조회 조건과 일치하는 임의 값을 반환하도록 조회 변환을 구성할 수 있습니다. 일치하는 임의 값을 반환하도록 조회 변환을 구성하면 변환이 조회 조건과 일치하는 첫 번째 값을 반환합니다. 이 변환은 모든 조회 변환 포트 대신 키 포트를 기반으로 인덱스를 작성합니다. 일치하는 임의 값을 사용하면 행 인덱싱 프로세스가 보다 간단하므로 성능이 향상될 수 있습니다.
- **모든 값 사용.** 조회 변환에서 일치하는 모든 행을 반환합니다. 이 옵션을 사용하려면 조회 변환을 작성할 때 모든 일치 항목을 반환하도록 이 변환을 구성해야 합니다. 해당 변환은 활성 변환이 됩니다. 변환을 작성한 후에는 수동과 활성 간에 모드를 변경할 수 없습니다.
- **오류 반환.** 조회 변환이 정적 캐시를 사용하거나 캐시를 사용하지 않는 경우 통합 서비스는 해당 행을 오류로 표시합니다. 조회 변환은 기본적으로 세션 로그에 해당 행을 쓰고 오류 개수를 1씩 늘립니다. 조회 변환에 동적 캐시가 있는 경우, 일치 항목을 여러 개 찾으면 통합 서비스에서 세션이 실패합니다. 통합 서비스가 조회 테이블을 캐시하고 있거나 중복 키 값을 조회하고 있는 동안 세션이 실패합니다. 또한 업데이트 시 이전 값을 출력하도록 조회 변환을 구성하는 경우 조회 변환이 일치 항목을 여러 개 찾으면 오류를 반환합니다. 이 변환은 모든 조회 변환 포트 대신 키 포트를 기반으로 인덱스를 작성합니다.

관련 항목:

- [“여러 행 반환” 페이지 286](#)

조회 캐시

조회 파일이나 테이블을 캐시하도록 조회 변환을 구성할 수 있습니다. 통합 서비스는 캐시된 조회 변환에서 데이터의 첫 번째 행을 처리할 때 캐시를 메모리에 빌드합니다. 사용자가 변환 또는 세션 속성에서 구성하는 양에 따라 캐시에 대한 메모리를 할당합니다. 통합 서비스는 조건 값을 인덱스 캐시에 저장하고 출력 값을 데이터 캐시에 저장합니다. 통합 서비스는 조회 변환을 시작하는 각 행에 대해 캐시를 쿼리합니다.

또한 통합 서비스는 기본적으로 \$PMCacheDir에서 캐시 파일을 작성합니다. 데이터가 메모리 캐시에 맞지 않을 경우 통합 서비스는 오버플로우 값을 캐시 파일에 저장합니다. 세션이 완료되면, 지속형 캐시를 사용하도록 조회 변환을 구성하는 경우를 제외하고 통합 서비스가 캐시 메모리를 해제하고 캐시 파일을 삭제합니다.

조회 캐시를 구성할 때 다음 옵션을 구성할 수 있습니다.

- 지속형 캐시
- 조회 소스에서 다시 캐시
- 정적 캐시
- 동적 캐시
- 공유 캐시
- 사전 빌드 조회 캐시

참고: 관계형 또는 플랫폼 파일 조회에 대한 동적 캐시를 사용할 수 있습니다.

여러 행 반환

일치하는 모든 행을 반환하도록 조회 변환을 구성한 경우 조회 변환은 조회 조건과 일치하는 모든 행을 반환합니다. 변환을 작성할 때 일치하는 모든 행을 반환하도록 변환을 구성해야 합니다. 해당 조회 변환은 활성 변환이 됩니다. 일치 항목이 여러 개인 경우의 조회 정책 속성이 모든 값 사용입니다. 속성은 읽기 전용이 됩니다. 변환을 작성한 후에는 속성을 변경할 수 없습니다.

관계형 테이블에 고객 주문 데이터가 있다고 가정합니다. 테이블에는 각 고객의 주문 여러 개가 있습니다. 조회에서 고객의 모든 주문을 반환하도록 조회 변환을 구성할 수 있습니다. 조회 테이블을 캐싱하여 성능을 개선할 수 있습니다.

캐싱을 사용하도록 조회 변환을 구성한 경우 통합 서비스는 조회 소스에서 읽는 모든 행을 캐싱합니다. 통합 서비스는 키 인덱스별로 조회 키의 모든 행을 캐싱합니다.

여러 행 반환에 대한 규칙 및 지침

여러 행을 반환하도록 조회 변환을 구성할 경우 다음 규칙 및 지침을 따르십시오.

- 통합 서비스는 캐싱되는 조회를 위해 조회 소스에 있는 모든 행을 캐싱합니다.
- 여러 행을 반환하는 캐싱되는 조회 또는 캐싱되지 않는 조회에 대한 **SQL** 재정의의 구성할 수 있습니다. **SQL** 재정의는 조회 키를 기준으로 정렬된 데이터를 반환해야 합니다. 데이터가 키로 정렬되지 않은 경우 예기치 않은 결과가 나올 수 있습니다.
- 여러 행을 반환하는 조회 변환에 대해 동적 캐시를 활성화할 수 없습니다.
- 연결되지 않은 조회 변환으로부터 여러 행을 반환할 수 없습니다.
- 여러 일치 정책에서 캐시 조회가 서로 일치하는 조회 변환이 여러 개 있으면 명명된 캐시를 공유하도록 해당 조회 변환을 구성할 수 있습니다.
- 여러 행을 반환하는 조회 변환은 각 입력 행에 대해 일치하는 행을 1개 반환하는 조회 변환과 캐시를 공유할 수 없습니다.

관련 항목:

- [“조회 변환 작성” 페이지 289](#)

연결되지 않은 조회 변환 구성

연결되지 않은 조회 변환은 소스 또는 대상에 연결되지 않은 조회 변환입니다. **LKP** 식을 사용하여 다른 변환에서 조회를 호출합니다.

식에서 조회를 호출할 때 다음과 같은 태스크를 수행할 수 있습니다.

- 식에서 조회 결과를 테스트합니다.
- 조회 결과에 따라 행을 필터링합니다.
- 조회 결과에 따라 업데이트를 위한 행을 표시하고 느린 변경 차원 테이블을 업데이트합니다.
- 하나의 매핑에서 여러 번 동일한 조회를 호출합니다.

연결되지 않은 조회 변환을 구성하려면 입력 포트를 추가하고, 조회 조건을 구성하고, 반환 값을 지정하고, 다른 변환의 조회 식을 구성합니다.

1단계. 입력 포트 추가

:LKP 식의 각 인수에 사용될 입력 포트를 조회 변환에 작성합니다. 작성하려는 각 조회 조건을 위해 입력 포트를 조회 변환에 추가해야 합니다. 각 조건별로 서로 다른 포트를 작성하거나 둘 이상의 조건에서 동일한 입력 포트를 사용할 수 있습니다.

예를 들어 소매 매장이 지난달에 모든 매장에서 가격을 인상한 것으로 가정합니다. 회계 부서는 가격이 인상된 품목에 대해서만 행을 대상으로 로드하려고 합니다. 이렇게 하려면 다음 태스크를 완료하십시오.

- 소스의 **ITEM_ID**를 대상의 **ITEM_ID**와 비교하는 조회 조건을 작성합니다.
- 소스에 있는 각 품목의 **PRICE**를 대상 테이블에 있는 가격과 비교합니다.
 - 대상 테이블에 해당 품목이 있고 소스에 있는 품목 가격이 대상 테이블에 있는 가격보다 낮거나 같으면 해당 행을 삭제하려고 합니다.
 - 소스에 있는 가격이 대상 테이블에 있는 품목 가격보다 높은 경우에는 해당 행을 업데이트하려고 합니다.
- 데이터 유형이 10진수(37,0)인 입력 포트(**IN_ITEM_ID**)를 작성하여 **ITEM_ID**와 일치시키고, 데이터 유형이 10진수(10,2)인 **IN_PRICE** 입력 포트를 작성하여 **PRICE** 조회 포트와 일치시킵니다.

2단계. 조회 조건 추가

포트를 구성한 후 변환 입력 값과 조회 소스 또는 캐시의 값을 비교하는 조회 조건을 정의하십시오. 성능을 향상 시키려면 등호가 있는 조건을 먼저 추가하십시오.

이 경우 다음 조회 조건을 추가합니다.

```
ITEM_ID = IN_ITEM_ID
```

```
PRICE <= IN_PRICE
```

항목이 매핑 소스 및 조회 소스에 존재하고 매핑 소스 가격이 조회 가격보다 적거나 같은 경우 조건이 참이고 조회가 반환 포트에서 지정된 값을 반환합니다. 조회 조건이 **false**일 경우 조회에서 **NULL**을 반환합니다. 업데이트 전략 식을 작성하는 경우 IIF 함수에 중첩된 **ISNULL**을 사용하여 **null** 값을 테스트합니다.

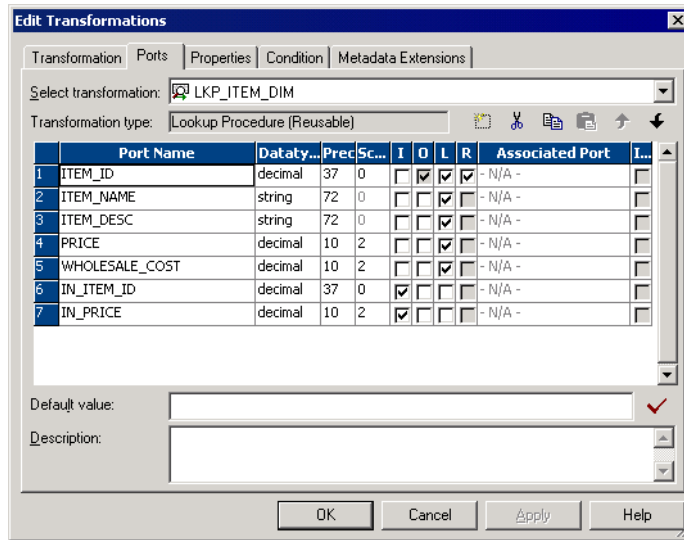
3단계. 반환 값 지정

여러 입력 값을 조회 변환으로 전달하고 데이터 열 1개를 반환할 수 있습니다. 연결되지 않은 조회 변환에서 여러 개의 일치 항목에 대한 모든 값을 사용하도록 조회 정책을 구성하지 마십시오. 조회/출력 포트 1개를 반환 포트로 지정하십시오. 조회 조건과 일치하는 첫 번째 값, 마지막 값 또는 임의의 값을 사용하도록 선택할 수 있습니다.

업데이트 전략 또는 필터 식에서 연결되지 않은 조회를 호출하는 경우, 일반적으로 **null** 값을 확인하게 됩니다. 이 경우 반환 포트는 무엇이든 될 수 있습니다. 계산을 수행하는 식에서 조회를 호출하는 경우, 반환 값은 계산에 포함하려는 값이어야 합니다.

업데이트 전략 예제를 계속하려면 **ITEM_ID** 포트를 반환 포트에 정의하면 됩니다. 업데이트 전략 식은 반환되는 **null** 값을 확인합니다. 조회 조건이 **true**이면 통합 서비스가 **ITEM_ID**를 반환합니다. 조건이 **false**이면 통합 서비스는 **NULL**을 반환합니다.

다음 그림은 조회 변환의 반환 포트를 보여 줍니다.



4단계. 식을 통해 조회 호출

연결되지 않은 조회 변환의 입력 값을 다른 변환의 :LKP 식에서 제공하십시오. 인수는 조회 조건에 사용되는 조회 변환 입력 포트와 일치하는 로컬 입력 포트입니다. :LKP 식에 다음 구문을 사용하십시오.

:LKP.lookup_transformation_name(argument, argument, ...)

소매 매장에 대한 예제를 계속하려면 업데이트 전략 식을 작성할 때 식에서의 포트 순서가 조회 조건에서의 순서와 일치해야 합니다. 이 경우에는 ITEM_ID 조건이 첫 번째 조회 조건이므로 이 조건이 업데이트 전략 식에서 첫 번째 인수입니다.

IIF(ISNULL(:LKP.lkpITEMS_DIM(ITEM_ID, PRICE)), DD_UPDATE, DD_REJECT)

다음 지침에 따라 연결되지 않은 조회 변환을 호출하는 식을 작성하십시오.

- 각 인수를 나열하는 순서는 조회 변환의 조회 조건 순서와 일치해야 합니다.
- 식에 있는 포트의 데이터 유형은 조회 변환에 있는 입력 포트의 데이터 유형과 일치해야 합니다. 데이터 유형이 일치하지 않는 경우, 디자이너는 식의 유효성을 검사하지 않습니다.
- 조회 조건에 있는 포트가 조회/출력 포트가 아니면 디자이너는 식의 유효성을 검사하지 않습니다.
- 식에 있는 인수 포트는 조회 조건에 있는 입력 포트와 동일한 순서로 되어 있어야 합니다.
- 잘못된 :LKP 구문을 사용하는 경우, 디자이너는 해당 매핑을 올바르게 않은 것으로 표시합니다.
- 연결된 조회 변환을 :LKP 식에서 호출하면 디자이너가 해당 매핑을 올바르게 않은 것으로 표시합니다.

팁: 식을 입력할 때 가리키고 클릭하는 방법을 사용해 함수 및 포트를 선택하여 구문 오류를 방지하십시오.

데이터베이스 교착 상태 복원력

조회 변환은 캐시되지 않은 조회에 대한 데이터베이스 교착 상태에 대해 복원력을 갖습니다. 데이터베이스 교착 상태 오류가 발생하는 경우 세션이 실패하지 않습니다. 통합 서비스는 지정된 다시 시도 기간 동안 마지막 문을 다시 실행하려고 시도합니다.

통합 서비스에 대한 교착 상태 중지 간격 및 교착 상태 다시 시도 횟수를 구성할 수 있습니다. 이러한 값은 관계형 기록기에 대한 데이터베이스 교착 상태에도 영향을 미칩니다. 사용자 지정 속성으로 세션 수준에서 해당 값을 재정의할 수 있습니다.

다음 통합 서비스 속성을 구성합니다.

- **NumOfDeadlockRetries.** 데이터베이스 교착 상태 시 PowerCenter 통합 서비스가 대상 쓰기를 다시 시도하는 횟수입니다. 최소값은 0입니다. 기본값은 10입니다. 교착 상태에서 세션이 실패하도록 하려면 NumOfDeadlockRetries를 0으로 설정합니다.
- **DeadlockSleep.** 데이터베이스 교착 상태 시 PowerCenter 통합 서비스가 대상 쓰기를 다시 시도하기까지의 시간(초)입니다.

교착 상태가 일어나면 통합 서비스가 해당 문을 실행하려고 시도합니다. 통합 서비스는 각각의 다시 시도 사이 지연 기간 동안 기다립니다. 모든 시도가 교착 상태로 인해 실패하는 경우 세션이 실패합니다. 통합 서비스는 문을 다시 시도할 때마다 세션 로그에 메시지를 기록합니다.

조회 변환 작성

변환 개발자에서 재사용 가능 조회 변환을 작성합니다. 매핑 디자이너에서 재사용 불가능 조회 변환을 작성합니다.

조회 변환을 작성하려면:

1. 재사용 가능 조회 변환을 작성하려면 변환 개발자를 엽니다.
재사용 불가능 조회 변환을 작성하려면 매핑 디자이너에서 매핑을 엽니다. 파이프라인 조회 변환을 작성할 경우 조회 소스로 사용할 소스 정의로 끝니다.
2. 변환 > 작성을 클릭합니다. 조회 변환을 선택합니다.
3. 변환 이름을 입력합니다. 작성을 클릭합니다.
조회 변환의 이름 지정 규칙은 `LKP_TransformationName`입니다.
4. 변환이 활성 또는 수동인지 여부를 선택합니다. 확인을 클릭합니다. 이 옵션은 변경할 수 없습니다.
5. 조회 테이블 선택 대화 상자에서 다음 옵션 중 하나를 선택하여 조회 정의를 가져옵니다.
 - 리포지토리의 소스 정의.
 - 리포지토리의 대상 정의.
 - 매핑의 소스 한정자.
 - 리포지토리에서 관계형 테이블 또는 파일 가져오기.

참고: 정의를 가져오는 대신 수동으로 조회 포트를 추가할 수 있습니다. 출력 포트이기도 한 조회 포트를 선택할 수 있습니다.

조회 소스를 선택하면 디자이너는 사용자가 선택하는 개체의 포트를 기반으로 변환에서 포트를 작성합니다. 디자이너는 각 포트를 조회 포트와 출력 포트 구성합니다. 조회 포트는 조회 소스의 열을 나타냅니다. 조회 변환은 각 조회 포트의 조회 소스에서 데이터를 받고 이 데이터를 대상에 전달합니다.

6. 일치하는 모든 행을 조회 변환이 반환하게 하려면 여러 개의 일치 항목에 대해 모든 행 반환을 활성화합니다. 변환을 작성한 후에는 이 옵션을 변경할 수 없습니다. 해당 조회 변환은 활성 변환이 됩니다.
7. 확인을 클릭하거나, 정의를 가져오는 대신 수동으로 조회 포트를 추가하려면 건너뛰기를 클릭합니다. 출력 포트이기도 한 조회 포트를 선택할 수 있습니다.
8. 연결된 조회 변환의 경우 입력 및 출력 포트를 추가합니다.

변환을 통해 데이터를 전달하고 조회 테이블에서 대상으로 데이터를 반환할 수 있습니다.

9. 연결되지 않은 조회 변환의 경우 조회에서 반환하려는 값에 대한 반환 포트를 작성합니다.
조회를 호출한 변환에 하나의 열을 반환할 수 있습니다.
10. 속성 탭을 클릭하여 조회 변환 속성을 구성합니다. 조회 캐싱을 구성합니다.
조회 캐싱은 파이프라인 및 플랫폼 파일 조회 변환에 대해 기본적으로 활성화되어 있습니다.
11. 동적 조회 캐시가 있는 조회 변환의 경우 입력 포트, 출력 포트 또는 시퀀스 ID를 각 조회 포트와 연결합니다.
통합 서비스는 각각의 연결된 식의 데이터를 사용하여 조회 캐시에서 행을 삽입하거나 업데이트합니다. 시퀀스 ID를 연결하는 경우 통합 서비스가 조회 캐시의 삽입된 행에 대한 기본 키를 생성합니다.
12. 조건 탭에서 조회 조건을 추가합니다.
조회 조건은 소스 열 값을 조회 소스의 값과 비교합니다. 조건 탭의 변환 포트는 소스 열 값을 나타냅니다.
조회 테이블은 조회 소스를 나타냅니다.

관련 항목:

- [“재사용 가능 파이프라인 조회 변환 작성” 페이지 290](#)
- [“재사용 불가능 파이프라인 조회 변환 작성” 페이지 290](#)

재사용 가능 파이프라인 조회 변환 작성

변환 개발자에서 재사용 가능 파이프라인 조회 변환을 작성합니다. 변환을 작성할 때 조회 테이블 위치에 대한 소스를 선택합니다. 변환 개발자에 리포지토리의 소스 정의 목록이 표시됩니다.

관계형 테이블 또는 플랫폼 파일 소스 정의를 나타내는 소스 한정자를 선택하면 디자이너가 관계형 또는 플랫폼 파일 조회 변환을 작성합니다. 소스 한정자가 응용 프로그램 소스를 나타내는 경우 디자이너가 파이프라인 조회 변환을 작성합니다. 관계형 또는 플랫폼 파일 조회 소스에 대한 파이프라인 조회 변환을 작성하려면 해당 변환을 작성한 후 소스 유형을 소스 한정자로 변경합니다. 조회 테이블 속성에서 소스 정의 이름을 입력합니다.

재사용 가능 파이프라인 조회 변환을 매핑으로 끝면 매핑 디자이너가 조회 테이블 속성의 소스 정의를 매핑에 추가합니다. 해당 디자이너가 소스 한정자 변환을 추가합니다.

조회 테이블 이름을 매핑의 다른 소스 한정자로 변경하려면 조회 테이블 이름 속성에서 열기 단추를 클릭합니다. 목록에서 소스 한정자를 선택합니다.

재사용 불가능 파이프라인 조회 변환 작성

매핑 디자이너에서 재사용 불가능 파이프라인 조회 변환을 작성합니다. 소스 정의를 매핑으로 끝니다. 매핑 디자이너에서 변환을 작성할 때 조회 테이블 위치로 소스 한정자를 선택합니다. 매핑 디자이너에 매핑의 소스 한정자 목록이 표시됩니다. 소스 한정자를 선택하면 매핑 디자이너는 사용자가 선택하는 소스 한정자의 특성 및 포트 이름으로 조회 변환을 채웁니다.

조회 변환에 대한 팁

조회 조건에서 사용되는 열에 인덱스를 추가합니다.

조회 테이블을 포함하는 데이터베이스를 수정할 권한이 있으면 캐싱되는 조회 및 캐싱되지 않는 조회의 성능을 향상시킬 수 있습니다. 조회 테이블이 매우 큰 경우에는 이렇게 하는 것이 중요합니다. 통합 서비스는 이러한 열

의 값을 쿼리하고 정렬하고 비교해야 하기 때문에 조회 조건에서 사용되는 모든 열이 인덱스에 포함되어야 합니다.

같은 연산자(=)를 맨 앞에 놓고 조건을 배치합니다.

둘 이상의 조회 조건을 포함하는 경우 다음과 같은 순서로 조건을 배치하여 조회 성능을 최적화합니다.

- 같은(=)
- 보다 작음(<), 보다 큼(>), 작거나 같음(<=), 크거나 같음(>=)
- 같지 않음(!=)

작은 조회 테이블을 캐싱합니다.

작은 조회 테이블을 캐싱하여 세션 성능을 향상시킵니다. 조회 테이블을 캐싱하는지 여부와 관계없이 조회 쿼리 및 처리 결과는 동일합니다.

데이터베이스의 테이블을 조인합니다.

조회 테이블이 매핑의 소스 테이블과 동일한 데이터베이스에 있으며 캐싱을 사용하기 어려운 경우에는 조회 변환을 사용하는 대신 소스 데이터베이스에서 테이블을 조인합니다.

지속형 조회 캐시를 정적 조회에 사용합니다.

조회 소스가 세션 간에 변경되지 않으면 지속형 조회 캐시를 사용하도록 조회 변환을 구성합니다. 그러면 통합 서비스가 세션 간에 캐시 파일을 저장하고 다시 사용하므로 조회 소스를 읽는 데 시간이 소요되지 않습니다.

:LKP 참조 한정자를 사용하여 연결되지 않은 조회 변환을 호출합니다.

:LKP 참조 한정자를 사용하여 식을 작성할 경우 연결되지 않은 조회 변환만 호출하십시오. 연결된 조회 변환을 호출하려고 하면 디자이너에서 오류를 표시하고 해당 매핑을 잘못된 것으로 표시합니다.

관계형 또는 플랫폼 파일 조회 소스를 처리할 때 성능이 향상되도록 파이프라인 조회 변환을 구성합니다.

조회 소스를 소스 한정자로 정의할 경우 관계형 또는 플랫폼 파일 조회 소스를 처리하기 위해 파티션을 작성할 수 있습니다. 재사용 불가능 파이프라인 조회 변환을 구성하고, 조회 소스를 처리하는 부분적 파이프라인에 파티션을 작성합니다.

제 18 장

조회 캐시

이 장에 포함된 항목:

- [조회 캐시 개요, 292](#)
- [연결된 조회 캐시 빌드, 294](#)
- [지속형 조회 캐시 사용, 295](#)
- [캐시되지 않은 조회 또는 정적 캐시 작업, 297](#)
- [조회 캐시 공유, 297](#)
- [조회 캐시에 대한 팁, 303](#)

조회 캐시 개요

조회 소스를 캐시하도록 조회 변환을 구성하여 조회 성능을 향상시킬 수 있습니다. 조회 테이블 또는 파일이 클 경우 조회 캐싱을 활성화합니다.

통합 서비스는 캐시된 조회 변환에서 데이터의 첫 번째 행을 처리할 때 메모리에 캐시를 작성합니다. 사용자가 변환 또는 세션 속성에서 구성하는 양에 따라 캐시에 대한 메모리를 할당합니다. 통합 서비스는 조건 값을 인덱스 캐시에 저장하고 출력 값을 데이터 캐시에 저장합니다. 통합 서비스는 조회 변환을 시작하는 각 행에 대해 캐시를 쿼리합니다.

데이터가 메모리 캐시에 맞지 않을 경우 통합 서비스는 오버플로우 값을 캐시 파일에 저장합니다. 또한 통합 서비스는 지정된 캐시 디렉터리에 캐시 파일을 작성합니다. 세션이 완료되면 지속형 캐시를 사용하도록 조회 변환을 구성하지 않는 한 통합 서비스는 캐시 메모리를 해제하고 캐시 파일을 삭제합니다.

플랫 파일 조회 또는 파이프라인 조회를 사용하는 경우 통합 서비스는 항상 조회 소스를 캐시합니다. 정렬된 입력을 사용하도록 플랫 파일 조회를 구성한 경우 조건 열이 그룹화되지 않으면 통합 서비스가 조회를 캐시하지 못합니다. 열이 그룹화되었지만 정렬되지 않은 경우 통합 서비스는 정렬된 입력이 구성되지 않은 것처럼 조회를 처리합니다.

조회 캐시를 구성할 때 다음과 같은 캐시 설정을 구성할 수 있습니다.

순차적 캐시와 동시 캐시

캐시를 순차적으로 또는 동시에 작성하도록 세션을 구성할 수 있습니다. 순차적 캐시를 작성하는 경우 통합 서비스는 소스 행이 조회 변환에 들어올 때 캐시를 작성합니다. 동시 캐시를 작성하도록 세션을 구성하는 경우 통합 서비스는 캐시를 작성하기 전에 첫 번째 행이 조회 변환에 들어올 때까지 기다리지 않습니다. 대신 여러 캐시를 동시에 작성합니다.

지속형 캐시

조회 캐시 파일을 저장하여 다음 번에 통합 서비스가 캐시를 사용하도록 구성된 조회 변환을 처리할 때 파일을 재사용할 수 있습니다.

소스에서 다시 캐시

지속형 캐시가 조회 소스와 동기화되지 않는 경우 조회 캐시를 다시 작성하도록 조회 변환을 구성할 수 있습니다.

정적 캐시

조회 소스에 대해 정적 캐시를 구성할 수 있습니다. 기본적으로 통합 서비스는 정적 캐시를 작성합니다. 통합 서비스는 조회 파일 또는 테이블을 캐시하고 변환에 들어오는 각 행에 대해 캐시에서 값을 조회합니다. 조회 조건이 **True**일 경우 통합 서비스가 조회 캐시의 값을 반환합니다. 통합 서비스는 조회 변환을 처리하는 동안 캐시를 업데이트하지 않습니다.

동적 캐시

조회 소스를 캐시하고 캐시를 업데이트하려면 동적 캐시를 사용하여 조회 변환을 구성합니다. 통합 서비스는 동적으로 데이터를 조회 캐시에서 삽입하거나 업데이트하며 데이터를 대상에 전달합니다. 동적 캐시는 대상과 동기화됩니다.

공유 캐시

여러 변환 사이에서 조회 캐시를 공유할 수 있습니다. 동일한 매핑의 변환 사이에서 명명되지 않은 캐시를 공유할 수 있습니다. 동일하거나 다른 매핑의 변환 사이에서 명명된 캐시를 공유할 수 있습니다. 조회 변환은 캐시 공유 규칙이 일치하는 경우 동일한 대상 로드 순서 그룹 내에서 명명되지 않은 정적 캐시를 공유할 수 있습니다. 조회 변환은 동일한 대상 로드 순서 그룹 내에서 동적 캐시를 공유할 수는 없습니다.

조회 변환에 캐싱을 구성하지 않은 경우 통합 서비스가 각 입력 행에 대해 조회 소스를 쿼리합니다. 조회 소스의 캐시 여부와 관계없이 조회 쿼리 및 처리 결과는 동일합니다. 그러나 조회 캐싱을 활성화하면 크기가 큰 조회 소스의 조회 성능이 개선될 수 있습니다.

캐시 비교

통합 서비스는 구성된 조회 캐시의 유형에 따라 다른 작업을 수행합니다.

다음 표에는 조회 변환의 캐시되지 않은 조회, 정적 캐시 및 동적 캐시가 비교되어 있습니다.

캐싱되지 않음	정적 캐시	동적 캐시
통합 서비스가 캐시를 삽입하거나 업데이트하지 않습니다.	통합 서비스가 캐시를 삽입하거나 업데이트하지 않습니다.	통합 서비스가 행을 대상에 전달할 때 캐시에 행을 삽입하거나 업데이트할 수 있습니다.
관계형 조회를 사용할 수 있습니다.	관계형, 플랫폼 파일 또는 파이프라인 조회를 사용할 수 있습니다.	관계형, 플랫폼 파일 또는 소스 한정자 조회를 사용할 수 있습니다.
조건이 true인 경우 통합 서비스가 조회 테이블 또는 캐시의 값을 반환합니다. 조건이 true가 아닌 경우 통합 서비스가 연결된 변환에 대해 기본값을 반환하고 연결되지 않은 변환에 대해 NULL을 반환합니다.	조건이 true인 경우 통합 서비스가 조회 테이블 또는 캐시의 값을 반환합니다. 조건이 true가 아닌 경우 통합 서비스가 연결된 변환에 대해 기본값을 반환하고 연결되지 않은 변환에 대해 NULL을 반환합니다.	조건이 true인 경우 통합 서비스가 행 유형에 따라 캐시의 행을 업데이트하거나 캐시를 변경되지 않은 상태로 유지합니다. 조건이 true인 경우 행이 캐시 및 대상 테이블에 있다는 것을 나타냅니다. 업데이트된 행을 대상에 전달할 수 있습니다. 조건이 true가 아닌 경우 통합 서비스가 행 유형에 따라 행을 캐시에 삽입하거나 캐시를 변경되지 않은 상태로 유지합니다. 조건이 true가 아닌 경우 행이 캐시 또는 대상에 없다는 것을 나타냅니다. 삽입된 행을 대상 테이블에 전달할 수 있습니다.

관련 항목:

- [“캐시되지 않은 조회 또는 정적 캐시 작업” 페이지 297](#)
- [“동적 조회 캐시 업데이트” 페이지 311](#)

연결된 조회 캐시 빌드

통합 서비스는 다음과 같은 방법으로 연결된 조회 변환에 대한 조회 캐시를 빌드할 수 있습니다.

- **순차 캐시.** 통합 서비스가 순차적으로 조회 캐시를 빌드합니다. 통합 서비스는 캐시된 조회 변환에서 첫 번째 데이터 행을 처리할 때 메모리에 캐시를 빌드합니다.
- **동시 캐시.** 통합 서비스가 동시에 조회 캐시를 빌드합니다. 데이터가 조회 변환에 도달하도록 기다릴 필요가 없습니다.

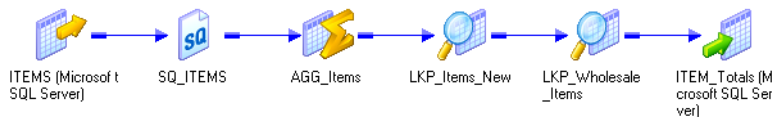
참고: 통합 서비스는 사용자가 캐시 빌드를 구성하는 방법에 관계없이 연결되지 않은 조회 변환에 대한 캐시를 순차적으로 빌드합니다. 연결되지 않은 조회 변환에 대한 동시 캐시를 빌드하도록 세션을 구성하는 경우 통합 서비스가 이 설정을 무시하고 연결되지 않은 조회 변환 캐시를 순차적으로 빌드합니다.

순차 캐시

기본적으로 통합 서비스는 캐싱된 조회 변환의 첫 번째 데이터 행을 처리할 때 메모리에 캐시를 작성합니다. 통합 서비스는 각 조회 캐시를 파이프라인에 순차적으로 작성합니다. 통합 서비스는 업스트림 활성 변환이 처리를 완료할 때까지 기다린 후, 조회 변환에 있는 행을 처리하기 시작합니다. 통합 서비스는 업스트림 조회 변환에서 캐시 작성을 완료할 때까지는 다운스트림 조회 변환에 대한 캐시를 작성하지 않습니다.

예를 들어 다음 매핑에는 비정렬 집계 변환과 2개의 조회 변환이 차례로 포함되어 있습니다.

수치 2. 순차적으로 조회 캐시 작성



통합 서비스는 비정렬 집계 변환의 모든 행을 처리하며, 비정렬 집계 변환이 완료된 후에 첫 번째 조회 변환을 처리하기 시작합니다. 통합 서비스는 첫 번째 입력 행을 처리할 때 첫 번째 조회 캐시를 작성하기 시작합니다. 통합 서비스는 첫 번째 조회 캐시 작성을 마친 후에 조회 데이터 처리를 시작할 수 있습니다. 첫 번째 데이터 행이 조회 변환에 도달하면 통합 서비스에서 다음 조회 캐시를 작성하기 시작합니다.

조회 변환에서 행 데이터를 처리할 수 없는 경우에는 조회 캐시를 순차적으로 처리하려 할 수 있습니다. 조건에 따라 서로 다른 파이프라인으로 데이터를 라우팅하도록 변환 논리가 구성된 경우, 조회 변환은 행 데이터를 처리하지 못할 수 있습니다. 순차적 캐싱을 구성하면 조회 캐시를 불필요하게 작성하는 것을 방지할 수 있습니다. 예를 들어 라우터 변환은 조건이 **true**로 확인되면 특정 파이프라인으로 데이터를 라우팅하고, 조건이 **false**로 확인되면 다른 파이프라인으로 데이터를 라우팅할 수 있습니다. 이 경우 조회 변환은 데이터를 전혀 받지 못할 수 있습니다.

동시 캐시

동시에 조회 캐시를 작성하도록 통합 서비스를 구성할 수 있습니다. 동시 캐시를 사용하여 세션 성능을 개선할 수 있습니다. 파이프라인에 조회 변환의 활성 변환 업스트림이 포함된 경우 특히 성능이 개선될 수 있습니다. 세션에서 각 조회 변환에 대한 캐시를 빌드해야 하는 것이 확실한 경우 동시 캐시를 작성하도록 세션을 구성하려고 할 수 있습니다.

동시 캐시를 작성하도록 조회 변환을 구성하는 경우 조회 캐시를 작성하기 전에 업스트림 변환이 완료되도록 기다리지 않으며, 다른 조회 캐시 빌드를 시작할 수 있기 전까지 조회 캐시 빌드를 마쳐야 할 필요가 없습니다.

다음 그림에는 동시에 빌드된 조회 변환 캐시가 나와 있습니다.



세션을 실행할 경우 통합 서비스가 동시에 조회 캐시를 빌드합니다. 업스트림 변환이 완료되도록 기다리지 않으며, 다른 조회 변환이 캐시 빌드를 완료하도록 기다리지 않습니다.

참고: 연결되지 않은 조회 변환에 대한 캐시를 동시에 처리할 수 없습니다.

동시 캐시를 작성하도록 세션을 구성하려면 조회 캐시 작성을 위한 추가 동시 파이프라인이라는 세션 구성 특성에 대한 값을 구성합니다.

지속형 조회 캐시 사용

비지속형 또는 지속형 캐시를 사용하도록 조회 변환을 구성할 수 있습니다. 통합 서비스는 성공적인 세션 이후에 지속형 조회 캐시 속성에 기반하여 조회 캐시 파일을 저장하거나 삭제합니다.

조회 테이블이 세션 간에 변경되지 않으면 지속형 조회 캐시를 사용하도록 조회 변환을 구성할 수 있습니다. 통합 서비스가 세션 간에 캐시 파일을 저장하고 다시 사용하므로 조회 테이블을 읽는 데 시간이 소요되지 않습니다.

비지속형 캐시 사용

기본적으로 통합 서비스는 조회 변환에서 캐시가 활성화된 경우 비지속형 캐시를 사용합니다. 세션이 끝날 때 통합 서비스가 캐시 파일을 삭제합니다. 다음에 세션을 실행할 때 통합 서비스는 데이터베이스에서 메모리 캐시를 작성합니다.

지속형 캐시 사용

캐시 파일을 저장하고 재사용하려는 경우, 지속형 캐시를 사용하도록 변환을 구성할 수 있습니다. 세션 실행 간에 조회 테이블이 바뀌지 않는 경우 지속형 캐시를 사용합니다.

통합 서비스는 지속형 조회 캐시를 사용하여 세션을 처음 실행할 때 캐시 파일을 삭제하는 대신 디스크에 저장합니다. 통합 서비스는 다음에 세션을 실행할 때 해당 캐시 파일로부터 메모리 캐시를 작성합니다. 조회 테이블이 가끔씩 변경되는 경우, 데이터베이스로부터 조회를 다시 캐싱하도록 세션 속성을 재정의할 수 있습니다.

지속형 조회 캐시를 사용할 경우 캐시 파일의 이름을 지정할 수 있습니다. 명명된 캐시를 지정하면 조회 캐시를 세션 간에 공유할 수 있습니다.

조회 캐시 다시 빌드

통합 서비스가 지속형 캐시를 마지막으로 작성한 후에 조회 소스가 변경된 것으로 생각될 경우, 통합 서비스를 통해 조회 캐시를 다시 작성할 수 있습니다.

캐시를 다시 작성하는 경우, 통합 서비스는 새 캐시 파일을 작성하여 기존 지속형 캐시 파일을 덮어씁니다. 통합 서비스는 캐시를 다시 작성할 때 세션 로그에 메시지를 기록합니다.

매핑이 조회 변환 1개를 포함하거나 매핑에서 캐시를 공유하는 여러 대상 로드 순서 그룹에 여러 조회 변환이 포함되어 있으면 캐시를 다시 작성할 수 있습니다. 동일한 매핑에서 동적 조회가 정적 조회와 캐시를 공유하는 경우에는 캐시를 다시 작성하지 않아도 됩니다.

캐시를 다시 사용할 수 없는 경우, 통합 서비스는 매핑 및 세션 속성에 따라 데이터베이스로부터 조회를 다시 캐싱하거나 세션이 실패한 것으로 처리합니다.

다음 테이블에는 명명된 캐시 및 명명되지 않은 캐시를 대상으로 지속형 캐싱을 어떻게 처리하는지 요약되어 있습니다.

세션 간의 매핑 또는 세션 변경	명명된 캐시	명명되지 않은 캐시
통합 서비스가 캐시 파일을 찾을 수 없습니다. 예를 들어 해당 파일이 더 이상 존재하지 않거나 통합 서비스가 그리드에서 실행되고 캐시 파일을 일부 노드에서만 사용할 수 있습니다.	캐시를 다시 작성합니다.	캐시를 다시 작성합니다.
세션 속성에서 많은 전체 자릿수 설정 옵션을 활성화하거나 비활성화합니다.	세션이 실패한 것으로 처리합니다.	캐시를 다시 작성합니다.
매핑 디자이너, Mapplet Designer 또는 재사용 가능 변환 개발자에서 변환을 편집합니다. ¹	세션이 실패한 것으로 처리합니다.	캐시를 다시 작성합니다.
매핑(조회 변환 제외)을 편집합니다.	캐시를 다시 사용합니다.	캐시를 다시 작성합니다.
조회 변환을 포함하는 파이프라인의 파티션 수를 변경합니다.	세션이 실패한 것으로 처리합니다.	캐시를 다시 작성합니다.
조회 테이블을 액세스하는 데 사용되는 데이터베이스 연결 또는 파일 위치를 변경합니다.	세션이 실패한 것으로 처리합니다.	캐시를 다시 작성합니다.
통합 서비스 데이터 이동 모드를 변경합니다.	세션이 실패한 것으로 처리합니다.	캐시를 다시 작성합니다.
유니코드 모드에서의 정렬 순서를 변경합니다.	세션이 실패한 것으로 처리합니다.	캐시를 다시 작성합니다.
통합 서비스 코드 페이지를 호환되는 코드 페이지로 변경합니다.	캐시를 다시 사용합니다.	캐시를 다시 사용합니다.
통합 서비스 코드 페이지를 호환할 수 없는 코드 페이지로 변경합니다.	세션이 실패한 것으로 처리합니다.	캐시를 다시 작성합니다.

¹ 변환 설명 또는 포트 설명 등의 속성을 편집해도 지속형 캐시 처리는 영향을 받지 않습니다.

캐시되지 않은 조회 또는 정적 캐시 작업

캐싱에 대해 조회 변환을 구성할 때 통합 서비스는 기본적으로 정적 조회 캐시를 생성합니다. 통합 서비스가 첫 번째 조회 요청을 처리할 때 캐시를 작성합니다. 변환에 전달되는 각 행의 조회 조건을 바탕으로 캐시를 쿼리합니다. 통합 서비스는 변환을 처리하는 동안 캐시를 업데이트하지 않습니다. 통합 서비스는 캐시를 작성하고 쿼리하는 대신 조회 소스를 쿼리하는 경우를 제외하고 캐시된 조회를 처리하는 것과 동일한 방식으로 캐시되지 않은 조회를 처리합니다.

조회 조건이 **true**이면 통합 서비스가 조회 소스 또는 캐시에서 값을 반환합니다. 연결된 조회 변환인 경우 통합 서비스는 조회/출력 포트가 나타내는 값을 반환합니다. 연결되지 않은 조회 변환인 경우 통합 서비스는 반환 포트가 나타내는 값을 반환합니다.

조건이 **true**가 아닌 경우 통합 서비스가 **NULL** 또는 기본값을 반환합니다. 연결된 조회 변환인 경우 조건이 충족되지 않으면 통합 서비스는 출력 포트의 기본값을 반환합니다. 연결되지 않은 조회 변환인 경우 조건이 충족되지 않으면 통합 서비스는 **NULL**을 반환합니다.

정적 캐시를 사용하는 파이프라인에서 여러 개의 파티션을 작성하는 경우 통합 서비스는 각 파티션과 각 변환에 대해 개별 메모리 캐시와 개별 디스크 캐시를 작성합니다.

조회 캐시 공유

매핑에 있는 여러 조회 변환이 단일 조회 캐시를 공유하도록 구성할 수 있습니다. 통합 서비스가 첫 번째 조회 변환을 처리할 때 캐시를 작성합니다. 통합 서비스는 캐시를 공유하는 이후의 조회 변환을 위해 조회를 수행하는데 동일한 캐시를 사용합니다.

명명되지 않은 캐시 및 명명된 캐시를 공유할 수 있습니다.

- **명명되지 않은 캐시.** 매핑에 있는 조회 변환이 호환되는 캐싱 구조를 갖고 있으면 통합 서비스는 기본적으로 캐시를 공유합니다. 정적인 명명되지 않은 캐시만 공유할 수 있습니다.
- **명명된 캐시.** 여러 매핑 간에 캐시 파일을 공유하거나 동적 및 정적 캐시를 공유하려는 경우 명명된 지속형 캐시를 사용합니다. 캐싱 구조는 명명된 캐시와 일치하거나 호환되어야 합니다. 명명된 캐시는 정적인지 또는 동적인지에 상관없이 공유할 수 있습니다.

통합 서비스는 조회 캐시를 공유할 경우 세션 로그에 메시지를 기록합니다.

명명되지 않은 조회 캐시 공유

기본적으로 통합 서비스는 호환되는 캐시 구조가 있는, 매핑의 조회 변환에 대한 캐시를 공유합니다. 예를 들어 동일한 재사용 가능 조회 변환의 인스턴스 2개가 매핑 1개에 있고 두 인스턴스에 대해 동일한 출력 포트를 사용할 경우, 조회 변환은 기본적으로 조회 캐시를 공유합니다.

두 조회 변환이 명명되지 않은 캐시를 공유하는 경우, 통합 서비스는 특정 조회 변환을 위해 해당 캐시를 저장하고 조회 캐시 구조가 동일한 이후의 조회 변환에 대해 해당 캐시를 사용합니다.

변환 속성 또는 캐시 구조가 공유를 허용하지 않으면 통합 서비스가 새 캐시를 작성합니다.

명명되지 않은 조회 캐시를 공유하기 위한 지침

다음 지침에 따라 명명되지 않은 캐시를 공유하도록 조회 변환을 구성하십시오.

- 명명되지 않은 정적 캐시를 공유할 수 있습니다.

- 공유되는 변환은 조회 조건에서 동일한 포트를 사용해야 합니다. 조건에서 서로 다른 연산자를 사용할 수 있지만 포트는 동일해야 합니다.
- 명명되지 않은 캐시 공유를 활성화하려면 일부 변환 속성을 구성해야 합니다.
- 공유되는 변환의 캐시 구조는 호환되어야 합니다.
 - 해시 자동 키 분할을 사용하는 경우 각 변환의 조회/출력 포트가 일치해야 합니다.
 - 해시 자동 키 분할을 사용하지 않는 경우 첫 번째 공유되는 변환의 조회/출력 포트가 일치하거나 이후 변환의 조회/출력 포트의 상위 집합이어야 합니다.
- 해시 자동 키 분할을 사용하는 조회 변환이 서로 다른 대상 로드 순서 그룹에 있을 경우 각 그룹에 대해 동일한 수의 파티션을 구성해야 합니다. 해시 자동 키 분할을 사용하지 않는 경우 각 대상 로드 순서 그룹에 대해 다른 수의 파티션을 구성할 수 있습니다.

다음 테이블은 명명되지 않은 정적 및 동적 캐시를 공유할 수 있는 경우를 보여 줍니다.

공유 캐시	변환의 위치
정적과 정적	매핑에서 원하는 모든 위치
동적과 동적	공유할 수 없습니다.
동적과 정적	공유할 수 없습니다.

다음 테이블에는 명명되지 않은 캐시를 공유하도록 조회 변환을 구성할 때 따라야 하는 지침이 설명되어 있습니다.

속성	명명되지 않은 공유 캐시에 대한 구성
조회 SQL 재정의	조회 SQL 재정의 속성을 사용하는 경우 모든 공유 변환에서 동일한 재정의의 사용해야 합니다.
조회 테이블 이름	일치해야 합니다.
조회 캐싱 설정	활성화해야 합니다.
일치 항목이 여러 개인 경우의 조회 정책	-
조회 조건	공유되는 변환은 조회 조건에서 동일한 포트를 사용해야 합니다. 조건에서 서로 다른 연산자를 사용할 수 있지만 포트는 동일해야 합니다.
연결 정보	연결이 동일해야 합니다. 세션을 구성할 때 데이터베이스 연결이 일치해야 합니다.
소스 유형	일치해야 합니다.
추적 수준	-
조회 캐시 디렉터리 이름	일치할 필요가 없습니다.
지속형 조회 캐시	선택 사항입니다. 지속형 및 비지속형 캐시를 공유할 수 있습니다.
조회 데이터 캐시 크기	통합 서비스는 각 파이프라인 단계에서 첫 번째 공유 변환에 대한 메모리를 할당하고 동일한 파이프라인 단계에서 이후의 공유 변환에 대한 추가 메모리는 할당하지 않습니다.

속성	명명되지 않은 공유 캐시에 대한 구성
조회 인덱스 캐시 크기	통합 서비스는 각 파이프라인 단계에서 첫 번째 공유 변환에 대한 메모리를 할당하고 동일한 파이프라인 단계에서 이후의 공유 변환에 대한 추가 메모리는 할당하지 않습니다.
동적 조회 캐시	명명되지 않은 동적 캐시를 공유할 수 없습니다.
업데이트 시 이전 값 출력	일치할 필요가 없습니다.
캐시 파일 이름 접두사	사용하지 마십시오. 명명된 캐시를 명명되지 않은 캐시와 공유할 수 없습니다.
조회 소스에서 다시 캐시	소스에서 다시 캐시하도록 조회 변환을 구성하는 경우 대상 로드 순서 그룹에서 이후 조회 변환이 소스에서 다시 캐시하도록 구성되었는지 여부에 상관없이 기존 캐시를 공유할 수 있습니다. 소스에서 다시 캐시하도록 이후의 조회 변환을 구성하는 경우 통합 서비스가 이후의 조회 변환을 처리할 때 캐시를 다시 작성하지 않고 캐시를 공유합니다. 소스에서 다시 캐시하도록 대상 로드 순서 그룹의 첫 번째 조회 변환을 구성하지 않고 소스에서 다시 캐시하도록 이후의 조회 변환을 구성하면 변환에서 캐시를 공유할 수 없습니다. 통합 서비스가 각 조회 변환을 처리할 때 캐시를 작성합니다.
조회/출력 포트	두 번째 조회 변환의 조회/출력 포트가 일치하거나 변환에서 통합 서비스가 캐시를 작성하는 데 사용하는 포트의 하위 집합이어야 합니다. 포트 순서는 일치하지 않아도 됩니다.
삽입 기타 항목 업데이트	-
업데이트 기타 항목 삽입	-
날짜/시간 형식	-
1000 단위 구분 기호	-
소수 구분 기호	-
대/소문자 구분 문자열 비교	일치해야 합니다.
Null 순서	일치해야 합니다.
정렬된 입력	-

명명된 조회 캐시 공유

지속형 조회 캐시를 사용하고 캐시 파일에 이름을 지정하여 여러 조회 변환 간에 캐시를 공유할 수도 있습니다. 동일한 매핑 내에서 또는 매핑 사이에서 조회 변환 간에 캐시 1개를 공유할 수 있습니다.

통합 서비스는 다음 프로세스를 사용하여 명명된 조회 캐시를 공유합니다.

1. 통합 서비스는 첫 번째 조회 변환을 처리할 때 캐시 디렉터리에서 파일 이름 접두사가 동일한 캐시 파일을 검색합니다.
2. 통합 서비스가 캐시 파일을 찾고 사용자가 소스에서 다시 캐싱하도록 지정하지 않은 경우, 통합 서비스는 저장된 캐시 파일을 사용합니다.
3. 통합 서비스가 캐시 파일을 찾지 않거나 사용자가 소스에서 다시 캐싱하도록 지정한 경우, 통합 서비스는 데이터베이스 테이블을 사용하여 조회 캐시를 작성합니다.

4. 통합 서비스는 각 대상 로드 순서 그룹을 처리한 후에 캐시 파일을 디스크에 저장합니다.
5. 통합 서비스는 다음 규칙에 따라 캐시 파일 이름 접두사가 동일한 두 번째 조회 변환을 처리합니다.
 - 동일한 대상 로드 순서 그룹에 변환이 있는 경우 통합 서비스는 메모리 캐시를 사용합니다.
 - 서로 다른 대상 로드 순서 그룹에 변환이 있는 경우 통합 서비스는 지속형 파일로부터 메모리 캐시를 다시 작성합니다.
 - 소스에서 다시 캐싱하도록 변환을 구성하고 첫 번째 변환이 다른 대상 로드 순서 그룹에 있는 경우에는 통합 서비스가 데이터베이스로부터 캐시를 다시 작성합니다.
 - 소스에서 다시 캐싱하도록 대상 로드 순서 그룹의 첫 번째 조회 변환을 구성하지 않고 소스에서 다시 캐싱하도록 이후의 조회 변환을 구성하면 통합 서비스가 캐시를 다시 구성하지 못합니다.
 - 캐시 구조가 일치하지 않으면 통합 서비스는 세션이 실패한 것으로 처리합니다.

조회 캐시를 공유하는 두 세션을 동시에 실행하는 경우 통합 서비스는 다음 규칙에 따라 캐시 파일을 공유합니다.

- 조회 변환이 캐시 파일을 읽기만 하면 되는 경우 통합 서비스는 여러 세션을 동시에 처리합니다.
- 특정 세션이 캐시 파일을 읽거나 업데이트하는 동안 다른 세션이 캐시 파일을 업데이트하면 통합 서비스는 세션이 실패한 것으로 처리합니다. 예를 들어 동적 캐시를 사용하거나 소스에서 다시 캐싱하도록 조회 변환을 구성한 경우, 조회 변환이 캐시 파일을 업데이트합니다.

명명된 조회 캐시를 공유하기 위한 지침

다음 지침에 따라 명명된 캐시를 공유하도록 조회 변환을 구성하십시오.

- 캐시가 동적 조회 캐시이거나 소스에서 다시 캐싱하도록 변환이 구성된 경우 세션 간에 조회 캐시를 공유하지 마십시오.
- 동적 조회 변환과 정적 조회 변환 간에 캐시를 공유할 수 있지만 캐시 위치에 대한 지침을 따라야 합니다.
- 명명된 캐시 공유를 활성화하려면 일부 변환 속성을 구성해야 합니다.
- 명명된 캐시에 중복 행이 있을 경우 동적 조회는 캐시를 공유할 수 없습니다.
- 여러 일치 항목 발견 시 오류 조회 정책이 있는 동적 조회 변환에서 작성된 명명된 캐시를 조회 정책이 있는 정적 또는 동적 조회 변환에서 공유할 수 있습니다.
- 첫 번째 값 사용, 마지막 값 사용 또는 모든 값 사용 조회 정책이 있는 동적 조회 변환에서 작성된 명명된 캐시를 동일한 조회 정책이 있는 조회 변환에서 공유할 수 있습니다.
- 공유되는 변환은 매핑에서 동일한 포트를 사용해야 합니다. 캐시의 조건 및 결과 열이 캐시 파일과 일치해야 합니다.
- 동적 조회 변환은 동일한 대상 로드 순서 그룹 내 다른 조회 변환과 캐시를 공유할 수 없습니다. 대상 로드 순서 그룹에서 캐시의 처리는 병렬로 수행되며 대상이 로드된 후 완료됩니다. **PowerCenter** 통합 서비스는 이후 조회 변환에 전체 캐시를 사용할 수 없으므로 캐시 공유를 허용하지 않습니다.

통합 서비스는 조회 변환의 유형 및 위치에 따라 메모리 캐시를 사용할 수도 있고 파일에서 메모리 캐시를 구성할 수도 있습니다.

다음 테이블은 정적 조회 변환과 동적 조회 변환 간에 명명된 캐시를 공유할 수 있는 경우를 보여 줍니다.

테이블 3. 명명된 캐시를 공유하기 위한 위치

공유 캐시	변환의 위치	공유되는 캐시
정적과 정적	<ul style="list-style-type: none"> - 동일한 대상 로드 순서 그룹 - 별개의 대상 로드 순서 그룹 - 별개의 매핑 	<ul style="list-style-type: none"> - 통합 서비스가 메모리 캐시를 사용합니다. - 통합 서비스가 메모리 캐시를 사용합니다. - 통합 서비스가 파일에서 메모리 캐시를 구성합니다.
동적과 동적	<ul style="list-style-type: none"> - 별개의 대상 로드 순서 그룹 - 별개의 매핑 	<ul style="list-style-type: none"> - 통합 서비스가 메모리 캐시를 사용합니다. - 통합 서비스가 파일에서 메모리 캐시를 구성합니다.
동적과 정적	<ul style="list-style-type: none"> - 별개의 대상 로드 순서 그룹 - 별개의 매핑 	<ul style="list-style-type: none"> - 통합 서비스가 파일에서 메모리 캐시를 구성합니다. - 통합 서비스가 파일에서 메모리 캐시를 구성합니다.

다음 테이블에는 명명된 캐시를 공유하도록 조회 변환을 구성할 때 따라야 하는 지침이 설명되어 있습니다.

테이블 4. 명명된 캐시를 공유하기 위한 속성

속성	명명된 공유 캐시에 대한 구성
조회 SQL 재정의	조회 SQL 재정의 속성을 사용하는 경우 모든 공유 변환에서 동일한 재정의의 사용해야 합니다.
조회 테이블 이름	일치해야 합니다.
조회 캐싱 설정	활성화해야 합니다.
일치 항목이 여러 개인 경우의 조회 정책	<ul style="list-style-type: none"> - 여러 일치 항목 발견 시 오류 조회 정책이 있는 동적 조회 변환에서 작성된 명명된 캐시를 조회 정책이 있는 정적 또는 동적 조회 변환에서 공유할 수 있습니다. - 첫 번째 값 사용 또는 마지막 값 사용 조회 정책이 있는 동적 조회 변환에서 작성된 명명된 캐시를 동일한 조회 정책이 있는 조회 변환에서 공유할 수 있습니다. - 모든 값 사용 조회 정책이 있는 동적 조회 변환이 동일한 조회 정책이 있는 다른 활성 조회 변환과 캐시를 공유하는 경우 동적 조회 변환에서 명명된 캐시를 공유할 수 있습니다.
조회 조건	공유되는 변환은 조회 조건에서 동일한 포트를 사용해야 합니다. 조건에서 서로 다른 연산자를 사용할 수 있지만 포트는 동일해야 합니다.
연결 정보	연결이 동일해야 합니다. 세션을 구성할 때 데이터베이스 연결이 일치해야 합니다.
소스 유형	일치해야 합니다.
추적 수준	-
조회 캐시 디렉터리 이름	일치해야 합니다.
지속형 조회 캐시	활성화해야 합니다.

속성	명명된 공유 캐시에 대한 구성
조회 데이터 캐시 크기	동일한 매핑 내 변환이 캐시를 공유하면 통합 서비스가 각 파이프라인 단계에서 첫 번째 공유되는 변환에 대한 메모리를 할당합니다. 동일한 파이프라인 단계에서 이후의 공유 변환에 대한 추가 메모리는 할당하지 않습니다.
조회 인덱스 캐시 크기	동일한 매핑 내 변환이 캐시를 공유하면 통합 서비스가 각 파이프라인 단계에서 첫 번째 공유되는 변환에 대한 메모리를 할당합니다. 동일한 파이프라인 단계에서 이후의 공유 변환에 대한 추가 메모리는 할당하지 않습니다.
동적 조회 캐시	정적 및 동적 캐시 공유에 대한 자세한 내용은 “명명된 조회 캐시를 공유하기 위한 지침” 페이지 300 을 참조하십시오.
업데이트 시 이전 값 출력	일치할 필요가 없습니다.
동적 캐시 업데이트	일치할 필요가 없습니다.
캐시 파일 이름 접두사	일치해야 합니다. 접두사만 입력합니다. .idx 또는 .dat을 입력하지 마십시오. 명명된 캐시를 명명되지 않은 캐시와 공유할 수 없습니다.
소스에서 다시 캐시	소스에서 다시 캐시하도록 조회 변환을 구성하는 경우 대상 로드 순서 그룹에서 이후 조회 변환이 소스에서 다시 캐시하도록 구성되었는지 여부에 상관없이 기존 캐시를 공유할 수 있습니다. 소스에서 다시 캐시하도록 이후의 조회 변환을 구성하는 경우 통합 서비스가 이후의 조회 변환을 처리할 때 캐시를 다시 작성하지 않고 캐시를 공유합니다. 소스에서 다시 캐시하도록 대상 로드 순서 그룹의 첫 번째 조회 변환을 구성하지 않고 소스에서 다시 캐시하도록 이후의 조회 변환을 구성하면 통합 서비스가 캐시를 다시 구성하지 못합니다.
조회/출력 포트	조회/출력 포트는 동일해야 하지만 순서가 동일할 필요는 없습니다.
삽입 기타 항목 업데이트	-
업데이트 기타 항목 삽입	-
1000 단위 구분 기호	-
소수 구분 기호	-
대/소문자 구분 문자열 비교	-
Null 순서	-
정렬된 입력	일치해야 합니다.

참고: 다른 운영 체제에서 작성된 조회 캐시는 공유할 수 없습니다. 예를 들어 UNIX의 통합 서비스만 UNIX의 통합 서비스에서 작성된 조회 캐시를 읽을 수 있고 Windows의 통합 서비스만 Windows의 통합 서비스에서 작성된 조회 캐시를 읽을 수 있습니다.

조회 캐시에 대한 팁

작은 조회 테이블을 캐싱합니다.

작은 조회 테이블을 캐싱하여 성능을 향상시킵니다.

지속형 조회 캐시를 정적 조회 테이블에 대해 사용합니다.

조회 테이블이 세션 간에 변경되지 않는 경우 지속형 조회 캐시를 사용하도록 조회 변환을 구성합니다. 통합 서비스가 캐시 파일을 저장하고 재사용하면 조회 테이블을 읽는 데 필요한 시간이 소요되지 않습니다.

제 19 장

동적 조회 캐시

이 장에 포함된 항목:

- [동적 조회 캐시 개요, 304](#)
- [동적 조회 속성, 305](#)
- [조회 변환 값, 309](#)
- [동적 조회 캐시 업데이트, 311](#)
- [동적 조회를 사용하는 매핑, 313](#)
- [조건부 동적 캐시 업데이트, 315](#)
- [식 결과로 동적 캐시 업데이트, 316](#)
- [조회 소스와 캐시 동기화, 317](#)
- [동적 조회 캐시 예제, 318](#)
- [동적 조회 캐시에 대한 규칙 및 지침, 319](#)

동적 조회 캐시 개요

동적 조회 캐시를 사용하면 캐시와 대상의 동기화 상태를 유지할 수 있습니다. 동적 캐시를 관계형 조회, 플랫폼 파일 조회 또는 파이프라인 조회와 함께 사용할 수 있습니다.

통합 서비스는 첫 번째 조회 요청을 처리할 때 동적 조회 캐시를 빌드합니다. 변환에 전달되는 각 행의 조회 조건을 바탕으로 캐시를 쿼리합니다. 통합 서비스는 각 행을 처리할 때 조회 캐시를 업데이트합니다.

통합 서비스는 소스에서 행을 읽을 때 조회 쿼리의 결과, 행 유형 및 조회 변환 속성에 따라 다음과 같은 동적 조회 캐시 작업 중 하나를 수행합니다.

행을 캐시에 삽입

행이 캐시에 없을 때 행을 캐시에 삽입하도록 조회 변환이 구성된 경우 통합 서비스가 행을 캐시에 삽입합니다. 입력 포트 또는 생성된 시퀀스 ID를 바탕으로 행을 캐시에 삽입하도록 변환을 구성할 수 있습니다. 삽입된 행에는 삽입 플래그가 지정됩니다.

캐시의 행을 업데이트

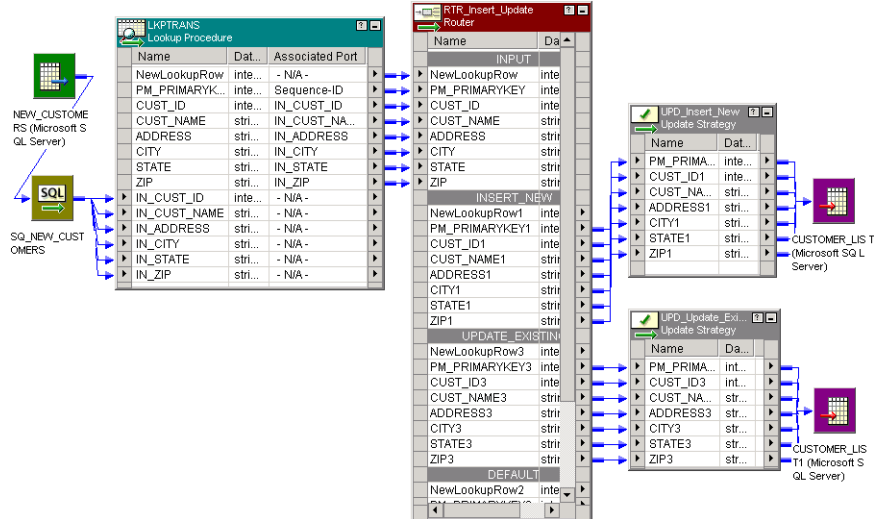
행이 캐시에 있을 때 캐시의 행을 업데이트하도록 조회 변환이 구성된 경우 통합 서비스가 캐시의 행을 업데이트합니다. 통합 서비스는 입력 포트를 바탕으로 캐시의 행을 업데이트합니다. 업데이트된 행에는 업데이트 플래그가 지정됩니다.

캐시를 변경하지 않음

행이 캐시에 있을 때 새 행만 삽입하도록 조회 변환이 구성된 경우 통합 서비스가 캐시를 변경하지 않습니다. 행이 캐시에 없을 때 기존 행만 업데이트하도록 지정한 경우 또는 행이 캐시에 있지만 조회 조건에 따라야 하는 경우에도 캐시가 변경되지 않습니다. 변경되지 않은 행에는 변경되지 않음 플래그가 지정됩니다.

NewLookupRow의 값에 따라 동적 조회 변환과 함께 라우터 또는 필터 변환을 구성하여 삽입 또는 업데이트 행을 대상 테이블로 라우팅할 수 있습니다. 변경되지 않은 행을 다른 대상 테이블 또는 플랫폼 파일로 라우팅하거나 삭제할 수 있습니다.

다음 그림은 동적 조회 캐시를 사용하는 조회 변환이 포함된 매핑을 보여 줍니다.



동적 조회 속성

동적 캐시를 사용하는 조회 변환에는 다음 속성이 있습니다.

- **NewLookupRow.** 디자이너는 동적 캐시를 사용하도록 구성된 조회 변환에 이 포트를 추가합니다. 통합 서비스가 캐시에서 행을 삽입 또는 업데이트하거나, 캐시를 변경하지 않는지 여부를 숫자 값으로 나타냅니다. 조회 캐시와 대상 테이블의 동기화를 유지하려면 NewLookupRow 값이 1 또는 2일 때 행을 대상에 전달하십시오.
- **연결된 식.** 조회 포트 또는 연결된 포트를 각각의 식, 입력/출력 포트 또는 시퀀스 ID와 연결합니다. 통합 서비스는 연결된 식의 데이터를 사용하여 조회 캐시에서 행을 삽입하거나 업데이트합니다. 시퀀스 ID를 연결하는 경우 통합 서비스가 조회 캐시의 삽입된 행에 대한 기본 키를 생성합니다.
- **업데이트에 대한 Null 입력 무시.** 동적 캐시를 사용하도록 조회 변환을 구성하는 경우 디자이너가 조회/출력 포트에 대해 이 포트 속성을 활성화합니다. 이 열의 데이터에 Null 값이 포함된 경우 통합 서비스에서 캐시의 열을 업데이트하지 않도록 하려면 이 속성을 선택합니다.
- **비교 시 무시.** 동적 캐시를 사용하도록 조회 변환을 구성하는 경우 디자이너가 조회 조건에 사용되지 않는 조회/출력 포트에 대해 이 포트 속성을 활성화합니다. 통합 서비스는 기본적으로 모든 조회 포트의 값을 연결된 입력 포트의 값과 비교합니다. 통합 서비스에서 행을 업데이트하기 전에 값을 비교할 때 포트를 무시하도록 하려면 이 속성을 선택합니다.
- **동적 캐시 조건 업데이트.** 통합 서비스에서 조건부로 동적 캐시를 업데이트하도록 허용합니다. 입력 행에 대한 캐시를 업데이트할지 여부를 결정하는 부울 식을 작성할 수 있습니다. 또는 통합 서비스를 활성화하여 입력 행에 대한 식 결과로 캐시를 업데이트할 수 있습니다. 식에는 입력 행 또는 조회 캐시의 값이 포함될 수 있습니다.

NewLookupRows

동적 캐시를 사용하도록 조회 변환을 구성하는 경우 디자이너가 **NewLookupRow** 포트를 변환에 추가합니다. 통합 서비스가 조회 캐시에 대해 수행하는 작업에 따라 포트에 값을 할당합니다.

다음 테이블에는 가능한 **NewLookupRow** 값이 나와 있습니다.

NewLookupRow 값	설명
0	통합 서비스가 캐시에서 행을 업데이트하거나 삽입하지 않습니다.
1	통합 서비스가 캐시에 행을 삽입합니다.
2	통합 서비스가 캐시에서 행을 업데이트합니다.

통합 서비스가 행을 읽는 경우 사용자가 정의하는 조회 변환 속성 및 조회 쿼리 결과에 따라 조회 캐시를 변경합니다. 값 0, 1 또는 2를 **NewLookupRow** 포트에 할당하여 캐시에서 행을 삽입 또는 업데이트하거나, 변경하지 않는지 여부를 나타냅니다.

NewLookupRow 값은 통합 서비스가 조회 캐시를 변경하는 방법을 나타냅니다. 행 유형은 변경하지 않습니다. 따라서 필터 또는 라우터 변환 및 업데이트 전략 변환을 사용하여 대상 테이블 및 조회 캐시를 동기화된 상태로 유지합니다.

새로운 행과 업데이트된 행을 캐시된 대상에 전달하기 전에 이러한 행을 업데이트 전략 변환에 전달하도록 필터 변환을 구성합니다. 업데이트 전략 변환을 사용하여 **NewLookupRow** 값에 따라 삽입 또는 업데이트할 각 행의 행 유형을 변경합니다.

캐시를 변경하지 않는 행을 삭제하거나 다른 대상으로 이러한 행을 전달할 수 있습니다.

NewLookupRow 값에 따라 필터 변환의 필터 조건을 정의합니다. 예를 들어 다음 조건을 사용하여, 삽입된 행 및 업데이트된 행 모두를 캐시된 대상에 전달합니다.

```
NewLookupRow != 0
```

관련 항목:

- [“업스트림 업데이트 전략 변환 구성” 페이지 313](#)

연결된 식

동적 조회 캐시를 구성하는 경우 입력 포트, 입력/출력 포트, 시퀀스 ID 또는 식과 각각의 조회 포트를 연결해야 합니다. 입력/출력 포트를 선택하는 경우 디자이너가 입력/출력 포트를 조회 조건에서 사용되는 조회/출력 포트와 연결합니다. 식을 구성하는 경우 통합 서비스는 연결된 식의 결과로 캐시를 업데이트합니다. 식에는 입력 값 또는 조회 캐시의 값이 포함될 수 있습니다.

조회 포트를 입력/출력 포트와 연결하려면 조회 포트에 대한 연결된 식 열을 클릭합니다. 목록에서 포트를 선택합니다.

식을 작성하려면 조회 포트에 대한 연결된 열을 클릭합니다. 목록에서 연결된 식을 선택합니다. 식 편집기가 표시됩니다.

대상 테이블에서 열에 대한 생성된 키를 작성하려면 연결된 식 열에서 **Sequence-ID**를 선택합니다. 조회 포트에 대한 입력 포트 대신 생성된 키를 **Bigint**, 정수 또는 작은 정수 데이터 유형과 연결할 수 있습니다. **Bigint** 조회 포트의 경우 생성된 키 최대값은 9,223,372,036,854,775,807입니다. 정수 또는 작은 정수 조회 포트의 경우 생성된 키 최대값은 2,147,483,647입니다.

연결된 식 열에서 **Sequence-ID**를 선택하는 경우 통합 서비스가 조회 캐시에 행을 삽입할 때 키를 생성합니다.

통합 서비스는 다음 프로세스를 사용하여 시퀀스 ID를 생성합니다.

1. 통합 서비스가 동적 조회 캐시를 작성할 때 동적 조회 캐시의 시퀀스 ID가 있는 각 포트에 대한 값 범위를 추적합니다.
2. 통합 서비스가 캐시에 데이터 행을 삽입할 때 가장 큰 시퀀스 ID 값을 1씩 증분시켜 포트에 대한 키를 생성합니다.
3. 통합 서비스가 생성된 시퀀스 ID의 최대 수에 도달하는 경우 1부터 다시 시작합니다. 통합 서비스는 기존의 가장 작은 값에서 1을 뺀 값에 도달할 때까지 각 시퀀스 ID를 1씩 증분시킵니다. 통합 서비스에서 고유한 시퀀스 ID 번호가 부족한 경우 세션이 실패합니다.

통합 서비스는 캐시에 삽입하는 각 행에 대한 시퀀스 ID를 생성합니다.

관련 항목:

- [“식 결과로 동적 캐시 업데이트” 페이지 316](#)

Null 값

동적 조회 캐시 및 대상 테이블을 업데이트하면 소스 데이터에 일부 Null 값이 포함될 수 있습니다. 통합 서비스는 다음과 같은 방법으로 Null 값을 처리할 수 있습니다.

- **Null 값 삽입.** 통합 서비스가 소스의 Null 값을 사용하고, 소스의 모든 값을 사용하여 조회 캐시 및 대상 테이블을 업데이트합니다.
- **Null 값 무시.** 통합 서비스가 소스의 Null 값을 무시하고 소스의 Null이 아닌 값만을 사용하여 조회 캐시 및 대상 테이블을 업데이트합니다.

소스 데이터에 Null 값이 포함된 것을 알고 있고, 통합 서비스가 조회 캐시나 대상을 Null 값으로 업데이트하길 원하지 않는 경우 해당 조회/출력 포트에 대해 Null 무시 속성을 선택합니다.

예를 들어 마스터 고객 테이블을 업데이트하려고 합니다. 소스에는 성이 변경된 현재 고객 및 새로운 고객이 포함되어 있습니다. 소스에는 이름이 변경된 고객의 고객 ID와 이름이 포함되어 있지만 주소 데이터에 대한 Null 값이 포함되어 있습니다. 마스터 고객 테이블에서 현재 주소 정보를 유지하면서, 새로운 고객을 삽입하고 현재 고객 이름을 업데이트하려고 합니다.

예를 들어 마스터 고객 테이블에 다음과 같은 데이터가 있습니다.

Primary Key	CUST_ID	CUST_NAME	ADDRESS	CITY	STATE	ZIP
100001	80001	Marion James	100 Main St.	Mt. View	CA	94040
100002	80002	Laura Jones	510 Broadway Ave.	Raleigh	NC	27601
100003	80003	Shelley Lau	220 Burnside Ave.	Portland	OR	97210

소스에는 다음 데이터가 있습니다.

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

매핑의 조회 변환에서 삽입 기타 항목 업데이트를 선택합니다. 조회 변환의 모든 조회/출력 포트에 대해 Null 무시 옵션을 선택합니다. 세션을 실행하면 통합 서비스가 소스 데이터의 Null 값을 무시하고 Null이 아닌 값을 사용하여 조회 캐시 및 대상 테이블을 업데이트합니다.

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

참고: NULL을 무시하도록 선택하는 경우 통합 서비스가 조회 캐시에 기록하는 동일한 값을 대상에 출력하는지 확인해야 합니다. NULL을 무시하도록 선택하는 경우 Null 입력 값을 대상에 전달하면 조회 캐시 및 대상 테이블이 동기화되지 않을 수 있습니다. 통합 서비스가 캐시에서 행을 업데이트할 때 조회/출력 포트에서 출력하게 할 값을 기준으로 매핑을 구성합니다.

- **새 값.** 조회/출력 포트만 조회 변환에서 대상으로 연결합니다.
- **이전 값.** 조회 변환 후 그리고 필터나 라우터 변환 전에 식 변환을 추가합니다. 대상 테이블의 각 포트마다 식 변환에서 출력 포트를 추가하고 대상에 Null 입력 값을 출력하지 않도록 확인하는 식을 작성하십시오.

비교 시 포트 무시

동적 조회 캐시를 사용하는 세션을 실행하는 경우 통합 서비스는 기본적으로 모든 조회 포트의 값을 연결된 입력 포트의 값과 비교합니다. 해당 값을 비교하여 조회 캐시의 행을 업데이트하는지 여부를 확인합니다. 입력 포트의 값이 조회 포트의 값과 다른 경우 통합 서비스가 캐시에서 행을 업데이트합니다.

일부 포트는 비교하지 않으려는 경우 통합 서비스가 포트를 비교할 때 무시하게 할 포트를 선택할 수 있습니다. 포트가 조회 조건에서 사용되지 않는 경우 디자이너는 조회/출력 포트에 대해 이 속성만 활성화합니다 비교 시 일부 포트를 무시하면 성능을 향상시킬 수 있습니다.

업데이트해야 하는 데이터가 행에 있는지 여부를 나타내는 열이 소스 데이터에 포함된 경우 이와 같이 하려고 할 수 있습니다. 캐시 및 대상 테이블에서 행을 업데이트하는지 여부를 나타내는 포트를 제외하고 모든 조회 포트에 대한 비교 속성에서 무시를 선택합니다.

하나 이상의 포트를 비교하도록 조회 변환을 구성해야 합니다. 모든 포트를 무시하는 경우 통합 서비스에서 세션이 실패합니다.

비교 속성에서 무시를 선택 취소하려면 포트를 조회 포트와 연결합니다. 비교 속성에서 무시를 선택 취소하는 경우 PowerCenter 통합 서비스가 캐시에서 행을 업데이트합니다.

SQL 재정의

동적 캐시를 사용하는 조회 변환에 WHERE 절을 추가할 때는 조회 변환 전에 필터 변환을 연결하여 캐시 또는 대상 테이블에 삽입하지 않을 행을 필터링합니다. 필터 변환을 포함하지 않으면 캐시 및 대상 테이블 간에 결과가 일관되지 않을 수 있습니다.

직원 테이블 EMP에 대해 동적 조회를 수행하여 EMP_ID를 기준으로 일치 행을 표시하도록 조회 변환을 구성하는 경우를 예로 들어 보겠습니다. 이 경우 다음과 같은 조회 SQL 재정의의 정의를 합니다.

```
SELECT EMP_ID, EMP_STATUS FROM EMP ORDER BY EMP_ID, EMP_STATUS WHERE EMP_STATUS = 4
```

세션을 처음 실행할 때 통합 서비스는 조회 SQL 재정의의 기준으로 대상 테이블에서 조회 캐시를 빌드합니다. 캐시의 모든 행은 WHERE 절의 조건(EMP_STATUS = 4)과 일치합니다.

예를 들어 통합 서비스는 지정한 조회 조건과 일치하는 소스 행을 읽지만 **EMP_STATUS** 값은 2입니다. 대상에는 **EMP_STATUS**가 2인 행이 있을 수도 있지만 통합 서비스는 **SQL** 재정의로 인해 캐시에서 해당 행을 찾지 않습니다. 통합 서비스는 캐시에 행을 삽입한 다음 대상 테이블에 행을 전달합니다. 통합 서비스가 대상 테이블에 이 행을 삽입할 때 해당 행이 이미 있으면 결과가 일관되지 않을 수 있습니다. 또한 캐시의 모든 행이 **SQL** 재정의에서 **WHERE** 절의 조건과 일치하지는 않습니다.

WHERE 절과 일치하는 행만 캐시에 삽입되는지 확인하려면 조회 변환 전에 필터 변환을 추가하고 필터 조건을 조회 **SQL** 재정의에서 **WHERE** 절의 조건으로 정의합니다.

위의 예제에서는 **SQL** 재정의의 **WHERE** 절 및 필터 변환에 다음 필터 조건을 입력합니다.

EMP_STATUS = 4

조회 변환 값

조회 변환에는 입력 포트, 조회 및 출력 포트에 대한 값이 포함됩니다. 동적 조회 캐시를 활성화한 경우 출력 포트 값은 동적 조회 캐시의 구성 방식에 따라 다릅니다.

조회 변환에는 다음 유형의 값이 포함됩니다.

입력 값

통합 서비스가 조회 변환에 전달하는 값입니다.

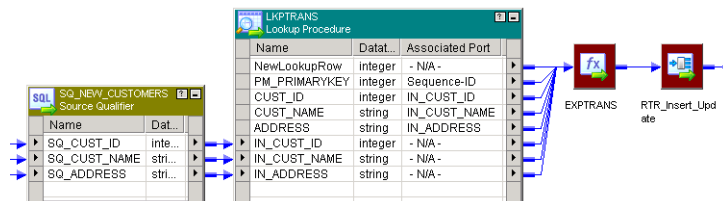
조회 값

통합 서비스가 캐시에 삽입하는 값입니다.

출력 값

통합 서비스가 조회 변환의 출력 포트에서 전달하는 값입니다. 조회/출력 포트의 출력 값은 통합 서비스가 행을 업데이트할 때 사용자가 출력하도록 선택한 값(이전 값 또는 새 값)에 따라 다릅니다.

예를 들어 다음 조회 변환은 동적 조회 캐시를 사용합니다.



다음 조회 조건을 정의합니다.

IN_CUST_ID = CUST_ID

기본적으로 조회 변환에 입력되는 모든 행의 행 유형이 삽입됩니다. 캐시 및 대상 테이블에서 삽입과 업데이트를 모두 수행하려면 조회 변환에서 **삽입 기타 항목 업데이트** 속성을 선택합니다.

다음 섹션에는 세션을 실행할 때 캐시, 입력 행, 조회 행 및 출력 행의 행 값이 설명되어 있습니다.

초기 캐시 값

세션을 실행할 경우 통합 서비스가 다음 데이터를 사용하여 대상 테이블에서 조회 캐시를 작성합니다.

PK_PRIMARYKEY	CUST_ID	CUST_NAME	ADDRESS
100001	80001	Marion James	100 Main St.

PK_PRIMARYKEY	CUST_ID	CUST_NAME	ADDRESS
100002	80002	Laura Jones	510 Broadway Ave.
100003	80003	Shelley Lau	220 Burnside Ave.

입력 값

대상 테이블에 있는 행과 대상 테이블에 없는 행이 소스에 포함되어 있습니다. 다음과 같은 행이 소스 한정자 변환에서 조회 변환으로 전달됩니다.

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.

조회 값

통합 서비스가 조회 조건을 기준으로 캐시의 값을 조회합니다. 기존 고객 ID 80001 및 80002에 대해 캐시의 행을 업데이트합니다. 고객 ID 99001에 대해 캐시에 행을 삽입합니다. 통합 서비스가 새 행에 대해 새 키 (PK_PRIMARYKEY)를 생성합니다.

다음 표에는 조회에서 반환된 행과 값이 표시되어 있습니다.

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.

출력 값

통합 서비스가 동적 캐시에서 수행한 삽입 및 업데이트를 기반으로 조회 변환의 행에 플래그 지정합니다. 이러한 행은 식 변환을 통과하여 라우터 변환으로 전달되고, 라우터 변환은 삽입된 행 및 업데이트된 행을 필터링하고 업데이트 전략 변환으로 행을 전달합니다. 업데이트 전략 변환은 NewLookupRow 포트의 값에 따라 행에 플래그를 지정합니다.

조회/출력 및 입력/출력 포트의 값은 통합 서비스가 행을 업데이트할 때 이전 값 또는 새 값 중 어느 것을 출력하도록 선택하는지에 따라 달라집니다. 그러나 새 행과 업데이트된 행의 경우 시퀀스 ID를 사용하는 NewLookupRow 포트 및 조회/출력 포트의 출력 값이 동일합니다.

새 값을 출력하도록 선택한 경우, 조회/출력 포트는 다음 값을 출력합니다.

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.

이전 값을 출력하도록 선택한 경우, 조회/출력 포트는 다음 값을 출력합니다.

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.

통합 서비스가 조회 캐시의 행을 업데이트하는 경우 캐시 및 대상 테이블의 행에 대해 기본 키 (PK_PRIMARYKEY) 값을 사용합니다.

통합 서비스는 캐시에서 찾지 않는 대상 고객에 대해 시퀀스 ID를 사용하여 기본 키를 생성합니다. 통합 서비스는 조회 캐시에 기본 키 값을 삽입하고 값을 조회/출력 포트에 반환합니다.

통합 서비스는 입력/출력 포트에서 입력 값과 일치하는 값을 출력합니다.

참고: 입력 값이 NULL이고 연결된 입력 포트에 대해 Null 무시 속성을 선택한 경우, 입력 값은 조회 값 또는 입력/출력 포트의 값과 일치하지 않습니다. Null 무시 속성을 선택한 경우 Null 값을 대상에 전달하면 조회 캐시 및 대상 테이블이 동기화되지 않을 수 있습니다. Null 값을 대상에 전달하지 않는지 확인해야 합니다.

동적 조회 캐시 업데이트

통합 서비스가 행 유형, 쿼리 결과 및 조회 변환 속성을 기반으로 동적 캐시를 업데이트합니다.

동적 조회 캐시를 사용하는 경우 조회 변환에 들어가는 행의 행 유형을 삽입 또는 업데이트로 정의합니다. 일부 행은 삽입으로 그리고 일부 행은 업데이트로 또는 모두 삽입으로 아니면 모두 업데이트로 정의할 수 있습니다. 기본적으로 조회 변환에 들어가는 모든 행의 행 유형은 삽입입니다. 조회 변환 앞에 업데이트 전략 변환을 추가하여 행 유형을 업데이트로 정의할 수 있습니다.

통합 서비스는 캐시에서 행을 삽입 또는 업데이트합니다. 또는 캐시를 변경하지 않습니다. 기본적으로 조회 변환은 행 유형이 삽입인 경우 캐시에 행을 삽입하고, 행 유형이 업데이트인 경우 행을 업데이트합니다.

하지만 다음 조회 속성을 구성하여 통합 서비스가 삽입을 처리하고 캐시에 업데이트하는 방법을 변경할 수 있습니다.

- **삽입 기타 항목 업데이트.** 이 옵션은 조회 변환을 시작하는 삽입 유형의 행에 적용됩니다. 데이터가 이미 캐시에 존재하는 경우 이 옵션을 사용하여 캐시에서 행을 업데이트합니다. 이 옵션을 활성화하지 않는 경우 통합 서비스는 행이 존재하는지 여부에 관계없이 캐시에 행을 삽입합니다.
- **업데이트 기타 항목 삽입.** 이 옵션은 조회 변환을 시작하는 업데이트 유형의 행에 적용됩니다. 데이터가 캐시에 존재하지 않는 경우 이 옵션을 사용하여 업데이트 행을 삽입합니다. 이 옵션을 활성화하지 않으면 행이 캐시에 존재하지 않는 경우 통합 서비스가 해당 행을 무시합니다.

참고: 삽입 기타 항목 업데이트 또는 업데이트 기타 항목 삽입 속성을 선택하거나, 두 속성 모두를 선택하거나 아무 속성도 선택하지 않을 수 있습니다.

조회 소스 및 동적 캐시를 동기화하도록 조회 변환을 구성할 수 있습니다. 조회 소스 및 캐시를 동기화하는 경우 조회 변환이 조회 소스에 직접 행을 삽입합니다. 사용자는 대상에 삽입 행을 전달하지 않습니다. 이 구성을 사용하여 동일한 대상에 행을 삽입하는 조회 변환으로 여러 세션을 실행합니다.

삽입 기타 항목 업데이트

동적 조회 캐시에서 행 유형이 삽입인 기존 행을 업데이트하려면 삽입 기타 항목 업데이트 속성을 사용합니다.

이 속성은 조회 변환을 시작하는 삽입 유형의 행에만 적용됩니다. 업데이트와 같은 다른 모든 유형의 행이 조회 변환을 시작하는 경우 **삽입 기타 항목 업데이트** 속성은 통합 서비스의 행 처리 방식에 영향을 미치지 않습니다.

삽입 기타 항목 업데이트를 선택하고 조회 변환을 시작하는 행 유형이 삽입인 경우 통합 서비스는 해당 행이 새 행일 경우 캐시에 삽입합니다. 인덱스 캐시에 존재하지만 데이터 캐시에는 존재하지 않는 행이 현재 행과 다른 경우 통합 서비스가 데이터 캐시에 해당 행을 업데이트합니다.

삽입 기타 항목 업데이트를 선택하지 않고 조회 변환을 시작하는 행 유형이 삽입인 경우 통합 서비스는 해당 행이 새 행일 경우 캐시에 삽입하고 이미 존재하는 행일 경우 캐시를 변경하지 않습니다.

다음 표에는 조회 변환을 시작하는 행의 유형이 삽입일 때 통합 서비스가 조회 캐시를 변경하는 방식이 설명되어 있습니다.

삽입 기타 항목 업데이트 옵션	행이 캐시에 있음	데이터 캐시가 다름	조회 캐시 결과	NewLookupRow 값
선택 취소 - 삽입만	예	-	변경 없음	0
선택 취소 - 삽입만	아니오	-	삽입	1
선택됨	예	예	업데이트	2 ¹
선택됨	예	아니오	변경 없음	0
선택됨	아니오	-	삽입	1

¹ **조회 조건에 없는 모든 조회 포트에 대해 Null 무시를 선택한 경우 이러한 모든 포트에 null 값이 포함되면 통합 서비스가 캐시를 변경하지 않으며 NewLookupRow 값은 0과 동일합니다.**

업데이트 기타 항목 삽입

업데이트 기타 항목 삽입 속성을 사용하여 행 유형이 업데이트인 경우 동적 조회 캐시에 새 행을 삽입할 수 있습니다.

조회 변환에서 **업데이트 기타 항목 삽입** 속성을 선택할 수 있습니다. 이 속성은 행 유형이 업데이트인 조회 변환에 입력되는 행에만 적용됩니다. 삽입과 같은 기타 유형의 행이 조회 변환에 입력되면 이 속성은 통합 서비스가 행을 처리하는 방법에 영향을 주지 않습니다.

조회 변환에 입력되는 행 유형이 업데이트일 때 이 속성을 선택하면 통합 서비스는 행이 인덱스 캐시에 있으며 캐시 데이터가 기존 행과 다른 경우 캐시의 행을 업데이트합니다. 새로운 행일 경우 통합 서비스가 캐시에 행을 삽입합니다.

조회 변환에 입력되는 행 유형이 업데이트일 때 이 속성을 선택하지 않으면 통합 서비스는 행이 있는 경우 캐시에서 행을 업데이트하며 새 행의 경우에는 캐시를 변경하지 않습니다.

조회 조건에 없는 모든 조회 포트에 null 값이 포함된 경우 해당 포트에 대해 **Null 무시**를 선택하면 통합 서비스는 캐시를 변경하지 않으며 NewLookupRow 값은 0이 됩니다.

다음 테이블에서는 조회 변환에 입력되는 행의 유형이 업데이트일 때 통합 서비스가 조회 캐시를 변경하는 방법을 설명합니다.

업데이트 기타 항목 삽입 옵션	행이 캐시에 있음	데이터 캐시가 다름	조회 캐시 결과	NewLookupRow 값
지워짐(업데이트만 해당)	예	예	업데이트	2
지워짐(업데이트만 해당)	예	아니오	변경 없음	0
지워짐(업데이트만 해당)	아니오	-	변경 없음	0
선택됨	예	예	업데이트	2
선택됨	예	아니오	변경 없음	0
선택됨	아니오	-	삽입	1

동적 조회를 사용하는 매핑

동적 조회 변환이 포함된 매핑을 구성하는 경우 조회 변환이 동적 캐시를 업데이트하는 방법에 영향을 주도록 조회 변환에서 변환 업스트림을 구성합니다. 통합 서비스가 대상에서 업데이트 표시된 행을 업데이트하고 삽입 표시된 행을 삽입하도록 조회 변환에서 변환 다운스트림을 구성합니다.

업스트림 업데이트 전략 변환 구성

조회 변환이 받는 행의 행 유형을 변경하도록 업스트림 업데이트 전략 변환을 구성할 수 있습니다.

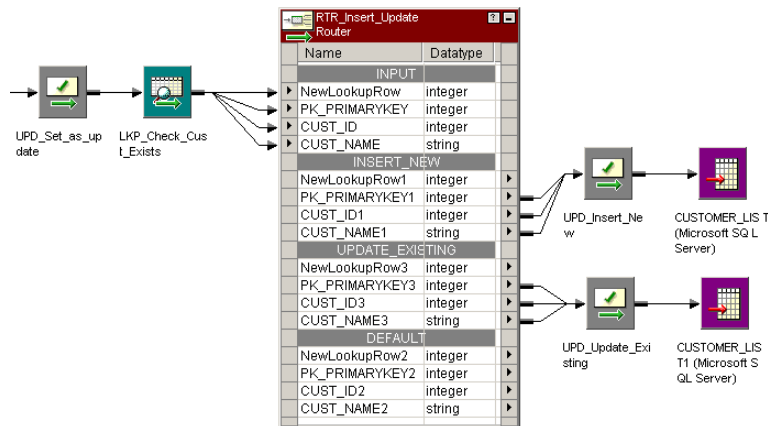
동적 조회 캐시를 사용하는 경우 업데이트 전략 변환을 사용하여 다음 행에 대한 행 유형을 정의합니다.

- **조회 변환에 들어가는 행.** 기본적으로 조회 변환에 들어가는 모든 행의 행 유형은 삽입입니다. 하지만 조회 변환 앞에 업데이트 전략 변환을 사용하여 행을 업데이트로 정의하거나, 일부는 업데이트로, 일부는 삽입으로 정의합니다.
- **조회 변환에서 나가는 행.** NewLookupRow 값은 통합 서비스가 조회 캐시를 변경한 방법을 나타내지만, 행 유형을 변경하지는 않습니다. 조회 변환 뒤에 필터 또는 라우터 변환을 사용하여 NewLookupRow 값을 기반으로 조회 변환에서 나가는 행을 전달합니다. 필터 또는 라우터 변환 뒤에 업데이트 전략 변환을 사용하여 매핑의 대상 정의 앞에 삽입 또는 업데이트로 행에 플래그를 지정합니다.

참고: 변경되지 않은 행을 삭제하려는 경우 필터 또는 라우터 변환에서 NewLookupRow가 0인 대상으로 행을 연결하지 마십시오.

조회 변환에 들어가는 행에 대해 삽입으로 행 유형을 정의하는 경우 조회 변환에서 삽입 기타 항목 업데이트 속성을 사용합니다. 조회 변환에 들어가는 행에 대해 업데이트로 행 유형을 정의하는 경우 조회 변환에서 업데이트 기타 항목 삽입 속성을 사용합니다. 조회 변환에 들어가는 일부 행은 업데이트로, 일부 행은 삽입으로 정의하는 경우, 업데이트 기타 항목 삽입 또는 삽입 기타 항목 업데이트 속성을 사용하거나 두 속성을 모두 사용합니다.

다음 그림에는 동적 캐시를 사용하는 조회 변환 및 여러 업데이트 전략 변환이 있는 매핑이 나와 있습니다.



이 예제에서 조회 변환 앞에 있는 업데이트 전략 변환은 모든 행에 업데이트로 플래그를 지정합니다. 조회 변환에서 업데이트 기타 항목 삽입 속성을 선택합니다. 라우터 변환은 삽입된 행을 **Insert_New** 업데이트 전략 변환에 보내고 업데이트된 행을 **Update_Existing** 업데이트 전략 변환에 보냅니다. 대상에 연결되지 않은 출력 행은 삭제됩니다. 조회 변환 플러그 오른쪽에 있는 두 개의 업데이트 전략 변환은 대상의 삽입 또는 업데이트로 행에 플래그를 지정합니다.

다운스트림 변환 구성

동적 조회 캐시와 대상이 동기화되도록 하려면 다운스트림 변환을 구성합니다.

동적 조회 캐시를 사용하는 경우 통합 서비스는 조회 캐시에 먼저 기록한 다음 대상 테이블에 기록합니다. 통합 서비스가 데이터를 대상에 기록하지 않을 경우 조회 캐시와 대상 테이블이 동기화되지 않을 수 있습니다. 예를 들어 대상 데이터베이스가 데이터를 거부할 수 있습니다.

조회 캐시를 조회 테이블과 동기화 상태로 유지하려면 다음 지침을 고려하십시오.

- **NewLookupRow** 값이 1 또는 2일 경우 행을 캐시된 대상에 전달하는 라우터 변환을 사용합니다.
- **NewLookupRow** 값이 0일 경우 행을 삭제하는 라우터 변환을 사용합니다. 또는 행을 다른 대상에 출력합니다.
- 행을 대상에 삽입 또는 업데이트하도록 플래그 지정하려면 조회 변환 후 업데이트 전략 변환을 사용합니다.
- 세션을 실행할 때 오류 임계값을 1로 설정합니다. 오류 임계값을 1로 설정한 경우 첫 번째 오류가 발생하면 세션이 실패합니다. 통합 서비스가 새 캐시 파일을 디스크에 기록하지 않습니다. 대신, 원본 캐시 파일이 있는 경우 이 파일을 복원합니다. 또한 세션 이전 대상 테이블을 대상 데이터베이스에 복원해야 합니다.
- 통합 서비스가 조회 캐시에 기록하는 동일한 값을 조회 변환이 대상에 출력하는지 확인하십시오. 업데이트 시 새 값을 출력하도록 선택한 경우 입력/출력 포트 대신 조회/출력 포트만 대상 테이블에 연결하십시오. 업데이트 시 이전 값을 출력하도록 선택한 경우 조회 변환 후 그리고 라우터 변환 전에 식 변환을 추가하십시오. 대상 테이블의 각 포트마다 식 변환에서 출력 포트를 추가하고 대상에 **Null** 입력 값을 출력하지 않도록 확인하는 식을 작성하십시오.
- 세션 속성에서 소스 행을 다음으로 처리 속성을 데이터 구동으로 설정하십시오.
- 업데이트 전략 대상 테이블 옵션을 정의할 경우 삽입 및 업데이트를 업데이트로 선택합니다. 이 경우 통합 서비스가 업데이트 표시된 행을 업데이트하고 삽입 표시된 행을 삽입하게 됩니다. 세션 속성에서 매핑 탭의 변환 보기에서 이 옵션을 선택합니다.

조회 조건 열의 Null 값

통합 서비스는 행이 캐시에 존재하는지 여부에 따라 조회 조건 열에 null 값이 포함된 행을 다르게 처리합니다.

행이 조회 캐시에 없을 경우 통합 서비스는 행을 캐시에 삽입하고 대상 테이블에 전달합니다. 행이 조회 캐시에 있을 경우 통합 서비스는 캐시 또는 대상 테이블에서 행을 업데이트하지 않습니다.

참고: 소스 데이터의 조회 조건 열에 null 값이 있을 경우 오류 임계값을 1로 설정합니다. 이렇게 하면 통합 서비스가 행을 캐시에 삽입하지만 데이터베이스가 Null이 아님 제약 조건으로 인해 행을 거부해도 조회 캐시와 테이블이 동기화된 상태로 유지됩니다.

동적 조회 캐시를 사용하여 세션 구성

업데이트 전략 변환 및 동적 조회 캐시를 사용하여 세션을 구성하는 경우 다음 업데이트 전략 테이블 옵션 중 하나를 구성해야 합니다.

- 삽입 선택
- 업데이트 시 업데이트 선택
- 삭제 선택 안 함

업데이트 전략 대상 테이블 옵션은 통합 서비스가 업데이트 표시된 행을 업데이트하고 삽입 표시된 행을 삽입하는지 확인합니다.

세션 속성에 있는 속성 탭의 일반 옵션 설정에서 소스 행을 다음으로 처리 옵션을 데이터 구동으로 정의합니다. 데이터 구동을 선택하지 않으면 통합 서비스는 사용자가 소스 행을 다음으로 처리 옵션에서 지정하는 행 유형에 대한 모든 행에 플래그를 지정하고 행에 플래그를 지정하는 데 매핑의 업데이트 전략 변환을 사용하지 않습니다. 통합 서비스는 올바른 행을 삽입하고 업데이트하지 않습니다. 업데이트 시 업데이트를 선택하지 않을 경우 통합 서비스는 대상 테이블에서 업데이트 플래그가 지정된 행을 올바르게 업데이트하지 않습니다. 따라서 조회 캐시와 대상 테이블이 동기화되지 않을 수 있습니다.

조건부 동적 캐시 업데이트

부울 식의 결과를 기반으로 동적 조회 캐시를 업데이트할 수 있습니다. 통합 서비스는 식이 True일 경우 캐시를 업데이트합니다.

대상 테이블에 제품 번호, 재고 수량 및 타임스탬프 열이 있는 경우를 예로 들겠습니다. 재고 수량을 최신 소스 값으로 업데이트해야 합니다. 소스 데이터의 타임스탬프가 동적 캐시의 타임스탬프보다 큰 경우 재고 수량을 업데이트할 수 있습니다. 조회 변환에서 다음과 유사한 식을 작성하십시오.

```
lookup_timestamp < input_timestamp
```

식에는 조회 및 입력 포트가 포함될 수 있습니다. 기본 제공 변수, 매핑 변수 및 매개 변수의 변수에 액세스할 수 있습니다. 사용자 정의 함수를 포함하고 연결되지 않은 변환을 참조할 수 있습니다.

식에서 true, false 또는 NULL을 반환합니다. 식 결과가 NULL일 경우 식이 false입니다. 통합 서비스에서 캐시를 업데이트하지 않습니다. 식 결과를 true로 변경해야 할 경우 식의 NULL 값 확인을 추가할 수 있습니다. 기본 식 값은 true입니다.

변환 개발자를 사용하여 식을 작성하십시오. 세션 수준에서 동적 캐시 업데이트 조건을 재정의할 수 없습니다.

세션 처리

조건부 동적 캐시 업데이트를 구성하면 데이터가 존재하지 않는 경우 통합 서비스가 조건부 식을 고려하지 않습니다. 삽입 기타 항목 업데이트를 활성화하면 데이터가 존재하지 않는 경우 통합 서비스가 캐시에서 행을 삽입합

니다. 데이터가 존재하면 조건부 식이 **true**인 경우 캐시를 업데이트합니다. 업데이트 기타 항목 삽입을 활성화하면 데이터가 캐시에 존재하고 조건부 식이 **true**인 경우 통합 서비스가 캐시를 업데이트합니다. 데이터가 캐시에 없는 경우에는 통합 서비스가 캐시에서 행을 삽입합니다.

식이 **true**인 경우 **NewLookupRow** 값이 1이며 통합 서비스가 행을 업데이트합니다. 식이 **false** 또는 **NULL**인 경우 **NewLookupRow** 값이 0입니다. 통합 서비스에서 행을 업데이트하지 않습니다.

동적 캐시를 동기화하도록 조회 변환을 구성한 경우 통합 서비스가 행을 캐시에 삽입하면 조회 소스에도 행이 삽입됩니다.

조건부 동적 캐시 조회 구성

조건부 식을 작성하려면 먼저 조건부 동적 캐시 조회를 수행하는 조회 변환을 활성화해야 합니다.

다음 단계를 사용하여 조건부 동적 캐시 조회를 구성합니다.

1. 변환 개발자에서 동적 조회 변환을 작성합니다.
2. 속성 탭을 클릭합니다.
3. 조회 캐싱 및 동적 조회 캐시 속성을 활성화합니다.
4. 식 편집기에 액세스하려면 동적 캐시 조건 업데이트 속성의 이동 단추를 클릭합니다.
5. 식 조건을 정의합니다. 식의 입력 포트, 조회 포트 및 함수를 선택할 수 있습니다.
6. 유효성 검사를 클릭하여 식이 유효한지 확인합니다.

식의 유효성을 검사하지 않으면 식 편집기를 닫을 때 디자이너가 식의 유효성을 검사합니다. 식이 유효하지 않으면 디자이너가 경고를 표시합니다.

식 결과로 동적 캐시 업데이트

동적 조회 캐시 값을 식 결과로 업데이트할 수 있습니다.

예를 들어, 제품 테이블 대상에 주문 개수를 포함하는 숫자 열이 있습니다. 조회 변환에서 제품에 대한 주문을 받을 때마다 다음 식 결과로 동적 캐시 **cache order_count**를 업데이트합니다.

```
order_count = order_count + 1
```

조회 변환에서 **order_count**를 반환합니다.

통합 서비스에서 식이 **Null**로 평가되는 경우를 처리하는 방법을 구성할 수 있습니다.

Null 식 값

식 값 중 하나가 **NULL**인 경우 식이 **NULL**을 반환합니다. 하지만 **NULL**이 아닌 값을 반환하도록 식을 구성할 수 있습니다.

식이 조회 포트를 참조하지만 소스 데이터가 신규인 경우 조회 포트에 기본값이 포함됩니다. 기본값은 **NULL**일 수 있습니다. **IsNull** 식을 구성하면 **NULL** 값을 확인할 수 있습니다.

예를 들어 다음 식은 **lookup_column**이 **NULL**인지 확인합니다.

```
iif (isnull(lookup_column), input_port, user_expression)
```

열이 **NULL**인 경우 **input_port** 값을 반환합니다. 그렇지 않은 경우 식의 값을 반환합니다.

세션 처리

통합 서비스는 식 결과에 따라 동적 조회 캐시에 행을 삽입 및 업데이트할 수 있습니다. 식의 결과는 조회 포트 값이 NULL인지 여부와 식에 포함되었는지 여부에 따라 달라질 수 있습니다.

삽입 기타 항목 업데이트를 활성화한 경우 데이터가 캐시에 없으면 통합 서비스가 식 결과에 따라 행을 삽입합니다. 데이터가 캐시에 없는 경우 조회 포트 값은 NULL입니다. 식이 조회 포트 값을 참조하는 경우 통합 서비스가 식의 기본 포트 값을 대체합니다. 삽입 기타 항목 업데이트를 활성화한 경우 데이터가 캐시에 있으면 통합 서비스가 식 결과에 따라 행을 업데이트합니다.

업데이트 기타 항목 삽입을 활성화한 경우 데이터가 캐시에 있으면 통합 서비스가 식 결과에 따라 캐시를 업데이트합니다. 데이터가 캐시에 없는 경우에는 통합 서비스가 식 결과를 포함하는 행을 삽입합니다. 식이 조회 포트 값을 참조하는 경우 통합 서비스가 식의 기본 포트 값을 대체합니다.

동적 캐시를 동기화하도록 조회 변환을 구성한 경우 통합 서비스는 식 결과에 따라 행을 조회 캐시에 삽입합니다. 통합 서비스는 식 결과에 따라 행을 조회 소스에 삽입합니다.

동적 캐시 업데이트를 위한 식 구성

조건부 식을 작성하려면 먼저 동적 조회를 수행하는 조회 변환을 활성화해야 합니다.

다음 단계를 사용하여 동적 캐시 조회에 대한 식을 구성합니다.

1. 변환 개발자에서 동적 조회 변환을 작성합니다.
2. 속성 탭을 엽니다.
3. 조회 캐싱 및 동적 조회 캐시 속성을 활성화합니다.
4. 식을 작성하려면 업데이트하려는 조회 포트에 대한 연결된 식 목록을 클릭합니다.
5. 연결된 식을 선택합니다.
식 편집기가 표시됩니다.
6. 입력 포트, 조회 포트 및 함수를 선택하여 식을 정의합니다. 식 반환 값이 조회 포트 유형과 일치해야 합니다.
7. 유효성 검사를 클릭하여 식이 유효한지 확인합니다.
식의 유효성을 검사하지 않으면 식 편집기를 닫을 때 디자이너가 식의 유효성을 검사합니다. 식이 유효하지 않으면 디자이너가 경고를 표시합니다.

조회 소스와 캐시 동기화

조회 변환은 대상으로 전달하는 행을 추적하기 위해 동적 조회 캐시를 유지합니다. 동일한 대상을 여러 세션이 업데이트하는 경우, 대상 대신 동일한 조회 소스에 맞춰 동적 조회 캐시를 동기화하도록 각 세션의 조회 변환을 구성할 수 있습니다.

캐시를 조회 소스와 동기화하도록 조회 변환을 구성하면 조회 변환이 조회 소스에 대해 조회를 수행합니다. 데이터가 조회 소스에 없는 경우 조회 변환은 동적 조회 캐시를 업데이트하기 전에 조회 캐시에 행을 삽입합니다.

다른 세션이 행을 삽입한 경우 조회 소스에 데이터가 있을 수도 있습니다. 조회 캐시를 조회 소스에 동기화하기 위해 통합 서비스는 조회 소스에서 최신 값을 검색합니다. 조회 변환은 조회 소스에 있는 값을 동적 조회 캐시에 삽입합니다. 조회 소스는 관계형 테이블이어야 합니다.

예를 들어 여러 세션이 동시에 실행되는 것으로 가정합니다. 각 세션은 새 제품 이름에 대해 제품 번호를 생성합니다. 세션이 제품 번호를 생성하면 다른 세션은 동일한 제품 번호를 사용하여 제품을 식별해야 합니다. 제품 번호는 한 번 생성되어 조회 소스에 삽입됩니다. 해당 제품을 포함하는 행을 다른 세션이 처리하는 경우, 해당 세션

은 조회 소스에 있는 제품 번호를 사용해야 합니다. 각 세션은 어떤 제품 번호가 이미 생성되었는지 확인하기 위해 조회 소스에 대한 조회를 수행합니다.

통합 서비스는 행 삽입을 위해 다음 태스크를 수행합니다.

- 통합 서비스는 동적 조회 캐시에 대한 조회를 수행합니다. 데이터가 동적 조회 캐시에 없는 경우 통합 서비스는 조회 소스에 대한 조회를 수행합니다.
- 데이터가 조회 소스에 있으면 통합 서비스가 조회 소스에서 데이터를 검색합니다. 통합 서비스는 조회 소스에 있는 열을 포함하는 동적 조회 캐시에 행을 삽입합니다. 소스 행을 포함하는 캐시는 업데이트하지 않습니다.
- 데이터가 조회 소스에 없으면 통합 서비스가 조회 소스에 데이터를 삽입하고 캐시에 행을 삽입합니다.

조회 소스는 조회 캐시와 동일한 열을 포함합니다. 열에 조회 변환의 결과가 반영되었거나 열이 조회 조건의 일부인 경우가 아니라면 통합 서비스는 조회 캐시에 열을 삽입하지 않습니다.

NewLookupRow

동적 캐시를 조회 소스와 동기화하는 경우 조회 변환 동작이 업데이트 행에 대해 변경되지 않습니다. 통합 서비스가 업데이트 행을 받으면 동적 조회 캐시를 업데이트하고 **NewLookupRow**의 값을 2로 설정합니다. 조회 소스를 업데이트하지 않습니다. 업데이트 전략 변환을 통해 업데이트 행을 라우팅하고 행을 대상에 전달할 수 있습니다.

통합 서비스가 행을 조회 소스에 삽입할 때 **NewLookupRow**를 1로 설정합니다. 통합 서비스가 동적 조회 캐시를 업데이트합니다. 변환이 동적 캐시를 동기화하도록 구성되고 **NewLookupRow**가 1이면 삽입 행을 대상에 전달할 필요가 없습니다.

조회 변환에서 삽입 기타 항목 업데이트 속성을 구성하고 소스 행이 삽입 행이면 통합 서비스가 행을 캐시 및 조회 소스에 삽입하거나 캐시를 업데이트합니다.

조회 변환에서 업데이트 기타 항목 삽입 속성을 구성하고 소스 행이 업데이트 행이면 통합 서비스가 캐시를 업데이트하거나 행을 캐시 및 조회 소스에 삽입합니다.

참고: 여러 세션에서 동일 테이블의 업데이트 행을 처리하는 경우 처리 순서가 다릅니다. 이 경우 예기치 않은 결과가 나타날 수 있습니다.

동적 캐시 동기화 구성

조회 변환을 작성할 때 동적 캐시를 삽입 행의 조회 소스에 동기화하도록 이 변환을 구성할 수 있습니다.

동적 캐시 동기화를 구성하려면:

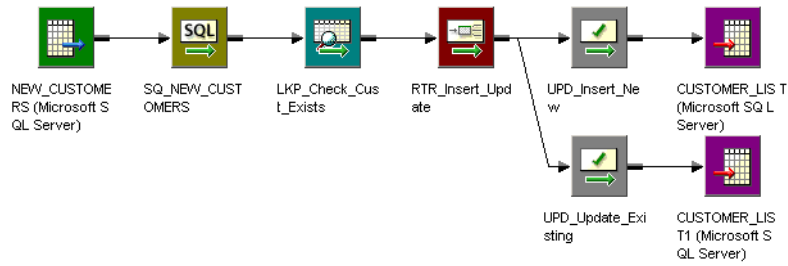
1. 조회 변환 속성 탭을 엽니다.
2. 동적 조회 캐시를 활성화합니다.
3. 동적 캐시 동기화를 활성화합니다.

동적 조회 캐시 예제

대상에서 행을 삽입 및 업데이트해야 하는 경우 동적 조회 캐시를 사용합니다. 동적 조회 캐시를 사용하는 경우 삽입 및 업데이트하기 위해 대상에 전달하는 동일한 데이터를 사용하여 캐시를 삽입 및 업데이트할 수 있습니다.

예를 들어 고객 데이터가 포함된 테이블을 업데이트해야 합니다. 소스 데이터에는 대상에 삽입하거나 업데이트할 고객 데이터의 행이 포함됩니다. 대상을 나타내는 동적 캐시를 작성합니다. 이 캐시의 고객을 조회하도록 조회 변환을 구성합니다.

다음 그림에는 동적 캐시가 있는 매핑이 나와 있습니다.



조회 변환은 동적 조회 캐시를 사용합니다. 세션을 시작하면 통합 서비스가 **Customer_List** 테이블에서 조회 캐시를 작성합니다. **Customer_List** 테이블은 매핑의 대상이기도 합니다. 통합 서비스가 조회 캐시에 없는 행을 읽으면 해당 행을 캐시에 삽입합니다.

조회 변환은 행을 라우터 변환에 반환합니다. 라우터 변환은 행을 **UPD_Insert_New** 또는 **UPD_Update_Existing** 변환에 전달합니다. 업데이트 전략 변환은 행을 삽입 또는 업데이트로 표시한 다음 대상에 전달합니다.

세션을 실행하면 **Customer_List** 테이블이 변경됩니다. 통합 서비스가 조회 캐시에 새 행을 삽입하고 기존 행을 업데이트합니다. 통합 서비스는 조회 캐시와 **Customer_List** 테이블의 동기화 상태를 유지합니다.

대상에 대한 키를 생성하려면 연결된 포트의 **Sequence-ID**를 사용합니다. 시퀀스 ID는 통합 서비스가 대상 테이블에 삽입하는 새 행에 대한 기본 키를 생성합니다.

동적 조회 캐시를 사용하면 데이터베이스에서 캐시를 한 번만 작성하므로 세션 성능이 개선됩니다. 대상 테이블의 데이터가 변경되어도 조회 캐시를 계속 사용할 수 있습니다.

동적 조회 캐시에 대한 규칙 및 지침

동적 조회 캐시를 사용할 때 특정 규칙 및 지침이 적용됩니다.

동적 조회 캐시를 사용하는 경우 다음 지침을 고려하십시오.

- 동일한 대상 로드 순서 그룹에서 동적 조회 변환과 정적 조회 변환 간에 캐시를 공유할 수 없습니다.
- 관계형, 플랫 파일 또는 소스 한정자 변환 조회의 경우 동적 조회 캐시를 활성화할 수 있습니다.
- 조회 변환은 연결된 변환이어야 합니다.
- 지속형 또는 비지속형 캐시를 사용할 수 있습니다.
- 동적 캐시가 비지속형인 경우 **조회 소스에서 다시 캐시**를 활성화하지 않는 경우에도 통합 서비스가 항상 데이터베이스에서 캐시를 재작성합니다.
- 동적 캐시 파일을 조회 소스 테이블과 동기화할 때 조회 변환이 행을 조회 소스 테이블 및 동적 조회 캐시에 삽입합니다. 소스 행이 업데이트 행인 경우 조회 변환이 동적 조회 캐시만 업데이트합니다.
- 같은 조회 조건만 작성할 수 있습니다. 동적 캐시에서는 데이터 범위를 조회할 수 없습니다.
- 조회 조건에 없는 각 조회 포트는 입력 포트, 시퀀스 ID 또는 연결된 식과 연결해야 합니다.
- NewLookupRow** 값이 1 또는 2일 경우 행을 캐시된 대상에 전달하는 라우터 변환을 사용합니다.
- NewLookupRow** 값이 0일 경우 행을 삭제하는 라우터 변환을 사용합니다. 그렇지 않으면 행을 다른 대상에 출력할 수 있습니다.
- 통합 서비스가 동일한 값을 조회 캐시에 쓰는 대상에 출력하는지 확인합니다. 업데이트 시 새 값을 출력하기로 선택하는 경우 입력/출력 포트 대신 조회/출력 포트만 대상 테이블에 연결합니다. 업데이트 시 이전 값을

출력하도록 선택한 경우 조회 변환 후 그리고 라우터 변환 전에 식 변환을 추가하십시오. 대상 테이블의 각 포트마다 식 변환에서 출력 포트를 추가하고 대상에 Null 입력 값을 출력하지 않도록 확인하는 식을 작성하십시오.

- 조회 SQL 재정의를 사용하는 경우 조회를 위해 적절한 대상에 올바른 열을 매핑합니다.
- WHERE 절을 조회 SQL 재정의에 추가하는 경우 조회 변환 전에 필터 변환을 사용합니다. 이렇게 하면 통합 서비스가 WHERE 절에 일치하는 동적 캐시 및 대상 테이블에 행을 삽입합니다.
- 동적 캐시를 사용하도록 재사용 가능 조회 변환을 구성하는 경우 매핑에서 **동적 조회 캐시** 속성을 비활성화하거나 조건을 편집할 수 없습니다.
- 조회 변환 후 업데이트 전략 변환을 사용하여 대상의 삽입 또는 업데이트를 위해 행에 플래그를 지정합니다.
- 조회 변환에서 업데이트 기타 항목 삽입 속성을 사용하려면 조회 변환 전에 업데이트 전략 변환을 사용하여 일부 또는 모든 행을 업데이트로 정의합니다.
- 세션 속성에서 행 유형을 데이터 구동으로 설정합니다.
- 세션 속성에서 대상 테이블 옵션에 대해 삽입 및 업데이트 시 업데이트를 선택합니다.

제 20 장

노멀라이저 변환

이 장에 포함된 항목:

- [노멀라이저 변환 개요, 321](#)
- [노멀라이저 변환 구성 요소, 322](#)
- [노멀라이저 변환 생성된 키, 325](#)
- [VSAM 노멀라이저 변환, 325](#)
- [파이프라인 노멀라이저 변환, 330](#)
- [매핑에서 노멀라이저 변환 사용, 333](#)
- [노멀라이저 변환 문제 해결, 336](#)

노멀라이저 변환 개요

노멀라이저 변환은 여러 번 발생하는 열이 포함된 행을 받고 여러 번 발생하는 데이터의 각 인스턴스마다 행을 반환합니다. 변환에서는 각 소스 행에서 다중 발생 열 또는 열의 다중 발생 그룹을 처리합니다. 노멀라이저 변환은 활성 변환입니다.

노멀라이저 변환은 COBOL 소스, 관계형 테이블 또는 그 밖의 소스에서 여러 번 발생하는 열을 구문 분석합니다. 이 변환은 REDEFINES 절을 포함하는 COBOL 소스의 여러 레코드 유형을 처리할 수 있습니다.

관계형 테이블에 매장별 4개의 판매 분기가 저장되는 경우를 예로 들어 보겠습니다. 이 경우 발생하는 각각의 판매에 대해 행을 작성해야 합니다. 분기별로 별개의 행을 반환하도록 노멀라이저 변환을 구성할 수 있습니다.

다음 소스 행에는 매장별 4개 분기 판매가 있습니다.

```
Store1 100 300 500 700
Store2 250 450 650 850
```

노멀라이저는 각 매장 및 판매 조합에 대해 행을 반환합니다. 또한 분기 번호를 식별하는 인덱스도 반환합니다.

```
Store1 100 1
Store1 300 2
Store1 500 3
Store1 700 4
Store2 250 1
Store2 450 2
Store2 650 3
Store2 850 4
```

노멀라이저 변환은 각 소스 행에 대해 키를 생성합니다. 통합 서비스는 소스 행을 처리할 때마다 생성된 키 시퀀스 번호를 늘립니다. 소스 행에 여러 번 발생하는 열 또는 열 그룹이 있을 경우 노멀라이저 변환은 각 발생에 대해 행을 반환합니다. 각 행에는 동일한 생성된 키 값이 포함됩니다.

노멀라이저가 한 소스 행에서 여러 행을 반환하는 경우 단일 발생 소스 열에 대해 중복 데이터를 반환합니다. 예를 들어 **Store1**과 **Store2**는 각 판매 인스턴스에 대해 반복됩니다.

VSAM 노멀라이저 변환 또는 **파이프라인 노멀라이저 변환**을 작성할 수 있습니다.

- **VSAM 노멀라이저 변환.** COBOL 소스의 소스 한정자 변환인 재사용 불가능 변환입니다. 매핑 디자이너가 매핑에서 COBOL 소스의 VSAM 노멀라이저 열을 작성합니다. 열 특성은 읽기 전용입니다. VSAM 노멀라이저는 하나의 입력 포트를 통해 여러 번 발생하는 소스 열을 받습니다.
- **파이프라인 노멀라이저 변환.** 관계형 테이블 또는 플랫폼 파일에서 여러 번 발생하는 데이터를 처리하는 변환입니다. 변환 개발자 또는 매핑 디자이너에서 수동으로 이러한 열을 작성하고 편집할 수 있습니다. 파이프라인 노멀라이저 변환은 각 소스 열 발생에 대해 하나의 입력 포트를 사용하여 여러 번 발생하는 열을 나타냅니다.

노멀라이저 변환 구성 요소

노멀라이저 변환에는 다음과 같은 탭이 있습니다.

- **변환.** 변환 이름 및 설명을 입력합니다. 노멀라이저 변환의 이름 지정 규칙은 **NRM_TransformationName**입니다. 파이프라인 노멀라이저 변환을 재사용 가능하게 만들 수도 있습니다.
- **포트.** 변환 포트 및 특성을 봅니다.
- **속성.** 추적 수준을 구성하여 세션 로그 파일에서 보고되는 트랜잭션 세부 정보의 양을 결정합니다. 다음 세션에서 생성된 키 시퀀스 값을 재설정하거나 다시 시작하려면 선택합니다.
- **노멀라이저.** 소스 데이터의 구조를 정의합니다. 노멀라이저 탭은 소스 데이터를 열 및 열 그룹으로 정의합니다.
- **메타데이터 확장.** 확장 이름, 데이터 유형, 전체 자릿수 및 값을 구성합니다. 또한 재사용 가능 메타데이터 확장을 작성할 수 있습니다.

포트 탭

노멀라이저 변환을 정의하는 경우 노멀라이저 탭에서 열을 구성합니다. 디자이너가 포트를 작성합니다. 포트 탭에서 노멀라이저 포트 및 특성을 볼 수 있습니다.

파이프라인 변환과 VSAM 노멀라이저 변환에서 다중 발생 소스 열이 서로 다르게 표시됩니다. VSAM 노멀라이저 변환에는 다중 발생 열에 대해 입력 포트가 하나만 표시됩니다. 파이프라인 노멀라이저 변환에는 다중 발생 열에 대해 입력 포트 여러 개가 표시됩니다.

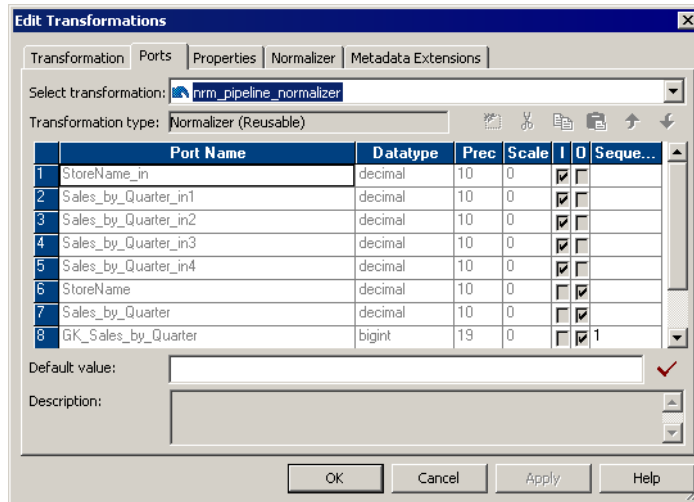
노멀라이저 변환에는 한 번 발생하는 입력 포트마다 출력 포트가 하나씩 표시됩니다. 소스 열이 다중 발생인 경우 파이프라인 및 VSAM 노멀라이저 변환에는 해당 열에 대해 출력 포트가 하나만 표시됩니다. 변환은 각 소스 열이 발생할 때마다 행을 반환합니다.

노멀라이저 변환에는 각 다중 발생 열에 대해 **GCID**(생성된 열 ID) 포트가 있습니다. 생성된 열 ID는 다중 발생 데이터의 인스턴스에 대한 인덱스입니다. 예를 들어 소스 레코드에서 열이 4번 발생하는 경우 노멀라이저는 다중 발생 데이터가 행에서 발생하는 인스턴스에 따라 생성된 열 ID에 값 1, 2, 3 또는 4를 반환합니다.

노멀라이저의 생성된 열 ID에 대한 이름 지정 규칙은 **GCID_<발생한_필드_이름>**입니다.

노멀라이저 변환에는 생성된 키 포트가 하나 이상 있습니다. 통합 서비스는 소스 행을 처리할 때마다 생성된 키 시퀀스 번호를 늘립니다.

다음 그림에서는 노멀라이저 변환 포트 탭을 보여 줍니다.



이 예에서 **Sales_By_Quarter**가 소스에서 여러 번 발생합니다. 노멀라이저 변환에는 **Sales_By_Quarter**에 대해 출력 포트 하나가 있습니다. 이 포트는 각 소스 행에 대해 네 개의 행을 반환합니다. 생성된 키 시작 값은 1입니다.

노멀라이저 탭에서 열을 편집하여 파이프라인 노멀라이저 변환의 포트를 변경할 수 있습니다. **VSAM** 노멀라이저 변환을 변경하려면 **COBOL** 소스를 변경하고 변환을 다시 작성해야 합니다.

속성 탭

속성 탭에서 노멀라이저 변환 일반 속성을 구성합니다.

다음과 같은 노멀라이저 변환 속성을 구성할 수 있습니다.

속성	필수/ 선택 사항	설명
재설정	필수	세션이 끝나면 생성된 키 값 각각의 값 시퀀스를 세션 이전의 값으로 재설정합니다.
다시 시작	필수	생성된 키 시퀀스를 1부터 시작합니다. 세션을 실행할 때마다 키 시퀀스 값은 1부터 시작되며 포트 탭의 시퀀스 값은 무시됩니다.
추적 수준	필수	이 변환을 포함하는 세션을 실행할 때 세션 로그에 포함되는 세부 정보의 양을 설정합니다.

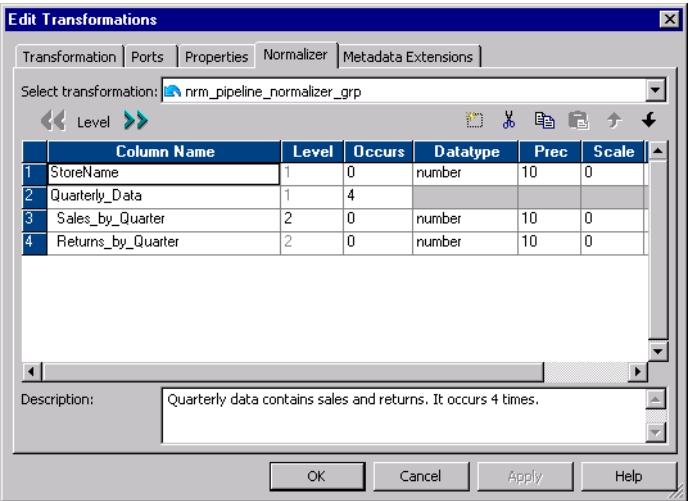
노멀라이저 탭

노멀라이저 탭에서는 소스 데이터의 구조를 정의합니다. 노멀라이저 탭은 소스 데이터를 열 및 열 그룹으로 정의합니다. 열 그룹은 **COBOL** 소스의 레코드를 정의하거나 소스에서 여러 번 발생하는 필드의 그룹을 정의할 수 있습니다.

열 수준 번호는 데이터에서 열 그룹을 식별합니다. 수준 번호는 데이터 계층을 정의합니다. 그룹의 열은 동일한 수준 번호를 가지며 그룹 수준 열 아래에 순차적으로 표시됩니다. 그룹 수준 열은 낮은 수준 번호를 가지며 데이터를 포함하지 않습니다.

다음 그림은 파이프라인 노멀라이저 변환의 노멀라이저 탭을 보여 줍니다.

수치 3. 노멀라이저 탭



Quarterly_Data는 그룹 수준 열이며 수준 1입니다. Quarterly_Data 그룹은 각 행에서 4번 발생합니다. Sales_by_Quarter 및 Returns_by_Quarter는 수준 2 열이며 그룹에 속합니다.

각 열에는 발생 수 특성이 있습니다. 발생 수 특성은 소스 행에서 두 번 이상 발생하는 열 또는 열 그룹을 식별합니다.

파이프라인 노멀라이저 변환을 작성하는 경우 열을 편집할 수 있습니다. VSAM 노멀라이저 변환을 작성하는 경우 노멀라이저 탭은 읽기 전용입니다.

다음 테이블에는 VSAM 및 파이프라인 노멀라이저 변환에 공통인 노멀라이저 탭 특성이 설명되어 있습니다.

특성	설명
열 이름	소스 열의 이름입니다.
수준	열을 그룹화합니다. 같은 그룹의 열은 열 아래에 하위 수준 번호가 지정되어 나타납니다. 각 열의 수준이 같으면 변환에는 열 그룹이 포함되지 않습니다
발생	소스 행의 열이나 열 그룹 인스턴스 수입니다.
데이터 유형	변환 열 데이터 유형은 문자열, Nstring 또는 숫자입니다.
Prec	전체 자릿수입니다. 열의 길이입니다.
배율	숫자 열의 소수점 위치 수입니다.

VSAM 노멀라이저 변환의 노멀라이저 탭에는 파이프라인 노멀라이저 변환과 동일한 특성이 있지만 COBOL 소스 정의에 고유한 특성도 있습니다.

노멀라이저 변환 생성된 키

노멀라이저 변환은 출력 행에서 하나 이상의 생성된 키 열을 반환합니다. 통합 서비스는 소스 행을 처리할 때마다 생성된 키 시퀀스 번호를 늘립니다. 통합 서비스는 노멀라이저 변환의 포트 탭에서 생성된 키 값을 기반으로 초기 키 값을 결정합니다. 노멀라이저 변환을 작성하는 경우 생성된 키 값은 기본적으로 1입니다. 노멀라이저 생성된 키의 이름 지정 규칙은 **GK_<제정의_필드_이름>**입니다.

관련 항목:

- [“키 값 생성” 페이지 335](#)

생성된 키 값 저장

현재 생성된 키 값을 노멀라이저 변환 포트 탭에서 볼 수 있습니다. 각 세션이 종료될 때 통합 서비스는 노멀라이저 변환의 생성된 키 값을 해당 세션에 대해 생성된 마지막 값 + 1로 업데이트합니다. 생성된 키의 최대값은 9,223,372,036,854,775,807입니다. 리포지토리에 노멀라이저 변환의 인스턴스가 여러 개 있는 경우, 통합 서비스는 세션을 실행할 때 모든 버전에서 생성된 키 값을 업데이트합니다.

참고: 현재 생성된 키 값을 노멀라이저 변환 포트 탭에서 변경할 수는 없습니다.

생성된 키 값 변경

다음과 같은 방법으로 생성된 키 값을 변경할 수 있습니다.

- **생성된 키 시퀀스 재설정.** 노멀라이저 변환 속성 탭에서 생성된 키 시퀀스를 재설정합니다. 생성된 키 값을 재설정하면 통합 서비스가 생성된 키 시작 값을 세션 시작 전 값으로 다시 재설정합니다. 세션을 실행할 때마다 동일한 생성된 키 값을 작성하려면 생성된 키 시퀀스를 재설정합니다.
- **생성된 키 시퀀스를 다시 시작.** 노멀라이저 변환 속성 탭에서 생성된 키 시퀀스를 다시 시작합니다. 생성된 키 시퀀스를 다시 시작하면 통합 서비스가 다음 번에 세션을 실행할 때 1에서 생성된 키 시퀀스를 시작합니다. 생성된 키 시퀀스를 다시 시작하면 세션을 다시 실행할 때까지 생성된 키 시작 값이 노멀라이저 변환에서 변경되지 않습니다. 세션을 실행하면 통합 서비스가 포트 탭에서 시퀀스 번호 값을 재정의합니다.

생성된 키 시퀀스를 재설정하거나 다시 시작하면 다음 번에 세션을 실행할 때 재설정 또는 다시 시작 결과가 생성된 키 시퀀스 값에 적용됩니다. 노멀라이저 변환에서 현재 생성된 키 시퀀스 값을 변경할 수 없습니다. 생성된 키 시퀀스를 재설정하거나 다시 시작하면 이 옵션을 비활성화할 때까지 모든 세션에서 옵션이 활성화됩니다.

VSAM 노멀라이저 변환

VSAM 노멀라이저 변환은 COBOL 소스 정의에 대한 소스 한정자입니다. COBOL 소스는 여러 번 발생하는 데이터와 여러 유형의 레코드를 동일한 파일에 포함할 수 있는 플랫폼 파일입니다.

VSAM(Virtual Storage Access Method)은 IBM 메인프레임 운영 체제의 파일 액세스 방법입니다. VSAM 파일은 인덱싱된 파일이나 순차적 플랫폼 파일로 레코드를 구성합니다. 하지만 COBOL 소스 정의로 정의한 모든 플랫폼 파일 소스에 대해 VSAM 노멀라이저 변환을 사용할 수 있습니다.

COBOL 소스 정의에는 여러 번 발생하는 열을 정의하는 OCCURS 문이 있을 수 있습니다. 또한 COBOL 소스 정의에는 파일에서 둘 이상의 레코드 유형을 정의하는 REDEFINES 문이 포함될 수 있습니다.

다음 COBOL copybook은 판매 레코드를 정의합니다.

```
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).

```

판매 파일은 두 유형의 판매 레코드를 포함할 수 있습니다. Store_Data는 매장을 정의하고 Detail_Data는 매장에서 판매된 상품을 정의합니다. REDEFINES 절은 레코드에서 Store_Data 필드 대신 Detail_Data 필드가 발생할 수 있음을 나타냅니다.

각 판매 레코드의 첫 세 문자는 헤더입니다. 헤더에는 레코드 유형과 매장 ID가 포함됩니다. Hdr_Rec_Type 값은 나머지 레코드가 매장 정보를 포함하는지, 아니면 상품 정보를 포함하는지를 정의합니다. 예를 들어 Hdr_Rec_Type이 "S"이면 레코드에 매장 데이터가 포함되어 있습니다. Hdr_Rec_Type이 "D"이면 레코드에 세부 데이터가 포함되어 있습니다.

레코드에 세부 데이터가 포함된 경우 Supplier_Info 필드가 포함됩니다. OCCURS 절은 각 Detail_Data 레코드에서 공급업체 네 개를 정의합니다.

다음 그림에서는 COBOL copybook에서 작성할 수 있는 Sales_File COBOL 소스 정의를 보여 줍니다.

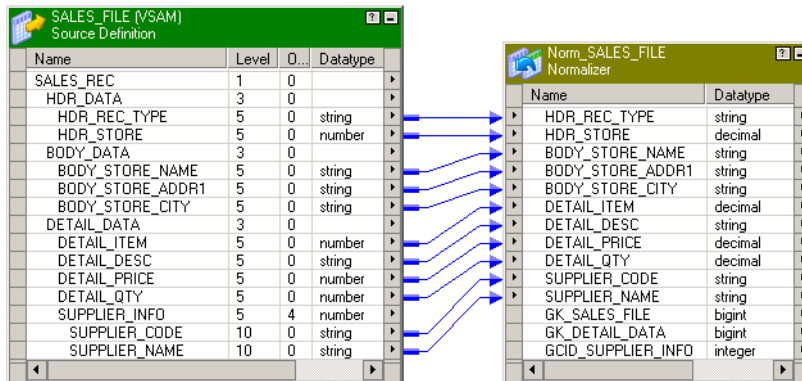
SALES_FILE (VSAM)					
K	Name	Level	Occurs	Datatype	Length
	SALES_REC	1	0		0
	HDR_DATA	3	0		0
	HDR_REC_TYPE	5	0	string	1
	HDR_STORE	5	0	number	2
	STORE_DATA	3	0		0
	STORE_NAME	5	0	string	30
	STORE_ADDR1	5	0	string	30
	STORE_CITY	5	0	string	30
	DETAIL_DATA	3	0		0
	DETAIL_ITEM	5	0	number	9
	DETAIL_DESC	5	0	string	30
	DETAIL_PRICE	5	0	number	6
	DETAIL_QTY	5	0	number	5
	SUPPLIER_INFO	5	4	number	0
	SUPPLIER_CODE	10	0	string	2
	SUPPLIER_NAME	10	0	string	8

Sales_Rec, Hdr_Data, Store_Data, Detail_Data 및 Supplier_Info 열은 하위 수준 데이터의 그룹을 식별하는 그룹 수준 열입니다. 그룹 수준 열은 데이터를 포함하지 않기 때문에 길이가 0입니다. 소스 정의에서 이러한 열에는 출력 포트가 없습니다.

Supplier_Info 그룹에는 Supplier_Code 및 Supplier_Name 열이 포함되어 있습니다. Supplier_Info 그룹은 각 Detail_Data 레코드에서 네 번 발생합니다.

COBOL 소스 정의에서 VSAM 노멀라이저 변환을 작성하면 COBOL 소스 정의를 기준으로 매핑 디자이너가 노멀라이저 변환의 입력/출력 포트를 작성합니다. 노멀라이저 변환에는 생성된 키 출력 포트가 하나 이상 포함되어 있습니다. COBOL 소스에 여러 번 발생하는 열이 있는 경우 노멀라이저 변환에는 생성된 열 ID 출력 포트가 있습니다.

다음 그림에서는 매핑 디자이너가 소스 정의에서 작성한 노멀라이저 변환 포트를 보여 줍니다.



Supplier_Info 열 그룹은 각 COBOL 소스 행에서 네 번 발생합니다.

COBOL 소스 행에는 다음 데이터가 포함될 수 있습니다.

```
Item1 ItemDesc 100 25 A Supplier1 B Supplier2 C Supplier3 D Supplier4
```

노멀라이저 변환은 Supplier_Code 및 Supplier_Name 열이 발생할 때마다 행을 반환합니다. 각 출력 행은 동일한 항목, 설명, 가격 및 수량 값을 포함합니다.

노멀라이저는 COBOL 소스 행에서 다음 세부 데이터 행을 반환합니다.

```
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
```

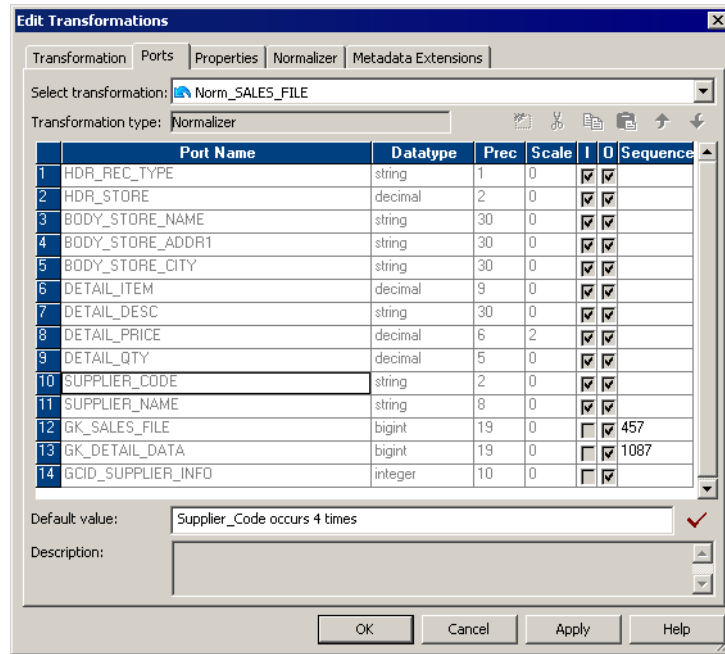
각 출력 행에는 생성된 키와 열 ID가 포함됩니다. 통합 서비스는 새 소스 행을 처리할 경우 생성된 키 값을 업데이트합니다. 세부 데이터 행에서 생성된 키 값이 1입니다.

열 ID는 Supplier_Info 열 발생 수를 정의합니다. 통합 서비스는 Supplier_Info가 발생할 때마다 열 ID를 업데이트합니다. 세부 데이터 행에서 열 ID 값은 1, 2, 3, 4입니다.

VSAM 노멀라이저 포트 탭

VSAM 노멀라이저 포트 탭에는 변환 입력 및 출력 포트가 표시됩니다. 각 COBOL 소스 열마다 입력/출력 포트가 하나씩 표시되고, 다중 발생 열인 경우 입력/출력 포트가 하나만 표시됩니다. 이 변환에서는 그룹 수준 열에 대한 입력 또는 출력 포트가 없습니다.

다음 그림에서는 VSAM 노멀라이저 포트 탭을 보여 줍니다.



이 예에서 Supplier_Code 및 Supplier_Name은 COBOL 소스에서 네 번 발생합니다. 포트 탭에는 Supplier_Code 포트 하나와 Supplier_Name 포트 하나가 표시됩니다. 생성된 키 시작 값은 457 및 1087입니다.

VSAM 노멀라이저 탭

VSAM 노멀라이저 변환을 작성하면 매핑 디자이너가 COBOL 소스로부터 열을 작성합니다. 노멀라이저 탭에는 COBOL 소스 정의와 동일한 정보가 표시됩니다. VSAM 노멀라이저 탭에 있는 열을 편집할 수는 없습니다.

다음 테이블에는 VSAM 노멀라이저 탭에 있는 특성이 설명되어 있습니다.

특성	설명
POffs	실제 오프셋입니다. 파일에서의 필드 위치입니다. 파일에서 첫 번째 바이트는 0입니다.
Plen	실제 길이입니다. 필드의 바이트 수입니다.
열 이름	소스 필드의 이름입니다.
수준	열 그룹 계층을 제공합니다. 수준 번호가 높을수록 계층에서 데이터의 수준은 낮습니다. 같은 그룹의 열은 열 아래에 하위 수준 번호가 지정되어 나타납니다. 각 열의 수준이 같으면 변환에는 열 그룹이 포함되지 않습니다
발생	소스 행의 열이나 열 그룹 인스턴스 수입니다.
데이터 유형	변환 데이터 유형은 문자열, Nstring 또는 숫자일 수 있습니다.
Prec	전체 자릿수입니다. 열의 길이입니다.
배율	숫자 열의 소수점 위치 수입니다.

특성	설명
사진	데이터가 소스에서 저장되거나 표시되는 방식입니다. 사진 99V99는 암시적 십진 소수점 2개를 포함하는 숫자 필드를 정의합니다. 사진 X(10)은 문자 10개를 나타냅니다.
사용	COMP, BINARY 및 COMP-3 등의 COBOL 데이터 저장 형식입니다. 사용이 DISPLAY인 경우, 사진 절은 소스 데이터를 볼 때 소스 데이터에 어떤 형식이 적용되는지 정의합니다.
키 유형	VSAM 파일의 필드에 적용할 키 제약 조건의 유형입니다. 다음 키 유형 중 하나를 선택합니다. <ul style="list-style-type: none"> - 키가 아님. 필드가 VSAM 파일에서 인덱스가 아닙니다. - 기본 키. 필드가 VSAM 파일에서 기본 인덱스입니다. 필드는 고유한 값을 포함합니다. - 대체 키. 필드가 VSAM 파일에서 보조 인덱스입니다. 필드는 고유한 값을 포함합니다. - 기본 중복 키. 필드가 VSAM 파일에서 기본 인덱스입니다. 필드가 중복 값을 포함할 수 있습니다. - 대체 중복 키. 필드가 VSAM 파일에서 보조 인덱스입니다. 필드가 중복 값을 포함할 수 있습니다.
서명됨(S)	숫자 값에 부호가 있는지 여부를 나타냅니다.
후행 기호(T)	필드의 마지막 자리에 부호(+ 또는 -)가 있음을 나타냅니다. 활성화되지 않은 경우, 부호가 필드에서 첫 번째 문자로 나타납니다.
포함 기호(I)	필드에 나타나는 모든 값에 기호가 포함되는지 여부를 나타냅니다.
실제 소수점(R)	소수점이 마침표(.)인지 또는 소수점이 숫자 필드에서 V 문자로 표현되는지를 나타냅니다.
재정의	열이 다른 열을 재정의함(REDEFINES)을 나타냅니다.
비즈니스 이름	설명이 포함된 이름이며 열에 지정됩니다.

VSAM 노멀라이저 변환 작성 단계

VSAM 노멀라이저 변환을 작성할 때 COBOL 소스를 매핑으로 끌어 오면 매핑 디자이너가 해당 소스로부터 변환 열을 작성합니다. 노멀라이저 변환은 매핑에서 COBOL 소스에 대한 소스 한정자입니다.

COBOL 소스를 매핑에 추가하면 매핑 디자이너가 노멀라이저 변환을 작성하고 구성합니다. 매핑 디자이너는 COBOL 소스에서 중첩된 레코드 및 여러 번 발생하는 필드를 식별합니다. 매핑 디자이너는 소스 열을 기반으로 노멀라이저 변환의 열 및 포트를 작성합니다.

VSAM 노멀라이저 변환을 작성하려면 다음을 수행하십시오.

1. 매핑 디자이너에서 새 매핑을 작성하거나 기존 매핑을 엽니다.
2. COBOL 소스 정의를 매핑으로 끌어 옵니다.

디자이너가 노멀라이저 변환을 추가하고 이 변환을 COBOL 소스 정의에 연결합니다. 소스 한정자를 기본적으로 작성하는 옵션을 활성화하지 않은 경우, 노멀라이저 변환 작성 대화 상자가 나타납니다.

3. 노멀라이저 변환 작성 대화 상자가 나타나면 다음 옵션 중에서 선택할 수 있습니다.

- **VSAM 소스.** 매핑에 있는 COBOL 소스 정의로부터 변환을 작성합니다.
- **파이프라인.** 변환을 작성하지만, COBOL 소스로부터 열을 정의하지는 않습니다. 노멀라이저 탭에서 수동으로 열을 정의하십시오. 매핑에 있는 다른 변환의 여러 번 발생하는 데이터를 처리하려는 경우 이 옵션을 선택할 수 있습니다.

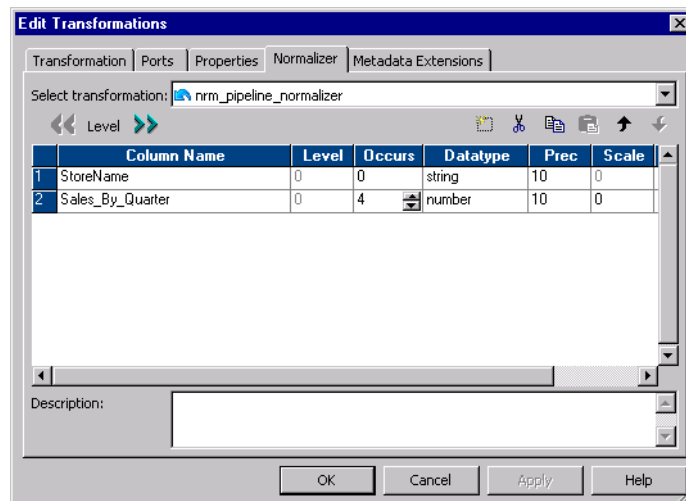
VSAM 노멀라이저 변환을 작성하려면 VSAM 노멀라이저 변환 옵션을 선택합니다. 매핑에 있는 COBOL 소스 정의의 이름이 대화 상자에 표시됩니다. COBOL 소스 정의를 선택하고 확인을 클릭합니다.

4. 노멀라이저 변환을 엽니다.

5. 포트 탭을 선택하여 노멀라이저 변환의 포트를 확인합니다.
디자이너는 기본적으로 COBOL 소스 정의로부터 포트를 작성합니다.
6. 노멀라이저 탭을 클릭하여 소스 열 구성을 검토합니다.
노멀라이저 탭에는 COBOL 소스의 열 탭과 동일한 정보가 포함됩니다. 그러나 열 특성을 노멀라이저 변환에서 수정할 수는 없습니다. 열 특성을 변경하려면 COBOL copybook을 변경하고 COBOL 소스를 가져온 후, 노멀라이저 변환을 다시 작성합니다.
7. 속성 탭을 선택하여 추적 수준을 설정합니다.
다음 세션이 시작될 때 생성된 키 시퀀스 번호를 재설정하도록 변환을 구성할 수도 있습니다.

파이프라인 노멀라이저 변환

변환 개발자에서 노멀라이저 변환을 작성하면 기본적으로 파이프라인 노멀라이저 변환이 작성됩니다. 파이프라인 노멀라이저 변환을 작성할 경우, 이 변환이 소스 한정자 변환 같은 다른 유형의 변환으로부터 받는 데이터에 기반하여 열을 정의합니다. 디자이너는 정의한 열에 근거하여 입력 및 출력 노멀라이저 변환 포트를 작성합니다. 다음 그림은 각 관계형 소스 행에서 4개의 매출 열을 받는 변환에 대한 노멀라이저 변환 열을 보여 줍니다.



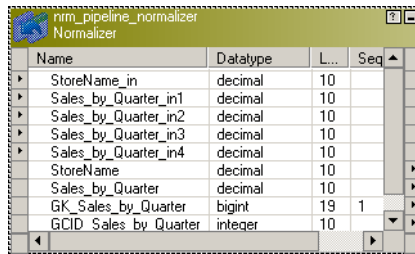
각 소스 행에는 StoreName 열 1개 및 Sales_By_Quarter 인스턴스 4개가 있습니다.

소스 행에는 다음과 같은 데이터가 포함될 수 있습니다.

```
Dellmark 100 450 650 780
Tonys    666 333 444 555
```

파이프라인 노멀라이저 변환에는 여러 번 발생하는 열의 각 인스턴스에 대한 입력 포트가 있습니다.

다음 그림은 디자이너가 노멀라이저 변환의 열에 근거하여 작성하는 포트를 보여 줍니다.



Name	Datatype	Length	Seq
StoreName_in	decimal	10	
Sales_by_Quarter_in1	decimal	10	
Sales_by_Quarter_in2	decimal	10	
Sales_by_Quarter_in3	decimal	10	
Sales_by_Quarter_in4	decimal	10	
StoreName	decimal	10	
Sales_by_Quarter	decimal	10	
GK_Sales_by_Quarter	bigint	19	1
GCID_Sales_by_Quarter	integer	10	

노멀라이저 변환은 여러 번 발생하는 열의 각 인스턴스에 대해 행을 1개 반환합니다.

```
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
```

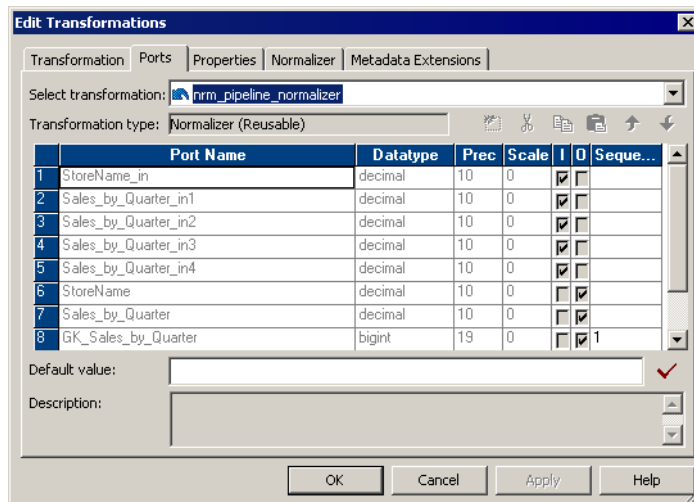
통합 서비스는 소스 행을 처리할 때마다 생성된 키 시퀀스 번호를 늘립니다. 생성되는 키는 각 분기 매출을 동일한 매장에 연결합니다. 이 예에서 **Dellmark** 행에 대해 생성되는 키는 1입니다. **Tonys** 매장에 대해 생성되는 키는 2입니다.

변환은 여러 번 발생하는 필드의 각 인스턴스에 대해 생성되는 열 ID(GCID)를 반환합니다. 이 예에서 **GCID_Sales_by_Quarter** 값은 항상 1, 2, 3 또는 4입니다.

파이프라인 노멀라이저 포트 탭

파이프라인 노멀라이저 포트 탭은 변환의 입력 및 출력 포트를 표시합니다. 이 탭에는 변환에서 정의한 각 단일 발생 열에 대해 입력/출력 포트가 1개 있습니다. 다중 발생 열의 경우 발생별로 포트가 1개씩 있습니다. 이 변환에서는 그룹 수준 열에 대한 입력 또는 출력 포트가 없습니다.

다음 그림은 파이프라인 노멀라이저 변환 포트 탭을 보여 줍니다.



Port Name	Datatype	Prec	Scale	I	O	Seque...
1 StoreName_in	decimal	10	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
2 Sales_by_Quarter_in1	decimal	10	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
3 Sales_by_Quarter_in2	decimal	10	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
4 Sales_by_Quarter_in3	decimal	10	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
5 Sales_by_Quarter_in4	decimal	10	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
6 StoreName	decimal	10	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
7 Sales_by_Quarter	decimal	10	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
8 GK_Sales_by_Quarter	bigint	19	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	1

Default value:

Description:

OK Cancel Apply Help

디자이너는 다중 발생 열의 각 발생에 대해 입력 포트를 작성합니다.

파이프라인 노멀라이저 변환의 포트를 변경하려면 노멀라이저 탭에서 열을 수정하십시오. 열 발생을 추가하면 디자이너에서 입력 포트를 추가합니다. 디자이너는 가장 낮은 수준의 열에 대해 포트를 작성합니다. 그룹 수준 열에 대해서는 포트를 작성하지 않습니다.

파이프라인 노멀라이저 탭

파이프라인 노멀라이저 변환을 작성할 때 노멀라이저 탭에서 열을 정의합니다. 디자이너는 노멀라이저 탭에서 사용자가 입력한 열에 따라 입력 및 출력 포트를 작성합니다.

다음 테이블에는 파이프라인 노멀라이저 탭 특성이 설명되어 있습니다.

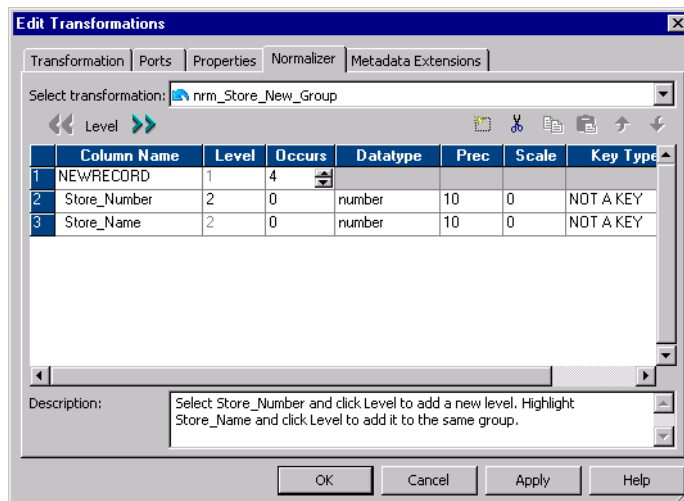
특성	설명
열 이름	열의 이름입니다.
수준	열 그룹을 식별합니다. 동일한 그룹에 있는 열은 수준 번호가 동일합니다. 기본값은 0입니다. 각 열의 수준이 같으면 변환에는 열 그룹이 포함되지 않습니다
발생	소스 행의 열이나 열 그룹 인스턴스 수입니다.
데이터 유형	열 데이터 유형은 문자열, Nstring 또는 숫자일 수 있습니다.
Prec	전체 자릿수입니다. 열의 길이입니다.
배율	숫자 값의 소수점 자릿수입니다.

노멀라이저 탭 열 그룹

소스 행에 반복되는 열이 있을 경우 노멀라이저 탭에서 열 그룹을 정의할 수 있습니다. 노멀라이저 변환은 각 열의 발생이 아닌 각 열 그룹의 발생에 대해 행을 반환합니다.

노멀라이저 탭의 수준 번호는 열의 계층 구조를 식별합니다. 그룹 수준 열은 열의 그룹을 식별합니다. 그룹 수준 열은 해당 그룹의 열보다 낮은 수준 번호를 갖습니다. 동일한 그룹의 열은 동일한 수준 번호를 갖고 노멀라이저 탭의 그룹 수준 열 아래 순차적으로 표시됩니다.

다음 그림은 노멀라이저 탭에서 여러 번 발생하는 열의 그룹을 보여 줍니다.



이 예제에서 NEWRECORD 열에는 데이터가 없습니다. 이 열은 수준 1 그룹 열입니다. 이 그룹은 각 소스 행에서 4번 발생합니다. Store_Number 및 Store_Name은 수준 2 열입니다. 이러한 열은 NEWRECORD 그룹에 속합니다.

파이프라인 노멀라이저 변환 작성 단계

파이프라인 노멀라이저 변환을 작성할 때 노멀라이저 탭에서 열을 정의합니다.

노멀라이저 변환은 변환 개발자 또는 매핑 디자이너에서 작성할 수 있습니다.

노멀라이저 변환을 작성하려면 다음을 수행하십시오.

1. 변환 개발자 또는 매핑 디자이너에서 변환 > 작성을 클릭합니다. 노멀라이저 변환을 선택합니다. 노멀라이저 변환 이름을 입력합니다.
노멀라이저 변환에 대한 명명 규칙은 `NRM_TransformationName`입니다.
2. 작성을 클릭하고 완료를 클릭합니다.
3. 노멀라이저 변환을 열고 노멀라이저 탭을 클릭합니다.
4. 추가를 클릭하여 새 열을 추가합니다.
디자이너가 기본 특성을 갖는 새 열을 작성합니다. 이름, 데이터 유형, 전체 자릿수 및 소수 자릿수를 변경할 수 있습니다.
5. 다중 발생 열을 작성하려면 발생 수 열에 발생 횟수를 입력합니다.
6. 여러 번 발생하는 열의 그룹을 작성하려면 1개 이상의 열을 노멀라이저 탭에 입력합니다. 열을 선택합니다. 수준을 클릭합니다.
디자이너가 **NEWRECORD** 그룹 수준 열을 선택된 열 위에 추가합니다. **NEWRECORD**가 수준 1이 됩니다. 선택한 열은 수준 2가 됩니다. **NEWRECORD** 열의 이름을 바꿀 수 있습니다.
기본적으로 모든 열은 수준이 동일합니다. 수준은 함께 그룹화되는 열을 정의합니다.
7. 다른 열의 열 수준을 변경하여 같은 그룹에 추가할 수 있습니다. 위에 있는 열과 동일한 수준으로 열을 변경하려면 해당 열을 선택하고 수준을 클릭합니다.
같은 그룹에 있는 열은 노멀라이저 탭에서 순차적으로 나타나야 합니다.
8. 그룹 수준에서 발생을 변경하여 열 그룹을 다중 발생으로 설정합니다.
9. 적용을 클릭하여 열을 저장하고 입력 및 출력 포트를 작성합니다.
디자이너가 노멀라이저 변환 입력 및 출력 포트를 작성합니다. 또한 디자이너는 생성되는 키 열을 작성하고, 여러 번 발생하는 각 열 또는 열의 그룹에 대한 열 ID를 작성합니다.
10. 속성 탭을 선택하여 추적 수준을 변경하거나, 다음 세션 이후에 생성되는 키 시퀀스 번호를 재설정합니다.

매핑에서 노멀라이저 변환 사용

노멀라이저 변환이 **COBOL** 소스로부터 두 가지 유형 이상의 데이터를 받는 경우, 노멀라이저 출력 포트를 각 행에 있는 데이터의 유형에 따라 서로 다른 대상에 연결해야 합니다. 다음 예제에서는 노멀라이저 변환을 통해 **Sales_File** **COBOL** 소스 정의를 여러 대상에 매핑하는 방법을 설명합니다.

Sales_File 소스 레코드는 매장 정보 또는 매장에서 판매하는 품목에 대한 정보를 포함합니다. 매출 파일에는 두 가지 유형의 레코드가 포함되어 있습니다.

두 가지 매출 파일 레코드가 다음 예에 나와 있습니다.

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 소스 정의 및 노멀라이저 변환에는 두 가지 유형의 레코드에 있는 필드를 나타내는 열이 있습니다. 품목 행에서 매장 행을 필터링하여 두 가지 행을 서로 다른 대상으로 전달해야 합니다.

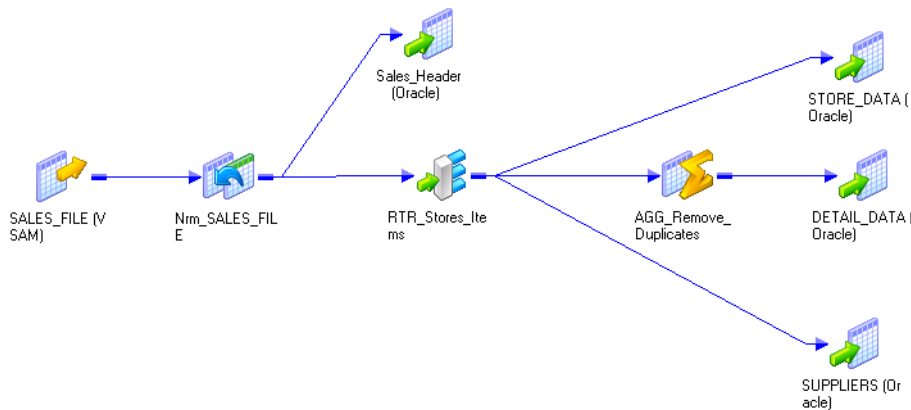
다음 그림은 상응하는 Store_Data(값이 “S”임) 및 Detail_Data(값이 “D”임)가 있는 Sales_File COBOL 소스를 보여 줍니다.

K	Name	Level	Occurs	Datatype	Length
	SALES_REC	1	0		0
	HDR_DATA	3	0		0
	HDR_REC_TYPE	5	0	string	1
	HDR_STORE	5	0	number	2
	STORE_DATA	3	0		0
	STORE_NAME	5	0	string	30
	STORE_ADDR1	5	0	string	30
	STORE_CITY	5	0	string	30
	DETAIL_DATA	3	0		0
	DETAIL_ITEM	5	0	number	9
	DETAIL_DESC	5	0	string	30
	DETAIL_PRICE	5	0	number	6
	DETAIL_QTY	5	0	number	5
	SUPPLIER_INFO	5	4	number	0
	SUPPLIER_CODE	10	0	string	2
	SUPPLIER_NAME	10	0	string	8

Hdr_Rec_Type은 레코드에 매장 정보 또는 상품 정보 중 어느 것이 포함되는지 정의합니다. Hdr_Rec_Type 값이 “S”이면 레코드에 Store_Data가 포함되고, Hdr_Rec_Type이 “D”이면 레코드에 Detail_Data가 포함됩니다. Detail_Data는 항상 Supplier_Info 필드의 발생 4개를 포함합니다.

데이터를 필터링하려면 노멀라이저 출력 행을 라우터 변환에 연결하여 매장, 품목 및 공급자 데이터를 서로 다른 대상으로 라우팅합니다. 라우터 변환에서 Hdr_Rec_Type의 값에 따라 행을 필터링할 수 있습니다.

다음 그림은 Sales_File 레코드를 서로 다른 대상으로 라우팅하는 매핑을 보여 줍니다.



이 매핑은 COBOL 소스에서 여러 관계형 대상으로 여러 가지 레코드 유형을 필터링합니다. 여러 번 발생하는 소스 열은 별도의 관계형 테이블에 매핑됩니다. 각 행은 소스 행에서의 발생을 기준으로 인덱싱됩니다.

이 매핑에는 다음 변환이 포함됩니다.

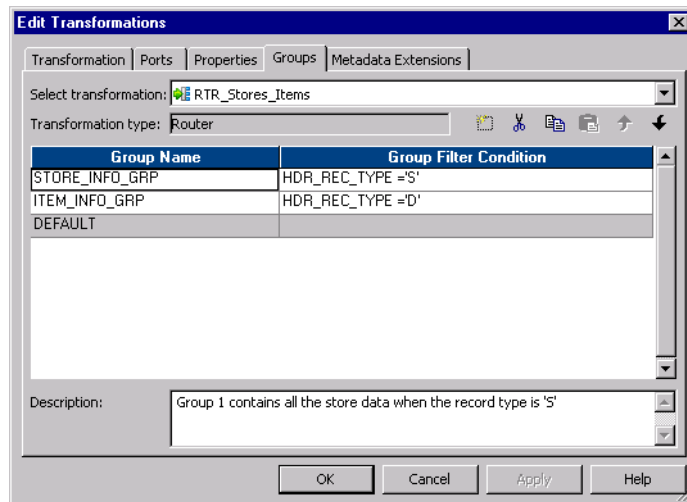
- **노멀라이저 변환.** 소스에 여러 번 발생하는 Detail_Data가 포함되어 있으면 노멀라이저 변환은 여러 행을 반환합니다. 또한 이 변환은 동일한 소스의 여러 레코드 유형을 처리합니다.
- **라우터 변환.** 라우터 변환은 Hdr_Rec_Type 값에 따라 여러 대상으로 데이터를 라우팅합니다.
- **집계 변환.** 집계 변환은 각 Supplier_Info 발생과 함께 발생하는 중복 Detail_Data 행을 제거합니다.

매핑에는 다음과 같은 기능이 포함됩니다.

1. 노멀라이저 변환이 헤더 레코드 유형 및 헤더 매장 번호 열을 Sales_Header 대상으로 전달합니다. 각 Sales_Header 레코드에는 Sales_Header 행을 Store_Data 또는 Detail_Data 대상 행에 연결하는 생성된 키가 있습니다. 노멀라이저는 Hdr_Data 및 Store_Data를 행마다 한 번씩 반환합니다.

2. 노멀라이저 변환이 모든 열을 라우터 변환으로 전달합니다. 이 변환은 **Detail_Data** 데이터를 행마다 4번씩 전달하고 **Supplier_Info** 열의 각 발생마다 1번씩 전달합니다. **Supplier_Info** 열을 제외한 **Detail_Data** 열은 중복 데이터를 포함합니다.
3. **Hdr_Rec_Type**이 “S”이면 라우터 변환이 매장 이름, 주소, 시 및 생성된 키를 **Store_Data**로 전달합니다. 생성된 키는 **Store_Data** 행을 **Sales_Header** 행에 연결합니다.
라우터 변환은 매장 데이터에 대한 사용자 정의 그룹 1개 및 상품 품목에 대한 사용자 정의 그룹 1개를 포함합니다.
4. **Hdr_Rec_Type**이 “D”이면 라우터 변환이 품목, 품목 설명, 가격, 수량 및 **Detail_Data** 생성된 키를 집계 변환으로 전달합니다.
5. **Hdr_Rec_Type**이 “D”이면 라우터 변환이 공급자 코드, 이름 및 열 ID를 공급자 대상으로 전달합니다. 이 변환은 **Suppliers** 행을 **Detail_Data** 행에 연결하는 생성된 키를 전달합니다.
6. 집계 변환이 중복 **Detail_Data** 열을 제거합니다. 집계는 품목, 설명, 가격, 수량 및 생성된 키의 인스턴스 1개를 **Detail_Data**로 전달합니다. **Detail_Data** 생성된 키는 **Detail_Data** 행을 **Suppliers** 행에 연결합니다. 또한 **Detail_Data**에는 **Detail_Data** 행을 **Sales_Header** 행에 연결하는 키가 있습니다.

다음 그림은 라우터 변환의 사용자 정의 그룹 및 필터 조건을 보여 줍니다.



라우터 변환은 레코드 유형에 따라 매장 데이터 또는 품목 데이터를 전달합니다.

키 값 생성

COBOL 소스에 다중 발생 열 그룹이 포함된 경우 노멀라이저 변환은 생성된 키를 작성합니다. 행의 기타 열이 아닌 다른 대상에 다중 발생 열 그룹을 전달할 수 있습니다. 생성된 키가 있는 대상 간에 기본-외래 키 관계를 작성할 수 있습니다.

다음 그림은 다중 발생 열 그룹이 포함된 COBOL 소스 정의를 보여 줍니다.

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

이 예에서 열의 **Detail_Suppliers** 그룹이 **Detail_Record**에서 네 번 발생합니다.

노멀라이저 변환은 각 소스 행에 대해 GK_Detail_Sales 키를 생성합니다. GK_Detail_Sales 키는 하나의 Detail_Record 소스 행을 나타냅니다.

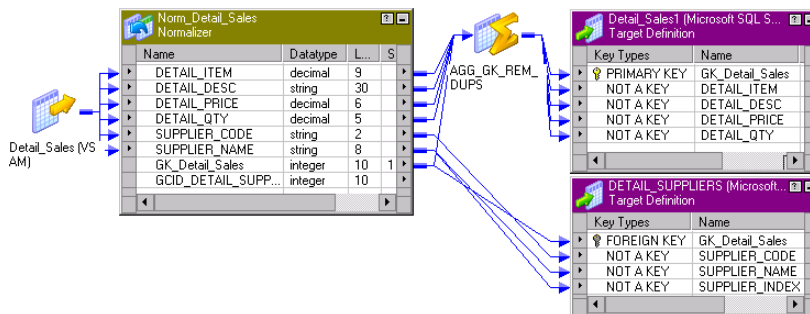
다음 그림은 대상 간 기본 외래 키 관계를 보여 줍니다.

Key Types	Name	Datatype	Length
FOREIGN KEY	GK_Detail_Sales	int	10
NOT A KEY	SUPPLIER_CODE	varchar	2
NOT A KEY	SUPPLIER_NAME	varchar	8

Key Types	Name	Datatype	Length
PRIMARY KEY	GK_Detail_Sales	int	10
NOT A KEY	DETAIL_ITEM	numeric	9
NOT A KEY	DETAIL_DESC	varchar	30
NOT A KEY	DETAIL_PRICE	numeric	6
NOT A KEY	DETAIL_QTY	numeric	5

다중 발생 Detail_Supplier 행에는 동일한 Detail_Sales 행에 이들을 연결하는 외래 키가 있습니다. Detail_Sales 대상에는 Detail_Suppliers 대상과 일대다 관계가 있습니다.

다음 그림은 대상의 기본 키 및 외래 키에 연결된 GK_Detail_Sales 생성된 키를 보여 줍니다.



GK_Detail_Sales를 Detail_Sales의 기본 키 및 Detail_Suppliers의 외래 키로 전달합니다.

노멀라이저 출력 열을 다음 개체에 연결합니다.

- **Detail_Sales_Target.** Detail_Item, Detail_Desc, Detail_Price 및 Detail_Qty 열을 Detail_Sales 대상으 로 전달합니다. GK_Detail_Sales 키를 Detail_Sales 기본 키로 전달합니다.
- **집계 변환.** 집계 변환을 통해 각 Detail_Sales 행을 전달하여 중복 행을 제거합니다. 노멀라이저는 Detail_Suppliers의 각 발생에 대한 중복 Detail_Sales 열을 반환합니다.
- **Detail_Suppliers.** Detail_Suppliers 열의 각 인스턴스를 Detail_Suppliers 대상에 전달합니다. GK_Detail_Sales 키를 Detail_Suppliers 외래 키에 전달합니다. Detail_Suppliers 열의 각 인스턴스에는 Detail_Suppliers 행을 Detail_Sales 행에 연결하는 외래 키가 있습니다.

노멀라이저 변환 문제 해결

관계형 소스를 사용할 경우 노멀라이저 변환에서 포트를 편집할 수 없습니다.

수동으로 포트를 작성할 때 변환의 포트 탭 대신 노멀라이저 탭에 포트를 추가하십시오.

COBOL 파일 가져오기가 실패하고 여러 오류가 발생했습니다. 어떻게 해야 하나요?

COBOL 프로그램이 공백, 탭 및 줄 바꿈 문자 등의 COBOL 표준을 따르는지 확인합니다. COBOL 파일의 위쪽 부분은 다음 텍스트와 비슷해야 합니다.

```
identification division.  
    program-id. mead.  
environment division.  
    select file-one assign to "fname".  
data division.  
file section.  
fd FILE-ONE.
```

디자이너는 COBOL 프로그램의 숨겨진 문자를 읽지 않습니다. 텍스트 전용 편집기를 사용하여 COBOL 파일을 변경하십시오. Word 또는 Wordpad는 사용하지 마십시오. 여분의 공백이 있으면 제거하십시오.

이진 데이터를 읽는 세션이 완료되었지만 대상 테이블에서 정보가 올바르지 않습니다.

워크플로우 관리자에서 세션을 편집하고 소스 파일 형식이 올바르게 설정되었는지 확인합니다. 파일 형식은 EBCDIC 또는 ASCII일 수 있습니다. 레코드 간에 건너뛰는 바이트 수는 0으로 설정되어야 합니다.

IBM COMP 이외의 유형을 사용하는 COBOL 필드 설명이 있습니다. 이 소스를 어떻게 가져와야 합니까?

소스 정의에서 IBM COMP 옵션을 선택 취소합니다.

매핑에서 식 변환 1개 및 조희 변환 1개를 사용하여 노멀라이저 변환의 출력 포트 2개를 수정합니다. 매핑이 출력 포트를 단일 변환에 연결하고, 모든 포트가 동일한 수준에 있습니다. 대상에서 로드된 데이터를 확인하면 데이터가 올바르지 않습니다. 그 이유가 무엇입니까?

수준 1의 포트만 연결할 수 있습니다. 연결을 제거하십시오.

제 21 장

순위 변환

이 장에 포함된 항목:

- [순위 변환 개요, 338](#)
- [순위 변환의 포트, 339](#)
- [그룹 정의, 340](#)
- [순위 변환 작성, 341](#)

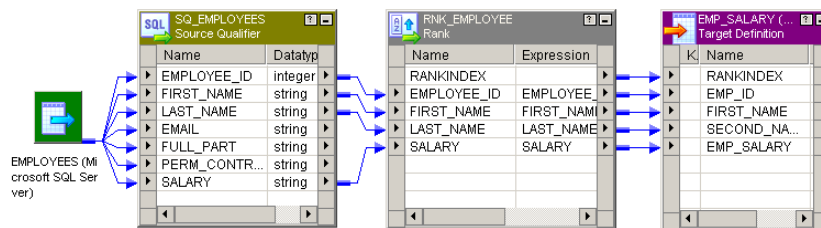
순위 변환 개요

순위 변환에서 상위 또는 하위 순위의 데이터만 선택할 수 있습니다. 순위 변환은 활성 변환입니다. 순위 변환을 사용하여 포트 또는 그룹에서 가장 크거나 가장 작은 숫자 값을 반환할 수 있습니다. 또한 순위 변환을 사용하여 세션 정렬 순서의 상위 또는 하위에 있는 문자열을 반환할 수 있습니다. 세션 중에는 통합 서비스가 순위 계산을 수행할 수 있을 때까지 입력 데이터를 캐싱합니다.

순위 변환은 값 하나만 선택할 수 있는 것이 아니라 상위 또는 하위 값의 그룹을 선택할 수 있다는 점에서 MAX 및 MIN 변환 함수와 다릅니다. 예를 들어 순위를 사용하여 지정된 지역에서 상위 10명의 판매자를 선택할 수 있습니다. 또는 재무 보고서를 생성하기 위해 순위 변환을 사용하여 급여 및 오버헤드에서 가장 낮은 비용을 사용하는 3개의 부서를 식별할 수도 있습니다. SQL 언어가 데이터 그룹을 처리하기 위해 설계된 여러 함수를 제공하지만 표준 SQL 함수를 사용해서는 행 집합에서 상위 또는 하위 계층을 식별할 수 없습니다.

동일한 행 집합을 나타내는 모든 포트를 변환에 연결합니다. 변환을 구성할 때 설정하는 몇 가지 측정에 따라 해당 순위에 속하는 행만 순위 변환을 통과합니다. 데이터를 변환하거나 계산을 수행하는 식을 작성할 수도 있습니다.

다음 그림에서는 인적 자원 테이블의 직원 데이터를 순위 변환을 통해 전달하는 매핑을 보여 줍니다. 순위 변환은 가장 높은 급여를 받는 10명의 직원에 대한 행만 다음 변환으로 전달합니다.



활성 변환인 순위 변환에서는 통과하는 행 수를 변경할 수 있습니다. 100개의 행을 순위 변환에 전달하지만 상위 10개의 행만 순위를 지정하도록 선택할 수 있으며, 이 상위 10개의 행이 순위 변환에서 다른 변환으로 전달됩니다.

한 변환에서만 순위 변환으로 포트를 연결할 수 있습니다. 또한 로컬 변수를 작성하고 비집게 식을 쓸 수도 있습니다.

문자열 값 순위 지정

통합 서비스가 ASCII 데이터 이동 모드로 실행되는 경우 이진 정렬 순서를 사용하여 세션 데이터를 정렬합니다.

통합 서비스가 유니코드 데이터 이동 모드로 실행되는 경우 세션에 구성된 정렬 순서를 사용합니다. 세션 속성에서 세션 정렬 순서를 선택합니다. 세션 속성에는 통합 서비스가 사용하는 코드 페이지를 기준으로 모든 사용 가능한 정렬 순서가 나열됩니다.

예를 들어 순위 변환이 문자열 포트의 상위 3개 값을 반환하도록 구성되어 있다고 가정합니다. 워크플로우를 구성할 때 워크플로우를 실행할 통합 서비스를 선택합니다. 세션 속성에 선택한 통합 서비스의 코드 페이지와 연관된 모든 정렬 순서(예: 프랑스어, 독일어 및 이진)가 표시됩니다. 이진 정렬 순서를 사용하도록 세션을 구성한 경우 통합 서비스는 각 문자열의 이진 값을 계산하여 문자열의 이진 값이 가장 큰 행 3개를 반환합니다.

순위 캐시

세션 중에 통합 서비스가 입력 행을 데이터 캐시의 행과 비교합니다. 입력 행이 캐시된 행의 순위 밖에 있는 경우 통합 서비스가 캐시된 행을 입력 행으로 바꿉니다. 여러 그룹에서 순위를 지정하도록 순위 변환을 구성한 경우 통합 서비스는 찾은 각 그룹에 대해 증분식으로 순위를 지정합니다.

통합 서비스가 그룹 정보를 인덱스 캐시에 저장하고 행 데이터를 데이터 캐시에 저장합니다. 파이프라인에서 여러 파티션을 작성하는 경우 통합 서비스가 각 파티션마다 별도의 캐시를 작성합니다.

순위 변환 속성

순위 변환을 작성하는 경우 다음 속성을 구성할 수 있습니다.

- 캐시 디렉터리를 입력합니다.
- 상위 또는 하위 순위를 선택합니다.
- 순위를 결정하는 데 사용되는 값이 포함된 입력/출력 포트를 선택합니다. 한 포트만 선택하여 순위를 정의할 수 있습니다.
- 순위에 드는 행 수를 선택합니다.
- 각 제조업체에 대해 10개의 가장 저렴한 제품과 같이 순위를 위한 그룹을 정의합니다.

순위 변환의 포트

순위 변환에는 매핑의 다른 변환에 연결된 입력 또는 입력/출력 포트가 포함됩니다. 또한 변수 포트와 순위 포트가 포함됩니다. 순위 포트를 사용하여 순위를 정하려는 열을 지정합니다.

다음 테이블에는 순위 변환의 포트가 설명되어 있습니다.

포트	필요한 수	설명
I	최소 1개	입력 포트. 다른 변환의 데이터를 받을 입력 포트를 작성합니다.
O	최소 1개	출력 포트. 다른 변환에 연결할 각 포트에 대한 출력 포트를 작성합니다. 입력 포트를 출력 포트로 지정할 수 있습니다.

포트	필요한 수	설명
V	필수 아님	변수 포트. 식에서 사용할 값 또는 계산을 저장하는 데 사용할 수 있습니다. 변수 포트는 입력 또는 출력 포트일 수 없습니다. 변수 포트는 변환 내에서만 데이터를 전달합니다.
R	1개만	순위 포트입니다. 값의 순위를 지정하려는 열을 지정하는 데 사용합니다. 순위 변환에서 순위 포트 하나만 지정할 수 있습니다. 순위 포트는 입력/출력 포트입니다. 순위 포트를 다른 변환에 연결해야 합니다.

순위 인덱스

디자이너는 각 순위 변환에 대해 **RANKINDEX** 포트를 작성합니다. 통합 서비스는 순위 인덱스 포트를 사용하여 그룹의 각 행에 대한 순위 위치를 저장합니다.

예를 들어, 각 분기에 대해 상위 5명의 영업사원 순위를 지정하는 순위 변환을 작성한 경우, 순위 인덱스에서 1에서 5까지 영업사원 번호를 지정합니다.

RANKINDEX	SALES_PERSON	SALES
1	Sam	10,000
2	Mary	9,000
3	Alice	8,000
4	Ron	7,000
5	Alex	6,000

RANKINDEX는 출력 포트 전용입니다. 순위 인덱스를 매핑의 다른 변환에 전달하거나 대상에 직접 전달할 수 있습니다.

그룹 정의

집계 변환과 같이 순위 변환을 통해 정보를 그룹화할 수 있습니다. 예를 들어, 제조업체별로 가장 비싼 항목 10개를 선택할 경우 먼저 각 제조업체에 대해 그룹을 정의합니다. 순위 변환을 구성한 경우 해당 입력/출력 포트 중 하나를 그룹 기준 포트로 설정할 수 있습니다. 그룹 포트의 고유한 값마다 변환에서 순위 정의에 해당하는 행 그룹을 작성합니다(위에서 아래로, 그리고 각 순위의 특정 번호).

따라서 순위 변환에서 행 수를 다음 두 가지 방법으로 변경합니다. 맨 위 또는 맨 아래 순위에 해당하는 거의 모든 행을 필터링하여 변환을 통과하는 행 수를 줄이십시오. 그룹을 정의하여 각 그룹에 대해 순위 지정된 행 집합을 1개 작성합니다.

예를 들어 회사에서 가장 높은 급여를 받는 50명의 직원을 식별하기 위한 순위 변환을 작성할 수 있습니다. 이러한 경우 **SALARY** 열을 순위를 측정하기 위해 사용되는 입력/출력 포트에 식별하고 상위 50개를 제외한 모든 행을 필터링하도록 변환을 구성합니다.

순위 변환이 상위 또는 하위 순위에 속하는 모든 행을 식별한 다음 순위 인덱스 값을 할당합니다. 급여로 측정된 상위 50명의 직원의 경우 가장 높은 급여를 받는 직원이 순위 인덱스 1을 받습니다. 다음으로 높은 급여를 받는 직원이 순위 인덱스 2를 받습니다. 이런 식으로 계속 진행됩니다. 재고에서 10개의 가장 싼 제품과 같이 하위 순

위를 측정하는 경우 순위 변환이 가장 낮은 값에서 가장 높은 값으로 순위 인덱스를 할당합니다. 그러므로 가장싼 항목이 순위 인덱스 1을 받습니다.

두 개의 순위 값이 일치하는 경우에는 순위 인덱스에서 동일한 값을 받고 변환은 다음 값을 건너뜁니다. 예를 들어 국가에 상위 5개의 소매점을 표시하려는데 두 상점의 판매량이 동일한 경우 반환 데이터가 다음과 유사하게 표시될 수 있습니다.

RANKINDEX	SALES	STORE
1	10000	Orange
1	10000	Brea
3	90000	Los Angeles
4	80000	Ventura

순위 변환 작성

소스 한정자 다음에 매핑의 어느 위치에나 순위 변환을 추가할 수 있습니다.

순위 변환을 작성하려면:

- 매핑 디자이너에서 변환 > 작성을 클릭합니다. 순위 변환을 선택합니다. 순위 이름을 입력합니다. 순위 변환의 이름 지정 규칙은 `RNK_TransformationName`입니다.
변환의 설명을 입력합니다. 이 설명은 Repository Manager에 나타납니다.
- 작성을 클릭한 다음 완료를 클릭합니다.
디자이너가 순위 변환을 작성합니다.
- 입력 변환에서 순위 변환으로 열을 연결합니다.
- 포트 탭을 클릭하고 순위 포트에 대한 순위(R) 옵션을 선택합니다.
순위 지정된 행에 대한 그룹을 작성하려면 그룹을 정의하는 포트에 대해 그룹 기준을 선택합니다.
- 속성 탭을 클릭하고 상위 또는 하위 순위를 원하는지 여부를 선택합니다.
- 순위 번호 옵션의 경우 순위를 위해 선택하려는 행 수를 입력합니다.
- 필요한 경우 기타 순위 변환 속성을 변경합니다.

다음 테이블에는 순위 변환 속성이 설명되어 있습니다.

설정	설명
캐시 디렉터리	통합 서비스가 인덱스 및 데이터 캐시 파일을 작성하는 로컬 디렉터리입니다. 기본적으로 통합 서비스는 워크플로우 관리자에서 프로세스 변수 <code>\$PMCacheDir</code> 에 대해 입력된 디렉터리를 사용합니다. 새 디렉터리를 입력하는 경우 디렉터리가 존재하고 캐시 파일용으로 충분한 디스크 공간이 있는지 확인합니다.
상위/하위	열에 대해 상위 또는 하위 순위를 지정할지 여부를 지정합니다.
순위 수	순위를 지정하려는 행 수입니다.

설정	설명
대/소문자 구분 문자열 비교	유니코드 모드로 실행되는 경우 통합 서비스는 세션에 대해 선택된 정렬 순서를 기준으로 문자열 순위를 매깁니다. 세션 정렬 순서가 대/소문자를 구분하는 경우 대/소문자 구분 문자열 비교를 활성화하려면 이 옵션을 선택하고, 통합 서비스가 문자열의 대/소문자 구분을 무시하게 하려면 이 옵션을 선택 취소합니다. 정렬 순서가 대/소문자를 구분하지 않는 경우 통합 서비스가 이 설정을 무시합니다. 기본적으로 이 옵션은 선택되어 있습니다.
추적 수준	통합 서비스가 세션에서 이 변환을 통해 전달되는 데이터에 대해 세션 로그에 기록하는 정보의 양을 결정합니다.
순위 데이터 캐시 크기	변환의 데이터 캐시 크기입니다. 기본값은 2,000,000바이트입니다. 구성된 총 세션 캐시 크기가 2GB(2,147,483,648바이트) 이상인 경우 64비트 통합 서비스에서 세션을 실행해야 합니다. 캐시에 숫자 값을 사용하거나, 매개 변수 파일의 캐시 값을 사용하거나, 자동 설정을 통해 캐시 크기를 설정하도록 통합 서비스를 구성할 수 있습니다. 캐시 크기를 결정하도록 통합 서비스를 구성하는 경우 통합 서비스가 캐시에 할당하는 최대 메모리 양도 구성할 수 있습니다.
순위 인덱스 캐시 크기	변환의 인덱스 캐시 크기입니다. 기본값은 1,000,000바이트입니다. 구성된 총 세션 캐시 크기가 2GB(2,147,483,648바이트) 이상인 경우 64비트 통합 서비스에서 세션을 실행해야 합니다. 캐시에 숫자 값을 사용하거나, 매개 변수 파일의 캐시 값을 사용하거나, 자동 설정을 통해 캐시 크기를 설정하도록 통합 서비스를 구성할 수 있습니다. 캐시 크기를 결정하도록 통합 서비스를 구성하는 경우 통합 서비스가 캐시에 할당하는 최대 메모리 양도 구성할 수 있습니다.
변환 범위	통합 서비스가 변환 논리를 수신 데이터에 적용하는 방식을 지정합니다. <ul style="list-style-type: none"> - 트랜잭션. 트랜잭션의 모든 행에 변환 논리를 적용합니다. 데이터 행이 동일한 트랜잭션의 모든 행에 종속되지만 다른 트랜잭션의 행에는 종속되지 않는 경우 트랜잭션을 선택합니다. - 모든 입력. 변환 논리를 모든 수신 데이터에 적용합니다. 모든 입력을 선택하면 PowerCenter가 수신 트랜잭션 경계를 삭제합니다. 데이터 행이 소스의 모든 행에 종속되는 경우 모든 입력을 선택합니다.

8. 확인을 클릭합니다.

제 22 장

라우터 변환

이 장에 포함된 항목:

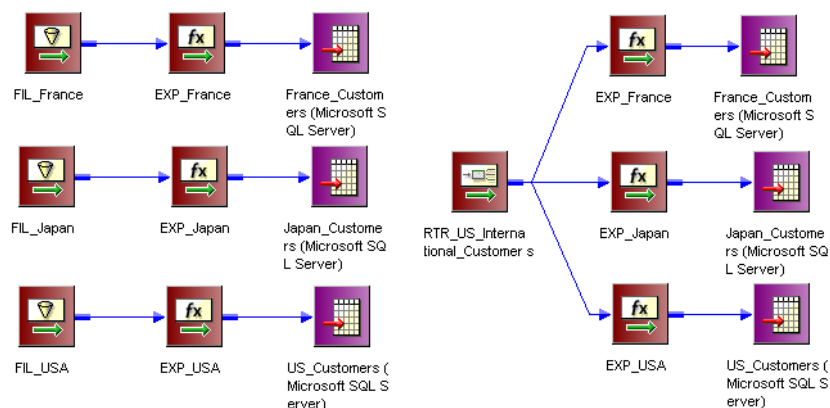
- [라우터 변환 개요, 343](#)
- [그룹 작업, 344](#)
- [포트 작업, 347](#)
- [매핑에서 라우터 변환 연결, 347](#)
- [라우터 변환 작성, 347](#)

라우터 변환 개요

라우터 변환은 조건을 사용하여 데이터를 테스트할 수 있다는 점에서 필터 변환과 유사합니다. 필터 변환은 조건 하나로 데이터를 테스트하여 조건을 충족하지 않는 데이터 행은 삭제합니다. 그에 반해 라우터 변환은 하나 이상의 조건에 대해 데이터를 테스트하고, 어떤 조건에도 부합하지 않는 데이터 행을 기본 출력 그룹으로 보낼 수 있는 옵션을 제공합니다. 라우터 변환은 활성 변환입니다.

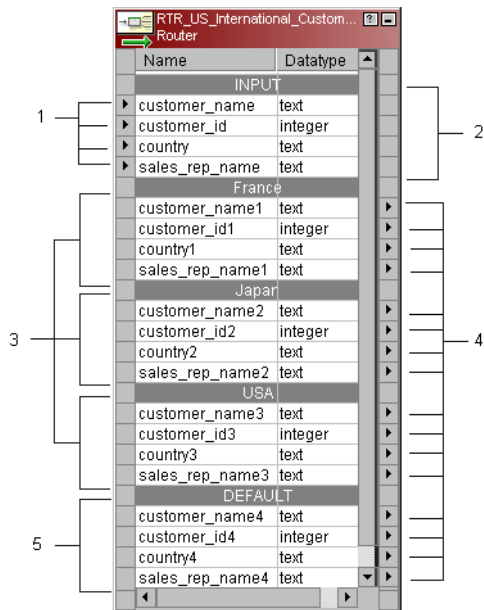
여러 조건에 따라 같은 입력 데이터를 테스트해야 하는 경우에는 같은 태스크를 수행하기 위해 여러 필터 변환을 작성하는 대신 매핑에서 라우터 변환을 사용하십시오. 라우터 변환이 더 효율적입니다. 예를 들어 세 가지 조건에 기반하여 데이터를 테스트하려는 경우 이 태스크를 수행하는 데 필터 변환 3개 대신 라우터 변환 1개만 필요합니다. 마찬가지로, 매핑에서 라우터 변환 1개를 사용할 경우 통합 서비스는 수신 데이터를 한 번만 처리합니다. 매핑에서 필터 변환을 여러 개 사용할 경우에는 통합 서비스가 각 변환별로 수신 데이터를 처리합니다.

다음 그림은 동일한 태스크를 수행하는 두 가지 매핑을 보여 줍니다. 첫 번째 매핑에서는 필터 변환 3개를 사용하지만 두 번째 매핑에서는 라우터 변환 1개를 사용하여 동일한 결과를 생성합니다.



라우터 변환은 디자이너에서 구성할 수 있는 입력 및 출력 그룹, 입력 및 출력 포트, 그룹 필터 조건 및 속성으로 구성됩니다.

다음 그림은 샘플 라우터 변환을 보여 줍니다.



1. 입력 포트.
2. 입력 그룹.
3. 사용자 정의 출력 그룹.
4. 출력 포트
5. 기본 출력 그룹

그룹 작업

라우터 변환에는 다음과 같은 유형의 그룹이 있습니다.

- 입력
- 출력

입력 그룹

디자이너는 입력 그룹의 입력 포트에서 속성 정보를 복사하여 각 출력 그룹에 대한 일련의 출력 포트를 작성합니다.

출력 그룹

출력 그룹에는 두 가지 유형이 있습니다.

- 사용자 정의 그룹
- 기본 그룹

출력 포트 또는 해당 포트의 속성을 수정하거나 삭제할 수 없습니다.

사용자 정의 그룹

들어오는 데이터를 기준으로 조건을 테스트하려면 사용자 정의 그룹을 생성합니다. 사용자 정의 그룹은 출력 포트와 그룹 필터 조건으로 구성됩니다. 디자이너를 사용하여 그룹 탭에서 사용자 정의 그룹을 작성하고 편집할 수 있습니다. 지정하려는 각 조건에 대해 사용자 정의 그룹을 하나씩 생성합니다.

통합 서비스는 조건을 사용하여 수신 데이터의 각 행을 평가합니다. 그리고 기본 그룹을 처리하기 전에 각 사용자 정의 그룹의 조건을 테스트합니다. 통합 서비스는 연결된 출력 그룹의 순서에 따라 각 조건의 평가 순서를 결정합니다. 통합 서비스는 매핑에 있는 변환 또는 대상에 연결된 사용자 정의 그룹을 처리합니다. 기본 그룹이 변환 또는 대상에 연결되어 있는 경우, 통합 서비스는 매핑에 있는 연결되지 않은 사용자 정의 그룹만 처리합니다.

하나 이상의 그룹 필터 조건을 충족하는 행이 있으면 통합 서비스는 이 행을 여러 번 전달합니다.

기본 그룹

사용자가 새 사용자 정의 그룹을 작성하면 디자이너에서 기본 그룹을 작성합니다. 디자이너 사용자는 이 기본 그룹을 편집하거나 삭제할 수 없습니다. 이 그룹에는 그룹 필터 조건이 연결되어 있지 않습니다. 모든 조건이 **FALSE**로 평가되면 통합 서비스는 해당 행을 기본 그룹으로 전달합니다. 통합 서비스가 기본 그룹에 있는 모든 행을 삭제하도록 하려면 매핑에 있는 변환 또는 대상에 기본 그룹을 연결하지 마십시오.

사용자가 마지막 사용자 정의 그룹을 목록에서 삭제하면 디자이너에서 기본 그룹을 삭제합니다.

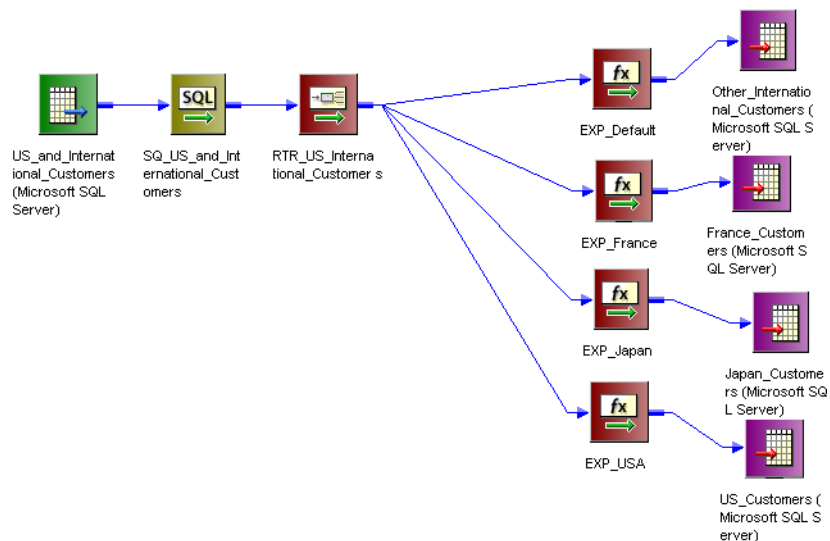
그룹 필터 조건 사용

하나 이상의 그룹 필터 조건에 따라 데이터를 테스트할 수 있습니다. 식 편집기를 사용하여 그룹 탭에서 그룹 필터 조건을 작성하십시오. 단일 값을 반환하는 식을 입력할 수 있습니다. 조건에 상수를 지정할 수도 있습니다. 그룹 필터 조건은 변환을 통과하는 각 행에 대해 행이 지정된 조건을 충족하는지 여부에 따라 **TRUE** 또는 **FALSE**를 반환합니다. 영(0)은 **FALSE**와 같고, 0이 아닌 값은 **TRUE**와 같습니다. 통합 서비스는 **TRUE**로 평가되는 데이터 행을 사용자 정의 그룹 각각과 연결된 각 변환 또는 대상으로 전달합니다.

9개 국가의 고객이 있고 그중 3개 국가에 한정하여 데이터에 대해 서로 다른 계산을 수행하려는 경우를 예로 들어 보겠습니다. 이 데이터를 3개의 서로 다른 식 변환으로 필터링하기 위해 매핑에 있는 라우터 변환을 사용할 수 있습니다.

기본 그룹과 연결된 그룹 필터 조건은 없습니다. 하지만 식 변환을 작성하면 나머지 6개 국가의 데이터를 기반으로 계산을 수행할 수 있습니다.

다음 그림은 여러 조건에 따라 데이터를 필터링하는 라우터 변환을 사용하는 매핑을 보여줍니다.



3개 국가의 데이터에 기반하여 여러 계산을 수행하려고 하므로 그룹 탭에서 사용자 정의 그룹 3개를 작성하고 그룹 필터 조건을 3개를 지정합니다.

다음 테이블은 고객 데이터를 필터링하는 그룹 필터 조건을 보여줍니다.

그룹 이름	그룹 필터 조건
프랑스	customer_name='France'
일본	customer_name='Japan'
USA	customer_name='USA'

세션에서 통합 서비스는 TRUE로 평가되는 데이터 행을 사용자 정의 그룹 각각, 즉 Japan, France 및 USA와 연결된 각 변환 또는 대상으로 전달합니다. 모든 조건이 FALSE로 평가되면 통합 서비스는 해당 행을 기본 그룹으로 전달합니다. 이 경우 통합 서비스는 다른 6개 국가의 데이터를 기본 그룹과 연결된 변환 또는 대상으로 전달합니다. 통합 서비스가 기본 그룹에 있는 모든 행을 삭제하도록 하려면 매핑에 있는 변환 또는 대상에 기본 그룹을 연결하지 마십시오.

라우터 변환이 조건을 충족하는 각 그룹을 통해 데이터를 전달합니다. 따라서 데이터가 출력 그룹 조건 3개를 충족하면 라우터 변환은 출력 그룹 3개를 통해 데이터를 전달합니다.

예를 들어 라우터 변환에서 다음 그룹 조건을 구성합니다.

그룹 이름	그룹 필터 조건
출력 그룹 1	employee_salary > 1000
출력 그룹 2	employee_salary > 2000

라우터 변환이 employee_salary=3000을 사용하여 입력 행 데이터를 처리하면 출력 그룹 1 및 2를 통해 데이터를 라우팅합니다.

그룹 추가

그룹 추가는 기타 변환에서의 포트 추가와 유사합니다. 디자이너는 입력 포트에서 출력 포트로 속성 정보를 복사합니다.

라우터 변환에 그룹을 추가하려면:

1. 그룹 탭을 클릭합니다.
2. 추가 단추를 클릭합니다.
3. 그룹 이름 섹션에 새 그룹 이름을 입력합니다.
4. 그룹 필터 조건 필드를 클릭하여 식 편집기를 엽니다.
5. 그룹 필터 조건을 입력합니다.
6. 유효성 검사를 클릭하여 조건의 구문을 검사합니다.
7. 확인을 클릭합니다.

포트 작업

라우터 변환에는 입력 포트 및 출력 포트가 있습니다. 입력 포트는 입력 그룹에 있고 출력 포트는 출력 그룹에 있습니다. 입력 포트는 다른 변환에서 복사하거나 포트 탭에서 수동으로 작성하는 방법으로 작성될 수 있습니다.

디자이너는 입력 포트에서 다음 속성을 복사하여 출력 포트를 작성합니다.

- 포트 이름
- 데이터 유형
- 전체 자릿수
- 배율
- 기본값

입력 포트를 변경하는 경우, 디자이너는 출력 포트를 업데이트하여 변경 내용을 반영합니다. 출력 포트는 편집하거나 삭제할 수 없습니다. 출력 포트는 라우터 변환의 일반 보기에 표시됩니다.

디자이너는 입력 포트 이름에 기반하여 출력 포트 이름을 작성합니다. 디자이너는 각 입력 포트에 대해 상응하는 출력 포트를 각 출력 그룹에 작성합니다.

매핑에서 라우터 변환 연결

매핑에서 변환을 라우터 변환에 연결할 경우 다음 규칙을 고려하십시오.

- 하나의 그룹을 하나의 변환 또는 대상에 연결할 수 있습니다.
- 그룹의 출력 포트 1개를 여러 변환 또는 대상에 연결할 수 있습니다.
- 1개 그룹의 여러 출력 포트를 여러 변환 또는 대상에 연결할 수 있습니다.
- 하나 이상의 그룹을 대상 1개 또는 단일 입력 그룹 변환에 연결할 수 없습니다.
- 각 출력 그룹을 다른 입력 그룹에 연결할 때 조이너 변환을 제외한 여러 입력 그룹 변환에 하나 이상의 그룹을 연결할 수 없습니다.

라우터 변환 작성

라우터 변환을 매핑에 추가하려면 다음 단계를 수행하십시오.

1. 매핑 디자이너에서 매핑을 엽니다.
2. 변환 > 작성을 클릭합니다.

라우터 변환을 선택하고 새 변환의 이름을 입력합니다. 라우터 변환의 이름 지정 규칙은 **RTR_TransformationName**입니다. 작성을 클릭한 다음 완료를 클릭합니다.

3. 변환에서 모든 포트를 선택하고 끌어서 라우터 변환에 추가하거나 포트 탭에서 수동으로 입력 포트를 작성할 수 있습니다.
4. 라우터 변환의 제목 표시줄을 두 번 클릭하여 변환 속성을 편집합니다.
5. 변환 탭을 클릭하고 변환 속성을 구성합니다.
6. 속성 탭을 클릭하고 추적 수준을 구성합니다.

7. 그룹 탭을 클릭한 후 추가 단추를 클릭하여 사용자 정의 그룹을 작성합니다.
첫 번째 사용자 정의 그룹을 작성하면 디자이너가 기본 그룹을 작성합니다.
8. 그룹 필터 조건 필드를 클릭하여 식 편집기를 엽니다.
9. 그룹 필터 조건을 입력합니다.
10. 유효성 검사를 클릭하여 입력한 조건의 구문을 검사합니다.
11. 확인을 클릭합니다.
12. 그룹 출력 포트를 변환 또는 대상에 연결합니다.

제 23 장

시퀀스 생성기 변환

이 장에 포함된 항목:

- [시퀀스 생성기 변환 개요, 349](#)
- [시퀀스 생성기 포트, 349](#)
- [시퀀스 생성기 변환 속성, 353](#)
- [시퀀스 생성기 고급 속성, 357](#)
- [시퀀스 생성기 변환 작성, 357](#)
- [시퀀스 생성기 변환 - 비원시 환경, 358](#)

시퀀스 생성기 변환 개요

시퀀스 생성기 변환은 숫자 값을 생성하는 수동 변환입니다. 시퀀스 생성기 변환을 사용하여 고유한 기본 키 값을 생성하거나 누락된 기본 키를 바꾸거나 순차적 숫자 범위를 반복합니다.

시퀀스 생성기 변환은 연결된 변환입니다. 이 변환은 하나 이상의 변환에 연결할 수 있는 두 출력 포트를 포함합니다. 통합 서비스는 행 블록이 연결된 변환에 입력될 때마다 시퀀스 번호 블록을 생성합니다. **CURRVAL**을 연결하면 통합 서비스가 각 블록에서 행 하나를 처리합니다. **NEXTVAL**을 다른 변환의 입력 포트에 연결하면 통합 서비스는 숫자 시퀀스를 생성합니다. **CURRVAL**을 다른 변환의 입력 포트에 연결하면 통합 서비스는 **NEXTVAL** 값과 증분 범위 값을 생성합니다.

단일 매핑에서 사용할 시퀀스 생성기 변환을 생성할 수도 있고 여러 매핑에서 사용할 재사용 가능 시퀀스 생성기 변환을 생성할 수도 있습니다. 재사용 가능 시퀀스 생성기 변환은 시퀀스 생성기 변환 인스턴스를 사용하는 각 매핑에서 시퀀스의 무결성을 유지 관리합니다.

시퀀스 생성기 변환을 재사용 가능하도록 지정하여 여러 매핑에서 사용할 수 있습니다. 단일 대상에 대한 여러 로드를 수행할 때 시퀀스 생성기 변환을 재사용할 수 있습니다.

예를 들어 큰 입력 파일을 병렬로 실행되는 3개 세션으로 구분하는 경우 시퀀스 생성기 변환을 사용하여 기본 키 값을 생성합니다. 다른 시퀀스 생성기 변환을 사용하는 경우에는 통합 서비스가 중복 키 값을 생성할 수 있습니다. 대신 3개 세션에 대해 모두 재사용 가능 시퀀스 생성기 변환을 사용하여 각 대상 행에 고유한 값을 제공합니다.

시퀀스 생성기 포트

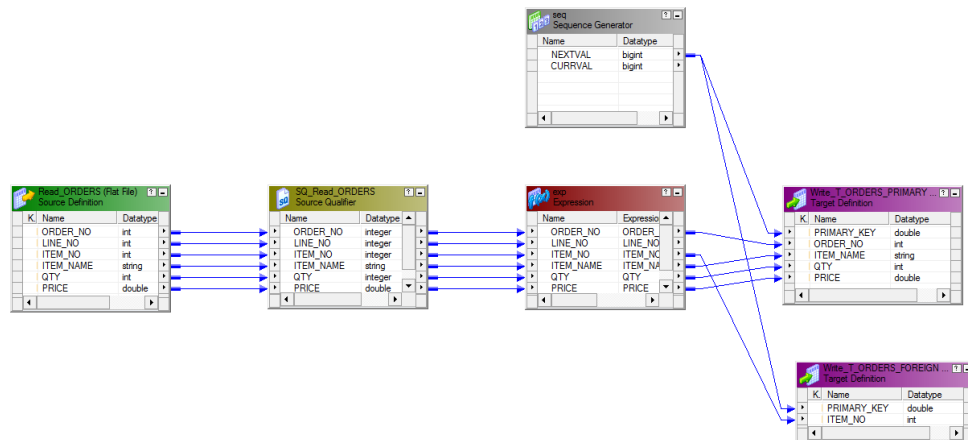
시퀀스 생성기 변환에는 **NEXTVAL** 및 **CURRVAL**의 두 가지 출력 포트가 있습니다. 이러한 포트는 편집하거나 삭제할 수 없습니다. 마찬가지로 포트를 변환에 추가할 수 없습니다.

NEXTVAL 포트

NEXTVAL을 변환에 연결하여 변환의 각 행에 대해 고유한 값을 생성할 수 있습니다. NEXTVAL 포트를 다운스트림 변환 또는 대상에 연결하여 숫자 시퀀스를 생성합니다. NEXTVAL을 여러 변환에 연결하는 경우 통합 서비스가 각 변환에 대해 동일한 숫자 시퀀스를 생성합니다.

NEXTVAL 포트를 연결하여 시작 값 및 증분 값 속성에 따라 시퀀스를 생성합니다. 시퀀스 생성기가 시퀀스를 반복하도록 구성되지 않은 경우 NEXTVAL 포트가 구성된 끝 값까지 시퀀스 번호를 생성합니다.

다음 이미지는 기본 및 외래 키 값을 생성하기 위해 두 대상에 연결된 시퀀스 생성기 변환 NEXTVAL 포트와의 매핑을 보여줍니다.



시작 값 = 1과 증분 값 = 1을 사용하여 시퀀스 생성기 변환을 구성하는 경우 통합 서비스가 T_ORDERS_PRIMARY 및 T_ORDERS_FOREIGN 대상 테이블에 대해 동일한 기본 키 값을 생성합니다.

NEXTVAL을 여러 변환에 연결하여 각 변환의 각 행에 대해 고유한 값을 생성합니다. NEXTVAL 포트를 다운스트림 변환 또는 대상에 연결하여 시퀀스 번호를 생성하려면 NEXTVAL 포트를 사용합니다. NEXTVAL 포트를 연결하여 현재 값 및 증분 범위 속성에 따라 시퀀스를 생성합니다. 시퀀스 생성기가 시퀀스를 반복하도록 구성되지 않은 경우 NEXTVAL 포트가 구성된 끝 값까지 시퀀스 번호를 생성합니다.

예를 들어 NEXTVAL을 매핑의 두 대상에 연결하여 고유한 기본 키 값을 생성할 수 있습니다. 통합 서비스가 각 대상 테이블에 대해 고유한 기본 키 값 열을 생성합니다. 고유한 기본 키 값 열은 시퀀스 번호 블록으로 하나의 대상 테이블에 전송됩니다. 첫 번째 대상이 시퀀스 번호 블록을 수신한 다음 다른 대상이 시퀀스 생성기 변환에서 시퀀스 번호 블록을 수신합니다.

예를 들어 다음과 같은 시퀀스 생성기 변환을 구성합니다. 현재 값 = 1, 증분 범위 = 1. 통합 서비스가 T_ORDERS_PRIMARY 및 T_ORDERS_FOREIGN 대상 테이블에 대해 다음 기본 키 값을 생성합니다.

**T_ORDERS_PRIMARY TABLE:
PRIMARY KEY**

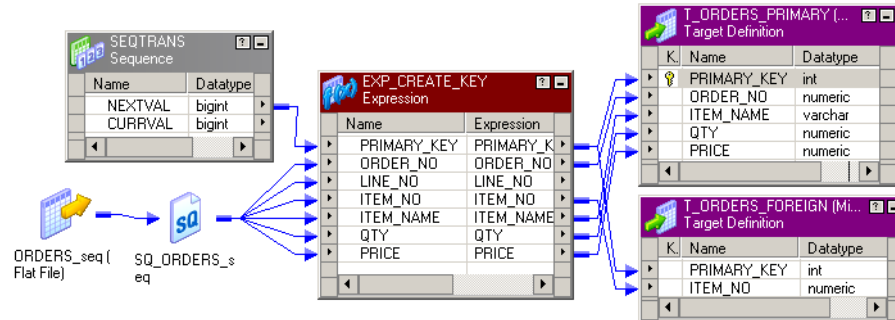
1
2
3
4
5

**T_ORDERS_FOREIGN TABLE:
PRIMARY KEY**

6
7
8
9
10

단일 변환에서 데이터를 수신하는 둘 이상의 대상에 동일한 값을 전송하려면 시퀀스 생성기 변환을 그 이전 변환에 연결하면 됩니다. 통합 서비스가 값을 시퀀스 번호 블록으로 처리합니다. 이를 통해 통합 서비스가 고유한 값을 변환에 전달한 다음 변환에서 대상으로 행을 라우팅할 수 있습니다.

다음 그림은 고유한 값을 식 변환에 전달하는 시퀀스 생성기와의 매핑을 보여줍니다.



식 변환이 두 대상을 동일한 기본 키 값으로 채웁니다.

예를 들어 다음과 같은 시퀀스 생성기 변환을 구성합니다. 현재 값 = 1, 증분 범위 = 1. 통합 서비스가 T_ORDERS_PRIMARY 및 T_ORDERS_FOREIGN 대상 테이블에 대해 다음 기본 키 값을 생성합니다.

T_ORDERS_PRIMARY TABLE: PRIMARY KEY

1
2
3
4
5

T_ORDERS_FOREIGN TABLE: PRIMARY KEY

1
2
3
4
5

참고: 그리드에서 분할된 세션을 실행하는 경우 시퀀스 생성기 변환은 각 파티션의 행 숫에 따라 값을 건너뛰니다.

키 작성

시퀀스 생성기 변환에서 NEXTVAL 포트를 대상 또는 다운스트림 변환에 연결하여 기본 또는 외래 키 값을 작성할 수 있습니다. 1에서 9,223,372,036,854,775,807 범위의 값을 최소 간격 1로 사용할 수 있습니다.

기본 또는 외래 키를 작성할 때는 주기 옵션을 사용하여 통합 서비스가 중복 기본 키를 작성하지 않도록 하십시오. 이렇게 하려면 세션 속성에서 대상 테이블 잘라내기 옵션을 선택하거나 복합 키를 작성합니다.

복합 키를 작성하려면 통합 서비스의 주기를 더 작은 값 집합으로 구성합니다. 예를 들어 세 곳의 매장에서 주문 번호가 생성되는 경우 시퀀스 생성기 변환의 주기 값을 1에서 3으로 구성하고 1씩 증분되도록 구성합니다. ORDER_NO 포트를 시퀀스 생성기 변환에 연결하면 생성된 값에서 고유한 복합 키가 작성됩니다.

다음 예는 복합 키 및 주문 순서를 보여 줍니다.

COMPOSITE_KEY

1
2

ORDER_NO

12345
12345

COMPOSITE_KEY	ORDER_NO
3	12345
1	12346
2	12346
3	12346

누락된 값 바꾸기

시퀀스 생성기 변환을 사용하여 IIF 및 ISNULL 함수와 함께 NEXTVAL을 사용하여 누락된 키를 바꿀 수 있습니다.

예를 들어 ORDER_NO 열에서 Null 값을 바꾸려면 속성을 사용하여 시퀀스 생성기 변환을 작성하고 NEXTVAL 포트를 식 변환으로 끕니다. 식 변환에서 ORDER_NO 포트를 다른 필요한 포트와 함께 변환으로 끕니다. 그런 다음 출력 포트, ALL_ORDERS를 작성합니다.

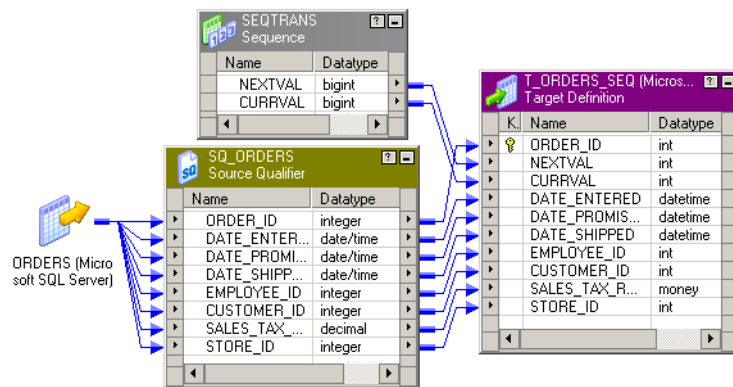
그런 다음 ALL_ORDERS에서 다음 식을 입력하여 Null 순서를 바꿉니다.

```
IIF( ISNULL( ORDER_NO ), NEXTVAL, ORDER_NO )
```

CURRVAL

CURRVAL은 NEXTVAL에 증분 값을 더한 것입니다. CURRVAL 포트는 NEXTVAL 포트가 이미 다운스트림 변환에 연결된 경우에만 연결합니다. 행이 CURRVAL 포트에 연결된 변환으로 들어가면 통합 서비스가 마지막으로 작성된 NEXTVAL 값에 1을 더한 값을 전달합니다.

다음 그림은 CURRVAL 및 NEXTVAL 포트를 대상에 연결하는 작업을 보여 줍니다.



예를 들어 다음과 같은 시퀀스 생성기 변환을 구성합니다. 현재 값 = 1, 증분 범위 = 1. 이 경우 통합 서비스는 NEXTVAL 및 CURRVAL에 대해 다음 값을 생성합니다.

NEXTVAL	CURRVAL
1	2
2	3
3	4

NEXTVAL

4

5

CURRVAL

5

6

NEXTVAL 포트를 연결하지 않고 CURRVAL 포트를 연결하면 통합 서비스가 각 행에 대해 상수 값을 전달합니다. 시퀀스 생성기 변환에서 CURRVAL 포트를 연결하면 통합 서비스가 각 블록에서 1개의 행을 전달합니다. 매핑에서 NEXTVAL 포트만 연결하면 성능을 최적화할 수 있습니다.

참고: 그리드에서 분할된 세션을 실행하는 경우 시퀀스 생성기 변환이 각 파티션의 행 수에 따라 값을 건너뛰는 수도 있습니다.

시퀀스 생성기 변환 속성

통합 서비스가 순차적 값을 생성하는 데 사용하는 변환 속성을 구성합니다.

다음 목록에는 구성할 수 있는 시퀀스 생성기 변환 속성이 설명되어 있습니다.

시작 값

주기 옵션을 사용하는 경우 통합 서비스가 사용할 생성된 시퀀스의 시작 값입니다. 주기를 선택하는 경우 끝 값에 도달하면 통합 서비스가 이 값으로 다시 순환합니다.

기본값은 0입니다.

최대값은 9,223,372,036,854,775,806입니다.

증분 범위

NEXTVAL 포트의 두 연속 값 사이의 차이입니다.

기본값은 1입니다.

양의 정수여야 합니다.

최대값은 2,147,483,647입니다.

끝 값

통합 서비스가 생성하는 최대값입니다. 세션 중 통합 서비스가 이 값에 도달하고 시퀀스가 순환하도록 구성되지 않은 경우 세션이 실패합니다.

최대값은 9,223,372,036,854,775,807입니다.

NEXTVAL 포트를 다운스트림 정수 포트에 연결한 경우 끝 값을 정수 최대값보다 작은 값으로 설정하십시오. NEXTVAL이 다운스트림 포트의 데이터 유형 최대값을 초과할 경우 세션이 실패합니다.

현재 값

시퀀스의 현재 값입니다. 통합 서비스에서 사용하도록 할 값을 시퀀스의 첫 번째 값으로 입력합니다. 일련의 값이 순환하게 하려면 값이 시작 값보다 크거나 같고 끝 값보다 작아야 합니다.

총 캐시된 값이 0으로 설정된 경우 통합 서비스는 현재 값을 업데이트하여 세션+1에 대해 마지막으로 생성된 값을 반영한 다음 업데이트된 현재 값을 이 세션의 다음 번 실행에 대한 기준으로 사용합니다. 그러나 재설정 옵션을 사용하면 통합 서비스는 각 세션 이후 이 값을 원래 값으로 재설정합니다.

참고: 이 설정을 편집하는 경우 시퀀스를 새 설정으로 재설정합니다. 현재 값을 10으로 재설정하고 증분이 1이면 다음 번에 세션을 사용할 때 통합 서비스가 첫 번째 값, 10을 생성합니다.

최대값은 9,223,372,036,854,775,806입니다. 현재 값이 최대값을 초과하면 통합 서비스가 값을 NULL로 설정합니다.

주기

활성화된 경우 통합 서비스가 시퀀스 범위를 순환하고 시작 값을 사용하여 다시 시작합니다.

비활성화된 경우 통합 서비스가 구성된 끝 값에서 시퀀스를 중지합니다. 끝 값에 도달했지만 여전히 처리할 행이 있는 경우 통합 서비스에서 오버플로우 오류가 발생하고 세션을 실행하지 못합니다.

총 캐시된 값

한 번에 통합 서비스가 캐시하는 순차적 값 수입니다. 여러 세션이 동시에 동일한 재사용 가능 시퀀스 생성기를 사용하는 경우 이 옵션을 사용하여 각 세션이 고유한 값을 수신하게 할 수 있습니다. 통합 서비스가 각 값을 캐시할 때 리포지토리를 업데이트합니다. 0으로 설정되면 통합 서비스가 값을 캐시하지 않습니다.

기본값은 0입니다.

재사용 가능 시퀀스 생성기에 대한 기본값은 1,000입니다.

최대값은 9,223,372,036,854,775,807입니다.

이 속성은 재사용 가능 시퀀스 생성기 변환에만 적용할 수 있습니다.

재설정

활성화된 경우 통합 서비스가 각 세션의 원래 현재 값에 따라 값을 생성합니다. 비활성화된 경우 통합 서비스가 현재 값을 업데이트하여 세션+1에 대해 마지막으로 생성된 값을 반영한 다음 업데이트된 현재 값을 다음 세션 실행에 대한 기준으로 사용합니다.

이 속성은 재사용 불가능 시퀀스 생성기 변환에만 적용할 수 있습니다.

추적 수준

통합 서비스가 세션 로그에 쓰는 변환에 대한 세부 정보 수준입니다.

시작 값

주기를 사용하여 반복 시퀀스를 생성할 수 있습니다(예: 1년의 달에 해당하도록 1부터 12까지의 숫자).

1. 통합 서비스가 시작 값으로 사용하게 하려는 시퀀스의 가장 낮은 값을 입력합니다.
2. 끝 값에 사용할 가장 높은 값을 입력합니다.
3. 주기를 선택합니다.

순환하는 동안 통합 서비스가 시퀀스의 구성된 끝 값에 도달하면 구성된 시작 값부터 시작하여 주기를 다시 시작합니다.

증분 범위

통합 서비스에서 NEXTVAL 포트의 시퀀스는 시퀀스 생성기 변환의 현재 값 및 증분 범위 속성을 바탕으로 생성됩니다.

현재 값 속성은 통합 서비스가 각 세션에 대한 시퀀스 작성을 시작할 때의 값입니다. 증분 범위는 통합 서비스가 시퀀스의 새 값을 작성할 때 기존 값에 추가하는 정수입니다. 기본적으로 현재 값은 1로 설정되고 증분 범위는 1로 설정됩니다.

예를 들어 현재 값이 1,000이고 증분 범위가 10인 시퀀스 생성기 변환을 작성할 수 있습니다. 매핑을 통해 행 3개를 전달하는 경우 통합 서비스는 다음과 같은 값 집합을 생성합니다.

1000
1010
1020

끝 값

통합 서비스에서 생성하도록 할 최대값이 끝 값입니다. 통합 서비스가 끝 값에 도달하고 시퀀스 생성기가 시퀀스를 반복하도록 구성되지 않은 경우 세션이 오버플로우 오류와 함께 실패합니다.

끝 값을 1과 9,233,372,036,854,775,807 사이의 정수로 설정하십시오. NEXTVAL 포트를 다운스트림 정수 포트에 연결한 경우 끝 값을 정수 최대값보다 작은 값으로 설정하십시오. 예를 들어, NEXTVAL 포트를 작은 정수 포트에 연결한 경우 끝 값을 최대 32,767로 설정하십시오. NEXTVAL이 다운스트림 포트의 데이터 유형 최대값을 초과할 경우 세션이 실패합니다.

값 범위 반복

시퀀스 생성기 변환에 대한 값 범위를 설정할 수 있습니다. 반복 옵션을 사용할 경우 시퀀스 생성기 변환에서 끝 값에 도달하면 범위를 반복합니다.

예를 들어, 시퀀스 범위가 10에서 시작하여 50에서 끝나도록 설정하고 증가 값을 10으로 설정한 경우 시퀀스 생성기 변환에서 10, 20, 30, 40, 50 값을 작성합니다. 시퀀스가 다시 10에서 시작됩니다.

현재 값

통합 서비스는 각 세션에 생성되는 값에 대한 기준으로 현재 값을 사용합니다. 통합 서비스가 시퀀스 생성기 변환을 처음 사용할 때 사용할 값을 표시하려면 해당 값을 현재 값으로 입력해야 합니다. 시퀀스 생성기 변환을 사용하여 일련의 값을 반복하려면 현재 값이 시작 값보다 크거나 같고 끝 값보다 작아야 합니다.

각 세션이 끝나면 통합 서비스는 캐시된 값의 시퀀스 생성기 번호가 0일 경우 세션에 생성된 마지막 값에 1을 더한 값으로 현재 값을 업데이트합니다. 예를 들어, 통합 서비스가 생성된 값 101로 세션을 종료한 경우 리포지토리의 시퀀스 생성기 현재 값을 102로 업데이트합니다. 다음에 시퀀스 생성기가 사용될 때 통합 서비스는 다음 생성되는 값의 기준으로 102를 사용합니다. 시퀀스 생성기 증분 범위가 1일 경우 통합 서비스가 시퀀스 생성기를 사용하여 다른 세션을 시작하면 첫 번째 생성되는 값은 102입니다.

여러 버전의 시퀀스 생성기 변환이 있는 경우 통합 서비스는 세션을 실행할 때 모든 버전의 현재 값을 업데이트합니다. 통합 서비스는 사용자가 시퀀스 생성기 변환 또는 상위 매핑을 체크 아웃했는지 여부와 상관없이 버전에서 현재 값을 업데이트합니다. 두 개 값이 다를 경우 업데이트된 현재 값이 시퀀스 생성기 변환의 편집된 현재 값을 재정의합니다.

예를 들어, 사용자 1이 시퀀스 생성기 변환을 작성하고 현재 값 10을 시퀀스 생성기 버전 1에 저장한 상태에서 체크 인합니다. 사용자 1이 시퀀스 생성기 변환을 체크 아웃하고 새로운 현재 값 100을 시퀀스 생성기 버전 2에 입력합니다. 사용자 1은 시퀀스 생성기 변환을 계속 체크 아웃합니다. 반면 사용자 2는 시퀀스 생성기 변환 버전 1을 사용하는 세션을 실행합니다. 통합 서비스는 사용자 2가 세션을 실행할 때 체크 인 값 10을 현재 값으로 사용합니다. 세션이 완료되면 현재 값은 150입니다. 통합 서비스는 사용자 1이 시퀀스 생성기 변환을 체크 아웃했더라도 시퀀스 생성기 변환의 버전 1과 버전 2에 대해 현재 값을 150으로 업데이트합니다.

세션을 실행한 후 매핑을 열 경우 현재 값이 세션에 생성된 마지막 값에 1을 더한 값으로 표시됩니다. 통합 서비스가 현재 값을 사용하여 각 세션의 첫 번째 값을 결정하기 때문에 시퀀스를 재설정할 경우에만 현재 값을 편집해야 합니다.

여러 버전의 시퀀스 생성기 변환이 있고 시퀀스를 재설정할 경우, 현재 값을 수정한 후 매핑 또는 재사용 가능한 시퀀스 생성기 변환을 체크 인해야 합니다.

참고: 시퀀스 생성기를 재설정하도록 구성한 경우 통합 서비스는 각 세션의 첫 번째 생성되는 값의 기준으로 현재 값을 사용합니다.

총 캐시된 값

총 캐시된 값에 따라 통합 서비스가 한 번에 캐시하는 값 수가 결정됩니다. 총 캐시된 값이 0보다 큰 경우 통합 서비스가 구성된 값 수를 캐시하고 값을 캐시할 때마다 현재 값을 업데이트합니다.

여러 세션이 동시에 동일한 재사용 가능 시퀀스 생성기 변환을 사용하는 경우 여러 인스턴스의 시퀀스 생성기 변환이 있을 수 있습니다. 각 세션에 대해 동일한 값을 생성하지 않도록 방지하려면 총 캐시된 값을 구성하여 각 세션에 대해 시퀀스 값 범위를 예약합니다.

팁: 그리드에서 세션을 실행할 때 성능을 향상시키려면 시퀀스 생성기 변환에 대해 총 캐시된 값을 늘립니다. 그러면 마스터 및 작업자 DTM 프로세스와 리포지토리 사이에 필요한 통신이 감소합니다.

재사용 불가능 시퀀스 생성기

재사용 불가능 시퀀스 생성기 변환에서는 총 캐시된 값이 0으로 기본 설정되며 통합 서비스는 세션 중에 값을 캐시하지 않습니다. 통합 서비스는 값을 캐시하지 않을 때 세션 시작 시 리포지토리에서 현재 값에 액세스합니다. 그런 다음 시퀀스에 대해 값을 생성합니다. 세션이 종료되면 통합 서비스는 리포지토리에서 현재 값을 업데이트합니다.

총 캐시된 값을 0보다 크게 설정하면 통합 서비스는 세션 중에 값을 캐시합니다. 세션이 시작될 때 통합 서비스는 리포지토리에서 현재 값에 액세스하여 구성된 값의 수를 캐시한 다음 그에 따라 현재 값을 업데이트합니다. 통합 서비스는 캐시의 모든 값을 사용하는 경우 리포지토리에서 다음 값 집합에 액세스하여 현재 값을 업데이트합니다. 세션이 종료되면 통합 서비스는 캐시에서 나머지 값을 무시합니다.

재사용 불가능 시퀀스 생성기 변환에서는 총 캐시된 값을 0보다 크게 설정하면 통합 서비스가 세션 중에 리포지토리에 액세스하는 횟수가 증가할 수 있습니다. 또한 각 세션이 종료될 때 사용되지 않은 캐시된 값이 무시되므로 특정 값 섹션을 건너뜁니다.

시퀀스 생성기 변환을 다음과 같이 구성하는 경우를 예로 들어 보겠습니다. 총 캐시된 값 = 50, 현재 값 = 1, 증분 범위 = 1. 통합 서비스는 세션을 시작할 때 세션에 대해 값 50개를 캐시하고 리포지토리에서 현재 값을 50으로 업데이트합니다. 통합 서비스는 세션에 대해 값 1~39를 사용하고 사용되지 않은 값인 40~49를 무시합니다. 통합 서비스는 해당 세션을 다시 실행할 때 리포지토리에서 현재 값인 50을 확인합니다. 그런 후 다음 50개의 값을 캐시하고 현재 값을 100으로 업데이트합니다. 세션 중에는 값 50~98이 사용됩니다. 두 세션에 대해 생성되는 값은 1~39 및 50~98입니다.

재사용 가능 시퀀스 생성기

여러 세션에 재사용 가능 시퀀스 생성기 변환이 있고 동시에 해당 세션이 실행될 때 총 캐시된 값을 사용하여 각 세션이 시퀀스에서 고유한 값을 수신하도록 할 수 있습니다. 기본적으로 재사용 가능 시퀀스 생성기에 대해 총 캐시된 값이 1000으로 설정됩니다.

여러 세션이 동시에 동일한 시퀀스 생성기 변환을 사용하는 경우 각 세션에 대해 동일한 값을 생성할 위험이 있습니다. 이를 방지하려면 총 캐시된 값을 구성하여 통합 서비스가 각 세션에 대해 총 값 집합을 캐시하게 합니다.

예를 들어 재사용 가능 시퀀스 생성기 변환을 다음과 같이 구성합니다. 총 캐시된 값 = 50, 현재 값 = 1, 증분 범위 = 1. 두 개의 세션이 시퀀스 생성기를 사용하고 거의 동시에 실행하도록 예약되어 있습니다. 통합 서비스가 첫 번째 세션을 시작할 때 세션에 대해 50개의 값을 캐시하고 리포지토리에서 현재 값을 50으로 업데이트합니다. 통합 서비스가 세션에서 1~50의 값을 사용하여 시작합니다. 통합 서비스가 두 번째 세션을 시작할 때 현재 값(50으로 설정)에 대해 리포지토리를 확인합니다. 그런 후 다음 50개의 값을 캐시하고 현재 값을 100으로 업데이트합니다. 그런 다음 두 번째 세션에서 51~100의 값을 사용합니다. 어떤 세션이든 해당 캐시된 값을 모두 사용하면 통합 서비스가 새 값 집합을 캐시하고 현재 값을 업데이트하여 이러한 값이 시퀀스 생성기에 대해 고유하게 유지되도록 합니다.

재사용 가능 시퀀스 생성기 변환의 경우 총 캐시된 값을 줄이면 삭제되는 값을 최소화할 수 있지만 1보다는 커야 합니다. 총 캐시된 값을 줄이는 경우 세션 중에 값을 캐시하기 위해 통합 서비스가 리포지토리에 액세스하는 횟수를 늘릴 수 있습니다.

시퀀스 생성기 고급 속성

통합 서비스가 순차적 값을 생성하는 데 사용하는 고급 속성을 구성합니다.

다음 목록에는 구성할 수 있는 시퀀스 생성기 고급 속성이 설명되어 있습니다.

재설정

활성화된 경우 통합 서비스가 각 세션의 원래 현재 값에 따라 값을 생성합니다. 비활성화된 경우 통합 서비스가 현재 값을 업데이트하여 세션+1에 대해 마지막으로 생성된 값을 반영한 다음 업데이트된 현재 값을 다음 세션 실행에 대한 기준으로 사용합니다.

이 속성은 재사용 가능 시퀀스 생성기 변환에 대해 비활성화됩니다.

추적 수준

통합 서비스가 세션 로그에 쓰는 변환에 대한 세부 정보 수준입니다.

재설정

재사용 불가능 시퀀스 생성기 변환에 대해 재설정을 선택하면 통합 서비스는 세션을 시작할 때마다 원래 시작 값을 기준으로 값을 생성합니다. 그렇지 않으면 통합 서비스는 마지막으로 생성된 값에 증분 값을 더한 결과를 반영하도록 현재 값을 업데이트하고 다음번에 시퀀스 생성기 변환을 사용할 때 업데이트된 값을 사용합니다.

1씩 증가하는 1에서 1,000까지의 값을 생성하도록 시퀀스 생성기 변환을 구성하는 경우를 예로 들어 보겠습니다. 이 경우 시작 값을 1로 재설정하도록 선택합니다. 첫 번째 세션 실행 중에 통합 서비스는 숫자 1~234를 생성합니다. 각 후속 매핑 실행에서 통합 서비스는 초기 값 1부터 숫자를 다시 생성합니다.

재설정을 수행하지 않으면 통합 서비스는 첫 번째 실행 종료 시에 현재 값을 235로 업데이트합니다. 그러면 다음번에 시퀀스 생성기 변환을 사용할 때 처음으로 생성되는 값이 235가 됩니다.

참고: 재사용 가능 시퀀스 생성기 변환에 대해서는 재설정이 비활성화됩니다.

시퀀스 생성기 변환 작성

시퀀스 생성기 변환을 매핑에서 사용하려면 변환을 매핑에 추가하고 변환 속성을 구성한 다음 NEXTVAL 또는 CURRVAL을 하나 이상의 변환에 연결합니다.

시퀀스 생성기 변환을 작성하려면 다음을 수행하십시오.

1. 매핑 디자이너에서 변환 > 작성을 클릭합니다. 시퀀스 생성기 변환을 선택합니다.
시퀀스 생성기 변환의 이름 지정 규칙은 SEQ_TransformationName입니다.
2. 시퀀스 생성기의 이름을 입력하고 작성을 클릭합니다. 완료를 클릭합니다.
디자이너가 시퀀스 생성기 변환을 작성합니다.
3. 변환의 제목 표시줄을 두 번 클릭합니다.
4. 변환의 설명을 입력합니다.
5. 속성 탭을 선택합니다. 설정을 입력합니다.
참고: 시퀀스 생성기 변환 속성을 세션 수준에서 재정의할 수 없습니다. 이는 생성된 시퀀스 값의 무결성을 보호하기 위한 것입니다.
6. 확인을 클릭합니다.

7. 세션 중에 새 시퀀스를 생성하려면 매핑에서 최소 1개의 변환에 NEXTVAL 포트를 연결합니다.
다른 변환의 식에서 NEXTVAL 또는 CURRVAL 포트를 사용합니다.

시퀀스 생성기 변환 - 비원시 환경

비원시 환경에서 시퀀스 생성기 변환의 처리는 변환을 실행하는 엔진에 따라 다릅니다.

다음과 같은 비원시 런타임 엔진에 대한 지원을 고려하십시오.

- **Blaze** 엔진. 제한적으로 지원됩니다.
- **Spark** 엔진. 일괄 매핑에서 제한적으로 지원됩니다. 스트리밍 매핑에서는 지원되지 않습니다.
- **Databricks Spark** 엔진. 도메인과 Databricks 클러스터가 AWS 또는 Azure Databricks 환경의 동일한 가상 네트워크에 있는 경우 지원됩니다.

시퀀스 생성기 변환 - Blaze 엔진

다음 조건이 참인 경우 시퀀스 생성기 변환이 포함된 매핑은 많은 양의 리소스를 소비합니다.

- 행 순서를 유지하도록 변환을 구성합니다.
- 매핑이 단일 파티션에서 실행됩니다.

시퀀스 생성기 변환 - Spark 엔진

시퀀스 생성기 변환은 출력 데이터에서 행 순서를 유지하지 않습니다. 변환에서 **행 순서 유지** 속성을 활성화하는 경우 데이터 통합 서비스는 이 속성을 무시합니다.

참고: 행 순서를 유지하는 **Spark** 시퀀스 생성기는 데이터 집합이 상대적으로 작고 단일 파티션에서 실행되거나 강제로 실행될 수 있는 경우 제대로 작동합니다.

제 24 장

분류기 변환

이 장에 포함된 항목:

- [분류기 변환 개요, 359](#)
- [데이터 정렬, 359](#)
- [분류기 변환 속성, 360](#)
- [분류기 변환 작성, 362](#)

분류기 변환 개요

분류기 변환으로 데이터를 정렬할 수 있습니다. 지정한 정렬 키에 따라 데이터를 오름차순이나 내림차순으로 정렬할 수 있습니다. 또한 대/소문자 구분 정렬을 수행하도록 분류기 변환을 구성할 수 있으며, 출력 행이 고유해야 하는지 여부를 지정할 수 있습니다. 분류기 변환은 활성 변환입니다. 즉, 데이터 흐름에 연결되어야 합니다.

관계형 또는 플랫폼 파일 소스의 데이터를 정렬할 수 있습니다. 또한 분류기 변환을 사용하여 정렬된 입력을 사용하도록 구성된 집계 변환을 통해 전달되는 데이터를 정렬할 수 있습니다.

매핑에서 분류기 변환을 작성할 때 하나 이상의 포트를 정렬 키로 지정하고 각 정렬 키 포트를 오름차순 또는 내림차순으로 정렬하도록 구성합니다. 또한 통합 서비스가 모든 정렬 키 포트에 적용하는 정렬 조건과 정렬 작업을 수행하기 위해 할당하는 시스템 리소스를 구성합니다.

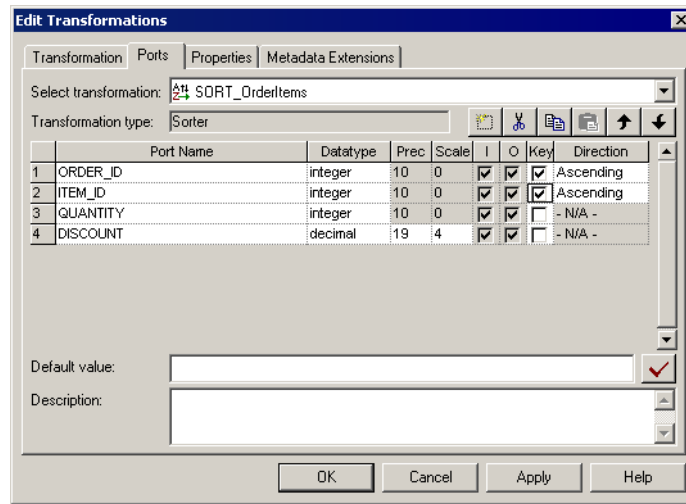
데이터 정렬

분류기 변환에는 입력/출력 포트만 포함됩니다. 분류기 변환을 통해 전달되는 모든 데이터는 정렬 키를 기준으로 정렬됩니다. 정렬 키는 정렬 조건으로 사용할 하나 이상의 포트입니다.

정렬 키로 둘 이상의 포트를 지정할 수 있습니다. 정렬 키에 대해 여러 포트를 지정하는 경우 통합 서비스는 각 포트를 순차적으로 정렬합니다. 포트 탭에 포트가 나타나는 순서에 따라 정렬 작업의 순서가 결정됩니다. 분류기 변환에서는 연속적인 각 정렬 키 포트를 통해 전달되는 데이터를 이전 포트의 2차 정렬로 취급합니다.

세션 런타임 시 통합 서비스는 세션 속성에 지정된 정렬 순서에 따라 데이터를 정렬합니다. 정렬 순서는 특수 문자 및 기호에 대한 정렬 조건을 결정합니다.

다음 그림에서는 주문 ID 및 항목 ID를 기준으로 오름차순으로 데이터를 정렬하는 분류기 변환에 대한 포트 탭 구성을 보여 줍니다.



세션 런타임 시 통합 서비스는 분류기 변환에 다음 행을 전달합니다.

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

데이터를 정렬하고 나면 통합 서비스가 분류기 변환에서 다음 행을 전달합니다.

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

분류기 변환 속성

분류기 변환에는 추가 정렬 조건을 지정하는 다양한 속성이 있습니다. 통합 서비스는 이러한 조건을 모든 정렬 키 포트에 적용합니다. 분류기 변환 속성은 통합 서비스가 데이터를 정렬할 때 할당하는 시스템 리소스도 결정합니다.

분류기 캐시 크기

통합 서비스는 분류기 캐시 크기 속성을 사용하여 정렬 작업을 수행하기 위해 할당할 수 있는 최대 메모리 양을 결정합니다. 통합 서비스는 정렬 작업을 수행하기 전에 모든 수신 데이터를 분류기 변환으로 전달합니다. 캐시에 숫자 값을 사용하거나, 매개 변수 파일의 캐시 값을 사용하거나, 자동 설정을 통해 캐시 크기를 설정하도록 통합 서비스를 구성할 수 있습니다. 캐시 크기를 결정하도록 통합 서비스를 구성하는 경우 통합 서비스가 캐시에 할당하는 최대 메모리 양도 구성할 수 있습니다. 구성된 총 세션 캐시 크기가 **2GB(2,147,483,648바이트)** 이상일 경우 **64비트** 통합 서비스에서 세션을 실행해야 합니다.

통합 서비스는 정렬 작업을 시작하기 전에 분류기 캐시 크기에 대해 구성된 양의 메모리의 할당합니다. 통합 서비스는 분할된 세션을 실행하는 경우 각 분할에 대해 지정된 양의 분류기 캐시 메모리를 할당합니다.

충분한 메모리를 할당할 수 없는 경우 통합 서비스에서 세션이 실패합니다. 최상의 성능을 얻으려면 분류기 캐시 크기를 통합 서비스 시스템에서 사용 가능한 실제 RAM 양과 같거나 작은 값으로 구성합니다. 분류기 변환을 사용하여 데이터를 정렬하려면 **16MB(16,777,216바이트)** 이상의 실제 메모리를 할당하십시오. 기본적으로 분류기 캐시 크기는 **16,777,216바이트**로 설정됩니다.

수신 데이터의 양이 분류기 캐시 크기의 양보다 큰 경우 통합 서비스는 데이터를 분류기 변환 작업 디렉터리에 임시로 저장합니다. 작업 디렉터리에 데이터를 저장할 경우 통합 서비스에 적어도 수신 데이터 양의 두 배에 이르는 디스크 공간이 필요합니다. 수신 데이터의 양이 분류기 캐시 크기보다 상당히 큰 경우에는 통합 서비스에 작업 디렉터리에서 사용할 수 있는 디스크 공간 양의 두 배보다 훨씬 많은 디스크 공간이 필요할 수 있습니다.

분류기 변환 추적 수준을 일반적으로 구성한 경우 통합 서비스는 분류기 변환이 사용하는 메모리의 양도 세션 로그에 기록합니다.

대/소문자 구분

대/소문자 구분 속성은 통합 서비스가 데이터를 정렬할 때 대/소문자를 고려하는지 여부를 결정합니다. 대/소문자 구분 속성을 활성화하면 통합 서비스는 대문자를 소문자보다 높게 정렬합니다.

작업 디렉터리

통합 서비스가 데이터를 정렬하는 동안 임시 파일을 작성하는 데 사용하는 작업 디렉터리를 지정해야 합니다. 통합 서비스는 데이터를 정렬한 후 임시 파일을 삭제합니다. 통합 서비스 시스템의 원하는 디렉터리를 작업 디렉터리로 사용하도록 지정할 수 있습니다. 기본적으로 통합 서비스는 **\$PMTempDir** 프로세스 변수에 지정된 값을 사용합니다.

분류기 변환을 사용하여 세션을 분할할 때 파이프라인의 각 파티션에 대해 서로 다른 작업 디렉터리를 지정할 수 있습니다. 세션 성능을 향상시키려면 통합 서비스 시스템의 물리적으로 분리된 디스크에 작업 디렉터리를 지정하십시오.

고유 출력 행

출력 행을 고유 행으로 처리하도록 분류기 변환을 구성할 수 있습니다. 고유 출력 행에 대해 분류기 변환을 구성하면 매핑 디자이너가 모든 포트를 정렬 키의 일부로 구성합니다. 통합 서비스는 정렬 작업 중에 비교된 중복 행을 삭제합니다.

추적 수준

통합 서비스가 세션 로그에 기록하는 분류기 오류 및 상태 메시지의 수와 유형을 제어하려면 분류기 변환 추적 수준을 구성합니다. 보통 추적 수준에서는 분류기 변환에 전달된 행의 크기와 분류기 변환이 정렬 작업을 위해 할당한 메모리의 양이 기록됩니다. 또한 통합 서비스가 분류기 변환으로 첫 번째 입력 행과 마지막 입력 행을 전달한 날짜와 시간이 기록됩니다.

분류기 변환 추적 수준을 자세한 정보 표시 데이터로 구성하면 분류기 변환이 파이프라인의 다음 변환으로 모든 데이터를 전달하는 작업을 마친 시간이 기록됩니다. 또한 분류기 변환이 메모리 리소스를 해제하고 작업 디렉터리에서 임시 파일을 제거한 시간이 세션 로그에 기록됩니다.

Null을 Low로 처리

분류기 변환이 **null** 값을 처리하는 방식을 구성할 수 있습니다. 통합 서비스가 정렬 작업을 수행할 때 **null** 값을 다른 모든 값보다 낮게 처리하도록 하려면 이 속성을 활성화합니다. 통합 서비스가 **null** 값을 다른 모든 값보다 높게 처리하도록 하려면 이 옵션을 비활성화합니다.

변환 범위

변환 범위는 통합 서비스가 수신 데이터에 변환 논리를 적용하는 방식을 지정합니다.

- **트랜잭션.** 트랜잭션의 모든 행에 변환 논리를 적용합니다. 데이터 행이 동일한 트랜잭션의 모든 행에 종속되지만 다른 트랜잭션의 행에는 종속되지 않는 경우 트랜잭션을 선택합니다.
- **모든 입력.** 변환 논리를 모든 수신 데이터에 적용합니다. 모든 입력을 선택하면 PowerCenter가 수신 트랜잭션 경계를 삭제합니다. 데이터 행이 소스의 모든 행에 종속되는 경우 모든 입력을 선택합니다.

분류기 변환 작성

분류기 변환을 매핑에 추가하려면 다음 단계를 수행하십시오.

1. 매핑 디자이너에서 변환 > 작성을 클릭합니다. 분류기 변환을 선택합니다.

분류기 변환의 이름 지정 규칙은 **SRT_TransformationName**입니다. 변환의 설명을 입력합니다. 이 설명은 Repository Manager에서 나타나며, 이 설명을 보고 변환이 수행하는 작업을 보다 쉽게 이해할 수 있습니다.

2. 분류기의 이름을 입력하고 작성을 클릭합니다.

디자이너가 분류기 변환을 작성합니다.

3. 완료를 클릭합니다.

4. 분류기 변환으로 정렬할 포트를 끕니다.

사용자가 포함하는 각 포트에 대해 디자이너가 입력/출력 포트를 작성합니다.

5. 변환의 제목 표시줄을 두 번 클릭하여 변환 편집 대화 상자를 엽니다.

6. 포트 탭을 선택합니다.

7. 정렬 키로 사용할 포트를 선택합니다.

8. 정렬 키의 일부로 선택된 각 포트에 대해 통합 서비스가 오름차순 또는 내림차순으로 데이터를 정렬하도록 할지 지정합니다.

9. 속성 탭을 선택합니다. 분류기 변환 속성을 수정합니다.

10. 메타데이터 확장 탭을 선택합니다. 분류기 변환에 대한 메타데이터 확장을 작성하거나 편집합니다.

11. 확인을 클릭합니다.

제 25 장

소스 한정자 변환

이 장에 포함된 항목:

- [소스 한정자 변환 개요, 363](#)
- [소스 한정자 변환 속성, 365](#)
- [기본 쿼리, 366](#)
- [소스 데이터 조인, 367](#)
- [SQL 쿼리 추가, 369](#)
- [사용자 정의 조인 입력, 370](#)
- [외부 조인 지원, 371](#)
- [소스 필터 입력, 378](#)
- [정렬된 포트 사용, 379](#)
- [고유 항목 선택, 379](#)
- [사전 세션 및 사후 세션 SQL 명령 추가, 380](#)
- [소스 한정자 변환 작성, 381](#)
- [소스 한정자 변환 문제 해결, 381](#)

소스 한정자 변환 개요

관계형 또는 플랫폼 파일 소스 정의를 매핑에 추가할 때는 소스 한정자 변환에 해당 정의를 연결해야 합니다. 소스 한정자 변환은 통합 서비스에서 세션을 실행할 때 읽는 행을 나타냅니다. 소스 한정자 변환은 활성 변환입니다.

소스 한정자 변환을 사용하여 다음 태스크를 완료하십시오.

- **동일한 소스 데이터베이스를 출처로 하는 데이터의 조인.** 여러 소스를 소스 한정자 변환 1개에 연결하여 기본 키-외래 키 관계에 있는 둘 이상의 테이블을 조인할 수 있습니다.
- **통합 서비스가 소스 데이터를 읽을 때 행 필터링.** 필터 조건을 포함시키면 통합 서비스가 기본 쿼리에 WHERE 절을 추가합니다.
- **기본 내부 조인 대신 외부 조인 지정.** 사용자 정의 조인을 포함하는 경우 통합 서비스는 SQL 쿼리의 메타데이터가 지정하는 조인 정보를 대체합니다.
- **정렬된 포트를 지정합니다.** 정렬된 포트에 대한 번호를 지정하면 통합 서비스가 ORDER BY 절을 기본 SQL 쿼리에 추가합니다.
- **소스에서 고유 값만 선택.** 고유 항목 선택을 선택하면 통합 서비스가 기본 SQL 쿼리에 SELECT DISTINCT 문을 추가합니다.

- 통합 서비스가 소스 데이터를 읽는 특수한 **SELECT** 문을 실행하기 위한 사용자 지정 쿼리 작성. 예를 들어 사용자 지정 쿼리를 사용하여 집계 계산을 수행할 수 있습니다.

변환 데이터 유형

소스 한정자 변환은 변환 데이터 유형을 표시합니다. 변환 데이터 유형은 통합 서비스가 데이터를 읽을 때 소스 데이터베이스에서 데이터를 바인딩하는 방식을 결정합니다. 소스 한정자 변환에서 데이터 유형을 변경하지 마십시오. 소스 정의와 소스 한정자 변환에서 데이터 유형이 일치하지 않는 경우, 매핑을 저장할 때 디자이너에서 해당 매핑을 잘못된 것으로 표시합니다.

대상 로드 순서

매핑에서 소스 한정자 변환에 기반하여 대상 로드 순서를 지정하십시오. 여러 소스 한정자 변환을 여러 대상에 연결한 경우 통합 서비스가 어떤 순서로 데이터를 대상으로 로드하는지 지정할 수 있습니다.

소스 한정자 변환 1개가 여러 대상에 데이터를 제공하는 경우, 세션에서 제약 조건 기반 로딩을 활성화하여 통합 서비스가 대상 테이블의 기본 및 외래 키 관계에 따라 데이터를 로드하도록 할 수 있습니다.

예를 들어 3개의 파이프라인을 포함하는 매핑이 있습니다. 각 파이프라인에는 매핑으로 연결되는 개별 소스, 소스 한정자, 변환 및 플랫폼 파일 대상이 포함됩니다. 통합 서비스가 각 대상 로드 순서 그룹을 순차적으로 처리하도록 매핑의 대상 로드 순서를 구성할 수 있습니다.

날짜/시간 값

SQL 쿼리에서 날짜/시간 값 또는 날짜/시간 매개 변수나 변수를 사용하는 경우 소스에 사용되는 형식으로 날짜 형식을 변경합니다. 통합 서비스는 SQL 쿼리에서 문자열로 소스 시스템에 날짜/시간 값을 전달합니다. 통합 서비스는 소스 데이터베이스를 기반으로 날짜/시간 값을 문자열로 변환합니다.

다음 테이블에는 각 데이터베이스 유형에 대한 날짜/시간 형식이 설명되어 있습니다.

소스	날짜 형식
DB2	YYYY-MM-DD-HH24:MI:SS
Informix	YYYY-MM-DD HH24:MI:SS
Microsoft SQL Server	MM/DD/YYYY HH24:MI:SS
ODBC	YYYY-MM-DD HH24:MI:SS
Oracle	MM/DD/YYYY HH24:MI:SS
Sybase	MM/DD/YYYY HH24:MI:SS
Teradata	YYYY-MM-DD HH24:MI:SS

일부 데이터베이스를 사용하려면 데이터베이스별 함수 또는 작은따옴표 등 추가적인 구두점으로 날짜/시간 값을 식별해야 합니다. 예를 들어 Oracle 소스에 대한 \$\$\$SessStartTime 값을 변환하려면 SQL 재정의에서 다음 Oracle 함수를 사용합니다.

```
to_date ('$$$SessStartTime', 'mm/dd/yyyy hh24:mi:ss')
```

Informix의 경우 SQL 재정의에서 다음 Informix 함수를 사용하여 \$\$\$SessStartTime 값을 변환합니다.

```
DATETIME ($$$SessStartTime) YEAR TO SECOND
```


데이터베이스별 함수에 대한 자세한 내용은 데이터베이스 설명서를 참조하십시오.

매개 변수 및 변수

소스 한정자 변환의 SQL 쿼리, 사용자 정의 조인, 소스 필터, 사전 및 사후 세션 SQL 명령에서 매개 변수 및 변수를 사용할 수 있습니다. 매개 변수 파일에 정의할 수 있는 모든 매개 변수 또는 변수 유형을 사용할 수 있습니다. SQL 문 내부에 매개 변수 또는 변수를 입력하거나 매개 변수 또는 변수를 SQL 쿼리로 사용할 수 있습니다. 예를 들어 세션 매개 변수 \$ParamMyQuery를 SQL 쿼리로 사용하고 매개 변수 파일에서 \$ParamMyQuery를 SQL 문으로 설정할 수 있습니다.

통합 서비스는 먼저 SQL 쿼리를 생성하고 각 매개 변수 또는 변수를 확장합니다. 통합 서비스는 각 매핑 매개 변수, 매핑 변수 및 워크플로우 변수를 시작 값으로 대체합니다. 그런 다음, 해당 쿼리를 소스 데이터베이스에 대해 실행합니다.

소스 한정자 변환에서 문자열 매핑 매개 변수 또는 변수를 사용할 경우, 소스 시스템에 적합한 문자열 식별자를 사용하십시오. 대부분의 데이터베이스는 작은따옴표를 문자열 식별자로 사용합니다. 예를 들어 문자열 매개 변수 \$\$IPAddress를 Microsoft SQL Server 데이터베이스 테이블에 대한 소스 필터에서 사용하려면 '\$\$IPAddress'와 같이 매개 변수를 작은따옴표로 둘러쌉니다.

날짜/시간 매핑 매개 변수 또는 변수를 사용하거나 기본 제공 변수 \$\$\$SessStartTime을 사용할 경우, 날짜 형식을 소스에서 사용되는 형식으로 변경합니다. 통합 서비스는 SQL 쿼리에서 문자열로 소스 시스템에 날짜/시간 값을 전달합니다.

팁: 날짜/시간 매개 변수 또는 변수의 형식이 소스에서 사용되는 형식과 일치하는지 확인하려면 SQL 쿼리의 유효성을 검사합니다.

소스 한정자 변환 속성

속성 탭에서 다음과 같은 소스 한정자 변환 속성을 구성하십시오.

옵션	설명
SQL 쿼리	이 소스 한정자 변환에서 나타내는 소스의 데이터를 읽기 위해 통합 서비스가 사용하는 기본 쿼리를 대체하는 사용자 지정 쿼리를 정의합니다. 사용자 지정 쿼리는 사용자 지정 조인 또는 소스 필터의 항목을 재정의합니다.
사용자 정의 조인	동일한 소스 한정자 변환에서 나타내는 여러 소스로부터 데이터를 조인하는 데 사용되는 조건을 지정합니다.
소스 필터	통합 서비스가 행을 쿼리할 때 적용하는 필터 조건을 지정합니다.
정렬된 포트 수	관계형 소스에서 쿼리한 행을 정렬할 때 사용되는 열의 수를 나타냅니다. 이 옵션을 선택하면 통합 서비스는 소스 행을 읽을 때 기본 쿼리에 ORDER BY를 추가합니다. ORDER BY는 변환의 맨 위에 있는 포트부터 시작하여 지정된 개수의 포트를 포함합니다. 이 옵션을 선택한 경우 데이터베이스 정렬 순서는 세션 정렬 순서와 일치해야 합니다.
추적 수준	이 변환을 포함하는 세션을 실행할 때 세션 로그에 포함되는 세부 정보의 양을 설정합니다.
고유 항목 선택	고유 행만 선택하려는 경우 지정합니다. 이 옵션을 선택하면 통합 서비스가 SELECT DISTINCT 문을 포함시킵니다.
SQL 전	통합 서비스가 소스를 읽기 전에 소스 데이터베이스에 대해 실행할 사전 세션 SQL 명령입니다.

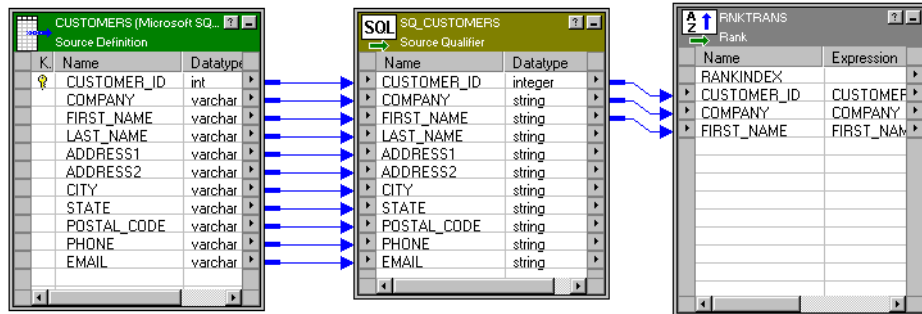
옵션	설명
SQL 후	통합 서비스가 대상에 기록한 후에 소스 데이터베이스에 대해 실행할 사후 세션 SQL 명령입니다.
확정 출력입니다.	입력 데이터가 세션 실행 사이에서 일관된 경우 세션 실행 간에 변경되지 않는 관계형 소스 또는 변환 출력입니다. 이 속성을 구성하면 파이프라인에서 변환이 항상 반복 가능 데이터를 생성할 경우, 통합 서비스는 복구를 위해 소스 데이터를 준비하지 않습니다.
출력은 반복 가능합니다.	입력 데이터의 순서가 일관된 경우 세션 실행 사이에서 동일한 순서에 있는 관계형 소스 또는 변환 출력입니다. 출력이 확정적이고 반복 가능한 경우 통합 서비스는 복구를 위해 소스 데이터를 준비하지 않습니다.

경고: 변환을 반복 가능 및 확정으로 구성하는 경우 데이터가 반복 가능 및 확정인지 확인하는 것은 사용자의 책임입니다. 세션과 복구 간에 동일한 데이터를 생성하지 않는 변환으로 세션을 복구하려는 경우 복구 프로세스로 인해 손상된 데이터가 발생할 수 있습니다.

기본 쿼리

관계형 소스의 경우 통합 서비스는 세션을 실행할 때 각 소스 한정자 변환에 대한 쿼리를 생성합니다. 기본 쿼리는 매핑에서 사용되는 각 소스 열에 대한 **SELECT** 문입니다. 다시 말해, 통합 서비스는 다른 변환에 연결된 열만 읽습니다.

다음 그림은 소스 한정자 변환에 연결되는 단일 소스 정의를 보여 줍니다. 소스 한정자 변환에는 많은 소스 정의 열이 있지만 세 개만 최종 변환에 연결됩니다.



소스 정의에 많은 열이 있지만 세 개의 열만 다른 변환에 연결됩니다. 이 경우 통합 서비스는 이러한 세 열만 선택하는 기본 쿼리를 생성합니다.

```
SELECT CUSTOMERS.CUSTOMER_ID, CUSTOMERS.COMPANY, CUSTOMERS.FIRST_NAME
FROM CUSTOMERS
```

테이블 이름 또는 열 이름에 데이터베이스 예약어가 있을 경우 예약어를 포함하는 **reswords.txt** 파일을 작성하고 유지 관리할 수 있습니다. 통합 서비스는 세션을 초기화할 때 통합 서비스 설치 디렉터리에서 **reswords.txt**를 검색합니다. 파일이 존재하면 통합 서비스는 데이터베이스에 대해 SQL을 실행할 때 일치하는 예약어를 따옴표로 묶습니다. SQL을 재정의하는 경우 예약어를 따옴표로 묶어야 합니다.

기본 쿼리를 생성할 때 디자이너는 다음 문자가 포함된 테이블 및 필드 이름을 큰따옴표로 구분합니다.

```
/ + - = ~ ` ! % & * ( ) [ ] { } ' ; , < > \ | <space>
```

기본 쿼리 보기

소스 한정자 변환의 기본 쿼리를 볼 수 있습니다.

기본 쿼리를 보려면 다음을 수행하십시오.

1. 속성 탭에서 **SQL** 쿼리를 선택합니다.

통합 서비스가 소스 데이터를 선택하기 위해 사용하는 기본 쿼리가 **SQL** 편집기에 표시됩니다.

2. **SQL** 생성을 클릭합니다.

3. 끝내려면 취소를 클릭합니다.

SQL 쿼리를 취소하지 않으면 통합 서비스가 사용자 지정 **SQL** 쿼리로 기본 쿼리를 재정의합니다.

소스 데이터베이스에 연결하지는 마십시오. 기본 쿼리를 재정의하는 **SQL** 쿼리를 입력할 때만 소스 데이터베이스에 연결하십시오.

기본 쿼리를 생성하려면 먼저 소스 한정자 변환의 열을 다른 변환 또는 대상에 연결해야 합니다.

기본 쿼리 재정의

변환 속성의 기본 설정을 변경하여 소스 한정자 변환의 기본 쿼리를 변경하거나 재정의할 수 있습니다. 선택한 포트의 목록을 변경하거나 해당 포트가 쿼리에 나타나는 순서를 변경하지 마십시오. 이 목록은 연결된 변환 출력 포트와 일치해야 합니다.

변환 속성을 편집하는 경우, 소스 한정자 변환은 이러한 속성을 기본 쿼리에 포함합니다. 그러나 사용자가 **SQL** 쿼리를 입력하면 통합 서비스는 정의된 **SQL** 문만 사용합니다. **SQL** 쿼리는 소스 한정자 변환의 사용자 정의 조인, 소스 필터, 정렬된 포트 수 및 고유 항목 선택 설정을 재정의합니다.

참고: 기본 **SQL** 쿼리를 재정의하는 경우에는 모든 데이터베이스 예약어를 따옴표로 묶어야 합니다.

소스 데이터 조인

하나의 소스 한정자 변환을 사용하여 여러 관계형 테이블의 데이터를 조인합니다. 이러한 테이블은 동일한 인스턴스 또는 데이터베이스 서버에서 액세스할 수 있어야 합니다.

매핑에서 관련된 관계형 소스를 사용하는 경우 하나의 소스 한정자 변환에서 두 소스를 모두 조인할 수 있습니다. 세션 동안 소스 데이터베이스는 데이터를 통합 서비스에 전달하기 전에 조인을 수행합니다. 따라서 소스 테이블을 인덱싱할 때 성능이 향상될 수 있습니다.

팁: 다른 유형 소스 및 플랫폼 파일 조인에 조이너 변환을 사용합니다.

기본 조인

관련 테이블을 하나의 소스 한정자 변환에서 조인하면 통합 서비스가 각 테이블의 관련 키에 따라 테이블을 조인합니다.

이 기본 조인은 **WHERE** 절에서 다음과 같은 구문을 사용하는 내부 동등 조인입니다.

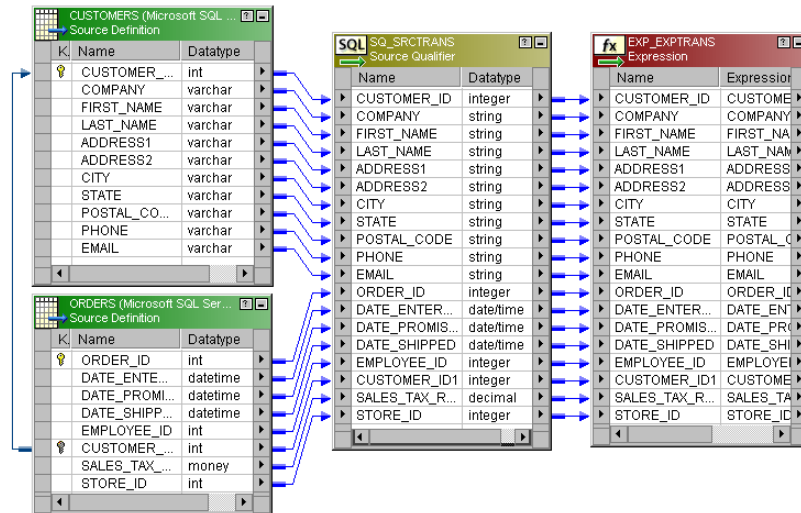
```
Source1.column_name = Source2.column_name
```

기본 조인의 열은 다음과 같은 항목을 포함해야 합니다.

- 기본 키-외래 키 관계
- 일치하는 데이터 유형

예를 들어 주문 번호, 주문량 및 고객 이름을 포함하여 해당 월의 모든 주문을 확인할 수 있습니다. **ORDERS** 테이블에는 주문 번호와 각 주문의 양이 포함되지만 고객 이름은 포함되지 않습니다. 고객 이름을 포함하려면 **ORDERS** 테이블과 **CUSTOMERS** 테이블을 조인해야 합니다. 두 테이블 모두 고객 ID를 포함하므로 하나의 소스 한정자 변환에서 테이블을 조인할 수 있습니다.

다음 그림은 하나의 소스 한정자 변환을 사용하여 두 개의 테이블을 조인하는 방식을 보여 줍니다.



여러 테이블을 포함하면 통합 서비스가 매핑에서 사용되는 모든 열에 대해 **SELECT** 문을 생성합니다. 이 경우 **SELECT** 문은 다음 문과 비슷합니다.

```
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
```

WHERE 절은 **ORDERS** 테이블과 **CUSTOMER** 테이블의 **CUSTOMER_ID**를 포함하는 동등 조인입니다.

사용자 지정 조인

기본 조인을 재정의해야 하는 경우 사용자 지정 쿼리에서 조인을 지정하는 **WHERE** 절의 내용을 입력할 수 있습니다. 쿼리가 외부 조인을 수행하는 경우 통합 서비스가 데이터베이스 구문에 따라 조인 구문을 **WHERE** 절 또는 **FROM** 절에 삽입할 수 있습니다.

다음과 같은 경우 기본 조인을 재정의해야 할 수 있습니다.

- 열에 기본 카-외래 키 관계가 없습니다.
- 조인에 사용되는 각 열의 데이터 유형이 일치하지 않습니다.
- 다른 유형의 조인(예: 외부 조인)을 지정하려고 합니다.

다른 유형 조인

다른 유형 조인을 수행하려면 조이너 변환을 사용합니다. 다음과 같은 유형의 소스를 조인해야 하는 경우 조이너 변환을 사용하십시오.

- 다른 소스 데이터베이스의 데이터 조인
- 다른 플랫폼 파일 시스템의 데이터 조인
- 관계형 소스 및 플랫폼 파일 조인

키 관계 작성

해당 테이블에 기본 키-외래 키 관계가 있을 경우 소스 한정자 변환에서 테이블을 조인할 수 있습니다. 하지만 여러 테이블에서 일치하는 열을 연결하여 소스 분석기에서 기본 키-외래 키 관계를 작성할 수 있습니다. 이러한 열은 키가 될 필요가 없지만 각 테이블의 인덱스에 포함되어야 합니다.

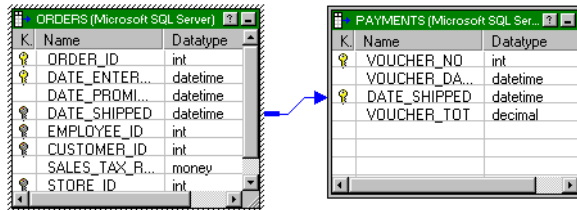
팁: 소스 테이블에 1,000개가 넘는 행이 있을 경우 기본 키-외래 키를 인덱싱하여 성능을 향상시킬 수 있습니다. 소스 테이블에 1,000개 미만의 행이 있을 경우 기본 키-외래 키를 인덱싱하면 성능이 저하될 수 있습니다.

예를 들어 소매 체인의 회사 사무실에서 주문을 기준으로 받은 지불금을 추출한다고 가정합니다. **ORDERS** 테이블과 **PAYMENTS** 테이블은 기본 키 및 외래 키를 공유하지 않습니다. 하지만 두 테이블 모두 **DATE_SHIPPED** 열을 포함합니다. 소스 분석기에서 메타데이터에 기본 키-외래 키 관계를 작성할 수 있습니다.

두 테이블은 연결되지 않았음에 유의하십시오. 따라서 디자이너는 **DATE_SHIPPED** 열에서 관계를 인식하지 못합니다.

DATE_SHIPPED 열을 연결하여 **ORDERS** 테이블과 **PAYMENTS** 테이블 간에 관계를 작성합니다. 디자이너가 **ORDERS** 및 **PAYMENTS** 테이블 정의에서 **DATE_SHIPPED** 열에 기본 키와 외래 키를 추가합니다.

다음 그림은 두 테이블 간의 **DATE_SHIPPED** 관계를 보여 줍니다.



두 열을 연결하지 않으면 디자이너가 관계를 인식하지 못합니다.

기본 키-외래 키 관계는 메타데이터에서만 존재합니다. SQL을 생성하거나 소스 테이블을 변경할 필요가 없습니다.

키 관계가 존재하면 소스 한정자 변환을 사용하여 두 테이블을 조인할 수 있습니다. 기본 조인은 **DATE_SHIPPED**를 기반으로 합니다.

SQL 쿼리 추가

소스 한정자 변환은 기본 쿼리를 재정의하는 SQL 쿼리 옵션을 제공합니다. 소스 데이터베이스에서 지원하는 SQL 문을 입력할 수 있습니다. 쿼리를 입력하기 전에 매핑에서 사용할 모든 입력 및 출력 포트를 연결하십시오.

SQL 쿼리를 편집할 때 기본 쿼리를 생성하고 편집할 수 있습니다. 디자이너는 기본 쿼리를 생성할 때 필터 또는 정렬된 포트의 수와 같은 모든 다른 구성된 옵션을 통합합니다. 결과 쿼리는 사용자가 이후에 변환에서 구성할 수 있는 모든 다른 옵션을 재정의합니다.

매개 변수 또는 변수를 SQL 쿼리로 사용하거나 매개 변수 및 변수를 SQL 쿼리 내에 포함할 수 있습니다. 문자열 매핑 매개 변수 또는 변수를 포함할 경우 소스 시스템에 적절한 문자열 식별자를 사용하십시오. 대부분의 데이터베이스의 경우 문자열 매개 변수 또는 변수의 이름을 작은따옴표로 묶어야 합니다.

날짜 시간 값이나 날짜 매핑 매개 변수 또는 변수를 SQL 쿼리에 포함하는 경우 소스에서 사용하는 형식과 일치하도록 날짜 형식을 변경합니다. 통합 서비스는 소스 시스템에 따라 날짜 시간 값을 문자열로 변환합니다.

사용자 지정 SQL 쿼리를 입력할 때 다음 규칙과 지침을 따르십시오.

- **SELECT** 문은 변환에서 포트 이름이 나타나는 순서대로 포트 이름을 나열해야 합니다.
- 소스가 **Microsoft SQL Server**인 경우 쿼리의 **SELECT** 문에 있는 열의 수가 소스 한정자 변환의 포트 수와 일치해야 합니다. 그렇지 않으면 세션이 다음과 같은 내용의 오류와 함께 실패합니다. **SQL 오류 [FnName: 가져오기 최적화 -- [Informatica][ODBC SQL Server Wire Protocol 드라이버] 바운드 열의 수가 결과 열의 수를 초과합니다.]**.

푸시다운 최적화가 구성된 세션에 대해 기본 SQL 쿼리를 재정의하면 통합 서비스가 SQL 재정의의 결과를 나타내는 보기를 작성합니다. 그런 다음 이 보기에 대해 SQL 쿼리를 실행하여 변환 논리를 데이터베이스로 푸시합니다.

SQL 쿼리를 편집하는 경우 모든 데이터베이스 예약어를 따옴표로 묶어야 합니다.

1. 소스 한정자 변환을 열고 속성 탭을 클릭합니다.
2. SQL 쿼리 필드에서 열기 단추를 클릭합니다.

SQL 편집기 대화 상자가 나타납니다.

3. SQL 생성을 클릭합니다.

디자이너가 소스 한정자 변환에 포함된 모든 소스에서 행을 쿼리할 때 생성하는 기본 쿼리를 표시합니다.

4. 기본 쿼리가 나타나는 공간에 쿼리를 입력합니다.

모든 쿼리 열은 해당 열이 나타나는 테이블, 보기 또는 동의어의 이름으로 규정되어야 합니다. 예를 들어 **ORDERS** 테이블의 **ORDER_ID** 열을 포함하려면 **ORDERS.ORDER_ID**를 입력합니다. 포트 창에서 표시되는 열 이름을 두 번 클릭하면 모든 열의 이름을 입력할 필요가 없습니다.

매개 변수 또는 변수를 쿼리로 사용하거나 매개 변수 및 변수를 쿼리에 포함할 수 있습니다.

문자열 매핑 매개 변수 및 변수를 문자열 식별자로 묶습니다. 필요한 경우 날짜 시간 매핑 매개 변수 및 변수의 날짜 형식을 변경합니다.

5. 쿼리에 포함된 소스가 있는 **ODBC** 데이터 소스를 선택합니다.
6. 이 데이터베이스에 연결할 사용자 이름과 암호를 입력합니다.

Kerberos 인증 사용 옵션은 연결의 데이터베이스가 **Kerberos** 인증을 사용하는 네트워크에서 실행됨을 나타냅니다. 이 옵션을 선택하면 사용자 이름과 암호를 입력할 수 없습니다. 연결에서는 디자이너가 실행되는 시스템에 로그인한 사용자 계정의 자격 증명을 사용합니다.

7. 유효성 검사를 클릭합니다.

디자이너가 쿼리를 실행하고 쿼리 구문이 올바른지 여부를 보고합니다.

8. 확인을 클릭하여 변환 편집 대화 상자로 돌아갑니다. 확인을 다시 클릭하여 디자이너로 돌아갑니다.

Tip: 식 편집기의 크기를 조정할 수 있습니다. 테두리에서 끌어서 대화 상자를 확대합니다. 디자이너가 대화 상자의 새로운 크기를 클라이언트 설정으로 저장합니다.

사용자 정의 조인 입력

사용자 정의 조인을 입력하는 것은 사용자 지정 SQL 쿼리를 입력하는 것과 비슷합니다. 하지만 전체 쿼리가 아닌 **WHERE** 절의 내용만 입력합니다. 외부 조인을 수행하면 통합 서비스가 데이터베이스 구문에 따라 쿼리의 **WHERE** 절 또는 **FROM** 절에 조인 구문을 삽입할 수 있습니다.

사용자 정의 조인을 추가하면 소스 한정자 변환이 기본 SQL 쿼리에 설정을 포함합니다. 하지만 사용자 정의 조인을 추가한 후 기본 쿼리를 수정하면 통합 서비스가 소스 한정자 변환의 SQL 쿼리 속성에 정의된 쿼리만 사용합니다.

매개 변수 또는 변수를 사용자 정의 조인으로 사용하거나 매개 변수 및 변수를 조인 내에 포함할 수 있습니다. 문자열 매핑 매개 변수 또는 변수를 포함할 경우 소스 시스템에 적절한 문자열 식별자를 사용하십시오. 대부분의 데이터베이스의 경우 문자열 매개 변수 또는 변수의 이름을 작은따옴표로 묶어야 합니다.

날짜 시간 매개 변수 또는 변수를 포함하는 경우 소스에서 사용하는 형식에 맞춰 날짜 형식을 변경해야 할 수 있습니다. 통합 서비스는 소스 시스템에 따라 날짜 시간 매개 변수 및 변수를 문자열로 변환합니다.

사용자 정의 조인을 작성하려면

1. 여러 소스 또는 연결된 소스의 데이터를 포함하는 소스 한정자 변환을 작성합니다.
2. 소스 한정자 변환을 열고 속성 탭을 클릭합니다.
3. 사용자 정의 조인 필드에서 열기 단추를 클릭합니다.

SQL 편집기 대화 상자가 나타납니다.

4. 조인 구문을 입력합니다.

조인 시작 부분에 **WHERE** 키워드를 입력하지 마십시오. 통합 서비스가 행을 쿼리할 때 이 키워드를 추가합니다.

문자열 매핑 매개 변수 및 변수를 문자열 식별자로 묶습니다. 필요한 경우 날짜 시간 매핑 매개 변수 및 변수의 날짜 형식을 변경합니다.

5. 확인을 클릭하여 변환 편집 대화 상자로 돌아간 후 확인을 클릭하여 디자이너로 돌아갑니다.

외부 조인 지원

소스 한정자 및 응용 프로그램 소스 한정자 변환을 사용하여 동일한 데이터베이스에 있는 두 소스의 외부 조인을 수행합니다. 외부 조인을 수행할 때 통합 서비스는 한 소스 테이블의 모든 행 및 조인 조건과 일치하는 두 번째 소스 테이블의 행을 반환합니다.

두 개의 테이블을 조인하고 이러한 테이블 중 하나에서 모든 행을 반환하려는 경우 외부 조인을 사용합니다. 예를 들어 등록된 고객의 테이블을 월별 구매 테이블과 조인하여 등록된 고객의 활동을 확인하려는 경우 외부 조인을 수행할 수 있습니다. 외부 조인을 사용하면 등록된 고객 테이블을 월별 구매 테이블과 조인하고 지난 달 구매를 하지 않은 고객을 포함하여 등록된 고객 테이블의 모든 행을 반환할 수 있습니다. 일반 조인을 수행하는 경우 통합 서비스는 해당 월에 구매를 한 등록된 고객과 등록된 고객의 구매만 반환합니다.

외부 조인을 사용하는 경우 조이너 변환에서 마스터 외부 조인 또는 상세 외부 조인과 동일한 결과를 생성할 수 있습니다. 하지만 외부 조인을 사용하면 데이터 흐름의 행 수를 줄일 수 있습니다. 이렇게 하려면 성능이 향상될 수 있습니다.

통합 서비스는 두 가지 종류의 외부 조인을 지원합니다.

- **왼쪽.** 통합 서비스가 조인 구문의 왼쪽에 있는 테이블의 모든 행과 조인 조건을 만족하는 두 테이블의 행을 반환합니다.
- **오른쪽.** 통합 서비스가 조인 구문의 오른쪽에 있는 테이블의 모든 행과 조인 조건을 만족하는 두 테이블의 행을 반환합니다.

참고: 기본 쿼리를 재정의하는 경우 중첩된 쿼리 문에서 외부 조인을 사용합니다.

Informatica 조인 구문

조인 구문을 입력할 때 **Informatica** 또는 데이터베이스별 조인 구문을 사용합니다. **Informatica** 조인 구문을 사용하면 통합 서비스가 세션 중에 구문을 변환하여 소스 데이터베이스에 전달합니다.

참고: 조인 조건에 대해 데이터베이스별 구문을 항상 사용합니다.

Informatica 조인 구문을 사용하는 경우 전체 조인 문을 괄호(**{Informatica syntax}**)로 둘러쌉니다. 데이터베이스 구문을 사용하는 경우 소스 데이터베이스에서 지원되는 구문을 괄호 없이 입력합니다.

Informatica 조인 구문을 사용하는 경우 테이블 이름을 접두사로 열 이름에 추가합니다. 예를 들어 REG_CUSTOMER 테이블에 FIRST_NAME이라는 이름의 열이 있을 경우 조인 구문에 "REG_CUSTOMER.FIRST_NAME"을 입력합니다. 또한 테이블 이름에 별칭을 사용하는 경우 Informatica 조인 구문 내에서 별칭을 사용하여 통합 서비스에서 별칭을 인식하도록 합니다.

다음 테이블에는 외부 조인을 작성할 때 다른 소스 한정자 변환의 여러 위치에서 입력할 수 있는 조인 구문이 나와 있습니다.

변환	변환 설정	설명
소스 한정자 변환	사용자 정의 조인	조인 재정의의 작성합니다. 통합 서비스는 기본 쿼리의 WHERE 또는 FROM 절에 조인 재정의의 추가합니다.
소스 한정자 변환	SQL 쿼리	기본 쿼리에서 WHERE 바로 다음에 조인 구문을 입력합니다.
응용 프로그램 소스 한정자 변환	조인 재정의	조인 재정의의 작성합니다. 통합 서비스는 기본 쿼리의 WHERE 절에 조인 재정의의 추가합니다.
응용 프로그램 소스 한정자 변환	추출 재정의	기본 쿼리에서 WHERE 바로 다음에 조인 구문을 입력합니다.

단일 소스 한정자에서 왼쪽 외부 조인 및 오른쪽 외부 조인을 일반 조인과 결합할 수 있습니다. 여러 일반 조인과 여러 왼쪽 외부 조인을 사용합니다.

조인을 결합할 때 다음 순서로 조인을 입력합니다.

1. 보통
2. 왼쪽 바깥쪽
3. 오른쪽 바깥쪽

참고: 일부 데이터베이스는 한 개의 오른쪽 외부 조인을 사용하도록 제한합니다.

일반 조인 구문

소스 한정자에서 조인 조건을 사용하여 일반 조인을 작성할 수 있습니다. 하지만 외부 조인을 작성할 경우 기본 조인을 재정의하여 외부 조인을 수행해야 합니다. 따라서 일반 조인을 조인 재정의에 포함해야 합니다. 일반 조인을 조인 재정의에 통합하는 경우 외부 조인 앞에 일반 조인을 나열합니다. 조인 재정의에서 여러 일반 조인을 입력할 수 있습니다.

일반 조인을 작성하려면 다음과 같은 구문을 사용합니다.

```
{ source1 INNER JOIN source2 on join_condition }
```

다음 테이블에는 조인 재정의의 일반 조인 구문이 나와 있습니다.

구문	설명
<i>source1</i>	소스 테이블 이름. 통합 서비스가 이 테이블에서 조인 조건과 일치하는 행을 반환합니다.
<i>source2</i>	소스 테이블 이름. 통합 서비스가 이 테이블에서 조인 조건과 일치하는 행을 반환합니다.
<i>join_condition</i>	조인 조건입니다. 소스 데이터베이스에서 지원되는 구문을 사용합니다. AND 연산자를 사용하여 여러 조인 조건을 결합할 수 있습니다.

예를 들어 등록된 고객에 대한 데이터가 포함된 REG_CUSTOMER 테이블이 있다고 가정합니다.

CUST_ID	FIRST_NAME	LAST_NAME
00001	Marvin	Chi
00002	Dinah	Jones
00003	John	Bowden
00004	J.	Marks

PURCHASES 테이블은 매달 업데이트되고 다음 데이터를 포함합니다.

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
06-2000-0007	NULL	6/24/2000	325.45

6월에 발생한 각 트랜잭션의 고객 이름을 표시하는 행을 반환하려면 다음과 같은 구문을 사용합니다.

```
{ REG_CUSTOMER INNER JOIN PURCHASES on REG_CUSTOMER.CUST_ID = PURCHASES.CUST_ID }
```

통합 서비스가 다음 데이터를 반환합니다.

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

통합 서비스에서 고객 ID가 일치하는 행을 반환합니다. 6월에 구매하지 않은 고객은 포함하지 않습니다. 또한 등록되지 않은 고객의 구매도 포함하지 않습니다.

왼쪽 외부 조인 구문

조인 재정의의 사용하여 왼쪽 외부 조인을 작성할 수 있습니다. 단일 조인 재정의에서 여러 개의 왼쪽 외부 조인을 입력할 수 있습니다. 왼쪽 외부 조인을 다른 조인과 함께 사용하는 경우 문에서 일반 조인 뒤에 모든 왼쪽 외부 조인을 함께 나열합니다.

왼쪽 외부 조인을 작성하려면 다음과 같은 구문을 사용합니다.

```
{ source1 LEFT OUTER JOIN source2 on join_condition }
```

다음 표에는 조인 재정의의 왼쪽 외부 조인 구문이 나와 있습니다.

구문	설명
<i>source1</i>	소스 테이블 이름. 왼쪽 외부 조인을 사용하면 통합 서비스가 이 테이블의 모든 행을 반환합니다.
<i>source2</i>	소스 테이블 이름. 통합 서비스가 이 테이블에서 조인 조건과 일치하는 행을 반환합니다.
<i>join_condition</i>	조인 조건입니다. 소스 데이터베이스에서 지원되는 구문을 사용합니다. AND 연산자를 사용하여 여러 조인 조건을 결합할 수 있습니다.

예를 들어 “일반 조인 구문” 페이지 372에 설명된 REG_CUSTOMER 및 PURCHASES 테이블을 사용하여 다음 조인 재정의로 6월에 무언가 구입한 고객의 수를 확인할 수 있습니다.

```
{ REG_CUSTOMER LEFT OUTER JOIN PURCHASES on REG_CUSTOMER.CUST_ID = PURCHASES.CUST_ID }
```

통합 서비스가 다음 데이터를 반환합니다.

CUST_ID	FIRST_NAME	LAST_NAME	DATE	AMOUNT
00001	Marvin	Chi	6/10/2000	255.56
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

통합 서비스가 6월에 구매하지 않은 고객에 null 값을 사용하여 REG_CUSTOMERS 테이블의 모든 등록 고객을 반환합니다. 등록되지 않은 고객의 구입은 포함하지 않습니다.

여러 개의 조인 조건을 사용하여 6월의 단일 구매에서 100달러가 넘게 소비한 등록된 고객의 수를 확인할 수 있습니다.

```
{REG_CUSTOMER LEFT OUTER JOIN PURCHASES on (REG_CUSTOMER.CUST_ID = PURCHASES.CUST_ID AND PURCHASES.AMOUNT > 100.00) }
```

통합 서비스가 다음 데이터를 반환합니다.

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

동일한 기간 동안의 반품에 대한 정보를 통합하려면 여러 개의 왼쪽 외부 조인을 사용할 수 있습니다. 예를 들어 RETURNS 테이블에 다음과 같은 데이터가 있습니다.

CUST_ID	CUST_ID	RETURN
00002	6/10/2000	55.79
00002	6/21/2000	104.45

6월에 구매하고 반품한 고객의 수를 확인하려면 두 개의 왼쪽 외부 조인을 사용합니다.

```
{ 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 }
```

통합 서비스가 다음 데이터를 반환합니다.

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
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

통합 서비스가 누락된 값에 NULL을 사용합니다.

오른쪽 외부 조인 구문

조인 재정의를 사용하여 오른쪽 외부 조인을 작성할 수 있습니다. 조인 구문에서 테이블의 순서를 반대로 지정하면 오른쪽 외부 조인이 왼쪽 외부 조인과 동일한 결과를 반환합니다. 조인 재정의에서 한 개의 오른쪽 외부 조인만 사용합니다. 오른쪽 외부 조인을 두 개 이상 작성하려는 경우 소스 테이블의 순서를 반대로 지정하고 조인 유형을 왼쪽 외부 조인으로 변경해 봅니다.

오른쪽 외부 조인을 다른 조인과 함께 사용하려면 조인 재정의의 끝에 오른쪽 외부 조인을 입력합니다.

오른쪽 외부 조인을 작성하려면 다음과 같은 구문을 사용합니다.

```
{ source1 RIGHT OUTER JOIN source2 on join_condition }
```

다음 표에는 조인 재정의의 오른쪽 외부 조인 구문이 나와 있습니다.

구문	설명
<i>source1</i>	소스 테이블 이름. 통합 서비스가 이 테이블에서 조인 조건과 일치하는 행을 반환합니다.
<i>source2</i>	소스 테이블 이름. 오른쪽 외부 조인을 사용하면 통합 서비스가 이 테이블의 모든 행을 반환합니다.
<i>join_condition</i>	조인 조건입니다. 소스 데이터베이스에서 지원되는 구문을 사용합니다. AND 연산자를 사용하여 여러 조인 조건을 결합할 수 있습니다.

오른쪽 외부 조인을 왼쪽 외부 조인과 함께 사용하여 두 테이블의 모든 데이터를 조인하고 반환함으로써 전체 외부 조인과 같은 결과를 얻을 수 있습니다. 예를 들어 다음과 같은 조인 재정의의 통해 등록된 모든 고객과 6월에 있었던 모든 구매를 추출할 수 있습니다.

```
{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 }
```

통합 서비스가 다음 데이터를 반환합니다.

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
NULL	NULL	NULL	06-2000-0006	6/23/2000	155.65
NULL	NULL	NULL	06-2000-0007	6/24/2000	325.45

외부 조인 작성

조인 재정의 또는 기본 쿼리 재정의의 일부로 외부 조인을 입력할 수 있습니다.

조인 재정의의 작성하면 디자이너가 기본 쿼리의 **WHERE** 절에 조인 재정의의 추가합니다. 세션 중에 통합 서비스가 **Informatica** 조인 구문을 변환하고 소스 데이터를 추출하는 데 사용되는 기본 쿼리에 이 구문을 포함시킵니다. 가능한 경우, 기본 쿼리를 재정의하는 대신 조인 재정의의를 입력합니다.

기본 쿼리를 재정의할 때 기본 쿼리의 **WHERE** 절에서 조인 구문을 입력합니다. 세션 중에 통합 서비스가 **Informatica** 조인 구문을 변환하고 소스 데이터를 추출하는 쿼리를 사용합니다. 재정의의를 작성한 후에 변환을 변경하면 통합 서비스가 이 변경 내용을 무시합니다. 따라서 가능하면, 외부 조인 구문을 조인 재정의의로 입력합니다.

외부 조인을 조인 재정의로 작성

외부 조인을 조인 재정의로 작성하려면:

1. 소스 한정자 변환을 열고 속성 탭을 클릭합니다.

2. 소스 한정자 변환에서 사용자 정의 조인 필드의 단추를 클릭합니다.
응용 프로그램 소스 한정자 변환에서 조인 재정의 필드의 단추를 클릭합니다.
3. 조인 구문을 입력합니다.
조인 시작 부분에 **WHERE**를 입력하지 마십시오. 통합 서비스가 행을 쿼리할 때 이 항목을 추가합니다.
Informatica 조인 구문을 괄호({ })로 묶습니다.
테이블 및 **Informatica** 조인 구문에 별칭을 사용하는 경우 **Informatica** 조인 구문 내에서 해당 별칭을 사용합니다.
테이블 이름을 사용하여 열 이름에 접두사를 추가합니다(예: "테이블.열").
소스 데이터베이스에서 지원하는 조인 조건을 사용합니다.
여러 개의 조인을 입력하는 경우 유형별로 조인을 그룹화한 후 일반, 왼쪽 외부, 오른쪽 외부의 순서로 조인을 나열합니다. 중첩 쿼리별로 하나의 오른쪽 외부 조인만 포함합니다.
정확성을 위해 포트 탭에서 포트 이름을 선택합니다.
4. 확인을 클릭합니다.

외부 조인을 추출 재정의로 작성

외부 조인을 추출 재정의로 작성하려면

1. 응용 프로그램 소스 한정자 변환에 대한 입력 및 출력 포트를 연결한 후 변환의 제목 표시줄을 두 번 클릭하고 속성 탭을 선택합니다.
2. 응용 프로그램 소스 한정자 변환의 추출 재정의 필드에서 단추를 클릭합니다.
3. **SQL** 생성을 클릭합니다.
4. **WHERE** 절에서 **WHERE** 바로 뒤에 조인 구문을 입력합니다.
Informatica 조인 구문을 괄호({ })로 묶습니다.
테이블 및 **Informatica** 조인 구문에 별칭을 사용하는 경우 **Informatica** 조인 구문 내에서 해당 별칭을 사용합니다.
테이블 이름을 사용하여 열 이름에 접두사를 추가합니다(예: "테이블.열").
소스 데이터베이스에서 지원하는 조인 조건을 사용합니다.
여러 개의 조인을 입력하는 경우 유형별로 조인을 그룹화한 후 일반, 왼쪽 외부, 오른쪽 외부의 순서로 조인을 나열합니다. 중첩 쿼리별로 하나의 오른쪽 외부 조인만 포함합니다.
정확성을 위해 포트 탭에서 포트 이름을 선택합니다.
5. 확인을 클릭합니다.

일반적인 데이터베이스 구문 제한 사항

데이터베이스별로 외부 조인 구문에 대한 제한 사항이 다릅니다. 외부 조인을 작성할 때 다음 제한 사항을 고려하십시오.

- 외부 조인 구문의 **ON** 절에서 **OR** 연산자와 조인 조건을 함께 결합하지 마십시오.
- 외부 조인 구문의 **ON** 절에서 열을 비교하는 데 **IN** 연산자를 사용하지 마십시오.
- 외부 조인 구문의 **ON** 절에 있는 하위 쿼리와 열을 비교하지 마십시오.
- 두 개 이상의 외부 조인을 결합하는 경우 동일한 테이블을 두 개 이상의 외부 조인의 내부 테이블로 사용하지 마십시오. 예를 들어 다음과 같은 외부 조인을 사용하지 마십시오.

```
{ 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}
```

- 일반 조인 조건에서 외부 조인의 두 테이블을 모두 사용하지 마십시오. 예를 들어 다음과 같은 조인 조건을 사용하지 마십시오.

```
{ TABLE1 LEFT OUTER JOIN TABLE2 ON TABLE1.COLUMNA = TABLE2.COLUMNA WHERE TABLE1.COLUMNB =
TABLE2.COLUMNC}
```

하지만 다음과 같은 필터 조건에서는 두 테이블을 모두 사용하십시오.

```
{ TABLE1 LEFT OUTER JOIN TABLE2 ON TABLE1.COLUMNA = TABLE2.COLUMNA WHERE TABLE1.COLUMNB = 32 AND
TABLE2.COLUMNC > 0}
```

참고: ON 절에 조건을 입력하면 WHERE 절에 동일한 조건을 입력할 때와 다른 결과가 반환될 수 있습니다.

- 테이블에 별칭을 사용하는 경우 별칭을 사용하여 테이블의 열에 접두사를 추가하십시오. 예를 들어 REG_CUSTOMER 테이블을 C라고 할 경우 FIRST_NAME 열을 참조할 때 "C.FIRST_NAME"을 사용하십시오.

소스 필터 입력

소스 필터를 입력하여 통합 서비스가 쿼리하는 행 수를 줄일 수 있습니다. 소스 필터에 문자열 'WHERE' 또는 대 형 개체가 포함된 경우 통합 서비스에서 세션이 실패합니다.

매핑의 소스 한정자 변환에 소스 필터를 추가하는 경우 기본 SQL 쿼리에 필터 조건이 포함됩니다. 하지만 소스 필터를 추가한 후 기본 쿼리를 수정하는 경우 통합 서비스가 소스 한정자 변환의 SQL 쿼리 부분에 정의된 쿼리 만 사용합니다.

매개 변수 또는 변수를 소스 필터로 사용하거나 소스 필터 내에 매개 변수 및 변수를 포함시킬 수 있습니다. 문자 열 매핑 매개 변수 또는 변수를 포함할 경우 소스 시스템에 적절한 문자열 식별자를 사용하십시오. 대부분의 데이터베이스의 경우 문자열 매개 변수 또는 변수의 이름을 작은따옴표로 묶어야 합니다.

날짜 시간 매개 변수 또는 변수를 포함하는 경우 소스에서 사용하는 형식에 맞춰 날짜 형식을 변경해야 할 수 있습니다. 통합 서비스는 소스 시스템에 따라 날짜 시간 매개 변수 및 변수를 문자열로 변환합니다.

참고: 세션 속성에 SQL 쿼리를 입력할 때 매핑 수준에서 SQL 쿼리 및 필터 조건을 재정의합니다.

소스 필터를 입력하려면:

1. 매핑 디자이너에서 소스 한정자 변환을 엽니다.
변환 편집 대화 상자가 표시됩니다.
2. 속성 탭을 선택합니다.
3. 소스 필터 필드에서 열기 단추를 클릭합니다.
4. SQL 편집기 대화 상자에 필터를 입력합니다.

테이블 이름 및 포트 이름을 포함시킵니다. 필터에 WHERE 키워드를 포함하지 않습니다.

문자열 매핑 매개 변수 및 변수를 문자열 식별자로 묶습니다. 필요한 경우 날짜 시간 매핑 매개 변수 및 변수의 날짜 형식을 변경합니다.

5. 확인을 클릭합니다.

정렬된 포트 사용

정렬된 포트를 사용할 경우 통합 서비스는 해당 포트를 기본 쿼리의 **ORDER BY** 절에 추가합니다. 통합 서비스는 소스 한정자 변환의 맨 위에 있는 포트부터 시작하여 구성된 개수의 포트를 추가합니다. 포트의 하위 집합이 다 운스트림 연결되어 있는 경우 기본 쿼리는 포트의 하위 집합만 포함합니다. 정렬된 포트는 소스 한정자 변환의 맨 위에서 시작하는 포트에 적용되지 않고 연결된 포트에 적용됩니다.

다음 변환 중 하나를 매핑에 포함할 때 성능을 향상시키기 위해 정렬된 포트를 사용할 수 있습니다.

- **집계.** 정렬된 입력에 대해 집계 변환을 구성하는 경우 정렬된 포트를 사용하여 정렬된 데이터를 보낼 수 있습니다. 집계 변환의 그룹 기준 포트가 소스 한정자 변환에 있는 정렬된 포트의 순서와 일치해야 합니다.
- **조이너.** 정렬된 입력에 대해 조이너 변환을 구성하는 경우 정렬된 포트를 사용하여 정렬된 데이터를 보낼 수 있습니다. 정렬된 포트의 순서는 각 소스 한정자 변환에서 동일하게 구성합니다.

참고: 또한 분류기 변환을 사용하여 집계 및 조이너 변환 전에 관계형 및 플랫폼 파일 데이터를 정렬할 수 있습니다.

정렬된 포트는 관계형 소스에 대해서만 사용하십시오. 정렬된 포트를 사용할 경우 소스 데이터베이스의 정렬 순서는 세션에 대해 구성된 정렬 순서와 일치해야 합니다. 통합 서비스는 정렬된 포트에 대한 **ORDER BY** 절을 포함하여 소스 데이터 추출에 사용되는 **SQL** 쿼리를 작성합니다. 데이터베이스 서버는 쿼리를 수행하고 결과 데이터를 통합 서비스에 전달합니다. 통합 서비스에서 요청하는 대로 데이터가 정렬되도록 하려면 데이터베이스 정렬 순서가 사용자 정의 세션 정렬 순서와 동일해야 합니다.

데이터 코드 페이지 유효성 검사를 하도록 통합 서비스를 구성하고 유니코드 데이터 이동 모드로 워크플로우를 실행하면 통합 서비스가 선택된 정렬 순서를 사용하여 문자 데이터를 정렬합니다.

낮은 수준의 데이터 코드 페이지 유효성 검사를 하도록 통합 서비스를 구성한 경우, 통합 서비스는 선택된 정렬 순서의 언어 범위에 들어 있는 모든 문자 데이터를 정렬하는 데 선택된 정렬 순서를 사용합니다. 통합 서비스는 선택된 정렬 순서의 언어 범위 밖에 있는 모든 문자 데이터를 표준 유니코드 정렬 순서에 따라 정렬합니다.

통합 서비스는 **ASCII** 모드로 실행될 경우, 이 설정을 무시하고 모든 문자 데이터를 이진 정렬 순서를 사용해 정렬합니다. 기본 정렬 순서는 통합 서비스의 코드 페이지에 따라 달라집니다.

소스 한정자 변환은 정렬된 포트 수를 기본 **SQL** 쿼리에 포함합니다. 그러나 정렬된 포트 수를 선택한 후에 기본 쿼리를 수정하면 통합 서비스는 **SQL** 쿼리 속성에 정의된 쿼리만 사용합니다.

정렬된 포트를 사용하려면 다음을 수행하십시오.

1. 매핑 디자이너에서 소스 한정자 변환을 열고 속성 탭을 클릭합니다.
2. 정렬된 포트 수를 클릭하고 정렬하려는 포트 수를 입력합니다.

통합 서비스는 소스 한정자 변환의 맨 위에 있는 열부터 시작하여 구성된 개수의 열을 **ORDER BY** 절에 추가합니다.

소스 데이터베이스 정렬 순서는 세션 정렬 순서와 일치해야 합니다.

팁: Sybase는 최대 16개의 열을 **ORDER BY** 절에서 지원합니다. 소스가 Sybase이면 17개 이상의 열을 정렬하지 마십시오.

3. 확인을 클릭합니다.

고유 항목 선택

통합 서비스가 소스에서 고유한 값을 선택하도록 하려면 고유 항목 선택 옵션을 사용합니다. 총 매출을 나열하는 테이블에서 고유한 고객 ID를 추출하려는 경우에 이 기능을 사용할 수 있습니다. 고유 항목 선택을 사용하면 불필요한 데이터를 데이터 흐름의 초기에서 필터링하여 성능이 향상될 수 있습니다.

기본적으로 디자이너는 **SELECT** 문을 생성합니다. 고유 항목 선택을 선택한 경우 소스 한정자 변환은 해당 설정을 기본 **SQL** 쿼리에 포함합니다.

[“소스 데이터 조인” 페이지 367](#)에 있는 소스 한정자 변환에서 고유 항목 선택 옵션을 활성화하는 경우를 예로 들어 보겠습니다. 디자이너는 다음과 같이 **SELECT DISTINCT**를 기본 쿼리에 추가합니다.

```
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
```

그러나 고유 항목 선택을 선택한 후에 기본 쿼리를 수정하면 통합 서비스는 **SQL** 쿼리 속성에 정의된 쿼리만 사용합니다. 즉, **SQL** 쿼리가 고유 항목 선택 설정을 무시하는 것입니다.

고유 항목 선택을 사용하려면 다음을 수행하십시오.

1. 매핑에서 소스 한정자 변환을 열고 속성 탭을 클릭합니다.
2. 고유 항목 선택을 선택하고 확인을 클릭합니다.

세션에서 고유 항목 선택 재정의

워크플로우 관리자에서 세션을 구성할 때 변환 수준 옵션을 고유 항목 선택으로 재정의할 수 있습니다.

고유 항목 선택 옵션을 재정의하려면 다음을 수행하십시오.

1. 워크플로우 관리자에서 세션 태스크를 열고 매핑 탭을 클릭합니다.
2. 변환 보기를 클릭하고 소스 노드 아래에서 소스 한정자 변환을 클릭합니다.
3. 속성 설정에서 고유 항목 선택을 활성화하고 확인을 클릭합니다.

사전 세션 및 사후 세션 SQL 명령 추가

소스 한정자 변환의 속성 탭에서 사전 세션 및 사후 세션 **SQL** 명령을 추가할 수 있습니다. 사전 세션 **SQL**을 사용하여 세션이 시작될 때 소스 테이블에 타임스탬프 행을 작성할 수 있습니다.

통합 서비스는 소스를 읽기 전에 소스 데이터베이스에 대해 사전 세션 **SQL** 명령을 실행하고, 대상에 기록한 후 소스 데이터베이스에 대해 사후 세션 **SQL** 명령을 실행합니다.

세션 속성의 매핑 탭에서 변환 보기의 **SQL** 명령을 재정의할 수 있습니다. 또한 사전 세션 또는 사후 세션 **SQL** 명령을 실행하는 중 오류가 발생하는 경우 중지 또는 계속하도록 통합 서비스를 구성할 수 있습니다.

소스 한정자 변환에서 사전 세션 및 사후 세션 **SQL** 명령을 입력할 때 다음 지침을 따르십시오.

- 데이터베이스 유형에 유효한 명령을 사용합니다. 데이터베이스가 중첩된 설명을 허용해도 통합 서비스는 이러한 설명을 허용하지 않습니다.
- 매개 변수 및 변수를 소스 사전 세션 및 사후 세션 **SQL** 명령에서 사용할 수도 있고 매개 변수 또는 변수를 명령으로 사용할 수도 있습니다. 매개 변수 파일에 정의할 수 있는 모든 매개 변수 또는 변수 유형을 사용할 수 있습니다.
- 세미콜론을 사용하여 여러 개의 문을 구분합니다. 통합 서비스는 각 문 후에 커밋을 실행합니다.
- 통합 서비스는 /* ... */ 내에 포함된 세미콜론은 무시합니다.

- 주석 외부에서 세미콜론을 사용해야 하는 경우 역슬래시(\)를 사용하여 이스케이프할 수 있습니다. 세미콜론을 이스케이프하면 통합 서비스가 역슬래시를 무시하며 세미콜론을 문 구분 기호로 사용하지 않습니다.
- Designer는 SQL의 유효성을 검사하지 않습니다.

참고: 또한 매핑에 있는 대상 인스턴스의 속성 탭에서 사전 세션 및 사후 세션 SQL 명령을 입력할 수 있습니다.

소스 한정자 변환 작성

소스를 매핑으로 끌어서 놓을 때 기본적으로 소스 한정자 변환을 작성하도록 디자이너를 구성하거나 수동으로 소스 한정자 변환을 작성할 수 있습니다.

수동으로 소스 한정자 변환 작성

매핑 디자이너에서 수동으로 소스 한정자 변환을 작성할 수 있습니다.

수동으로 소스 한정자 변환을 작성하려면

1. 매핑 디자이너에서 변환 > 작성을 클릭합니다.
2. 변환 이름을 입력하고 작성을 클릭합니다.
3. 소스를 선택하고 확인을 클릭합니다.
4. 완료를 클릭합니다.

소스 한정자 변환 옵션 구성

소스 한정자 변환을 작성한 후 몇 가지 옵션을 구성할 수 있습니다.

소스 한정자 변환을 구성하려면

1. 디자이너에서 매핑을 엽니다.
2. 소스 한정자 변환의 제목 표시줄을 두 번 클릭합니다.
3. 변환 편집 대화 상자에서 이름 바꾸기를 클릭하고 변환에 대한 설명 이름을 입력한 후 확인을 클릭합니다.
소스 한정자 변환의 이름 지정 규칙은 `SQ_TransformationName`(예: `SQ_AllSources`)입니다.
4. 속성 탭을 클릭합니다.
5. 소스 한정자 변환 속성을 입력합니다.
6. 소스 탭을 클릭하고 이 변환에 대해 정의하려는 연결된 소스 정의를 지정합니다.
여러 데이터베이스 또는 플랫폼 파일 시스템의 데이터를 조인해야 할 경우에만 연결된 소스를 식별합니다.
7. 확인을 클릭합니다.

소스 한정자 변환 문제 해결

포트 연결 같은 끌어서 놓기 작업을 수행할 수 없습니다.

상태 표시줄의 오류 메시지에서 세부 정보를 검토하십시오.

소스 정의를 대상 정의에 연결할 수 없습니다.

소스를 대상에 직접 연결할 수는 없습니다. 대신, 관계형 및 플랫폼 파일 소스의 경우 소스 한정자 변환을 통해 연결하고 COBOL 소스의 경우 노멀라이저 변환을 통해 연결해야 합니다.

여러 소스를 하나의 대상에 연결할 수 없습니다.

디자이너에서는 여러 개의 소스 한정자 변환을 단일 대상에 연결하도록 허용하지 않습니다. 두 가지 해결 방법이 있습니다.

- **대상을 재사용합니다.** 대상 정의가 재사용 가능하므로 동일한 대상을 매핑에 여러 번 추가할 수 있습니다. 그런 다음 각 소스 한정자 변환을 각 대상에 연결합니다.
- **소스를 소스 한정자 변환에 조인합니다.** 그런 다음, SQL 쿼리에서 WHERE 절을 제거합니다.

소스에서 일부 열에 QNAN(숫자가 아님) 값이 있지만 대상에서는 1.#QNAN이 표시됩니다.

운영 체제별로 NaN의 문자열 표현이 서로 다릅니다. 통합 서비스는 Win64EMT 플랫폼에서 QNAN 값을 1.#QNAN으로 변환합니다. 1.#QNAN은 QNAN의 올바른 표현입니다.

사용자 지정 쿼리를 입력했지만, 세션을 포함하는 워크플로우를 실행하면 해당 쿼리가 작동하지 않습니다.

워크플로우를 실행하기 전에 소스 한정자 변환에 대해 이 설정을 테스트해야 합니다. 소스 한정자 변환으로 돌아가 사용자 지정 쿼리를 입력했던 대화 상자를 다시 엽니다. 데이터베이스에 연결하고 유효성 검사 단추를 클릭하여 SQL을 테스트할 수 있습니다. 디자이너에서 오류가 표시됩니다. 더 자세한 정보가 필요하면 세션 로그 파일을 검토하십시오.

세션이 실패하는 가장 일반적인 이유는 세션 및 소스 한정자 변환에서 데이터베이스 로그인인 테이블 소유자가 아니기 때문입니다. 세션에서 테이블 소유자를 지정해야 하고 소스 한정자 변환에서 SQL 쿼리를 생성할 때도 테이블 소유자를 지정해야 합니다.

SQL 쿼리를 잘라내어 데이터베이스 클라이언트 도구(예: Oracle Net)에 붙여 넣고 오류 반환 여부를 확인하는 방법으로 SQL 쿼리를 테스트할 수 있습니다.

소스 필터에서 매핑 변수를 사용했는데 세션이 실패합니다.

소스 한정자 변환에서 SQL을 생성하고 유효성 검사를 수행하여 쿼리를 테스트해 보십시오. 변수 또는 매개 변수가 문자열인 경우 작은따옴표로 문자열을 둘러싸야 할 수도 있습니다. 날짜/시간 변수 또는 매개 변수이면 소스 시스템의 날짜/시간 형식을 변경해야 할 수도 있습니다.

제 26 장

SQL 변환

이 장에 포함된 항목:

- [SQL 변환 개요, 383](#)
- [스크립트 모드, 384](#)
- [쿼리 모드, 385](#)
- [데이터베이스에 연결, 390](#)
- [세션 처리, 393](#)
- [입력 행 대 출력 행의 카디널리티, 396](#)
- [SQL 변환 속성, 400](#)
- [SQL 문, 403](#)
- [SQL 변환 작성, 404](#)

SQL 변환 개요

SQL 변환은 파이프라인에서 SQL 쿼리 미드스트림을 처리합니다. SQL 변환은 활성 또는 수동 변환일 수 있습니다. 데이터베이스의 행을 삽입, 삭제, 업데이트 및 검색할 수 있습니다. 데이터베이스 연결 정보를 런타임에 입력 데이터로 SQL 변환에 전달할 수 있습니다. 이 변환은 SQL 편집기에서 작성한 외부 SQL 스크립트 또는 SQL 쿼리를 처리합니다. SQL 변환은 쿼리를 처리하여 행 및 데이터베이스 오류를 반환합니다.

예를 들어 새 트랜잭션을 추가하기 전에 데이터베이스 테이블을 작성해야 하는 경우가 있습니다. 워크플로우에서 테이블을 작성하는 SQL 변환을 작성할 수 있습니다. SQL 변환은 출력 포트에서 데이터베이스 오류를 반환합니다. SQL 변환이 오류를 반환하지 않는 경우 다른 워크플로우를 실행하도록 구성할 수 있습니다.

SQL 변환을 작성할 때 다음 옵션을 구성합니다.

- **모드.** SQL 변환은 다음 모드 중 하나로 실행됩니다.
 - **스크립트 모드.** SQL 변환이 외부에 위치하는 ANSI SQL 스크립트를 실행합니다. 각 입력 행에서 변환에 스크립트 이름을 전달합니다. SQL 변환은 입력 행마다 행 하나를 출력합니다.
 - **쿼리 모드.** SQL 변환은 쿼리 편집기에서 정의된 쿼리를 실행합니다. 문자열 또는 매개 변수를 쿼리로 전달하여 동적 쿼리를 정의하거나 선택 매개 변수를 변경할 수 있습니다. 쿼리에 SELECT 문이 있는 경우 여러 행을 출력할 수 있습니다.
- **수동 또는 활성 변환.** SQL 변환은 기본적으로 활성 변환입니다. 변환을 작성할 때 수동 변환으로 구성할 수 있습니다.
- **데이터베이스 유형.** SQL 변환이 연결되는 데이터베이스의 유형입니다.
- **연결 유형.** 데이터베이스 연결 정보를 SQL 변환으로 전달하거나 연결 개체를 사용합니다.

스크립트 모드

스크립트 모드에서 실행되는 SQL 변환은 텍스트 파일의 SQL 스크립트를 실행합니다. 소스에서 SQL 변환의 ScriptName 포트로 각 스크립트 파일 이름을 전달합니다. 스크립트 파일 이름에는 스크립트 파일에 대한 전체 경로가 포함됩니다.

변환을 스크립트 모드로 실행되도록 구성할 경우 수동 변환을 작성합니다. 변환은 입력 행마다 행 하나를 반환합니다. 출력 행에는 쿼리 결과와 모든 데이터베이스 오류가 포함되어 있습니다.

SQL 변환이 스크립트 모드로 실행되는 경우 쿼리 문 및 쿼리 데이터가 변경되지 않습니다. 스크립트 모드에서 다른 쿼리를 실행해야 하는 경우 소스 데이터에서 스크립트를 전달합니다. 스크립트 모드를 사용하여 테이블 작성 또는 삭제와 같은 데이터 정의 쿼리를 실행합니다.

스크립트 모드에서 실행되도록 SQL 변환을 구성하는 경우 디자이너가 변환에 ScriptName 입력 포트를 추가합니다. 매핑 작성 시 ScriptName 포트를 각 행에 대해 실행할 스크립트 이름이 포함된 포트에 연결합니다. 각 입력 행에 대해 서로 다른 SQL 스크립트를 실행할 수 있습니다. 디자이너는 쿼리 결과에 대한 정보를 반환하는 기본 포트를 작성합니다.

스크립트 모드로 구성된 SQL 변환에는 다음과 같은 기본 포트가 있습니다.

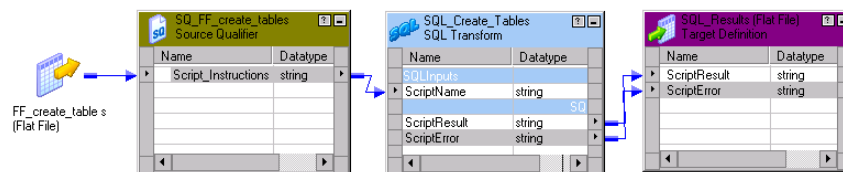
포트	유형	설명
ScriptName	입력	현재 행에 대해 실행할 스크립트의 이름을 받습니다.
ScriptResult	출력	행에 대한 스크립트 실행이 성공한 경우 PASSED를 반환합니다. 그렇지 않은 경우 FAILED를 포함합니다.
ScriptError	출력	행에 대한 스크립트가 실패한 경우 발생한 오류를 반환합니다.

예제

새 데이터를 테이블에 추가하기 전에 순서 및 인벤토리 테이블을 작성해야 합니다. SQL 스크립트를 작성하여 테이블을 작성하고 SQL 변환을 구성하여 이 스크립트를 실행할 수 있습니다.

테이블을 작성하기 위한 SQL 문이 포함된 create_order_inventory.txt라는 파일을 작성합니다.

다음 매핑에는 SQL 변환에 스크립트 이름을 전달하는 방법이 나와 있습니다.



통합 서비스가 소스에서 행을 읽습니다. 소스 행에는 SQL 스크립트 파일 이름 및 경로가 포함되어 있습니다.

C:\81\server\shared\SrcFiles\create_order_inventory.txt

변환에서 ScriptName 포트의 파일 이름을 수신합니다. 통합 서비스가 스크립트 파일을 찾고 이 스크립트를 구문 분석합니다. SQL 프로시저를 작성하고 이 프로시저를 데이터베이스에 보내 처리합니다. 데이터베이스가 SQL의 유효성을 검사하고 쿼리를 실행합니다.

SQL 변환이 ScriptResults 및 ScriptError를 반환합니다. 스크립트가 실행되면 ScriptResult 출력 포트가 전달된 PASSED를 반환합니다. 그렇지 않으면 ScriptResult 포트가 FAILED 상태를 반환합니다. ScriptResult가 FAILED인 경우 SQL 변환에서 ScriptError 포트의 오류 메시지를 반환합니다. SQL 변환이 수신하는 각 입력 행에 대해 하나의 행을 반환합니다.

스크립트 모드에 대한 규칙 및 지침

스크립트 모드에서 실행되는 SQL 변환의 경우 다음 규칙 및 지침을 사용하십시오.

- 스크립트 모드에는 정적 또는 동적 데이터베이스 연결을 사용할 수 있습니다.
- 스크립트에 여러 쿼리 문을 포함시키려면 세미콜론으로 쿼리 문을 구분할 수 있습니다.
- 스크립트 파일 이름에서 매핑 변수 또는 매개 변수를 사용할 수 있습니다.
- 스크립트 코드 페이지는 기본적으로 운영 체제 로캘입니다. 스크립트의 로캘을 변경할 수 있습니다.
- 통합 서비스에서 스크립트 파일에 액세스할 수 있어야 합니다. 통합 서비스에는 스크립트가 포함된 디렉터리에 대한 읽기 사용 권한이 있어야 합니다. 통합 서비스가 운영 체제 프로필을 사용하는 경우 운영 체제 프로필의 운영 체제 사용자에게 스크립트가 포함된 디렉터리에 대한 읽기 사용 권한이 있어야 합니다.
- 통합 서비스는 SQL 스크립트에 포함되어 있는 모든 SELECT 문의 출력을 무시합니다. 스크립트 모드의 SQL 변환은 각 입력 행에 대해 하나 이상의 데이터 행을 출력하지 않습니다.
- 스크립트에서 Oracle PL/SQL 또는 Microsoft/Sybase T-SQL 같은 스크립팅 언어를 사용할 수 없습니다.
- SQL 스크립트가 다른 SQL 스크립트를 호출하는 중첩된 스크립트를 사용할 수 없습니다.
- 스크립트가 런타임 인수를 받을 수 없습니다.

쿼리 모드

SQL 변환이 쿼리 모드에서 실행되는 경우 변환에 정의되어 있는 SQL 쿼리를 실행합니다. 변환 입력 포트에서 쿼리 문자열 또는 매개 변수를 전달하여 쿼리 문 또는 쿼리 데이터를 변경합니다.

SQL 변환을 쿼리 모드로 실행되도록 구성할 경우 활성 변환을 작성합니다. 변환에서는 각 입력 행에 대해 여러 행을 반환할 수 있습니다.

SQL 변환의 SQL 편집기에서 쿼리를 작성합니다. 쿼리를 작성하려면 SQL 편집기의 기본 창에 쿼리 문을 입력합니다. SQL 편집기는 쿼리에서 참조할 수 있는 변환 포트의 목록을 제공합니다. 포트 이름을 두 번 클릭하여 쿼리 매개 변수로 추가할 수 있습니다.

쿼리를 생성하면 SQL 편집기가 쿼리에서 포트 이름의 유효성을 검사합니다. 또한 문자열 대체에 사용하는 포트가 문자열 데이터 유형인지도 확인합니다. SQL 편집기는 SQL 쿼리의 구문 유효성은 검사하지 않습니다.

SQL 변환에서 다음 유형의 SQL 쿼리를 작성할 수 있습니다.

- **정적 SQL 쿼리.** 쿼리 문은 변경되지 않지만 쿼리 매개 변수를 사용하여 데이터를 변경할 수 있습니다. 통합 서비스는 쿼리를 한 번만 준비하고 모든 입력 행에 대해 해당 쿼리를 실행합니다.
- **동적 SQL 쿼리.** 쿼리 문 및 데이터를 변경할 수 있습니다. 통합 서비스는 각 입력 행에 대해 쿼리를 준비합니다.

정적 쿼리를 작성하면 통합 서비스가 SQL 프로시저를 한 번만 준비하고 각 행에 대해 해당 프로시저를 실행합니다. 동적 쿼리를 작성하면 통합 서비스가 각 입력 행마다 SQL을 준비합니다. 정적 쿼리를 작성하면 성능을 최적화할 수 있습니다.

정적 SQL 쿼리 사용

각 입력 행에 대해 동일한 쿼리 문을 실행하지만 각 입력 행의 쿼리에서 데이터를 변경하려는 경우 정적 SQL 쿼리를 작성합니다. 정적 SQL 쿼리를 작성할 경우 SQL 편집기에서 매개 변수 바인딩을 사용하여 쿼리 데이터로 사용할 매개 변수를 정의합니다.

쿼리에서 데이터를 변경하려면 쿼리 매개 변수를 구성하고 변환의 입력 포트에 해당 매개 변수를 바인딩합니다. 입력 포트에 매개 변수를 바인딩할 때는 쿼리의 이름을 기준으로 포트를 식별합니다. SQL 편집기에서는 이름을 물음표(?)로 묶습니다. 쿼리 데이터는 입력 포트의 데이터 값에 따라 변경됩니다.

SQL 변환의 입력 포트는 쿼리의 데이터 값에 대한 데이터나 쿼리의 WHERE 절에 포함된 값을 받습니다.

다음 정적 쿼리에서는 매개 변수 바인딩을 사용합니다.

```
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
```

다음 정적 SQL 쿼리에는 SQL 변환의 Employee_ID 및 Dept 입력 포트에 바인딩되는 쿼리 매개 변수가 있습니다.

```
SELECT Name, Address FROM Employees WHERE Employee_Num =?Employee_ID? and Dept = ?Dept?
```

소스에는 다음 행이 포함될 수 있습니다.

Employee_ID	Dept
100	Products
123	HR
130	Accounting

통합 서비스는 행에서 다음 쿼리 문을 생성합니다.

```
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'
```

여러 데이터베이스 행 선택

SQL 쿼리에 SELECT 문이 포함된 경우 변환은 검색하는 데이터베이스 행 하나마다 행 하나를 반환합니다.

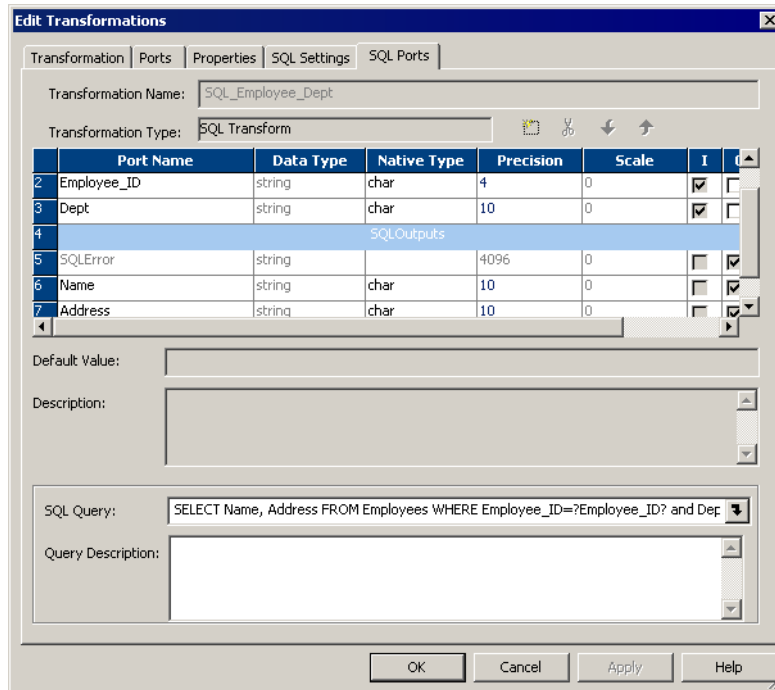
SELECT 문에서 각 열에 대한 출력 포트를 구성해야 합니다. 출력 포트는 SELECT 문의 열과 동일한 순서여야 합니다.

데이터베이스 열에 대한 출력 포트를 구성할 경우 선택한 각 데이터베이스 열의 데이터 유형을 구성해야 합니다. 목록에서 원시 데이터 유형을 선택합니다. 원시 데이터 유형을 선택하는 경우 디자이너가 자동으로 변환 데이터 유형을 구성합니다.

변환의 원시 데이터 유형은 데이터베이스 열 데이터 유형과 일치해야 합니다. 통합 서비스는 런타임에 데이터베이스의 열 데이터 유형을 변환의 원시 데이터베이스 유형과 일치시킵니다. 데이터 유형이 일치하지 않으면 통합 서비스가 행 오류를 생성합니다.

참고: Teradata 데이터베이스에서는 Bigint 열을 사용할 수 있지만 변환 개발자의 SQL 변환에서 사용할 수 있는 원시 데이터 유형에는 Bigint 데이터 유형이 포함되어 있지 않습니다.

다음 그림에서는 쿼리 모드에서 실행되도록 구성된 변환의 포트를 보여 줍니다.



입력 포트는 WHERE 절의 데이터를 수신합니다. 출력 포트는 SELECT 문의 열을 반환합니다. SQL 쿼리는 employees 테이블에서 이름 및 주소를 선택합니다. SQL 변환은 검색한 각 데이터베이스 행에 대해 대상에 행을 기록합니다.

동적 SQL 쿼리 사용

동적 SQL 쿼리를 사용하면 입력 행마다 서로 다른 쿼리 문을 실행할 수 있습니다. 동적 SQL 쿼리를 작성할 경우 문자열 대체를 사용하여 쿼리에서 문자열 매개 변수를 정의하고 변환의 입력 포트에 해당 매개 변수를 연결합니다.

쿼리 문을 변경하려면 쿼리에서 변경하려는 부분을 문자열 변수로 구성합니다. 문자열 변수를 구성하려면 쿼리에서 이름으로 입력 포트를 식별하고 이름을 물결표(~)로 묶습니다. 쿼리는 포트의 데이터 값에 따라 변경됩니다. 쿼리 매개 변수를 포함하는 변환 입력 포트는 문자열 데이터 유형이어야 합니다. 문자열 대체를 사용하여 쿼리 문과 쿼리 데이터를 변경할 수 있습니다.

동적 SQL 쿼리를 작성하면 통합 서비스가 각 입력 행마다 쿼리를 준비합니다. 입력 포트에서 전체 쿼리를 전달하거나 쿼리 일부를 전달할 수 있습니다.

- **전체 쿼리.** 전체 SQL 쿼리를 소스 데이터의 쿼리 문으로 대체할 수 있습니다.
- **부분 쿼리.** 쿼리 문의 일부(예: 테이블 이름)를 대체할 수 있습니다.

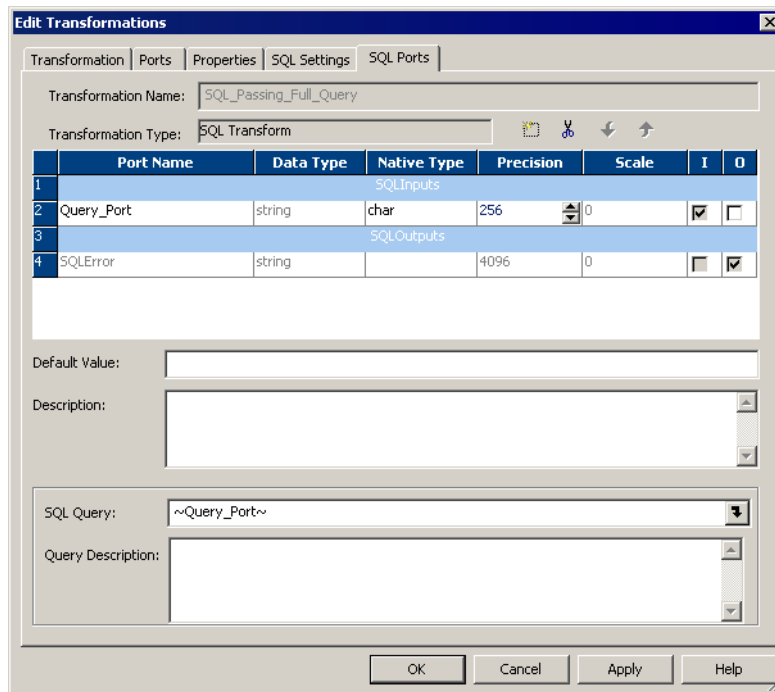
전체 쿼리 전달

변환의 입력 포트를 통해 전체 SQL 쿼리를 전달할 수 있습니다. 전체 쿼리를 전달하려면 SQL 편집기에서 전체 쿼리를 나타내는 문자열 변수 하나로 구성된 쿼리를 작성합니다.

~Query_Port~

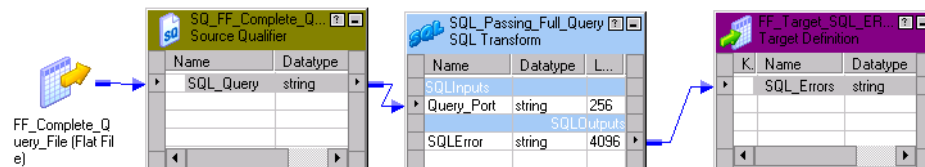
변환은 Query_Port 입력 포트에서 쿼리를 수신합니다.

다음 그림에서는 SQL 변환의 포트를 보여 줍니다.



통합 서비스는 동적 쿼리의 ~Query_Port~ 변수를 소스의 SQL 문으로 대체합니다. 그런 다음 쿼리를 준비하고 데이터베이스로 보내 처리합니다. 데이터베이스가 쿼리를 실행합니다. SQL 변환이 데이터베이스 오류를 SQLError 포트에 반환합니다.

다음 매핑에서는 SQL 변환으로 쿼리를 전달하는 방법을 보여 줍니다.



전체 쿼리를 전달할 경우 각 입력 행에 대해 둘 이상의 쿼리 문을 전달할 수 있습니다. 예를 들어 소스에 다음 행이 포함되어 있다고 가정합니다.

```
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';
```

소스 데이터에서 원하는 유형의 쿼리를 전달할 수 있습니다. 쿼리에서 SELECT 문을 구성하는 경우 데이터베이스에서 검색하는 데이터베이스 열에 대한 출력 포트를 구성해야 합니다. SELECT 문과 다른 쿼리 유형을 혼합할 경우 데이터베이스 열이 검색되지 않으면 데이터베이스 열을 나타내는 출력 포트에 null 값이 포함됩니다.

문자열에서 테이블 이름 대체

쿼리에서 테이블 이름을 대체할 수 있습니다. 테이블 이름을 대체하려면 각 입력 행에서 테이블 이름을 수신하도록 입력 포트를 구성합니다. 쿼리에서 이름으로 입력 포트를 식별하고 이름을 물결표(~)로 묶습니다.

다음 동적 쿼리는 문자열 변수, ~Table_Port~를 포함합니다.

```
SELECT Emp_ID, Address from ~Table_Port~ where Dept = 'HR'
```


소스가 다음 값을 **Table_Port** 열에 전달할 수 있습니다.

Table_Port

Employees_USA

Employees_England

Employees_Australia

통합 서비스가 ~**Table_Port**~ 변수를 입력 포트의 테이블 이름으로 바꿉니다.

```
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'
```

관련 항목:

- [“동적 업데이트 예제” 페이지 406](#)

통과 포트 구성

SQL 변환에 통과 포트를 추가할 수 있습니다. 통과 포트는 변환을 통해 데이터를 전달하는 입력/출력 포트입니다. SQL 변환은 SQL 쿼리가 행을 반환하는지 여부에 관계없이 통과 포트의 데이터를 반환합니다.

소스 행에 **SELECT** 쿼리 문이 포함되어 있으면 SQL 변환은 데이터베이스에서 반환하는 각 행에서 통과 포트의 데이터를 반환합니다. 쿼리 결과에 행이 여러 개 포함되어 있으면 SQL 변환은 각 행에서 통과 데이터를 반복합니다.

쿼리가 행을 반환하지 않으면 SQL 변환은 출력 열에 null 값이 포함된 통과 열 데이터를 반환합니다. 예를 들어 **INSERT**, **UPDATE** 및 **DELETE** 문이 포함된 쿼리는 행을 반환하지 않습니다. 쿼리에 오류가 있으면 SQL 변환은 출력 포트의 통과 열 데이터, **SQLError** 메시지 및 null 값을 반환합니다.

통과 포트를 작성하려면 다음을 수행하십시오.

- 입력 포트를 작성하고 출력에 대해 활성화합니다. 디자이너는 출력 포트를 작성한 다음 포트 이름에 "_output" 접미사를 추가합니다.
- 소스 한정자 변환에서 SQL 변환으로 포트를 끕니다. 디자이너가 통과 포트를 작성합니다.

수동 모드 구성

SQL 변환을 작성할 때 활성 모드가 아닌 수동 모드로 실행되도록 SQL 변환을 구성할 수 있습니다. 수동 변환은 변환을 통해 전달되는 행의 수를 변경하지 않습니다. 트랜잭션 경계를 유지하고 행 유형을 유지합니다.

쿼리 모드에서 실행되도록 변환을 구성한 경우 변환을 작성할 때 수동 모드를 구성할 수 있습니다. 변환을 작성한 후에는 모드를 변경할 수 없습니다.

수동 모드에 대한 규칙 및 지침

수동 모드에서 실행되도록 SQL 변환을 구성할 때 다음 규칙 및 지침을 사용하십시오.

- **SELECT** 쿼리가 둘 이상의 행을 반환하는 경우 통합 서비스는 첫 번째 행과 오류를 **SQLError** 포트에 반환합니다. 이 오류는 SQL 변환이 여러 행을 생성했음을 나타냅니다.
- SQL 쿼리에 여러 SQL 문이 있는 경우 통합 서비스는 모든 문을 실행하고, 첫 번째 SQL 문에 대한 데이터만 반환합니다. SQL 변환은 행 하나만 반환합니다. **SQLError** 포트에는 모든 SQL 문의 오류가 포함되어 있습니다. 여러 오류가 발생한 경우 **SQLError** 포트에서 세미콜론으로 오류가 구분됩니다.

- SQL 쿼리에 여러 SQL 문이 있고 통계 포트가 활성화된 경우 통합 서비스는 첫 번째 SQL 문에 대한 데이터와 통계를 반환합니다. `SQL_Error` 포트에는 모든 SQL 문에 대한 오류가 포함되어 있습니다.

쿼리 모드에 대한 규칙 및 지침

쿼리 모드에서 실행되도록 SQL 변환을 구성할 때 다음 규칙 및 지침을 사용하십시오.

- 출력 포트의 수 및 순서는 쿼리 `SELECT` 절에 있는 필드의 수 및 순서와 일치해야 합니다.
- 변환에서 출력 포트의 원시 데이터 유형은 데이터베이스에서 해당 열의 데이터 유형과 일치해야 합니다. 데이터 유형이 일치하지 않으면 통합 서비스가 행 오류를 생성합니다.
- SQL 쿼리에 `INSERT`, `UPDATE` 또는 `DELETE` 절이 포함된 경우 변환은 `SQL_Error` 포트, 통과 포트 및 `NumRowsAffected` 포트(활성화된 경우)에 데이터를 반환합니다. 출력 포트를 추가하는 경우 해당 포트는 `NULL` 데이터 값을 받습니다.
- SQL 쿼리에 `SELECT` 문이 포함되어 있고 변환에 통과 포트가 있는 경우 쿼리에서 데이터베이스 데이터가 반환되는지 여부와 관계없이 변환은 통과 포트에 데이터를 반환합니다. SQL 변환은 출력 포트에 `NULL` 데이터가 포함된 행을 반환합니다.
- 작성한 출력 포트 이름에 `"_output"` 접미사를 추가할 수 없습니다.
- 통과 포트를 사용하여 `SELECT` 쿼리에서 데이터를 반환할 수 없습니다.
- 출력 포트의 수가 `SELECT` 절에 있는 열의 수보다 많으면 추가 포트는 `NULL` 값을 받습니다.
- 출력 포트의 수가 `SELECT` 절의 열 수보다 적으면 통합 서비스에서 행 오류를 생성합니다.
- 쿼리에서 바인딩할 때 매개 변수 대신 문자열 대체를 사용할 수 있습니다. 하지만 입력 포트는 문자열 데이터 유형이어야 합니다.

데이터베이스에 연결

정적 데이터베이스 연결을 사용하거나, 런타임 시 데이터베이스 연결 정보를 SQL 변환에 전달할 수 있습니다.

SQL 변환에서 연결 환경 SQL 문 또는 트랜잭션 SQL 문을 사용할 수 있습니다. 관계형 연결 개체에서 SQL 문을 구성합니다. 통합 서비스는 데이터베이스에 연결할 때 연결 환경 SQL을 실행하고 각 트랜잭션의 시작 전에 트랜잭션 SQL 문을 실행합니다.

다음 연결 유형 중 하나를 사용하여 SQL 변환을 데이터베이스에 연결할 수 있습니다.

- **정적 연결.** 세션에서 연결 개체를 구성합니다. 먼저 워크플로우 관리자에서 연결 개체를 작성해야 합니다.
- **논리적 연결.** 런타임 시 연결 이름을 입력 데이터로 SQL 변환에 전달합니다. 먼저 워크플로우 관리자에서 연결 개체를 작성해야 합니다.
- **전체 데이터베이스 연결.** 런타임 시 연결 문자열, 사용자 이름, 암호 및 그 밖의 연결 정보를 SQL 변환 입력 포트에 전달합니다.

참고: 세션에 파티션이 여러 개 있을 경우 SQL 변환은 파티션별로 별개의 데이터베이스 연결을 작성합니다.

정적 데이터베이스 연결 사용

SQL 변환을 구성하여 정적 연결로 데이터베이스에 연결할 수 있습니다. 정적 데이터베이스 연결은 워크플로우 관리자에서 정의된 데이터베이스 연결입니다.

정적 연결을 사용하려면 세션을 구성할 때 관계형 연결 개체를 선택합니다. 데이터 유형 변환 오류를 방지하려면 변환에 구성되어 있는 동일한 데이터베이스 유형에 대한 관계형 연결을 사용합니다.

논리적 데이터베이스 연결 전달

논리적 데이터베이스 연결을 통해 데이터베이스에 연결하도록 SQL 변환을 구성할 수 있습니다. 논리적 데이터베이스 연결은 런타임에 변환으로 전달하는 연결 개체 이름입니다. 워크플로우 관리자에서 관계형 연결 개체를 정의하십시오. 논리적 데이터베이스 연결을 사용하도록 변환을 구성하면 디자이너가 **LogicalConnectionObject** 입력 포트를 작성합니다.

각 입력 행에 대한 논리적 연결을 전달할 수 있습니다. 연결 개체 이름을 **LogicalConnectionObject** 포트로 전달하도록 매핑을 구성하십시오. 데이터 유형 변환 오류를 방지하려면 변환에 구성되어 있는 동일한 데이터베이스 유형에 대한 관계형 연결을 사용합니다.

전체 연결 정보 전달

모든 데이터베이스 연결 정보를 입력 포트 데이터로 SQL 변환에 전달할 수 있습니다. 전체 연결을 통해 데이터베이스에 연결하도록 SQL 변환을 구성하면 디자이너가 연결 구성 요소의 입력 포트를 작성합니다. 데이터베이스 유형은 기본적으로 변환에 대해 구성된 데이터베이스 유형입니다.

다음 테이블에는 전체 연결을 통해 데이터베이스에 연결하도록 SQL 변환을 구성한 경우 디자이너에서 작성하는 포트가 설명되어 있습니다.

포트	필수/ 선택 사항	설명
ConnectionString	필수	데이터베이스 이름 및 데이터베이스 서버 이름을 포함합니다.
DBUser	필수	데이터베이스에 대한 읽기 및 쓰기 사용 권한이 있는 사용자의 이름입니다.
DBPasswd	필수	DBUser 암호입니다.
CodePage	선택 사항	데이터베이스에 대한 읽기 또는 쓰기를 위해 통합 서비스가 사용하는 코드 페이지입니다. ISO-8859-6과 같은 ISO 코드 페이지 이름을 사용합니다. 코드 페이지는 대/소문자를 구분하지 않습니다.
AdvancedOptions	선택 사항	연결 특성입니다. 이름 값 쌍으로 특성을 전달합니다. 각 특성을 세미콜론으로 서로 구분합니다. 특성 이름은 대/소문자를 구분하지 않습니다.

연결 문자열 전달

원시 연결 문자열은 데이터베이스 이름 및 데이터베이스 서버 이름을 포함합니다. 연결 문자열을 통해 **PowerCenter** 및 데이터베이스 클라이언트는 올바른 데이터베이스를 직접 호출할 수 있습니다.

다음 테이블에는 각 데이터베이스에 대한 원시 연결 문자열 구문이 나열되어 있습니다.

데이터베이스	연결 문자열 구문	예
IBM DB2	<i>dbname</i>	mydatabase
Microsoft SQL Server	<i>servername@dbname</i>	sqlserver@mydatabase
Oracle	<i>dbname.world</i> (TNSNAMES 항목과 같음)	oracle.world
Sybase ASE ¹	<i>servername@dbname</i>	sambrown@mydatabase

데이터베이스	연결 문자열 구문	예
Teradata ²	<i>ODBC_data_source_name</i> 또는 <i>ODBC_data_source_name@db_name</i> 또는 <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 *servername*은 인터페이스 파일의 Adaptive Server 이름입니다.

². Teradata ODBC 드라이버를 사용하여 소스 및 대상 데이터베이스에 연결합니다.

고급 옵션 전달

선택적인 연결 특성을 구성할 수 있습니다. 이 특성을 구성하려면 이름-값 쌍으로 특성을 전달하십시오. 특성이 여러 개이면 세미콜론으로 구분합니다. 특성 이름은 대/소문자를 구분하지 않습니다.

예를 들어, 연결 옵션을 구성하기 위해 다음 문자열을 전달할 수 있습니다.

Use Trusted Connection = 1; Connection Retry Period = 5

다음 테이블에는 구성할 수 있는 고급 옵션이 설명되어 있습니다.

특성	데이터베이스 유형	설명
연결 다시 시도 기간	모두	정수. 연결이 실패할 경우 통합 서비스가 데이터베이스에 다시 연결하려고 시도하는 시간(초)입니다.
데이터 소스 이름	Teradata	문자열. Teradata ODBC 데이터 소스의 이름입니다.
데이터베이스 이름	Sybase ASE Microsoft SQL Server Teradata	문자열. ODBC의 기본 데이터베이스 이름을 재정의합니다. 데이터베이스 이름을 입력하지 않으면 연결과 관련된 메시지에서 데이터베이스 이름이 표시되지 않습니다.
도메인 이름	Microsoft SQL Server	문자열. Microsoft SQL Server가 실행되는 도메인의 이름입니다.
병렬 모드 설정	Oracle	정수. 대량 모드로 데이터를 로드할 때 병렬 처리를 활성화합니다. 0이면 활성화되지 않습니다. 1이면 활성화됩니다. 기본값은 활성화됩니다.
소유자 이름	모두	문자열. 테이블 소유자 이름입니다.
패킷 크기	Sybase ASE Microsoft SQL Server	정수. Sybase ASE 및 Microsoft SQL Server에 대한 ODBC 연결을 최적화합니다.

특성	데이터베이스 유형	설명
서버 이름	Sybase ASE Microsoft SQL Server	문자열. 데이터베이스 서버의 이름입니다.
트러스트된 연결 사용	Microsoft SQL Server	정수. 활성화되면 통합 서비스는 Windows 인증을 사용하여 Microsoft SQL Server 데이터베이스에 액세스합니다. 통합 서비스를 시작하는 사용자 이름은 Microsoft SQL Server 데이터베이스에 대한 액세스 권한이 있는 유효한 Windows 사용자여야 합니다. 0이면 활성화되지 않습니다. 1이면 활성화됩니다.

데이터베이스 연결에 대한 규칙 및 지침

SQL 변환을 위한 데이터베이스 연결을 구성할 때 다음 규칙 및 지침을 따르십시오.

- 여러 데이터베이스 유형에 연결하려면 **PowerCenter** 라이선스 키가 필요합니다. 데이터베이스에 연결하는데 필요한 **PowerCenter** 라이선스가 없는 경우, 세션이 실패합니다.
- 성능을 향상시키려면 정적 데이터베이스 연결을 사용합니다. 동적 연결을 구성하면 통합 서비스가 각 입력 행별로 새로운 연결을 설정하게 됩니다.
- 세션에서 사용할 수 있는 연결의 수가 제한되어 있으면 여러 개의 **SQL** 변환을 구성할 수 있습니다. 각 **SQL** 변환이 서로 다른 정적 연결을 사용하도록 구성하십시오. 행에 있는 연결 정보에 따라 행을 **SQL** 변환으로 보내려면 라우터 변환을 사용합니다.
- 전체 연결 데이터를 사용하도록 **SQL** 변환을 구성하는 경우 데이터베이스 암호는 일반 텍스트입니다. 세션에서 사용해야 연결의 수가 제한되어 있으면 논리적 연결을 전달할 수 있습니다. 논리적 연결은 전체 연결과 동일한 기능을 제공하며, 데이터베이스 암호가 안전하게 보호됩니다.
- 논리적 데이터베이스 연결을 **SQL** 변환으로 전달하는 경우, 통합 서비스는 리포지토리에 액세스하여 각 입력 행의 연결 정보를 검색합니다. 처리할 행이 많은 경우, 논리적 데이터베이스 연결을 전달하면 성능이 영향을 받을 수 있습니다.
- 기본적으로 **SQL** 변환은 원시 데이터베이스 유형을 사용합니다. **ODBC**를 사용하여 세션을 실행하려면 **ODBC** 데이터베이스 유형의 변환을 구성하십시오. 변환에 **datetime** 또는 **datetime2** 포트가 포함되어 있으면 대응하는 원시 유형을 타임스탬프로 설정하십시오.

세션 처리

통합 서비스가 **SQL** 변환을 처리할 때 파이프라인의 미드스트림에서 **SQL** 쿼리를 실행합니다. **SELECT** 쿼리가 데이터베이스 행을 검색할 경우 **SQL** 변환은 출력 포트에서 데이터베이스 열을 반환합니다. 다른 유형의 쿼리에서는 **SQL** 변환이 출력 포트에서 쿼리 결과, 통과 데이터 또는 데이터베이스 오류를 반환합니다.

스크립트 모드에서 실행되도록 구성된 **SQL** 변환은 항상 각 입력 행에 대해 하나의 행을 반환합니다. 쿼리 모드에서 실행되는 **SQL** 변환은 각 입력 행에 대해 서로 다른 수의 행을 반환할 수 있습니다. **SQL** 변환이 반환하는 행 수는 실행하는 쿼리의 유형과 쿼리 성공 여부에 따라 달라집니다.

통합 서비스가 처리를 위해 데이터베이스로 전달한 **SQL** 쿼리의 로그를 볼 수 있습니다. 자세한 정보를 표시하도록 로깅을 설정한 경우 통합 서비스는 각 **SQL** 쿼리를 세션 로그에 기록합니다. **SQL** 변환이 포함된 세션을 디버깅해야 할 경우 로깅을 자세한 정보를 표시하도록 설정하십시오.

정적 데이터베이스 연결을 사용하도록 변환을 구성한 경우 SQL 변환에서 트랜잭션 제어를 사용할 수 있습니다. 또한 쿼리에서 커밋 및 롤백 문을 실행할 수 있습니다.

SQL 변환은 몇 가지 데이터베이스 연결 복구 기능을 제공하며, 데이터베이스 교착 상태에 대한 복원력도 제공합니다.

관련 항목:

- [“입력 행 대 출력 행의 카디널리티” 페이지 396](#)
- [“고가용성” 페이지 394](#)

트랜잭션 제어

스크립트 모드에서 실행되는 SQL 변환은 업스트림 소스 또는 트랜잭션 생성기의 모든 수신 트랜잭션 경계를 삭제합니다. 통합 서비스는 SQL 변환의 각 입력 행에 대한 스크립트를 실행한 후 커밋을 실행합니다. 트랜잭션에는 스크립트의 영향을 받는 행 집합이 포함됩니다.

쿼리 모드에서 실행되는 SQL 변환은 다음과 같은 데이터베이스 연결 유형을 기준으로 다양한 지점에서 트랜잭션을 커밋합니다.

- **동적 데이터베이스 연결.** 통합 서비스는 각 입력 행에 대한 SQL을 실행한 후 커밋을 실행합니다. 트랜잭션은 스크립트의 영향을 받는 행 집합입니다. 쿼리 모드에서 동적 연결로 트랜잭션 제어 변환을 사용할 수 없습니다.
- **정적 연결.** 통합 서비스는 모든 입력 행을 처리한 후 커밋을 실행합니다. 트랜잭션에는 업데이트할 모든 데이터베이스 행이 포함되어 있습니다. 트랜잭션 제어 변환을 통해 트랜잭션을 제어하거나 SQL 쿼리에서 커밋 및 롤백 문을 사용하여 기본 동작을 재정의할 수 있습니다.

SQL 문을 구성하여 행을 커밋 또는 롤백하는 경우 트랜잭션 생성 변환 속성으로 트랜잭션을 생성하도록 SQL 변환을 구성하십시오. 사용자 정의 커밋을 위한 세션을 구성합니다.

다음 트랜잭션 제어 SQL 문은 SQL 변환에 유효하지 않습니다.

- **SAVEPOINT.** 트랜잭션에서 롤백 지점을 식별합니다.
- **SET TRANSACTION.** 트랜잭션 옵션을 변경합니다.

관련 항목:

- [“동적 연결 예제” 페이지 411](#)

자동 커밋

자동 커밋 모드에서 작동하도록 SQL 변환 데이터베이스 연결을 구성할 수 있습니다. 자동 커밋 모드를 구성하면 SQL 문이 완료될 때 커밋이 수행됩니다.

자동 커밋을 활성화하려면 SQL 변환의 SQL 설정 탭에서 AutoCommit을 선택합니다.

자동 커밋과 함께 HA 복구가 활성화되어 있으면 쿼리에 삽입, 업데이트 또는 삭제 문이 있을 경우 데이터 무결성을 보장할 수 없다는 경고 메시지가 세션 로그에 포함됩니다.

고가용성

고가용성이 있는 경우 SQL 변환에서는 정적 및 동적 연결에 대한 데이터베이스 연결 복구를 제공합니다. 통합 서비스가 데이터베이스에 연결하지 못하는 경우 연결을 다시 시도합니다. 연결에 대한 연결 다시 시도 기간을 구성할 수 있습니다. 통합 서비스가 사용자가 구성하는 기간 내에 데이터베이스에 연결하지 못하는 경우 동적 연결에 대한 행 오류를 생성하거나 정적 연결에 대한 세션이 실패합니다.

참고: SQL 변환 데이터베이스 연결은 Informix 또는 ODBC 연결에 대해 복원력이 없습니다.

실시간 세션에 대한 정확히 한 번 처리

통합 서비스는 SQL 변환을 사용하여 실시간 소스의 메시지에 대한 정확히 한 번 배달을 제공합니다. 실시간 메시지는 SQL 변환에 한 번 배달되어야 합니다. 처리 시 중단이 되는 경우 통합 서비스는 메시지를 다시 보내도록 요구하지 않고 복구할 수 있습니다. 메시지가 다시 전송되면 통합 서비스가 메시지에서 DML 문을 두 번 실행하지 않습니다. 통합 서비스는 SELECT 또는 SET 등 기타 SQL 문을 다시 실행할 수 있습니다.

정확히 한 번 처리를 수행하기 위해 통합 서비스는 PM_REC_STATE 테이블의 검사점에 대한 작업 상태를 저장합니다. 각 SQL 변환에는 별도의 작업 상태가 있습니다. 각 SQL 변환은 일관된 상태를 유지 관리하며 연결을 공유하지 않습니다. 정확히 한 번 처리를 위해 고가용성이 있어야 합니다.

SQL 변환을 통한 실시간 세션 복구에 대한 다음 규칙 및 지침을 사용하십시오.

- SQL 변환이 포함된 세션에 대해 복구를 활성화하려면 변환 범위를 트랜잭션으로 설정해야 합니다.
- SQL 쿼리에 자동 커밋, 커밋 문 또는 DDL 문을 포함할 수 없습니다.
- SQL 변환이 수동 변환인 경우 또는 동적 연결이나 스크립트 모드에 대해 SQL 변환을 구성한 경우 SQL 변환에 대한 HA 복구를 활성화할 수 없습니다.
- SQL 변환에 대해 복구를 활성화하고 워크플로우가 동시 실행에 대해 활성화되거나 세션이 여러 파티션에서 실행되는 경우 세션이 실패할 수 있습니다. 데이터베이스 교착 상태가 PM_REC_STATE 테이블에서 발생할 수 있습니다.

쿼리 모드 복원력

통합 서비스가 입력 행에 대해 SQL 쿼리를 실행하는 동안 데이터베이스 연결 오류가 발생하는 경우, 통합 서비스는 데이터베이스에 다시 연결하려고 시도합니다.

쿼리 모드에서의 복구에 대한 규칙 및 지침은 다음과 같습니다.

- 통합 서비스가 목록의 첫 번째 쿼리를 실행하는 동안 데이터베이스 연결 오류가 발생하면 통합 서비스는 목록의 첫 번째 쿼리를 해당 행에 대해 다시 실행합니다. 통합 서비스는 나머지 쿼리를 해당 행에 대해 실행합니다.
- 첫 번째 쿼리문이 SELECT 문이고 파이프라인의 다운스트림에서 일부 행이 플러시된 후에 통신 오류가 발생하는 경우, 세션이 실패합니다.
- 통합 서비스가 목록의 첫 번째 쿼리가 아닌 쿼리를 실행하는 동안 연결 오류가 발생하는 경우, 통합 서비스는 데이터베이스에 다시 연결하고 현재 행에 대해 나머지 쿼리를 건너뛵니다. 이전 행에 대한 쿼리 결과는 손실될 수 있습니다.
- 통합 서비스가 데이터베이스에 다시 연결할 수 없는 경우, SQL 변환에서 정적 연결을 사용하면 세션이 실패합니다. SQL 변환에서 동적 연결을 사용하면 세션이 계속됩니다.
- 목록에 있는 SQL 쿼리에 명시적인 커밋 또는 롤백 문이 있는 경우, 세션이 계속되면 커밋 지점 이후에 작성된 쿼리 결과가 손실됩니다.

스크립트 모드 복원력

통합 서비스가 입력 행에 대해 스크립트를 실행하는 동안 통신 오류가 발생하는 경우, 통합 서비스는 데이터베이스에 다시 연결하려고 시도합니다.

스크립트 모드에서의 복구에 대한 규칙 및 지침은 다음과 같습니다.

- 통합 서비스가 데이터베이스에 연결하지 못하고 연결이 정적이면 세션이 실패하게 됩니다. SQL 변환에서 동적 연결을 사용하면 세션이 계속됩니다.
- 통합 서비스가 데이터베이스에 다시 연결하는 경우, 통합 서비스는 현재 행에 대한 처리를 건너뛰고 다음 행을 계속합니다.

데이터베이스 교착 상태 복구

교착 상태 시 세션 다시 시도 세션 속성을 활성화하면 SQL 변환이 데이터베이스 교착 상태 오류에 대해 복원력이 있습니다. SQL 변환은 쿼리 모드에서 데이터베이스 교착 상태 오류에 복원력이 있지만 스크립트 모드에서는 교착 상태 오류에 복원력이 없습니다. 쿼리 모드에서 교착 상태가 발생하면 통합 서비스는 사용자가 구성한 교착 상태 다시 시도 횟수 동안 데이터베이스에 다시 연결하려고 시도합니다. 교착 상태 다시 시도 횟수와 교착 상태 절전 기간을 설정하도록 통합 서비스를 구성합니다.

교착 상태가 발생할 경우 현재 행에 DML 문이 없으면 통합 서비스는 현재 행에서 SQL 문을 다시 시도합니다. 행에 INSERT, UPDATE 또는 DELETE와 같은 DML 문이 있을 경우 통합 서비스는 현재 행을 다시 처리하지 않습니다.

동적 연결의 경우 다시 시도가 실패하면 통합 서비스는 **SQLError** 포트에서 오류를 반환합니다. 통합 서비스는 행 내의 SQL 오류에 대해 계속 속성을 기반으로 다음 문을 처리합니다. 속성이 비활성화되면 통합 서비스는 현재 행을 건너뛵니다. 현재 행에 INSERT, UPDATE 또는 DELETE와 같은 DML 문이 있을 경우 통합 서비스는 오류 수를 증분시킵니다.

정적 연결의 경우 다시 시도가 실패하면 통합 서비스는 **SQLError** 포트에서 오류를 반환합니다. 현재 행에 DML 문이 있으면 통합 서비스는 세션을 중지합니다. 통합 서비스는 행 내의 SQL 오류에 대해 계속 속성을 기반으로 다음 문을 처리합니다. 속성이 비활성화되면 통합 서비스는 현재 행을 건너뛵니다.

SQL 쿼리 로그

통합 서비스가 처리를 위해 데이터베이스로 전달한 SQL 쿼리의 로그를 볼 수 있습니다. 자세한 정보를 표시하도록 로깅을 설정한 경우 통합 서비스는 각 SQL 쿼리를 세션 로그에 기록합니다. SQL 변환이 포함된 세션을 디버깅해야 할 경우 로깅을 자세한 정보를 표시하도록 설정하십시오.

쿼리에 CLOB 또는 BLOB 데이터가 포함된 경우 세션 로그에 쿼리가 포함되지 않습니다. 세션 로그에는 데이터를 설명하는 메시지(예: CLOB 데이터)가 포함됩니다.

입력 행 대 출력 행의 카디널리티

통합 서비스에서 SELECT 쿼리를 실행하면 SQL 변환은 검색한 각 행에 대해 행을 반환합니다. 쿼리에서 데이터가 검색되지 않을 경우 SQL 변환은 각 입력 행에 대해 0 또는 1개의 행을 반환합니다.

SQL 변환이 반환하는 출력 행의 수는 다음과 같은 요인에 따라 다릅니다.

- **쿼리 문 처리.** 쿼리에 SELECT 문이 포함되면 통합 서비스가 여러 출력 행을 검색할 수 있습니다. SELECT 쿼리가 성공할 경우 SQL 변환이 여러 행을 검색할 수 있습니다. 쿼리에 다른 문이 포함되면 통합 서비스가 SQL 오류 또는 영향을 받는 행의 수가 포함된 행을 생성할 수 있습니다.
- **포트 구성.** NumRowsAffected 출력 포트에는 한 입력 행의 업데이트, 삽입 또는 삭제에 영향을 받는 총 행 수가 포함됩니다. SQL 문에 통과 포트가 포함되면 변환이 각 소스 행에 대해 한 번 이상 열 데이터를 반환합니다.
- **최대 행 수 구성.** 최대 출력 행 개수는 SQL 변환이 SELECT 쿼리에서 반환하는 행 수를 제한합니다.
- **오류 행.** 통합 서비스는 연결 또는 구문 오류를 발견하면 행 오류를 반환합니다. SQL 문은 쿼리 모드에서 실행될 경우 **SQLError** 포트에 오류를 반환합니다. SQL 문은 스크립트 모드에서 실행될 경우 **ScriptError** 포트에 오류를 반환합니다.
- **SQL 오류에 대해 계속.** SQL 문에 오류가 있을 경우 처리를 계속하도록 SQL 변환을 구성할 수 있습니다. SQL 변환이 행 오류를 생성하지 않습니다.

쿼리 문 처리

쿼리 유형은 SQL 변환에서 반환하는 행의 개수를 결정합니다. 쿼리 모드로 실행되는 SQL 변환은 행을 반환하지 않거나 1개 이상의 행을 반환할 수 있습니다. 쿼리에 SELECT 문이 포함된 경우 SQL 변환이 데이터베이스의 각 열을 출력 포트에 반환합니다. 변환이 모든 인가된 행을 반환합니다.

다음 테이블에는 쿼리 모드에서 오류가 발생하지 않을 경우 SQL 변환이 여러 유형의 쿼리 문에 대해 생성하는 출력 행이 나열되어 있습니다.

쿼리 문	출력 행
UPDATE, INSERT, DELETE만	0개 행입니다.
하나 이상의 SELECT 문	검색한 데이터베이스 총 행 수.
CREATE, DROP, TRUNCATE 등의 DDL 쿼리	0개 행입니다.

영향을 받는 행 수

각 입력 행에서 INSERT, UPDATE 또는 DELETE 쿼리 문의 영향을 받는 행 수를 NumRowsAffected 출력 포트가 반환하도록 설정할 수 있습니다. 통합 서비스는 쿼리의 각 문에 대해 NumRowsAffected를 반환합니다. NumRowsAffected는 기본적으로 비활성화됩니다.

쿼리 모드에서 NumRowsAffected를 활성화했지만 SQL 쿼리에 INSERT, UPDATE 또는 DELETE 문이 포함되어 있지 않은 경우 각 출력 행에서 NumRowsAffected가 0이 됩니다.

참고: NumRowsAffected를 활성화했지만 변환이 스크립트 모드로 실행되도록 구성되어 있는 경우 NumRowsAffected는 항상 NULL입니다.

다음 테이블에는 쿼리 모드에서 NumRowsAffected를 활성화한 경우 SQL 변환이 생성하는 출력 행이 나열되어 있습니다.

쿼리 문	출력 행
UPDATE, INSERT, DELETE만	문에 NumRowsAffected를 사용하여 각 문에 대해 한 행씩
하나 이상의 SELECT 문	검색한 데이터베이스 총 행 수. 각 행의 NumRowsAffected는 0입니다.
CREATE, DROP, TRUNCATE 등의 DDL 쿼리	NumRowsAffected가 0인 한 개의 행

SQL 변환이 쿼리 모드로 실행되고 쿼리에 여러 문이 포함되어 있으면 통합 서비스는 각 문에 대해 NumRowsAffected를 반환합니다. NumRowsAffected는 입력 행에서 각 INSERT, UPDATE 및 DELETE 문의 영향을 받는 행의 합을 포함합니다.

예를 들어 쿼리는 다음 문을 포함합니다.

```
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 문은 행 하나에 적용됩니다. SELECT 문은 행에 적용되지 않습니다. INSERT 문은 행 하나에 적용됩니다.

통합 서비스는 **DELETE** 문에서 행 하나를 반환하고, **NumRowsAffected**는 1과 같습니다. **SELECT** 문에서는 행 하나를 반환하지만 **NumRowsAffected**가 0입니다. **INSERT** 문에서는 행 하나를 반환하고 **NumRowsAffected**는 1입니다.

다음 조건이 모두 참인 경우 **NumRowsAffected** 포트는 0을 반환합니다.

- 데이터베이스가 Informix입니다.
- 변환이 쿼리 모드에서 실행되고 있습니다.
- 쿼리에 매개 변수가 포함되어 있지 않습니다.

최대 출력 행 개수

SQL 변환에서 **SELECT** 쿼리에 대해 반환하는 행의 수를 제한할 수 있습니다. 행의 수를 제한하려면 최대 출력 행 개수 속성을 구성합니다. 쿼리에 여러 **SELECT** 문이 포함되어 있으면 SQL 변환은 모든 **SELECT** 문의 총 행 개수를 제한합니다.

최대 출력 행 개수를 100으로 설정하는 경우를 예로 들어 보겠습니다. 쿼리에는 다음의 두 **SELECT** 문이 포함되어 있습니다.

```
SELECT * FROM table1; SELECT * FROM table2;
```

첫 번째 **SELECT** 문이 행 200개를 반환하고 두 번째 **SELECT** 문이 행 50개를 반환하면 SQL 변환은 첫 번째 **SELECT** 문에서 행 100개를 반환합니다. 두 번째 문에서는 행이 반환되지 않습니다.

무제한 출력 행을 구성하려면 최대 출력 행 개수를 0으로 설정합니다.

오류 행 이해

통합 서비스는 연결 오류 또는 구문 오류가 발생할 경우 행 오류를 반환합니다. SQL 변환에는 오류 텍스트를 출력하는 다음 기본 포트가 있습니다.

- **SQLException**. SQL 변환이 쿼리 모드에서 실행될 때 데이터베이스 오류를 반환합니다.
- **ScriptError**. SQL 변환이 스크립트 모드에서 실행될 때 데이터베이스 오류를 반환합니다.

SQL 쿼리에 구문 오류가 있으면 오류 포트에는 데이터베이스의 오류 텍스트가 포함됩니다. 예를 들어 다음 SQL 쿼리는 Oracle 데이터베이스에서 행 오류를 생성합니다.

```
SELECT Product_ID FROM Employees
```

Employees 테이블에는 **Product_ID**가 없습니다. 통합 서비스는 행을 하나 생성합니다. **SQLException** 포트의 한 행에 오류 텍스트가 포함되어 있습니다.

```
ORA-0094: "Product_ID": invalid identifier Database driver error... Function Name: Execute SQL Stmt:  
SELECT Product_ID from Employees Oracle Fatal Error
```

쿼리에 문 여러 개가 포함되어 있고 SQL 오류가 발생해도 계속 진행되도록 SQL 변환을 구성하면 SQL 변환이 특정 쿼리 문에 대해서는 데이터베이스의 행을 반환하지만 다른 쿼리 문에 대해서는 데이터베이스 오류를 반환할 수 있습니다. SQL 변환은 데이터베이스 오류를 개별 행에 반환합니다.

통과 포트 또는 **NumRowsAffected** 포트를 구성한 경우 SQL 변환이 각 소스 행에 대해 최소 1개의 행을 반환합니다. 쿼리에서 데이터가 반환되지 않을 경우 SQL 변환은 통과 데이터 및 **NumRowsAffected** 값을 반환하지만 출력 포트의 Null 값도 반환합니다. 필터 변환을 통해 출력 행을 전달하면 Null 값을 포함하는 행을 제거할 수 있습니다.

다음 테이블에는 쿼리 문의 유형을 기준으로 SQL 변환이 반환하는 출력 행이 설명되어 있습니다.

다음 테이블에는 SQL 변환이 UPDATE, INSERT 또는 DELETE 쿼리 문에 대해 생성하는 행이 설명되어 있습니다.

NumRowsAffected 포트 또는 통과 포트 구성	SQLError	행 출력
포트 둘 다 구성되지 않음	아니요	0개 행입니다.
포트 둘 다 구성되지 않음	예	SQLError 포트의 오류를 포함하는 행 1개가 생성됩니다.
둘 중 한 포트가 구성됨	아니요	NumRowsAffected 또는 통과 열 데이터를 포함하는 각 쿼리 문에 대한 행 1개가 생성됩니다.
둘 중 한 포트가 구성됨	예	SQLError 포트의 오류, NumRowsAffected 포트 또는 통과 포트 데이터를 포함하는 행 1개가 생성됩니다.

다음 테이블에는 SQL 변환이 SELECT 문에 대해 생성하는 출력 행의 수가 설명되어 있습니다.

NumRowsAffected 포트 또는 통과 포트 구성	SQLError	행 출력
포트 둘 다 구성되지 않음	아니요	각 SELECT 문에서 반환된 행에 따라 0개 이상의 행이 생성됩니다.
포트 둘 다 구성되지 않음	예	성공한 문에 대한 출력 행의 합계보다 큰 행 1개가 생성됩니다. 마지막 행에는 SQLError 포트의 오류가 포함됩니다.
둘 중 한 포트가 구성됨	아니요	각 SELECT 문에 대해 반환된 행에 따라 1개 이상의 행이 생성됩니다. - NumRowsAffected가 활성화된 경우 각 행에 값이 0인 NumRowsAffected 열이 포함됩니다. - 통과 포트가 구성된 경우 각 행에 통과 열 데이터가 포함됩니다. 쿼리에서 여러 행이 반환되는 경우 각 행에 통과 열 데이터가 중복됩니다.
둘 중 한 포트가 구성됨	예	각 SELECT 문에 대해 반환된 행에 따라 1개 이상의 행이 생성됩니다. 마지막 행에는 SQLError 포트의 오류가 포함됩니다. - NumRowsAffected가 활성화된 경우 각 행에 값이 0인 NumRowsAffected 열이 포함됩니다. - 통과 포트가 구성된 경우 각 행에 통과 열 데이터가 포함됩니다. 쿼리에서 여러 행이 반환되는 경우 각 행에 통과 열 데이터가 중복됩니다.

다음 테이블에는 SQL 변환이 CREATE, DROP 또는 TRUNCATE와 같은 DDL 쿼리에 대해 생성하는 출력 행의 수가 설명되어 있습니다.

NumRowsAffected 포트 또는 통과 포트 구성	SQLError	행 출력
포트 둘 다 구성되지 않음	아니요	- 0개 행입니다.
포트 둘 다 구성되지 않음	예	- SQLError 포트의 오류를 포함하는 행 1개가 생성됩니다.

NumRowsAffected 포트 또는 통과 포트 구성	SQLError	행 출력
둘 중 한 포트가 구성됨	아니요	- 값이 0인 NumRowsAffected 열 및 통과 열 데이터를 포함하는 행 1개가 생성됩니다.
둘 중 한 포트가 구성됨	예	- SQLError 포트의 오류, 값이 0인 NumRowsAffected 열 및 통과 열 데이터를 포함하는 행 1개가 생성됩니다.

SQL 오류에 대해 계속

행 내의 SQL 오류에 대해 계속 옵션을 활성화하여 문에서 SQL 오류를 무시하도록 선택할 수 있습니다. 통합 서비스가 행에 대해 SQL 문의 나머지 부분을 계속 실행합니다. 통합 서비스는 행 오류를 생성하지 않습니다. 그러나 SQLError 포트에는 실패한 SQL 문 및 오류 메시지가 포함됩니다. 행 오류 수가 세션 오류 임계값을 초과하면 세션이 실패하게 됩니다.

예를 들어 쿼리에 다음 문이 포함될 수 있습니다.

```
DELETE FROM Persons WHERE FirstName = 'Ed';
```

```
INSERT INTO Persons (LastName, Address)VALUES ('Gein', '38 Beach Rd')
```

DELETE 문이 실패한 경우 SQL 변환이 데이터베이스에서 오류 메시지를 반환합니다. 통합 서비스는 INSERT 문을 계속해서 처리합니다.

팁: 데이터베이스 오류를 디버그하려면 SQL 오류에 대해 계속 옵션을 비활성화합니다. 그렇지 않으면 오류를 발생시킨 쿼리 문과 오류를 연결하지 못할 수 있습니다.

SQL 변환 속성

SQL 변환을 작성한 후 다음 변환 탭에서 포트를 정의하고 특성을 설정할 수 있습니다.

- **포트.** SQL 포트 탭에서 작성한 변환 포트 및 특성을 표시합니다.
- **속성.** SQL 변환 일반 속성입니다.
- **SQL 설정.** SQL 변환의 고유한 특성입니다.
- **SQL 포트.** SQL 변환 포트 및 특성입니다.

참고: 포트 탭에서 열을 업데이트할 수 없습니다. SQL 포트 탭에서 포트를 정의한 경우 포트 탭에 표시됩니다.

속성 탭

속성 탭에서 SQL 변환 일반 속성을 구성합니다. 일부 변환 속성은 SQL 변환에 적용되지 않거나 구성할 수 없습니다.

다음 테이블에는 SQL 변환 속성이 설명되어 있습니다.

속성	설명
RunTime 위치	DLL 또는 공유 라이브러리가 포함된 위치입니다. SQL 변환 세션을 실행하는 통합 서비스 노드에 상대적인 경로를 입력합니다. 이 속성이 비어 있는 경우 통합 서비스는 통합 서비스 노드에 정의된 환경 변수를 사용하여 DLL 또는 공유 라이브러리를 찾습니다. 모든 DLL 또는 공유 라이브러리를 런타임 위치나 통합 서비스 노드에 정의된 환경 변수의 위치로 복사해야 합니다. 통합 서비스가 DLL, 공유 라이브러리 또는 참조된 파일을 찾을 수 없으면 해당 절차를 로드하지 못합니다.
추적 수준	이 변환을 포함하는 세션을 실행할 때 세션 로그에 포함되는 세부 정보의 양을 설정합니다. SQL 변환 추적 수준을 자세한 정보 표시 데이터로 구성하면 통합 서비스는 준비한 각 SQL 쿼리를 세션 로그에 기록합니다.
IsPartitionable	파이프라인의 여러 파티션에서 이 변환을 사용할 수 있습니다. 다음과 같은 옵션을 사용합니다. - 아니요. 변환을 분할할 수 없습니다. 변환과 동일한 파이프라인에 있는 다른 변환이 하나의 파티션으로 제한됩니다. 변환에서 데이터 정리처럼 입력 데이터를 모두 함께 처리하는 경우 아니요를 선택할 수 있습니다. - 로컬로. 변환을 분할할 수 있지만 통합 서비스가 같은 노드의 파이프라인에서 모든 파티션을 실행해야 합니다. 변환의 여러 파티션이 메모리에서 개체를 공유해야 하는 경우 로컬로 선택합니다. - 그리드에서. 변환을 분할할 수 있고 통합 서비스가 각 파티션을 여러 노드에 분산시킬 수 있습니다. 기본값은 아니요입니다.
업데이트 전략 변환	변환이 출력 행에 대해 업데이트 전략을 정의합니다. 쿼리 모드 SQL 변환의 경우 이 속성을 활성화할 수 있습니다. 기본값이 비활성화됩니다.
변환 범위	통합 서비스가 변환 논리를 수신 데이터에 적용하는 방식입니다. 다음과 같은 옵션을 사용합니다. - 행 - 트랜잭션 - 모든 입력 정적 쿼리 모드에서 트랜잭션 제어를 사용할 경우 트랜잭션 범위를 트랜잭션으로 설정합니다. 스크립트 모드 변환의 경우 기본값은 행입니다. 쿼리 모드 변환의 경우 기본값은 모든 입력입니다.
출력은 반복 가능합니다.	출력 데이터의 순서가 세션 실행 간에 일치하는지 여부를 나타냅니다. - 사용 안 함 출력 데이터 순서는 세션 실행 간에 일관되지 않습니다. - 입력 순서 기반. 입력 데이터 순서가 세션 실행 간에 일관된 경우 입력 순서는 세션 실행 간에 일관됩니다. - 항상 입력 데이터 순서가 세션 실행 간에 일관되지 않더라도 출력 데이터 순서는 세션 실행 간에 일관됩니다. 기본값은 사용 안 함입니다.
트랜잭션 생성	변환이 트랜잭션 행을 생성합니다. SQL 쿼리에서 데이터를 커밋하는 쿼리 모드 SQL 변환의 경우 이 속성을 활성화합니다. 기본값이 비활성화됩니다.

속성	설명
파티션당 단일 스레드 필요	통합 서비스가 프로시저의 각 파티션을 개별 스레드로 처리하는지 여부를 나타냅니다.
확정 출력입니다.	변환이 세션 실행 간에 일관된 출력 데이터를 생성합니다. 이 변환을 사용하는 세션에 대해 복구를 수행하려면 이 속성을 활성화합니다. 기본값은 활성화됨입니다.

경고: 변환을 반복 가능 및 확정으로 구성하는 경우 데이터가 반복 가능 및 확정인지 확인하는 것은 사용자의 책임입니다. 세션과 복구 간에 동일한 데이터를 생성하지 않는 변환으로 세션을 복구하려는 경우 복구 프로세스로 인해 손상된 데이터가 발생할 수 있습니다.

SQL 설정 탭

SQL 설정 탭에서 SQL 변환 특성을 구성합니다. SQL 특성은 SQL 변환에 대해 고유합니다.

다음 테이블에는 SQL 설정 탭에서 구성할 수 있는 특성이 나열되어 있습니다.

옵션	설명
행 내의 SQL 오류에 대해 계속	SQL 오류가 발생한 후 쿼리에서 남은 SQL 문 처리를 계속합니다.
통계 출력 포트 추가	NumRowsAffected 출력 포트를 추가합니다. 이 포트는 입력 행에 대한 INSERT, DELETE 및 UPDATE 쿼리 문의 영향을 받는 총 데이터베이스 행 수를 반환합니다.
AutoCommit	각 데이터베이스 연결에 대한 자동 커밋을 활성화합니다. 쿼리의 각 SQL 문이 트랜잭션을 정의합니다. SQL 문이 완료되거나 다음 문이 실행되는 경우 커밋이 발생합니다.
최대 출력 행 개수	SQL 변환이 SELECT 쿼리에서 출력할 수 있는 최대 행 수를 정의합니다. 무제한 행 수를 구성하려면 [최대 출력 행 개수]를 0으로 설정하십시오.
스크립트 로깅	SQL 스크립트의 코드 페이지를 식별합니다. 목록에서 코드 페이지를 선택합니다. 기본값은 운영 체제 로깅입니다.
연결 풀링 사용	데이터베이스 연결에 대한 연결 풀을 유지합니다. 동적 연결에 대해서만 연결 풀링을 활성화할 수 있습니다.
풀의 최대 연결 수	연결 풀에서 사용 가능한 최대 활성 연결 수입니다. 최소값은 1입니다. 최대값은 20입니다. 기본값은 10입니다.

SQL 포트 탭

SQL 변환을 작성할 때 변환을 구성하는 방식에 따라 디자이너가 기본 포트를 추가합니다. 변환을 작성한 후 SQL 포트 탭에서 변환에 포트를 추가할 수 있습니다.

다음 테이블에는 SQL 변환 포트가 나열되어 있습니다.

포트	유형	모드	설명
ScriptName	입력	스크립트	각 행에 대해 실행할 스크립트의 이름입니다.
ScriptError	출력	스크립트	스크립트가 실패하는 경우 데이터베이스가 변환으로 반환하는 SQL 오류입니다.
ScriptResult	출력	스크립트	쿼리 실행의 결과입니다. 쿼리가 성공적으로 실행되면 PASSED를 포함합니다. 쿼리가 성공하지 못하면 FAILED를 포함합니다.
SQLError	출력	쿼리	데이터베이스가 변환으로 반환하는 SQL 오류입니다.
LogicalConnectionObject	입력	쿼리	동적 연결에 해당합니다. 워크플로우 관리자 연결에 정의되어 있는 연결의 이름입니다.
연결 문자열	입력	쿼리 스크립트	연결 개체에만 해당합니다. 데이터베이스 이름 및 데이터베이스 서버 이름입니다.
DBUser	입력	쿼리 스크립트	전체 연결에만 해당합니다. 데이터베이스에서 읽고 쓸 수 있는 사용 권한이 있는 사용자의 이름입니다.
DBPasswd	입력	쿼리 스크립트	전체 연결에만 해당합니다. 데이터베이스 암호입니다.
CodePage	입력	쿼리 스크립트	전체 연결에만 해당합니다. 통합 서비스가 데이터베이스에서 읽고 쓰기 위해 사용하는 코드 페이지입니다.
고급 옵션	입력	쿼리 스크립트	전체 연결에만 해당합니다. 패킷 크기, 연결 재시도 기간 및 병렬 모드 활성화 같은 선택적인 연결 특성입니다.
NumRowsAffected	출력	쿼리 스크립트	입력 행에 대한 INSERT, DELETE 및 UPDATE 쿼리 문의 영향을 받는 총 데이터베이스 행 수입니다.

SQL 문

다음 테이블에는 SQL 변환에서 사용할 수 있는 문이 나열되어 있습니다.

문 유형	문	설명
데이터 정의	ALTER	데이터베이스 구조를 수정합니다.
데이터 정의	COMMENT	데이터 사전에 설명을 추가합니다.

문 유형	문	설명
데이터 정의	CREATE	데이터베이스, 테이블 또는 인덱스를 작성합니다.
데이터 정의	DROP	인덱스, 테이블 또는 데이터베이스를 삭제합니다.
데이터 정의	RENAME	데이터베이스 개체의 이름을 바꿉니다.
데이터 정의	TRUNCATE	테이블에서 모든 행을 제거합니다.
데이터 조작	CALL	PL/SQL 또는 Java 하위 프로그램을 호출합니다.
데이터 조작	DELETE	테이블에서 행을 삭제합니다.
데이터 조작	EXPLAIN PLAN	데이터베이스 Explain 테이블에 문에 대한 액세스 계획을 작성합니다.
데이터 조작	INSERT	테이블에 행을 삽입합니다.
데이터 조작	LOCK TABLE	응용 프로그램 프로세스가 테이블을 동시에 사용하거나 변경하지 않도록 방지합니다.
데이터 조작	MERGE	소스 데이터로 테이블을 업데이트합니다.
데이터 조작	SELECT	데이터베이스에서 데이터를 검색합니다.
데이터 조작	UPDATE	테이블의 행 값을 업데이트합니다.
데이터 제어 언어	GRANT	데이터베이스 사용자에게 권한을 부여합니다.
데이터 제어 언어	REVOKE	데이터베이스 사용자의 액세스 권한을 제거합니다.
트랜잭션 제어	COMMIT	작업 단위를 저장하고 해당 작업 단위에 대한 데이터베이스 변경을 수행합니다.
트랜잭션 제어	ROLLBACK	마지막 COMMIT 이후의 데이터베이스 변경 내용을 취소합니다.

SQL 변환 작성

변환 개발자 또는 매핑 디자이너에서 SQL 변환을 작성할 수 있습니다.

SQL 변환을 작성하려면

1. 변환 > 작성을 클릭합니다.
2. SQL 변환을 선택합니다.
3. 변환 이름을 입력합니다.

SQL 변환의 이름 지정 규칙은 SQL_TransformationName입니다.

4. 변환에 대한 설명을 입력하고 작성을 클릭합니다.

5. SQL 변환 모드를 구성합니다.
 - **쿼리 모드.** 동적 SQL 쿼리를 실행하는 활성 변환을 구성합니다.
 - **스크립트 모드.** 외부 SQL 스크립트를 실행하는 수동 변환을 구성합니다.
6. SQL 변환이 연결되는 데이터베이스 유형을 구성합니다. 목록에서 데이터베이스 유형을 선택합니다.
7. SQL 변환 연결 옵션을 구성합니다.

옵션	설명
정적 연결	세션에 대해 구성하는 연결 개체를 사용합니다. 한 세션 동안 SQL 변환은 데이터베이스에 한 번 연결됩니다.
동적 연결.	매핑에서 변환에 전달하는 연결 정보에 따라 데이터베이스에 연결합니다. 동적 연결을 구성하는 경우 변환에 연결 개체 이름이 필요한지, 아니면 모든 연결 정보가 필요한지 선택합니다. 기본값은 연결 개체입니다.
연결 개체	동적 연결만 해당됩니다. LogicalConnectionObject 포트를 사용하여 연결 개체 이름을 받습니다. 워크플로우 관리자 연결에서 연결 개체를 정의합니다.
전체 연결 정보	동적 연결만 해당됩니다. 입력 포트를 사용하여 모든 연결 구성 요소를 받습니다.

8. 수동 모드에서 실행되도록 SQL 변환을 구성하려면 SQL 변환을 수동 모드로 실행을 선택합니다.
9. 확인을 클릭하여 변환을 구성합니다.
디자이너가 사용자가 선택한 옵션에 따라 변환에서 기본 포트를 작성합니다. 데이터베이스 유형을 제외하고 구성을 변경할 수 없습니다.
10. 포트 탭을 클릭하여 포트를 변환에 추가합니다. 데이터베이스 포트 뒤에 통과 포트를 추가합니다.

제 27 장

매핑에서 SQL 변환 사용

이 장에 포함된 항목:

- [SQL 변환 예제 개요, 406](#)
- [동적 업데이트 예제, 406](#)
- [동적 연결 예제, 411](#)

SQL 변환 예제 개요

SQL 변환은 파이프라인에서 SQL 쿼리 미드스트림을 처리합니다. 이 변환은 SQL 편집기에서 작성한 외부 SQL 스크립트 또는 SQL 쿼리를 처리합니다. 데이터베이스 연결 정보를 런타임에 입력 데이터로 SQL 변환에 전달할 수 있습니다.

이 장에서는 SQL 변환 기능을 설명하는 두 가지 예제를 제공합니다. 이 장의 예제를 사용하여 동적 SQL 쿼리를 작성 및 실행하고 동적으로 데이터베이스에 연결하십시오. 이 장에서는 매핑에 포함할 수 있는 변환의 설명 및 샘플 데이터를 제공합니다.

이 장에서 제공하는 예제는 다음과 같습니다.

- **데이터베이스를 업데이트하기 위한 동적 SQL 쿼리 작성.** 동적 쿼리 업데이트 예제에서는 소스 파일로부터 받은 가격 코드에 따라 테이블에서 제품 가격을 업데이트하는 방법을 보여 줍니다.
- **동적 데이터베이스 연결 구성.** 동적 연결 예제에서는 소스 행에 있는 고객 위치 값에 따라 여러 데이터베이스에 연결하는 방법을 보여 줍니다.

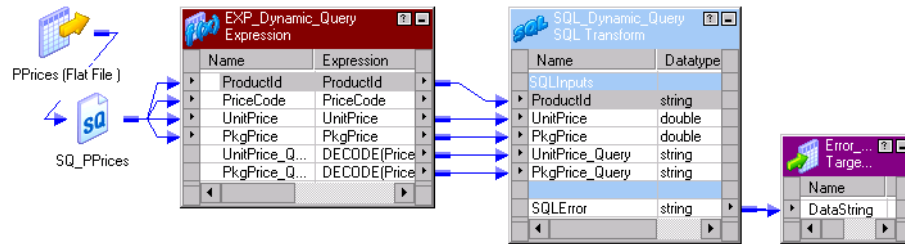
동적 업데이트 예제

이 예제에는 소스 파일의 열 값을 기반으로 SQL 쿼리를 생성하도록 식 변환 및 SQL 변환을 구성하는 방법이 나와 있습니다.

이 예제에서는 제품 가격이 포함된 데이터베이스 테이블이 있습니다. 트랜잭션 파일에서 가격을 업데이트해야 합니다. 각 트랜잭션 행은 가격 코드 열을 기반으로 데이터베이스에서 도매, 소매 또는 제조 가격을 업데이트합니다.

소스 파일은 플랫폼 파일입니다. 각 소스 행의 가격 코드 열 값을 기반으로 업데이트하기 위해 열 이름을 반환하도록 식 변환을 구성할 수 있습니다. 식 변환은 SQL 변환에 열 이름을 전달합니다. SQL 변환은 수신하는 열 이름을 기반으로 `Prod_Cost` 테이블에서 열을 업데이트하는 동적 SQL 쿼리를 실행합니다. SQL 변환은 `Error_File` 대상에 데이터베이스 오류를 반환합니다.

다음 그림에는 식 변환이 SQL 변환에 열 이름을 전달하는 방법이 나와 있습니다.



매핑에는 다음과 같은 구성 요소가 있습니다.

- **PPrices 소스 정의.** PPrices 플랫폼 파일에는 제품 ID, 패키지 가격, 단가 및 가격 코드가 포함되어 있습니다. 가격 코드는 패키지 가격 및 단가가 도매, 소매 또는 제조 가격인지 정의합니다.
- **Error_File 플랫폼 파일 대상 정의.** 대상에는 SQL 변환에서 데이터베이스 오류를 수신하는 Datastring 필드가 포함되어 있습니다.
- **Exp_Dynamic_Expression 변환.** 식 변환은 PriceCode 열의 값을 기반으로 업데이트할 Prod_Cost 열 이름을 정의합니다. 이 변환은 UnitPrice_Query 및 PkgPrice_Query 포트의 열 이름을 반환합니다.
- **SQL_Dynamic_Query 변환.** SQL 변환에는 Prod_Cost 테이블에서 UnitPrice 열 및 PkgPrice 열을 업데이트하기 위한 동적 SQL 쿼리가 있습니다. 이 변환은 UnitPrice_Query 및 PkgPrice_Query 열에 명명된 열을 업데이트합니다.

참고: 해당 매핑에는 Prod_Cost 테이블에 대한 관계형 테이블 정의가 포함되지 않습니다. SQL 변환에는 Prod_Cost 테이블이 포함된 데이터베이스에 대한 정적 연결이 있습니다. 이 변환이 테이블의 단가 및 패키지 가격을 업데이트하는 SQL 문을 생성합니다.

소스 파일 정의

트랜잭션 파일은 데이터베이스에서 업데이트할 가격이 포함된 플랫폼 파일 소스입니다. 각 행에는 가격이 도매, 소매 또는 제조업체 가격인지를 정의하는 코드가 있습니다. 소스 파일 이름은 PPrices.dat입니다.

Srcfiles에서 다음 행이 포함된 PPrices.dat 파일을 작성할 수 있습니다.

```
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 파일을 가져와서 리포지토리에서 PPrices 소스 정의를 작성할 수 있습니다.

PPrices 파일에는 다음과 같은 열이 포함됩니다.

열	데이터 유형	전체 자릿수	설명
ProductID	문자열	10	업데이트할 제품을 식별하는 고유 번호입니다.
PriceCode	문자열	2	M, W 또는 R. 가격이 제조, 도매 또는 소매 가격인지 정의합니다.

열	데이터 유형	전체 자릿수	설명
UnitPrice	숫자	10	제품 단위당 가격입니다.
PkgPrice	숫자	10	제품 패키지의 가격입니다.

대상 정의 작성

Error_File은 SQL 변환에서 데이터베이스 오류 메시지를 받는 플랫폼 파일 대상 정의입니다.

Error_File 정의에는 다음과 같은 입력 포트가 포함됩니다.

포트 이름	데이터 유형	전체 자릿수	소수 자릿수
DataStream	문자열	4000	0

대상 디자이너에서 대상 정의를 작성합니다.

데이터베이스 테이블 작성

SQL 변환에서 **Prod_Cost** 관계형 테이블에 제품 가격을 씁니다. **Prod_Cost** 테이블에는 각 제품의 단가 및 패키지 가격이 있습니다. 단위 및 패키지별로 도매, 소매 및 제조 가격이 저장됩니다.

사용자는 리포지토리에서 **Prod_Cost** 테이블에 대한 관계형 대상 정의를 작성하지 않습니다. SQL 변환이 데이터베이스에서 테이블을 업데이트하는 SQL 문을 생성합니다.

Prod_Cost 테이블에는 다음 열이 있습니다.

비용 유형	데이터 유형	설명
ProductID	varchar	업데이트할 제품을 식별하는 고유 번호입니다.
WUnitPrice	숫자	도매 단가입니다.
WPkgPrice	숫자	도매 패키지 가격입니다.
RUnitPrice	숫자	소매 단가입니다.
RPkgPrice	숫자	소매 패키지 가격입니다.
MUnitPrice	숫자	제조업체 단가입니다.
MPkgPrice	숫자	제조업체 패키지 가격입니다.

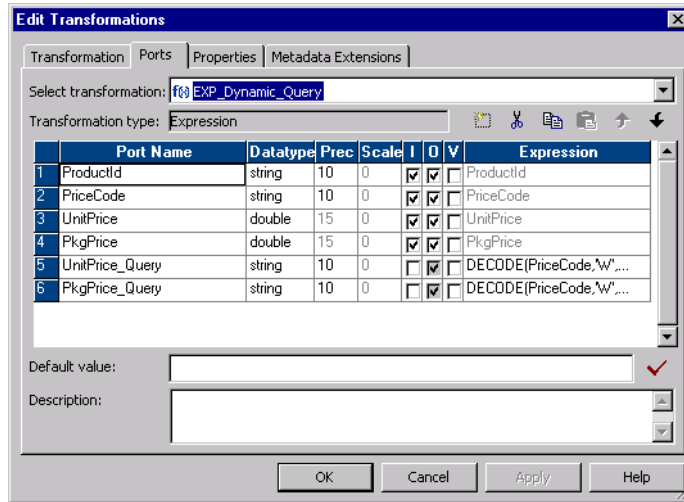
다음 SQL 문은 Oracle 데이터베이스에서 **Prod_Cost** 테이블 및 세 개의 제품 행을 작성합니다.

```
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;
```

식 변환 구성

식 변환은 각 소스 열에 대한 입력/출력 포트를 포함하며 PriceCode 열의 값에 따라 열 이름을 SQL 변환에 전달합니다.

다음 그림은 열 이름을 반환하는 식 변환의 포트를 보여 줍니다.



SQL 변환에는 식 결과를 포함하는 다음과 같은 열이 있습니다.

- **UnitPrice_Query.** 가격 코드 "M", "R" 또는 "W"에 따라 열 이름 "MUnitprice", "RUnitPrice" 또는 "WUnitprice"를 반환합니다.
`DECODE(PriceCode, 'M', 'MUnitPrice', 'R', 'RUnitPrice', 'W', 'WUnitPrice')`
- **PkgPrice_Query.** 가격 코드 "M", "R" 또는 "W"에 따라 열 이름 "MPkgPrice", "RPkgPrice" 또는 "WPkgPrice"를 반환합니다.
`DECODE(PriceCode, 'M', 'MPkgPrice', 'R', 'RPkgPrice', 'W', 'WPkgPrice')`

SQL 변환 정의

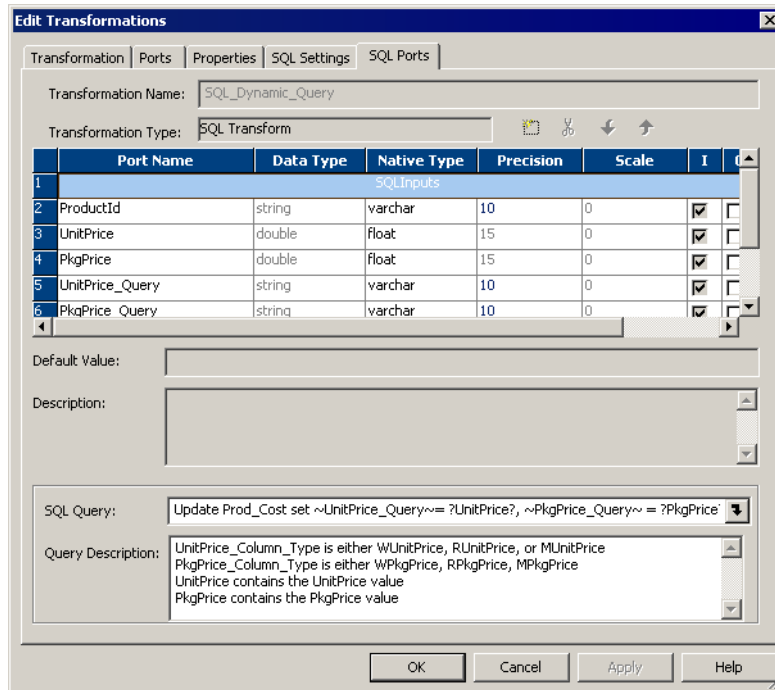
SQL 변환은 단가 및 패키지 가격 데이터를 Prod_Cost 테이블에 삽입하는 동적 SQL 쿼리를 실행합니다. SQL 변환은 열 이름을 받아 UnitPrice_Query 및 PkgPrice_Query 포트를 업데이트합니다.

SQL 변환을 작성할 때 변환 모드, 데이터베이스 유형 및 연결 유형을 정의합니다. 변환을 작성한 후에는 모드 또는 연결 유형을 변경할 수 없습니다.

다음과 같은 속성을 사용하여 SQL 변환을 작성합니다.

- **쿼리 모드.** SQL 변환이 동적 SQL 쿼리를 실행합니다.
- **정적 연결.** SQL 변환이 사용자가 워크플로우 관리자에서 정의하는 연결 개체를 사용하여 데이터베이스에 한 번 연결합니다.

다음 그림은 SQL 쿼리 및 쿼리 설명이 포함된 SQL 변환 포트 탭을 보여 줍니다.



SQL 변환은 UnitPrice_Query 및 PkgPrice_Query 포트에서 받는 열 이름에 따라 Prod_Cost 테이블에서 UnitPrice 열 중 하나와 PkgPrice 열 중 하나를 업데이트하는 동적 SQL 쿼리를 포함합니다.

SQL 변환에 다음 쿼리가 있습니다.

```
Update Prod_Cost set ~UnitPrice_Query~= ?UnitPrice?, ~PkgPrice_Query~= ?PkgPrice? where ProductId = ?
ProductId?;
```

SQL 변환은 UnitPrice_Query 및 PkgPrice_Query 문자열 변수를 업데이트할 열 이름으로 대체합니다.

SQL 변환은 쿼리의 ProductId, UnitPrice 및 PkgPrice 매개 변수를 해당 포트에서 받는 데이터에 바인딩합니다.

예를 들어 다음 소스 행에는 제품 100에 대한 단가와 패키지 가격이 있습니다.

```
100,M,100,110
```

PriceCode가 "M"이면 가격이 제조 가격입니다. 식 변환은 MUnitprice 및 MPkgPrice 열 이름을 업데이트할 SQL 변환에 전달합니다.

SQL 변환은 다음 쿼리를 실행합니다.

```
Update Prod_Cost set MUnitprice = 100, MPkgPrice = 110 where ProductId = '100';
```

다음 소스 행에는 제품 100에 대한 도매 가격이 있습니다.

```
100,W,120,200
```

식 변환은 WUnitprice 및 WPkgPrice 열 이름을 SQL 변환에 전달합니다. SQL 변환은 다음 쿼리를 실행합니다.

```
Update Prod_Cost set WUnitprice = 120, WPkgPrice = 200 where ProductId = '100';
```

세션 특성 구성

세션을 구성할 때 소스 파일, 대상 파일 및 데이터베이스 연결을 구성합니다.

특성	값
소스 파일 이름	PPrices.dat
출력 파일 이름	ErrorFile.dat
SQL_Dynamic_Query 관계형 연결	Prod_Cost 테이블이 포함된 테스트 데이터베이스에 대한 연결입니다.

대상 데이터 결과

샘플 데이터로 세션을 실행하면 통합 서비스가 **Prod_Cost** 대상 테이블을 다음 데이터로 채웁니다.

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

데이터베이스가 SQL 변환으로 오류를 반환하는 경우, 오류 텍스트는 **Error_File** 대상에 포함됩니다.

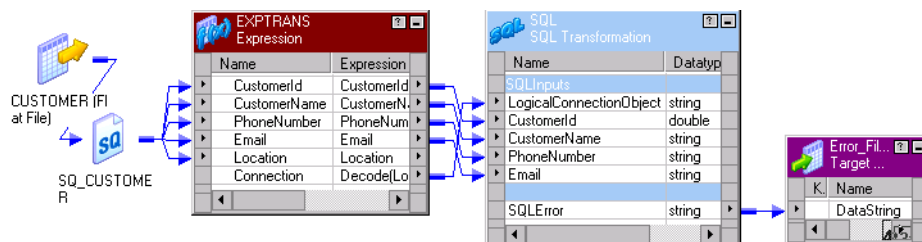
동적 연결 예제

동적 연결 SQL 변환 예제는 소스 파일 데이터에 따라 동적으로 데이터베이스에 연결하는 방법을 보여 줍니다.

이 예제에서는 미국, 영국 및 캐나다의 고객 데이터베이스를 사용합니다. 트랜잭션 파일의 고객 데이터를 고객의 위치에 따라 데이터베이스에 삽입해야 합니다.

식 변환은 위치 열의 값에 따라 데이터베이스 연결 개체 이름을 반환합니다. 식 변환은 연결 개체 이름을 SQL 변환 **LogicalConnectionObject** 포트에 전달합니다. SQL 변환은 **LogicalConnectionObject** 열의 값에 따라 데이터베이스에 연결합니다.

다음 그림은 매핑 내 식 변환과 SQL 변환을 보여 줍니다.



매핑에는 다음과 같은 구성 요소가 있습니다.

- **고객 소스 정의.** 고객 정보를 포함하는 플랫폼 파일 소스 정의입니다. 고객 위치에 따라 SQL 변환이 고객 데이터를 삽입할 때 연결하는 데이터베이스가 결정됩니다.
- **Error_File 대상 정의.** 대상에 SQL 변환에서 데이터베이스 오류를 받는 Datastring 필드가 있습니다.
- **Exp_Dynamic_Connection 변환.** 식 변환은 위치 열의 값에 따라 연결할 데이터베이스를 정의합니다. 식 변환은 연결 포트에서 연결 개체 이름을 받습니다. 연결 개체는 워크플로우 관리자에서 정의된 데이터베이스 연결입니다.
- **SQL_Dynamic_Connection 변환.** SQL 변환은 LogicalConnectionPort에서 연결 개체 이름을 받고, 데이터베이스에 연결하여 고객 데이터를 데이터베이스에 삽입합니다.

소스 파일 정의

트랜잭션 파일은 데이터베이스에 추가할 고객이 포함된 플랫폼 파일 소스입니다. 각 행에는 고객이 거주하는 국가를 정의하는 위치가 있습니다. 소스 파일 이름은 Customer.dat입니다.

고객 파일 레코드에는 다음 필드가 포함됩니다.

포트 이름	데이터 유형	전체 자릿수	소수 자릿수	설명
CustomerID	숫자	10	0	고객을 고유하게 식별하는 고객 번호입니다.
CustomerName	문자열	25	0	고객 이름입니다.
PhoneNumber	문자열	15	0	전화 번호에는 공백 없이 지역 번호 및 7자리 숫자가 포함됩니다.
전자 메일	문자열	25	0	고객 전자 메일 주소입니다.
위치	문자열	10	0	고객 위치입니다. 값은 US, UK, CAN입니다.

Srcfiles에서 다음 행이 포함된 Customer.dat 파일을 작성할 수 있습니다.

```
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 파일을 가져와서 리포지토리에서 고객 소스 정의를 작성할 수 있습니다.

대상 정의 작성

Error_File은 SQL 변환에서 데이터베이스 오류 메시지를 받는 플랫폼 파일 대상 정의입니다.

Error_File 정의에는 다음과 같은 입력 포트가 포함됩니다.

포트 이름	데이터 유형	전체 자릿수	소수 자릿수
DataString	문자열	4000	0

대상 디자이너에서 대상 정의를 작성합니다.

데이터베이스 테이블 작성

이 예제에서는 3개의 Oracle 데이터베이스, 즉 미국 데이터베이스, 영국 데이터베이스 및 캐나다 데이터베이스를 작성해야 합니다. 각 데이터베이스에는 해당 지역의 고객 데이터를 포함하는 **Cust** 테이블이 있습니다.

다음 SQL 문은 Oracle 데이터베이스에서 **Cust** 테이블을 작성합니다.

```
Create table Cust (CustomerId number, CustomerName varchar2(25), PhoneNumber varchar2(15), Email
varchar2(25));
```

참고: 이 예제에는 동적 데이터베이스 연결을 보여 주는 3개의 데이터베이스가 있습니다. 테이블이 동일한 데이터베이스에 있을 경우 정적 데이터베이스 연결을 사용하면 성능이 향상됩니다.

데이터베이스 연결 작성

워크플로우 관리자를 사용하여 **US**, **UK** 및 **CAN** 데이터베이스에 대한 데이터베이스 연결을 작성할 수 있습니다.

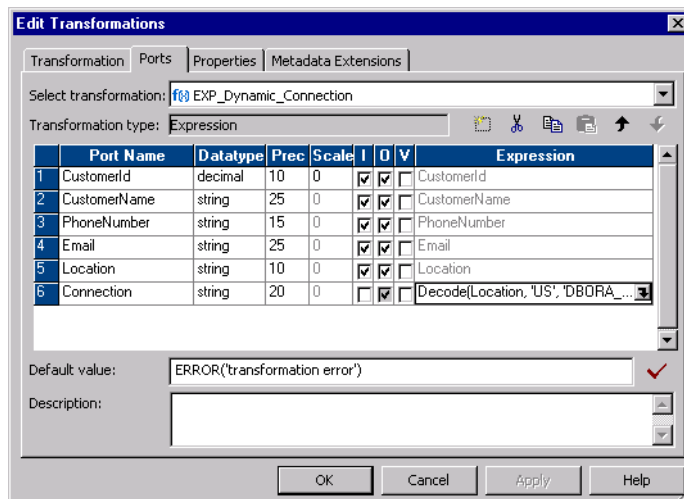
이 예제에서 SQL 변환은 다음과 같은 연결 이름을 사용하여 데이터베이스에 연결합니다.

데이터베이스	연결 개체 이름
미국	DBORA_US
영국	DBORA_UK
CAN	DBORA_CAN

식 변환 구성

식 변환은 각 소스 열에 대한 입력/출력 포트를 포함하며 해당 변환은 식의 결과를 기반으로 연결 포트의 연결 개체 이름을 반환합니다.

다음 그림에는 식 변환의 포트 탭 및 각 소스 열에 대한 입력/출력 포트 열이 나와 있습니다.



연결 포트에는 다음 식이 있습니다.

```
Decode(Location, 'US', 'DBORA_US', 'UK', 'DBORA_UK', 'CAN', 'DBORA_CAN', 'DBORA_US')
```

해당 식은 위치가 **US**, **UK** 또는 **CAN**인지 여부에 따라 연결 개체 이름을 반환합니다. 해당 위치가 식의 위치와 일치하지 않는 경우 연결 개체 이름은 기본적으로 “**DBORA_US**”입니다.

예를 들어 미국의 고객에 대한 다음 소스 행 고객 정보가 있습니다.

1,John Smith,6502345677,jsmith@catgary.com,US

고객 위치가 “US”인 경우 식 변환이 “DBORA_US” 연결 개체 이름을 반환합니다. 식 변환이 “DBORA_US”를 SQL 변환 LogicalConnectionObject 포트에 전달합니다.

SQL 변환 정의

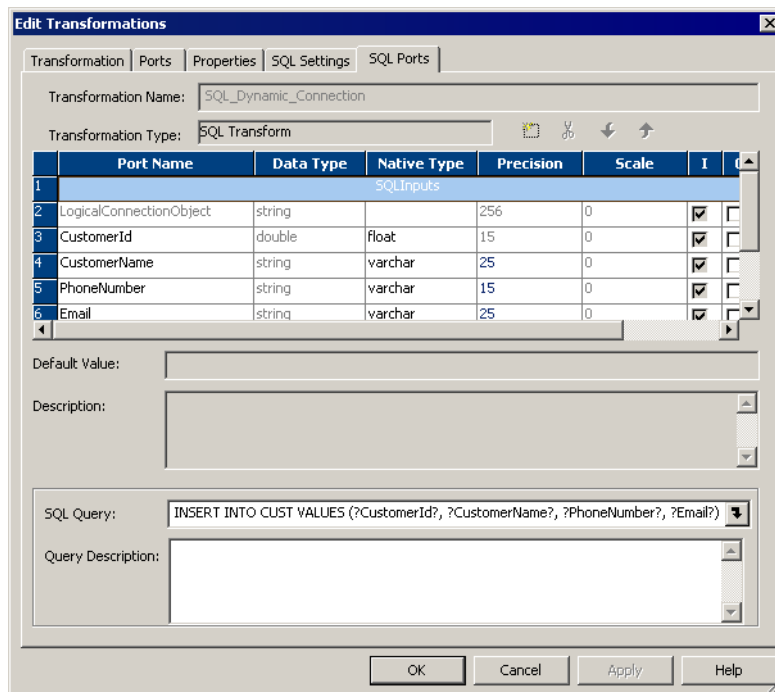
SQL 변환은 데이터베이스에 연결하고 고객 데이터를 CUST 테이블에 삽입하는 동적 SQL 쿼리를 실행합니다.

SQL 변환을 작성할 때 변환 모드, 데이터베이스 유형 및 연결 유형을 정의합니다. 변환을 작성한 후에는 모드 또는 연결 유형을 변경할 수 없습니다.

다음과 같은 속성을 사용하여 SQL 변환을 작성합니다.

- **쿼리 모드.** SQL 변환이 동적 SQL 쿼리를 실행합니다.
- **동적 연결에 해당합니다.** SQL 변환이 사용자가 매핑에서 변환에 전달하는 연결 정보에 따라 데이터베이스에 연결합니다.
- **연결 개체에 해당합니다.** SQL 변환에 연결 개체 이름을 받는 LogicalConnectionObject 포트가 있습니다. 위크플로우 관리자 연결에서 연결 개체를 정의해야 합니다.

다음 그림은 SQL 변환의 포트를 보여 줍니다.



SQL 변환은 LogicalConnectionObject 포트에서 연결 개체 이름을 받고, 행을 처리할 때마다 연결 개체 이름을 사용하여 데이터베이스에 연결합니다.

변환에 고객 데이터를 CUST 테이블에 삽입하는 다음과 같은 동적 SQL 쿼리가 있습니다.

```
INSERT INTO CUST VALUES (?CustomerId?,?CustomerName?,?PhoneNumber?,?Email?);
```

SQL 변환은 쿼리의 매개 변수를 변환의 입력 포트에서 받는 고객 데이터로 대체합니다. 예를 들어 다음 소스 행에는 고객 번호 1에 대한 고객 정보가 있습니다.

1,John Smith,6502345677,jsmith@catgary.com,US

SQL 변환은 DBORA_US 연결 개체를 사용하여 데이터베이스에 연결하고 다음과 같은 SQL 쿼리를 실행합니다.

```
INSERT INTO CUST VALUES (1,'John Smith','6502345677','jsmith@catgary.com');
```

세션 특성 구성

다음 소스 및 대상에 액세스하도록 세션을 구성합니다.

특성	값
소스 파일 이름	Customer.dat
출력 파일 이름	Error_File.dat

참고: 세션에 대한 데이터베이스 연결은 구성하지 마십시오. SQL 변환은 입력 행을 처리할 때마다 다른 데이터베이스에 연결할 수 있습니다.

대상 데이터 결과

샘플 데이터로 세션을 실행하는 경우, 통합 서비스는 United States, United Kingdom 또는 Canada 데이터베이스의 Cust 테이블을 채웁니다.

United States 데이터베이스에는 다음과 같은 행이 있습니다.

CustomerID	CustomerName	PhoneNumber	전자 메일
1	John Smith	6502345677	jsmith@catagary.com
4	Robert Gregson	5423123453	rgregson@networkmail.com

United Kingdom 데이터베이스에는 다음과 같은 행이 있습니다.

CustomerID	CustomerName	PhoneNumber	전자 메일
2	Nigel Whitman	5123456754	nwhitman@nwhitman.com

Canadian 데이터베이스에는 다음과 같은 행이 있습니다.

CustomerID	CustomerName	PhoneNumber	전자 메일
3	Girish Goyal	5423123453	ggoyal@netcompany.com

데이터베이스가 SQL 변환으로 오류를 반환하는 경우, 오류 텍스트는 Error_File 대상에 포함됩니다.

제 28 장

저장 프로시저 변환

이 장에 포함된 항목:

- [저장 프로시저 변환 개요, 416](#)
- [매핑에서 저장 프로시저 사용, 419](#)
- [저장 프로시저 작성, 420](#)
- [저장 프로시저 변환 작성, 422](#)
- [연결된 변환 구성, 426](#)
- [연결되지 않은 변환 구성, 426](#)
- [오류 처리, 430](#)
- [지원되는 데이터베이스, 430](#)
- [식 규칙, 431](#)
- [저장 프로시저 변환에 대한 팁, 432](#)
- [저장 프로시저 변환 문제 해결, 432](#)

저장 프로시저 변환 개요

저장 프로시저 변환은 데이터베이스를 채우고 유지 관리하기 위한 중요한 도구입니다. 데이터베이스 관리자는 저장 프로시저를 작성하여 표준 SQL 문으로 처리하기에는 복잡한 태스크를 자동화합니다. 저장 프로시저 변환은 수동 변환입니다. 연결되거나 연결되지 않은 저장 프로시저 변환을 구성할 수 있습니다.

저장 프로시저는 Transact-SQL, PL-SQL 또는 실행 가능한 스크립트와 유사한 기타 데이터베이스 프로시저 문 및 선택적 흐름 제어 문의 사전 컴파일된 컬렉션입니다. 저장 프로시저는 데이터베이스 내에서 저장되고 실행됩니다. 데이터베이스 클라이언트 도구에서 SQL 문을 실행하는 것처럼 EXECUTE SQL 문으로 저장 프로시저를 실행할 수 있습니다. 하지만 표준 SQL과 달리 저장 프로시저에서는 사용자 정의 변수, 조건문 및 기타 강력한 프로그래밍 기능을 사용할 수 있습니다.

일부 데이터베이스는 저장 프로시저를 지원하지 않으며 저장 프로시저 구문은 데이터베이스마다 다릅니다. 저장 프로시저를 사용하여 다음 태스크를 완료할 수도 있습니다.

- 대상 데이터베이스에 데이터를 로드하기 전에 해당 데이터베이스의 상태를 확인합니다.
- 데이터베이스에 충분한 공간이 있는지 확인합니다.
- 특수화된 계산을 수행합니다.
- 인덱스 삭제 및 다시 작성.

저장 프로시저가 SQL 문보다 훨씬 유연하기 때문에 데이터베이스 개발자와 프로그래머는 데이터베이스 내의 다양한 태스크에 저장 프로시저를 사용합니다. 또한 저장 프로시저에는 중요 태스크에 필요한 오류 처리 및 로깅

기능이 있습니다. 개발자는 데이터베이스와 함께 제공된 클라이언트 도구를 사용하여 데이터베이스에서 저장 프로시저를 작성합니다.

저장 프로시저가 데이터베이스에 존재해야 저장 프로시저 변환을 작성할 수 있으며, 저장 프로시저는 소스, 대상 또는 통합 서비스에 대한 유효한 연결이 있는 모든 데이터베이스에 존재할 수 있습니다.

저장 프로시저를 사용하여 일반적으로 매핑으로 처리해야 하는 쿼리 또는 계산을 수행할 수 있습니다. 예를 들어 판매세 계산을 위한 효율적으로 테스트된 저장 프로시저가 이미 있는 경우 식 변환에서 같은 계산을 다시 작성하는 대신 저장 프로시저를 사용하여 해당 계산을 수행할 수 있습니다.

입력 및 출력 데이터

저장 프로시저의 가장 유용한 기능 중 하나는 데이터를 저장 프로시저에 전송하고 저장 프로시저로부터 데이터를 수신하는 기능입니다. 통합 서비스와 저장 프로시저 간에 전달되는 데이터는 세 가지 유형이 있습니다.

- 입력/출력 매개 변수
- 반환 값
- 상태 코드

데이터베이스 구현에 따라 데이터 전달에 제한이 존재하며 이에 대해 이 장 전체에서 논의합니다. 또한 모든 저장 프로시저가 데이터를 전송하고 수신하는 것은 아닙니다. 예를 들어 세션이 끝날 때 데이터베이스 인덱스를 다시 구성하도록 저장 프로시저를 작성하면 세션이 이미 완료되었기 때문에 데이터를 수신할 수 없습니다.

입력/출력 매개 변수

많은 저장 프로시저에서 값을 제공하고 반환 값을 받습니다. 이러한 값을 입력 및 출력 매개 변수라고 합니다. 예를 들어 판매 세금 계산 저장 프로시저에서 품목의 가격과 같은 단일 매개 변수를 취할 수 있습니다. 계산을 수행한 후 저장 프로시저는 두 개의 출력 매개 변수, 즉 세액 및 세금을 포함한 품목의 총 가격을 반환합니다.

저장 프로시저 변환은 포트, 변수를 사용하거나 식에 **10** 또는 **SALES** 등의 값을 입력하여 입력 및 출력 매개 변수를 보내고 받습니다.

반환 값

대부분의 데이터베이스는 저장 프로시저를 실행한 후 반환 값을 제공합니다. 데이터베이스 구현에 따라 이 값은 사용자가 정의할 수 있어 단일 출력 매개 변수와 유사하게 사용할 수 있는 값이거나 단순히 정수 값만 반환하는 형태일 수 있습니다.

저장 프로시저 변환은 입력/출력 매개 변수가 표현된 메서드에 따라 입력/출력 매개 변수와 유사한 방식으로 반환 값을 표현합니다. 경우에 따라서는 매개 변수 또는 반환 값만 표현할 수 있습니다.

저장 프로시저가 단일 반환 값 대신 결과 집합을 반환하는 경우 저장 프로시저 변환은 프로시저에서 반환된 첫 번째 값만 가져옵니다.

참고: Oracle 저장 함수는 Oracle 저장 프로시저와 유사하지만, 저장 함수가 출력 매개 변수 또는 반환 값을 지원하지 않는다는 점이 다릅니다. 따로 언급되어 있지 않는 한, 이 장에서 저장 프로시저와 관련된 모든 설명은 저장 함수에도 적용됩니다.

상태 코드

상태 코드는 워크플로우 중에 오류 처리 기능을 통합 서비스에 제공합니다. 저장 프로시저는 저장 프로시저의 성공적 완료 여부를 알려 주는 상태 코드를 발급합니다. 사용자가 이 값을 볼 수는 없습니다. 통합 서비스는 이 값을 사용하여 세션을 계속 실행하지 또는 중지할지 결정합니다. 저장 프로시저 오류 시에 세션을 계속하거나 중지하려면 워크플로우 관리자에서 옵션을 구성합니다.

연결됨 및 연결되지 않음

저장 프로시저는 연결된 모드 또는 연결되지 않은 모드에서 실행됩니다. 사용하는 모드는 저장 프로시저가 무엇을 수행하는지 그리고 사용자가 세션에서 이 저장 프로시저를 어떻게 사용할 계획인지에 따라 다릅니다. 매핑에서 연결된 저장 프로시저 변환 및 연결되지 않은 저장 프로시저 변환을 구성할 수 있습니다.

- **연결됨.** 연결된 모드에서 매핑을 통과하는 데이터 흐름은 저장 프로시저 변환을 통해서도 전달됩니다. 입력 포트를 통해 변환에 들어가는 모든 데이터는 저장 프로시저에 영향을 미칩니다. 저장 프로시저에 입력 매개 변수로 전송된 입력 포트의 데이터 또는 다른 변환에 출력 매개 변수로 전송된 저장 프로시저의 결과가 필요한 경우 연결된 저장 프로시저 변환을 사용해야 합니다.
- **연결되지 않음.** 연결되지 않은 저장 프로시저 변환은 매핑 흐름에 직접 연결되지 않습니다. 이 변환은 세션 전 또는 후에 실행되거나, 매핑의 다른 변환에서 식에 의해 호출됩니다.

다음 테이블에는 연결된 변환과 연결되지 않은 변환이 비교되어 있습니다.

수행할 작업	다음 모드 사용
세션 전 또는 후에 저장 프로시저를 실행합니다.	연결되지 않음
사전 세션 또는 사후 세션 등 매핑 중에 저장 프로시저를 한 번 실행합니다.	연결되지 않음
행이 저장 프로시저 변환을 통해 전달될 때마다 저장 프로시저를 실행합니다.	연결됨 또는 연결되지 않음
특정 포트가 Null 값을 포함하는 않는 경우 등 매핑을 통해 전달되는 데이터를 기반으로 저장 프로시저를 실행합니다.	연결되지 않음
저장 프로시저에 매개 변수를 전달하고 단일 출력 매개 변수를 수신합니다.	연결됨 또는 연결되지 않음
저장 프로시저에 매개 변수를 전달하고 여러 출력 매개 변수를 수신합니다. 참고: 연결되지 않은 저장 프로시저 변환에서 여러 출력 매개 변수를 가져오려면 각 출력 매개 변수에 대한 변수를 작성해야 합니다.	연결됨 또는 연결되지 않음
중첩 저장 프로시저를 실행합니다.	연결되지 않음
매핑 내에서 여러 번 호출합니다.	연결되지 않음

관련 항목:

- [“연결되지 않은 변환 구성” 페이지 426](#)
- [“연결된 변환 구성” 페이지 426](#)

저장 프로시저 실행 시기 지정

저장 프로시저 변환의 모드를 지정할 수 있을 뿐만 아니라 실행 시기도 지정할 수 있습니다. 위의 연결되지 않은 저장 프로시저에서 식 변환이 저장 프로시저를 참조합니다. 즉, 식 변환을 통해 행이 전달될 때마다 저장 프로시저가 실행됩니다. 저장 프로시저 변환을 참조하는 변환이 없는 경우에는 세션 전 또는 후에 저장 프로시저를 한 번 실행하는 옵션을 사용할 수 있습니다.

다음 목록에는 저장 프로시저 변환을 실행하는 옵션이 설명되어 있습니다.

- **보통.** 매핑에서 행별 기준으로 변환이 존재할 때마다 저장 프로시저가 실행됩니다. 이 옵션은 입력 포트에 대한 계산을 실행하는 것처럼, 매핑을 통해 전달하는 데이터의 각 행에 대해 저장 프로시저를 호출하는 경우 유용합니다. 연결된 저장 프로시저는 일반 모드에서만 실행됩니다.

- **소스의 사전 로드.** 세션이 소스에서 데이터를 검색하기 전에 저장 프로시저가 실행됩니다. 임시 테이블에서 데이터 조인을 수행하거나 테이블이 있는지 확인하는 데 유용합니다.
- **소스의 사후 로드.** 세션이 소스에서 데이터를 검색한 후에 저장 프로시저가 실행됩니다. 임시 테이블을 제거하는 데 유용합니다.
- **대상의 사전 로드.** 세션이 대상에 데이터를 보내기 전에 저장 프로시저가 실행됩니다. 대상 시스템에서 디스크 공간이나 대상 테이블을 확인하는 데 유용합니다.
- **대상의 사후 로드.** 세션이 대상에 데이터를 보낸 후에 저장 프로시저가 실행됩니다. 데이터베이스에서 인덱스를 다시 작성하는 데 유용합니다.

동일한 매핑에서 둘 이상의 저장 프로시저 변환을 서로 다른 모드로 실행할 수 있습니다. 예를 들어 사전 로드 소스 저장 프로시저는 테이블 무결성을 확인하고, 일반 저장 프로시저는 테이블을 채우고, 사후 로드 저장 프로시저는 데이터베이스에서 인덱스를 다시 작성할 수 있습니다. 하지만 한 매핑에서 저장 프로시저 변환의 동일한 인스턴스를 동시에 연결된 모드와 연결되지 않은 모드로 실행할 수 없습니다. 변환에 대한 다른 인스턴스를 작성해야 합니다.

매핑에서 둘 이상의 소스 또는 대상 사전 또는 사후 로드 저장 프로시저가 호출되는 경우 통합 서비스는 매핑에 지정되어 있는 실행 순서대로 저장 프로시저를 실행합니다.

통합 서비스는 변환 속성에 지정되어 있는 데이터베이스 연결을 사용하여 각 저장 프로시저를 실행합니다. 통합 서비스는 첫 번째 저장 프로시저를 발견했을 때 데이터베이스 연결을 엽니다. 이 데이터베이스 연결은 통합 서비스가 해당 연결의 모든 저장 프로시저 처리를 마칠 때까지 유지됩니다. 통합 서비스는 다른 데이터베이스 연결을 사용하는 저장 프로시저를 발견한 경우 기존 데이터베이스 연결을 닫고 새 데이터베이스 연결을 엽니다.

동일한 데이터베이스 연결을 사용하는 여러 개의 저장 프로시저를 실행하려면 해당 저장 프로시저가 연속적으로 실행되도록 설정하십시오. 저장 프로시저가 연속적으로 실행되도록 설정하지 않으면 대상에서 예기치 않은 결과가 발생할 수 있습니다. 예를 들어 두 개의 저장 프로시저, 저장 프로시저 A와 저장 프로시저 B가 있다고 가정합니다. 저장 프로시저 A는 트랜잭션을 시작하고 저장 프로시저 B는 트랜잭션을 커밋합니다. 저장 프로시저 B보다 먼저 다른 데이터베이스 연결을 사용하는 저장 프로시저 C를 실행하면 저장 프로시저 B가 트랜잭션을 커밋할 수 없습니다. 이는 통합 서비스가 저장 프로시저 C를 실행하면서 기존 데이터베이스 연결을 닫기 때문입니다.

한 데이터베이스 연결에서 여러 저장 프로시저를 실행하려면 다음 지침을 따르십시오.

- 저장 프로시저는 저장 프로시저 속성에 정의되어 있는 동일한 데이터베이스 연결 문자열을 사용해야 합니다.
- 저장 프로시저가 연속적인 순서로 실행되도록 설정해야 합니다.
- 저장 프로시저는 저장 프로시저 유형이 동일해야 합니다.
 - 소스 사전 로드
 - 소스 사후 로드
 - 대상 사전 로드
 - 대상 사후 로드

매핑에서 저장 프로시저 사용

매핑에서 저장 프로시저 변환을 사용하려면 여러 단계를 수행해야 합니다. 저장 프로시저가 데이터베이스 내에 존재하므로 매핑 및 세션만이 아니라 데이터베이스의 저장 프로시저도 구성해야 합니다.

저장 프로시저 변환을 사용하려면 다음 단계를 완료하십시오.

1. 데이터베이스에서 저장 프로시저를 작성합니다.

디자인러를 사용하여 변환을 작성하기 전에 데이터베이스에서 저장 프로시저를 작성해야 합니다. 또한 제공된 데이터베이스 클라이언트 도구를 통해 저장 프로시저를 테스트해야 합니다.

2. 저장 프로시저 변환을 가져오거나 작성합니다.
디자이너를 사용하여 모든 필요한 입력/출력 및 반환 값에 대한 포트를 제공하고 저장 프로시저 변환을 가져오거나 작성합니다.
3. 변환을 연결된 변환으로 사용할지, 아니면 연결되지 않은 변환으로 사용할지를 결정합니다.
변환을 구성하기 전에 저장 프로시저와 매핑을 연결하는 방법을 결정해야 합니다.
4. 연결되는 경우 적절한 입력 및 출력 포트를 매핑하십시오.
대부분의 다른 변환을 사용하는 것과 마찬가지로 연결된 저장 프로시저 변환을 사용합니다. 적절한 입력 흐름 포트를 변환으로 끌어 출력 포트에서 다른 변환으로의 매핑을 작성합니다.
5. 연결되지 않는 경우 사전 또는 사후 세션을 실행하도록 저장 프로시저를 구성하거나 다른 변환의 식에서 실행되도록 저장 프로시저를 구성합니다.
저장 프로시저는 세션 이전이나 이후에 실행될 수 있으므로 연결되지 않은 변환이 실행되는 시기를 지정해야 할 수 있습니다. 반대로, 저장 프로시저가 다른 변환에서 호출되는 경우에는 다른 변환에서 저장 프로시저를 호출하는 식을 작성해야 합니다. 식은 변수를 포함할 수 있으며 반환 값을 포함하거나 포함하지 않을 수 있습니다.
6. 세션을 구성합니다.
워크플로우 관리자의 세션 속성에는 저장 프로시저 실행 시의 오류 처리를 위한 옵션과 다양한 SQL 재정의 옵션이 포함되어 있습니다.

저장 프로시저 작성

데이터베이스에서 SQL 문을 작성하여 저장 프로시저를 작성하고 다른 Transact-SQL 문과 데이터베이스별 함수를 추가할 수 있습니다. 여기에는 사용자 정의 데이터 유형과 실행 순서 문이 포함될 수 있습니다.

샘플 저장 프로시저

다음 예에서 소스 데이터베이스에는 직원 ID 번호를 입력 매개 변수로 가져와 직원 이름을 출력 매개 변수로 반환하는 저장 프로시저가 있습니다. 또한 저장 프로시저가 성공적으로 완료되면 알림으로 반환 값 0이 반환됩니다.

직원 ID와 이름을 포함하는 데이터베이스 테이블은 다음과 같이 표시됩니다.

직원 ID	직원 이름
101	Bill Takash
102	Louis Li
103	Sarah Ferguson

저장 프로시저는 직원 ID 101을 입력 매개 변수로 받아 이름 Bill Takash를 반환합니다. 매핑에서 이 저장 프로시저를 호출하는 방식에 따라 ID 중 하나 또는 전부가 저장 프로시저로 전달될 수 있습니다.

구문은 데이터베이스마다 다르므로 이 저장 프로시저를 작성하는 SQL 문이 다를 수 있습니다. SQL 문을 데이터베이스로 전달하는 데 사용되는 클라이언트 도구도 달라집니다. 대부분의 데이터베이스는 표준 SQL 편집기를 비롯한 일련의 클라이언트 도구를 제공합니다. Microsoft SQL Server와 같은 일부 데이터베이스는 초기 SQL 문 중 일부를 작성하는 도구를 제공합니다.

참고: 세션에 저장 프로시저 인수로 대형 개체가 포함된 경우 통합 서비스에서 세션이 실패합니다.

Informix

Informix에서 출력 매개 변수를 선언하는 구문은 다른 데이터베이스와 다릅니다. 대부분의 데이터베이스에서는 변수가 입력 또는 출력 매개 변수로 작동하는지 지정하기 위해 **IN** 또는 **OUT**을 사용하여 변수를 선언합니다. Informix에서는 **RETURNING** 키워드를 사용하므로 입력/출력 매개 변수와 반환 값을 구분하기 어렵습니다. 예를 들어 **RETURN** 명령을 사용하여 하나 이상의 출력 매개 변수를 반환할 수 있습니다.

```
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;
```

이 경우 **RETURN** 문은 **outVAR**의 값을 전달합니다. 하지만 다른 데이터베이스와 달리 **outVAR**는 반환 값이 아닌 출력 매개 변수입니다. 다음과 같은 방식으로 여러 출력 매개 변수가 반환됩니다.

```
return outVAR1, outVAR2, outVAR3
```

Informix는 반환 값을 전달하지 않습니다. 반환 값은 사용자 정의되지 않고 오류 확인 값으로 생성됩니다. 변환에서 **R** 값을 확인해야 합니다.

Oracle

Oracle에서는 값을 반환하는 모든 저장 프로시저를 저장 함수라고 합니다. **CREATE PROCEDURE** 문을 사용하여 예제에 기반한 새 저장 프로시저를 작성하는 대신 **CREATE FUNCTION** 문을 사용합니다. 이 샘플에서 변수는 **IN** 및 **OUT**으로 선언되어 있지만 Oracle은 **INOUT** 매개 변수 유형도 지원합니다. 이 매개 변수 유형을 사용하면 매개 변수에 값을 전달하고 수정한 다음 수정된 값을 반환할 수 있습니다.

```
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;
/
```

반환 값은 **VARCHAR2** 데이터 유형의 문자열 값(Success)입니다. Oracle은 반환 값에 문자열 데이터 유형을 사용할 수 있는 유일한 데이터베이스입니다.

Sybase ASE 및 Microsoft SQL Server

Sybase와 Microsoft는 다음 구문에 표시된 것처럼 저장 프로시저를 동일하게 구현합니다.

```
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
```

반환 값이 변수일 필요는 없습니다. 이 경우 **SELECT** 문이 성공하면 반환 값으로 **0**이 반환됩니다.

IBM DB2

다음 텍스트는 IBM DB2 기반 SQL 저장 프로시저의 예제입니다.

```
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 ' ';
-- Declare 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

다음 텍스트는 Teradata의 SQL 저장 프로시저 예입니다. 이 저장 프로시저는 입력 매개 변수로 직원 ID 번호를 가져와 출력 매개 변수로 직원 이름을 반환합니다.

```

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;

```

저장 프로시저 변환 작성

데이터베이스에서 저장 프로시저를 구성하고 테스트한 후 매핑 디자이너에서 저장 프로시저 변환을 작성해야 합니다. 저장 프로시저 변환을 구성하는 방법은 두 가지가 있습니다.

- 저장 프로시저 가져오기 대화 상자를 사용하여 저장 프로시저에서 사용하는 포트를 구성합니다.
- 입력 또는 출력 매개 변수에 대한 적절한 포트를 작성하여 변환을 수동으로 구성합니다.

저장 프로시저 변환은 기본적으로 일반 유형으로 작성되므로 세션 전 또는 후가 아닌 매핑 중에 실행됩니다.

새 저장 프로시저 변환은 재사용 가능 변환으로 작성되지 않습니다. 재사용 가능 변환을 작성하려면 변환을 작성한 후 변환 속성에서 재사용 가능 설정을 클릭합니다.

참고: 변환에 대해 전역적으로 변경을 수행하려면 매핑 디자이너가 아닌 변환 개발자에서 재사용 가능 변환의 속성을 구성합니다.

저장 프로시저 가져오기

저장 프로시저를 가져오는 경우 디자이너가 저장 프로시저 입력 및 출력 매개 변수를 기반으로 포트를 작성합니다. 가능할 때마다 저장 프로시저를 가져와야 합니다.

매핑 디자이너에서 저장 프로시저를 가져오는 세 가지 방법이 있습니다.

- 저장 프로시저 아이콘을 선택하고 저장 프로시저 변환을 추가합니다.
- 변환 > 저장 프로시저 가져오기를 클릭합니다.
- 변환 > 작성을 클릭하고 저장 프로시저를 선택합니다.

저장 프로시저 이름에 마침표(.)가 포함된 저장 프로시저를 가져오는 경우 디자이너가 저장 프로시저 변환 이름의 마침표를 밑줄(_)로 대체합니다.

저장 프로시저를 가져오려면:

1. 매핑 디자이너에서 변환 > 저장 프로시저 가져오기를 클릭합니다.

2. ODBC 소스 목록에서 저장 프로시저가 포함된 데이터베이스를 선택합니다. 데이터베이스에 연결하기 위해 사용자 이름, 소유자 이름 및 암호를 입력하고 연결을 클릭합니다.

대화 상자의 폴더에 **FUNCTIONS**가 표시됩니다. 이 폴더의 저장 프로시저에 입력 매개 변수, 출력 매개 변수 또는 반환 값이 포함되어 있습니다. 매개 변수 또는 반환 값이 포함되지 않은 데이터베이스에 저장 프로시저가 존재하는 경우 **PROCEDURES**라는 폴더에 이 저장 프로시저가 나타납니다. 이는 주로 **Oracle** 저장 프로시저에 적용됩니다. 함수 목록에 나타나는 연결된 일반 저장 프로시저의 경우 하나 이상의 입력/출력 포트가 필요합니다.

팁: 건너뛰기를 선택하여 저장 프로시저를 가져오지 않고 저장 프로시저 변환을 추가할 수 있습니다. 이러한 경우 수동으로 포트를 추가하고 변환 내에서 정보를 연결해야 합니다.

3. 필요에 따라, 검색 필드를 사용하여 나타나는 프로시저 수를 제한합니다.

4. 가져올 프로시저를 선택하고 확인을 클릭합니다.

저장 프로시저 변환이 매핑에 나타납니다. 저장 프로시저 변환 이름은 선택한 저장 프로시저 이름과 같아야 합니다. 저장 프로시저에 입력 매개 변수, 출력 매개 변수 또는 반환 값이 포함된 경우 저장 프로시저 변환에서 각 매개 변수 또는 반환 값과 일치하는 해당 포트가 표시됩니다.

이 저장 프로시저 변환에서 다음 값과 매개 변수가 해당 저장 프로시저에 포함되어 있는지 볼 수 있습니다.

- 정수 반환 값, **RETURN_VALUE**, 출력 포트 포함.
- 문자열 입력 매개 변수, **nNAME**, 입력 포트 포함.
- 정수 출력 매개 변수, **outVar**, 입력 및 출력 포트 포함.

참고: 변환 이름을 변경하는 경우 변환 속성에서 저장 프로시저 이름을 구성해야 합니다. 매핑에 동일한 저장 프로시저의 인스턴스가 여러 개 있는 경우 저장 프로시저 이름도 구성해야 합니다.

5. 변환을 열고 속성 탭을 클릭합니다.

연결 정보 행에서 저장 프로시저가 있는 데이터베이스를 선택합니다. 저장 프로시저 변환 이름을 저장 프로시저 이름이 아닌 다른 이름으로 변경한 경우 해당 저장 프로시저 이름을 입력합니다.

6. 확인을 클릭합니다.

수동으로 저장 프로시저 변환 작성

수동으로 저장 프로시저 변환을 작성하려면 저장 프로시저의 입력 매개 변수, 출력 매개 변수 및 반환 값(있는 경우)을 알아야 합니다. 또한 이러한 매개 변수의 데이터 유형과 저장 프로시저의 이름도 알아야 합니다. 이러한 모든 항목은 저장 프로시저 가져오기를 통해 구성합니다.

저장 프로시저 변환을 작성하려면

1. 매핑 디자이너에서 변환 > 작성을 클릭한 후 저장 프로시저를 선택합니다.

저장 프로시저 변환의 이름 지정 규칙은 자동으로 구성되는 저장 프로시저의 이름입니다. 변환 이름을 변경하려면 변환 속성에서 저장 프로시저의 이름을 구성해야 합니다. 매핑에 동일한 저장 프로시저의 인스턴스가 여러 개 있을 경우 이 단계를 수행해야 합니다.

2. 건너뛰기를 클릭합니다.

저장 프로시저 변환이 매핑 디자이너에 표시됩니다.

3. 변환을 열고 포트 탭을 클릭합니다.

저장 프로시저의 입력 매개 변수, 출력 매개 변수 및 반환 값을 기반으로 포트를 작성해야 합니다. 다음과 같은 저장 프로시저 매개 변수 각각에 대해 저장 프로시저 변환의 포트를 구성하십시오.

- 정수 입력 매개 변수
- 문자열 출력 매개 변수

- 반환 값

정수 입력 매개 변수의 경우 정수 입력 포트를 작성합니다. 매개 변수와 포트는 데이터 유형 및 전체 자릿수가 같아야 합니다. 출력 매개 변수 및 반환 값에 대해 이 작업을 반복합니다.

R 열을 선택해야 하고 반환 값에 대해 출력 포트를 선택해야 합니다. 매개 변수가 여러 개 있는 저장 프로시저의 경우 저장 프로시저에 표시되는 순서와 동일하게 포트를 나열해야 합니다.

4. 속성 탭을 클릭합니다.

저장 프로시저 이름 행에 저장 프로시저의 이름을 입력하고 연결 정보 행에서 저장 프로시저가 있는 데이터베이스를 선택합니다.

5. 확인을 클릭합니다.

리포지토리에서는 매핑의 유효성을 검사하고 매핑을 저장하지만 디자이너는 수동으로 입력된 저장 프로시저 변환의 유효성을 검사하지 않습니다. 적절한 매개 변수 또는 반환 값이 저장 프로시저에 있는지 확인하기 위한 검사가 전혀 수행되지 않습니다. 저장 프로시저 변환이 제대로 구성되지 않으면 세션이 실패합니다.

저장 프로시저 설정 옵션

다음 테이블에는 저장 프로시저 변환의 속성이 설명되어 있습니다.

설정	설명
저장 프로시저 이름	데이터베이스에 있는 저장 프로시저의 이름입니다. 변환의 이름이 데이터베이스에 있는 실제 저장 프로시저 이름과 다른 경우 통합 서비스는 이 텍스트를 사용하여 저장 프로시저를 호출합니다. 변환 이름이 저장 프로시저 이름과 일치하면 이 필드를 비워 두십시오. 저장 프로시저 가져오기 기능을 사용할 경우 이 이름은 저장 프로시저와 일치합니다.
연결 정보	<p>저장 프로시저를 포함하는 데이터베이스를 지정합니다. 매핑, 세션 또는 매개 변수 파일에서 데이터베이스를 정의할 수 있습니다.</p> <ul style="list-style-type: none"> - 매핑. 관계형 연결 개체를 선택합니다. - 세션. <code>\$Source</code> 또는 <code>\$Target</code> 연결 변수를 사용합니다. 이러한 변수 중 하나를 사용하는 경우, 세션을 실행할 때 지정된 소스 또는 대상 데이터베이스에 저장 프로시저가 있어야 합니다. 세션 속성에서 각 변수에 대해 데이터베이스 연결을 지정합니다. - 매개 변수 파일. 세션 매개 변수 <code>\$DBConnectionName</code>을 사용하고 매개 변수 파일에서 이 매개 변수를 정의합니다. <p>기본적으로 디자이너는 일반 저장 프로시저 유형에 대해 <code>\$Target</code>을 지정합니다. 소스 사전 로드 및 사후 로드의 경우 디자이너가 <code>\$Source</code>를 지정하고, 대상 사전 로드 및 사후 로드의 경우 디자이너가 <code>\$Target</code>을 지정합니다. 세션 속성에서 해당 값을 재정의할 수 있습니다.</p>
호출 텍스트	<p>저장 프로시저를 호출하는 데 사용되는 텍스트입니다. 저장 프로시저 유형이 일반이 아닌 경우에만 사용됩니다. 저장 프로시저로 전달되는 모든 입력 매개 변수를 호출 텍스트 내에 포함해야 합니다.</p> <p>PowerCenter 매개 변수 또는 변수를 호출 텍스트에서 사용할 수도 있습니다. 매개 변수 파일에 정의할 수 있는 모든 매개 변수 또는 변수 유형을 사용할 수 있습니다.</p>
저장 프로시저 유형	통합 서비스가 저장 프로시저를 호출하는 시기를 결정합니다. 이 옵션에는 일반(매핑 중)과 소스 또는 대상 데이터베이스에서의 사전 로드 또는 사후 로드가 포함됩니다. 기본값은 보통입니다.

설정	설명
추적 수준	세션 로그 파일에서 보고되는 트랜잭션 세부 정보의 양입니다. 다음과 같은 추적 수준을 사용합니다. <ul style="list-style-type: none"> - 간단 - 보통 - 자세한 정보 표시 초기화 - 자세한 정보 표시 데이터 기본값은 보통입니다.
실행 순서	변환에서 사용되는 저장 프로시저를 통합 서비스가 호출하는 순서이며, 이 순서는 동일한 매핑 내의 다른 저장 프로시저에 상대적입니다. 저장 프로시저 유형이 일반 이외의 다른 유형으로 설정되고 둘 이상의 저장 프로시저가 있는 경우에만 사용됩니다.
초 단위 이하 정밀도	날짜/시간 포트에 대한 초 단위 이하 정밀도를 지정합니다. 날짜/시간 데이터의 소수 자릿수가 편집 가능한 데이터베이스에 대해 전체 자릿수를 변경할 수 있습니다. Oracle 타임스탬프, Informix 날짜/시간 및 Teradata 타임스탬프 데이터 유형에 대한 초 단위 이하 정밀도를 변경할 수 있습니다. 0 ~ 9의 양의 정수 값을 입력합니다. 기본값은 6(마이크로초)입니다.
출력은 반복 가능합니다.	변환이 세션 실행 간에 동일한 순서로 행을 생성하는지 여부를 나타냅니다. 출력이 반복 가능하거나 확정 출력인 경우 통합 서비스가 마지막 검사점에서 세션을 다시 시작할 수 있습니다. 다음 값을 사용하십시오. <ul style="list-style-type: none"> - 항상 입력 데이터 순서가 세션 실행 간에 일관되지 않더라도 출력 데이터 순서는 세션 실행 간에 일관됩니다. - 입력 순서 기반. 모든 입력 그룹의 입력 데이터 순서가 세션 실행 간에 일관적인 경우 변환에서 세션 실행 간에 반복 가능한 데이터를 생성합니다. 입력 그룹의 입력 데이터가 정렬되지 않으면 출력이 정렬되지 않습니다. - 사용 안 함 출력 데이터 순서는 세션 실행 간에 일관되지 않습니다. 변환이 반복 가능 데이터를 생성하지 않으면 마지막 검사점에서 다시 시작하도록 복구를 구성할 수 없습니다. 기본값은 입력 순서 기반입니다.
확정 출력입니다.	변환이 세션 실행 간에 일치하는 출력 데이터를 생성하는지 여부를 나타냅니다. 이 변환을 사용하는 세션에서 복구를 수행하려면 이 속성을 활성화해야 합니다. 기본값이 비활성화됩니다.

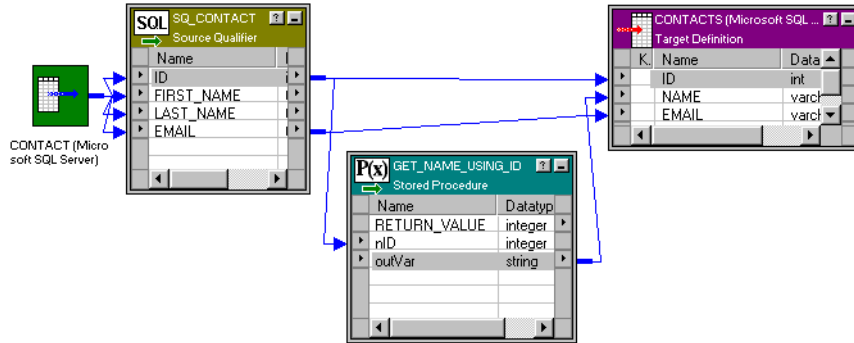
경고: 변환을 반복 가능 및 확정으로 구성하는 경우 데이터가 반복 가능 및 확정인지 확인하는 것은 사용자의 책임입니다. 세션과 복구 간에 동일한 데이터를 생성하지 않는 변환으로 세션을 복구하려는 경우 복구 프로세스로 인해 손상된 데이터가 발생할 수 있습니다.

저장 프로시저 변경

저장 프로시저의 매개 변수 개수 또는 반환 값이 변경되면 해당 값을 다시 가져오거나 수동으로 저장 프로시저 변환을 편집할 수 있습니다. 매핑을 열 때마다 디자이너는 저장 프로시저 변환을 확인하지 않습니다. 변환을 가져오거나 작성한 후 디자이너는 저장 프로시저의 유효성을 검사하지 않습니다. 저장 프로시저가 변환과 일치하지 않으면 세션이 실패합니다.

연결된 변환 구성

다음 그림에는 소스 한정자에서 저장 프로시저 변환의 입력 매개 변수로 ID를 보내는 매핑이 나와 있습니다.



저장 프로시저 변환이 출력 매개 변수를 대상에 전달합니다. 소스 한정자 변환의 각 데이터 행은 저장 프로시저 변환을 통해 데이터를 전달합니다.

필요하지는 않지만, 연결된 거의 모든 저장 프로시저 변환에는 입력 및 출력 매개 변수가 포함되어 있습니다. 필요한 입력 매개 변수는 저장 프로시저 변환의 입력 포트에 지정됩니다. 출력 매개 변수는 변환에서 출력 포트에 나타납니다. 또한 반환 값은 출력 포트이며, 변환 포트 구성에서 선택된 R 값을 가지고 있습니다. 함수 목록에 나타나는 연결된 일반 저장 프로시저의 경우 하나 이상의 입력/출력 포트가 필요합니다.

저장 프로시저의 출력 매개 변수 및 반환 값은 변환에서 기타 출력 포트에 사용됩니다. 이러한 포트를 다른 변환 또는 대상에 연결할 수 있습니다.

연결된 저장 프로시저 변환을 구성하려면:

1. 매핑에서 저장 프로시저 변환을 작성합니다.
2. 저장 프로시저의 출력 포트를 다른 변환이나 대상으로 끕니다.
3. 저장 프로시저 변환을 열고 속성 탭을 선택합니다.
4. 변환을 작성할 때 선택하지 않은 경우 연결 정보에서 해당 데이터베이스를 선택합니다.
5. 변환에 대한 추적 수준을 선택합니다.

매핑을 테스트할 경우, 변환 실패 시 자세한 정보를 제공하도록 자세한 정보 표시 초기화 옵션을 선택합니다.

6. 확인을 클릭합니다.

연결되지 않은 변환 구성

연결되지 않은 저장 프로시저 변환은 매핑을 통해 데이터 흐름에 직접 연결되지 않습니다. 대신 저장 프로시저가 다음 중 하나를 실행합니다.

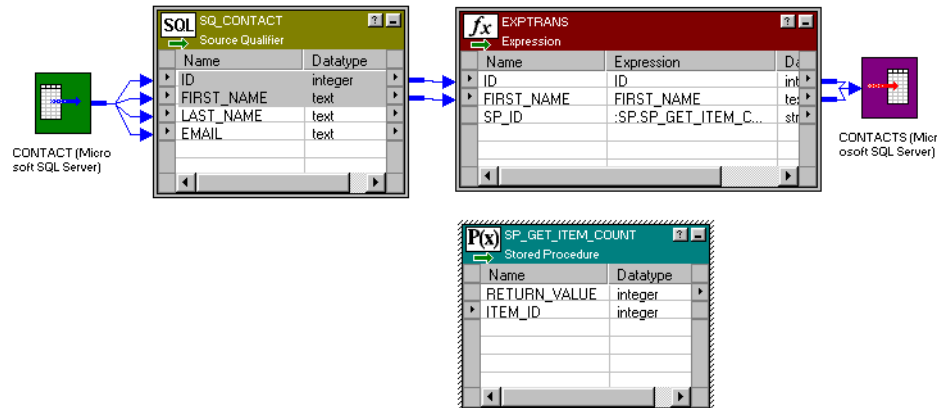
- **식에서.** 매핑의 다른 변환 내 식 편집기에 기록된 식에서 호출됩니다.
- **사전 세션 또는 사후 세션.** 세션 전 또는 후에 실행됩니다.

다음 섹션에는 연결되지 않은 저장 프로시저 변환을 실행할 수 있는 방법이 설명되어 있습니다.

식에서 저장 프로시저 호출

연결되지 않은 매핑에서는 저장 프로시저 변환이 파이프라인에 연결하지 않습니다.

다음 그림은 저장 프로시저 변환을 참조하는 식 변환과 매핑을 보여 줍니다.



하지만 연결된 매핑과 같이, 매핑을 통해 데이터 흐름에 저장 프로시저를 적용할 수 있습니다. 사실, 저장 프로시저를 호출하는 데 식을 사용하므로 유연성이 더 많이 있습니다. 즉, 입력 매개 변수로서 저장 프로시저에 전달하는 데이터를 선택할 수 있습니다.

연결되지 않은 저장 프로시저 변환을 식에 사용하는 경우 출력 매개 변수 값을 포트에 반환하는 방법이 필요합니다. 다음 방법 중 하나를 사용하여 출력 값을 캡처합니다.

- 로컬 변수에 출력 값을 할당합니다.
- 시스템 변수 **PROC_RESULT**에 출력 값을 할당합니다.

PROC_RESULT를 사용하여 출력 포트에 직접 반환 매개 변수 값을 할당합니다. 그러면 대상에 직접 적용할 수 있습니다. 또한 하나의 출력 매개 변수를 **PROC_RESULT**로 할당하고 다른 매개 변수를 변수로 할당하여 두 옵션을 결합할 수 있습니다.

식 내에서만 **PROC_RESULT**를 사용합니다. **PROC_RESULT** 또는 변수를 사용하지 않는 경우 식이 포함된 포트가 **NULL**을 캡처합니다. 연결된 조회 변환에서 또는 저장 프로시저 변환에 대한 호출 텍스트 내에서 **PROC_RESULT**를 사용할 수 없습니다.

하나의 저장 프로시저의 출력 매개 변수가 다른 저장 프로시저에 전달되는 중첩 저장 프로시저가 필요한 경우, **PROC_RESULT**를 사용하여 값을 전달합니다.

통합 서비스는 연결되지 않은 저장 프로시저 변환을 식 변환에서 호출합니다. 저장 프로시저 변환에는 두 개의 입력 포트와 한 개의 출력 포트가 있습니다. 세 개의 포트 모두가 문자열 데이터 유형입니다.

식 내에서 저장 프로시저를 호출하려면:

1. 매핑에서 저장 프로시저 변환을 작성합니다.
2. 출력 및 변수 포트를 지원하는 모든 변환에서, 저장 프로시저를 호출하는 변환의 새로운 출력 포트를 작성합니다. 출력 포트 이름을 지정합니다.

저장 프로시저를 호출하는 출력 포트는 식을 지원해야 합니다. 식이 구성된 방법에 따라 출력 포트에는 반환 값 또는 출력 매개 변수 값이 포함됩니다.

3. 포트에 대한 식 편집기를 엽니다.

새 포트의 값은 변환 언어의 **:SP** 키워드를 사용하는 저장 프로시저에 대한 호출로 식 편집기에서 설정됩니다. 이 값을 제대로 설정하는 가장 쉬운 방법은 식 편집기에서 저장 프로시저 노드를 선택하고 나열된 저장 프로시저 이름을 클릭하는 것입니다. 함수 목록에 나타나는 연결된 일반 저장 프로시저의 경우 하나 이상의 입력/출력 포트가 필요합니다.

저장 프로시저는 빈 괄호 쌍과 함께 식 편집기에 나타납니다. 필요한 입력 및/또는 출력 매개 변수는 식 편집기의 왼쪽 하단 모서리에 표시됩니다.

4. 입력 매개 변수를 보내고 출력 매개 변수나 반환 값을 캡처하도록 식을 구성합니다.

식 편집기에 표시된 매개 변수가 입력 또는 출력 매개 변수인지 알고 있어야 합니다. 저장 프로시저에 나타나는 순서대로 괄호 사이에 변수 또는 포트 이름을 삽입합니다. 포트 및 변수의 데이터 유형은 저장 프로시저에 전달되는 매개 변수의 데이터 유형과 일치해야 합니다.

예를 들어 저장 프로시저를 클릭하면 다음과 비슷한 내용이 나타납니다.

```
:SP.GET_NAME_FROM_ID()
```

이 특정 저장 프로시저는 입력 매개 변수로 정수 값을 필요로 하며 문자열 값을 출력 매개 변수로 반환합니다. 출력 매개 변수 또는 반환 값이 캡처되는 방식은 출력 매개 변수 개수 및 반환 값을 캡처해야 하는지 여부에 따라 다릅니다.

저장 프로시저가 단일 출력 매개 변수 또는 반환 값(하지만 둘 다는 아님)을 반환하는 경우 예약된 변수 **PROC_RESULT**를 출력 변수로 사용해야 합니다. 이전 예에서 해당 식은 다음과 같이 나타납니다.

```
:SP.GET_NAME_FROM_ID(inID, PROC_RESULT)
```

inID는 변환에 대한 입력 포트 또는 변환의 변수일 수 있습니다. **PROC_RESULT** 값은 식에 대한 출력 포트에 적용됩니다.

저장 프로시저가 여러 출력 매개 변수를 반환하는 경우 각 출력 매개 변수에 대한 변수를 작성해야 합니다. 예를 들어 저장 프로시저 식에 대해 **varOUTPUT2**라는 포트를 작성하고 **varOUTPUT1**이라는 변수를 작성하는 경우 해당 식은 다음과 같이 나타납니다.

```
:SP.GET_NAME_FROM_ID(inID, varOUTPUT1, PROC_RESULT)
```

두 번째 출력 포트 값은 식에 대한 출력 포트에 적용되며 첫 번째 출력 포트 값은 **varOUTPUT1**에 적용됩니다. 출력 매개 변수는 저장 프로시저에서 선언된 순서대로 반환됩니다.

이러한 모든 식과 함께, 포트 및 변수에 대한 데이터 유형은 입력/출력 변수 및 반환 값에 대한 데이터 유형과 일치해야 합니다.

5. 유효성 검사를 클릭하여 식을 확인한 다음 확인을 클릭하여 식 편집기를 닫습니다.

식 유효성 검사는 저장 프로시저의 매개 변수에 대한 데이터 유형이 식에 입력된 데이터 유형과 일치하는지 확인합니다.

6. 확인을 클릭합니다.

매핑을 저장하면 디자이너가 저장 프로시저 식에 대한 유효성을 검사하지 않습니다. 저장 프로시저 식이 제대로 구성되지 않은 경우 세션이 실패합니다. 저장 프로시저를 사용하여 매핑을 테스트하는 경우 추적 재정의의 세션 옵션을 자세한 정보 표시 모드로 설정하고 저장 프로시저가 실패하면 실행을 중지하도록 저장 프로시저 세션 옵션을 구성합니다. 세션 속성에 있는 구성 개체 탭의 오류 처리 설정에서 이러한 세션 옵션을 구성합니다.

포트에 대해 입력한 식의 저장 프로시저는 포트를 통과하는 일부 값에는 영향을 미치지 않아도 됩니다. 예를 들어 **IIF** 문을 사용하여, **5**로 시작하는 **ID** 번호 등 특정 값만 저장 프로시저에 전달하고 다른 모든 값은 건너뛸 수 있습니다. 또한 저장 프로시저 하나의 반환 값이 두 번째 저장 프로시저에 대한 입력 매개 변수가 되도록 중첩 저장 프로시저를 설정할 수 있습니다.

사전 세션 또는 사후 세션 저장 프로시저 호출

세션별로 한 번 저장 프로시저를 실행하려고 할 수 있습니다. 예를 들어 매핑을 실행하기 전에 대상 데이터베이스에 테이블이 존재하는지 확인해야 하는 경우, 사전 로드 대상 저장 프로시저가 테이블을 확인한 다음 계속 워크플로우를 실행하거나 중지할 수 있습니다. 소스, 대상 또는 다른 연결된 데이터베이스에서 저장 프로시저를 실행할 수 있습니다.

사전 로드 또는 사후 로드 저장 프로시저를 작성하려면:

1. 매핑에서 저장 프로시저 변환을 작성합니다.
2. 저장 프로시저 변환을 두 번 클릭하고 속성 탭을 선택합니다.
3. 저장 프로시저 이름을 입력합니다.

저장 프로시저를 가져온 경우 저장 프로시저 이름이 기본적으로 나타납니다. 수동으로 저장 프로시저를 설정하는 경우 저장 프로시저 이름을 입력합니다.

4. 연결 정보에서 저장 프로시저가 포함된 데이터베이스를 선택합니다.

5. 저장 프로시저의 호출 텍스트를 입력합니다.

호출 텍스트에서는 저장 프로시저 이름 다음에 괄호 안에 모든 해당 입력 매개 변수가 나옵니다. 입력 매개 변수가 있는 경우 빈 괄호 쌍을 포함해야 합니다. 아니면 저장 프로시저에 대한 호출이 실패합니다. SQL 문 EXEC를 포함하거나 :SP 키워드를 사용할 필요가 없습니다. 예를 들어 **check_disk_space**라는 저장 프로시저를 호출하려면 다음 텍스트를 입력합니다.

```
check_disk_space()
```

문자열 입력 매개 변수를 전달하려면 따옴표 없이 이 매개 변수를 입력합니다. 문자열이 안에 공백을 가지고 있는 경우 큰따옴표로 매개 변수를 묶습니다. 예를 들어 저장 프로시저 **check_disk_space**에 입력 매개 변수로 시스템 이름이 필요한 경우 다음 텍스트를 입력합니다.

```
check_disk_space(oracle_db)
```

사전 세션 또는 사후 세션 저장 프로시저를 통해 날짜/시간 값을 전달하는 경우 다음과 같이 이 값이 Informatica 기본 날짜 형식으로 되어 있고 큰따옴표로 묶여 있어야 합니다.

```
SP("12/31/2000 11:45:59")
```

호출 텍스트에서 **PowerCenter** 매개 변수 및 변수를 사용할 수 있습니다. 매개 변수 파일에 정의할 수 있는 모든 매개 변수 또는 변수 유형을 사용할 수 있습니다. 호출 텍스트 내에 매개 변수나 변수를 입력하거나, 매개 변수나 변수를 호출 텍스트로 사용할 수 있습니다. 예를 들어 세션 매개 변수 **\$ParamMyCallText**를 호출 텍스트로 사용하고 **\$ParamMyCallText**를 매개 변수 파일에서 호출 텍스트로 설정할 수 있습니다.

참고: 사전 세션 및 사후 세션 프로시저의 입력 매개 변수에 대한 저장 매개 변수 대신 값을 입력해야 합니다.

6. 저장 프로시저 유형을 선택합니다.

저장 프로시저 유형의 옵션은 다음과 같습니다.

- **소스 사전 로드.** 세션이 소스에서 데이터를 검색하기 전에 저장 프로시저가 실행됩니다. 임시 테이블에서 데이터 조인을 수행하거나 테이블이 있는지 확인하는 데 유용합니다.
- **소스 사후 로드.** 세션이 소스에서 데이터를 검색한 후에 저장 프로시저가 실행됩니다. 임시 테이블을 제거하는 데 유용합니다.
- **대상 사전 로드.** 세션이 대상에 데이터를 보내기 전에 저장 프로시저가 실행됩니다. 대상 시스템에서 디스크 공간이나 대상 테이블을 확인하는 데 유용합니다.
- **대상 사후 로드.** 세션이 대상에 데이터를 보낸 후에 저장 프로시저가 실행됩니다. 데이터베이스에서 인덱스를 다시 작성하는 데 유용합니다.

7. 실행 순서를 선택하고, 필요한 경우 위쪽 또는 아래쪽 화살표를 클릭하여 순서를 변경합니다.

소스 사후 로드에서 둘 다 실행되는 두 프로시저 등 세션에서 동일한 지점에서 실행되는 여러 저장 프로시저를 추가한 경우, 저장 프로시저 실행 계획을 세워 통합 서비스가 이러한 저장 프로시저를 호출하는 순서를 결정할 수 있습니다. 변경하려는 각 저장 프로시저에 대해 이 단계를 반복해야 합니다.

8. 확인을 클릭합니다.

리포지토리가 매핑의 유효성을 검사하고 매핑을 저장하더라도, 디자이너는 저장 프로시저 식이 오류 없이 실행되는지 확인하지 않습니다. 저장 프로시저 식이 제대로 구성되지 않은 경우 세션이 실패합니다. 저장 프로시저를 사용하여 매핑을 테스트하는 경우 추적 재정의 세션 옵션을 자세한 정보 표시 모드로 설정하고 저장 프로시저가 실패하면 실행을 중지하도록 저장 프로시저 세션 옵션을 구성합니다. 세션 속성에 있는 구성 개체 탭의 오류 처리 설정에서 이러한 세션 옵션을 구성합니다.

값을 캡처할 곳이 없기 때문에, 사전 세션 또는 사후 세션 저장 프로시저 중에 호출되는 출력 매개 변수나 반환 값이 손실됩니다. 값을 캡처해야 하는 경우 데이터베이스의 테이블에 값을 저장하도록 저장 프로시저를 구성하려고 할 수 있습니다.

오류 처리

때때로 저장 프로시저는 "0으로 나누기" 또는 "추가 행 없음"과 같은 데이터베이스 오류를 반환합니다. 저장 프로시저에서 나타나는 데이터베이스 오류의 최종 결과는 저장 프로시저가 실행되는 때와 세션이 구성된 방식에 따라 다릅니다.

사전 세션 또는 사후 세션 저장 프로시저 오류가 발생할 때 세션 실행을 중지하거나 계속하도록 세션을 구성할 수 있습니다. 기본적으로 통합 서비스는 사전 세션 또는 사후 세션 저장 프로시저 데이터베이스 오류가 발생할 경우 세션을 중지합니다.

사전 세션 오류

사전 읽기 및 사전 로드 저장 프로시저는 사전 세션 저장 프로시저로 간주됩니다. 둘 모두 통합 서비스가 소스 데이터 읽기를 시작하기 전에 실행됩니다. 사전 세션 저장 프로시저 중에 데이터베이스 오류가 발생하는 경우 통합 서비스는 세션 구성에 따라 다른 작업을 수행합니다.

- 저장 프로시저 오류가 발생하면 세션을 중지하도록 구성한 경우 통합 서비스에서 세션이 실패합니다.
- 저장 프로시저 오류가 발생해도 세션이 계속되도록 구성한 경우 통합 서비스에서 세션이 계속됩니다.

사후 세션 오류

사후 읽기 및 사후 로드 저장 프로시저는 사후 세션 저장 프로시저로 간주됩니다. 둘 모두 통합 서비스가 데이터베이스에 모든 데이터를 커밋한 후 실행됩니다. 사후 세션 저장 프로시저 중에 데이터베이스 오류가 발생하는 경우 통합 서비스는 세션 구성에 따라 다른 작업을 수행합니다.

- 저장 프로시저 오류가 발생하면 세션을 중지하도록 구성한 경우 통합 서비스에서 세션이 실패합니다.
하지만 통합 서비스는 이미 모든 데이터를 세션 대상에 커밋했습니다.
- 저장 프로시저 오류가 발생해도 세션이 계속되도록 구성한 경우 통합 서비스에서 세션이 계속됩니다.

세션 오류

세션 중에 발생하는 연결되거나 연결되지 않은 저장 프로시저 오류는 세션 오류 처리 옵션의 영향을 받지 않습니다. 데이터베이스의 특정 행에서 오류가 반환되는 경우 통합 서비스는 해당 행을 건너뛰고 다음 행을 계속 처리합니다. 다른 행 변환 오류와 마찬가지로 건너뀀 행은 세션 로그에 나타납니다.

지원되는 데이터베이스

Oracle과 기타 데이터베이스(Informix, Microsoft SQL Server, Sybase 등)에 대해 지원되는 옵션이 아래에 설명되어 있습니다.

관련 항목:

- [“저장 프로시저 작성” 페이지 420](#)

SQL 선언

데이터베이스에서 저장 프로시저를 작성하는 문은 다음 Oracle 저장 프로시저와 비슷하게 표시됩니다.

```
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;

```

이 경우 Oracle 문은 CREATE OR REPLACE PROCEDURE로 시작합니다. Oracle은 저장 프로시저와 저장 함수를 모두 지원하므로 Oracle에서만 선택적인 CREATE FUNCTION 문을 사용합니다.

매개 변수 유형

저장 프로시저에서 세 가지 매개 변수 유형을 사용할 수 있습니다.

- **IN.** 매개 변수를 저장 프로시저로 전달되어야 하는 대상으로 정의합니다.
- **OUT.** 매개 변수를 저장 프로시저에서 반환된 값으로 정의합니다.
- **INOUT.** 매개 변수를 입력 및 출력 모두로 정의합니다. Oracle만 이 매개 변수 유형을 지원합니다.

매핑의 입력/출력 포트

Oracle은 INOUT 매개 변수 유형을 지원하므로 저장 프로시저 변환의 포트가 동일한 저장 프로시저 매개 변수에 대해 입력 및 출력 포트 둘 다로 작동할 수 있습니다. 다른 데이터베이스의 경우에는 포트에 대해 입력 및 출력 확인란을 둘 다 선택할 수 없습니다.

지원되는 반환 값 유형

데이터베이스마다 서로 다른 유형의 반환 값 데이터 유형을 지원하며 Informix만 사용자 정의 반환 값을 지원하지 않습니다.

식 규칙

연결되지 않은 저장 프로시저 변환을 다른 변환의 식에서 호출할 수 있습니다. 식 구성 시 다음 규칙과 지침을 따르십시오.

- 단일 출력 매개 변수는 변수 PROC_RESULT를 사용하여 반환됩니다.
- 식에 저장 프로시저를 사용하는 경우 :SP 참조 한정자를 사용합니다. 입력 오류를 방지하기 위해 식 편집기에서 저장 프로시저 노드를 선택하고 저장 프로시저 이름을 두 번 클릭합니다.
- 하지만 저장 프로시저 변환의 동일한 인스턴스가 매핑의 연결된 모드 및 연결되지 않은 모드 중 하나에서 실행되지 않을 수 있습니다. 변환에 대한 다른 인스턴스를 작성해야 합니다.
- 식의 입력/출력 매개 변수는 저장 프로시저 변환의 입력/출력 포트와 일치해야 합니다. 저장 프로시저에 입력 매개 변수가 있는 경우 저장 프로시저 변환에도 입력 포트가 있어야 합니다.
- 저장 프로시저가 포함된 식을 작성하는 경우 저장 프로시저 및 저장 프로시저 변환에 나타나는 동일한 순서로 매개 변수를 나열합니다.
- 식의 매개 변수에는 저장 프로시저 변환의 모든 매개 변수가 포함되어야 합니다. 입력 매개 변수는 제외할 수 없습니다. 필요한 경우 저장 프로시저에 더미 변수를 전달합니다.
- 식의 인수는 저장 프로시저 변환의 인수와 데이터 유형 및 전체 자릿수가 동일해야 합니다.

- **PROC_RESULT**를 사용하여 저장 프로시저 식의 출력 매개 변수를 대상에 직접 적용합니다. 출력 매개 변수에 대한 변수를 사용하여 대상에 결과를 직접 전달할 수 없습니다. 로컬 변수를 사용하여 동일한 변환 내 출력 포트에 결과를 전달합니다.
- 중첩 저장 프로시저는 다른 저장 프로시저의 입력 매개 변수로 저장 프로시저 하나의 반환 값 전달을 허용합니다. 예를 들어 다음 두 개의 저장 프로시저가 있습니다.
 - `get_employee_id (employee_name)`
 - `get_employee_salary (employee_id)`
 그리고 `get_employee_id`의 반환 값은 직원 ID 번호이고, 중첩 저장 프로시저의 구문은 다음과 같습니다.


```
:sp.get_employee_salary (:sp.get_employee_id (employee_name))
```

 여러 수준의 중첩 저장 프로시저가 있을 수 있습니다.
- 문자열 매개 변수 주변에 작은따옴표를 사용하지 마십시오. 입력 매개 변수에 공백이 포함되지 않은 경우 따옴표를 사용하지 마십시오. 입력 매개 변수에 공백이 포함된 경우 큰따옴표를 사용합니다.

저장 프로시저 변환에 대한 팁

불필요한 저장 프로시저 인스턴스를 실행하지 마십시오.

매핑 중에 저장 프로시저가 실행될 때마다 데이터베이스에서 저장 프로시저가 완료될 때까지 세션이 대기해야 합니다. 이 대기를 방지하려면 두 가지 옵션을 사용할 수 있습니다.

- **행 수를 줄입니다.** 저장 프로시저 변환 전에 활성 변환을 사용하여 저장 프로시저로 전달되어야 하는 행 수를 줄입니다. 또는 저장 프로시저로 전달하기 전에 값을 테스트하는 식을 작성하여 실제로 전달해야 하는 값인지 확인합니다.
- **식을 작성합니다.** 저장 프로시저에서 사용되는 대부분의 논리는 디자이너에서 식을 사용하여 쉽게 재현할 수 있습니다.

저장 프로시저 변환 문제 해결

세션 로그 파일에 "저장 프로시저를 찾을 수 없음" 오류가 나타납니다.

저장 프로시저가 올바른 데이터베이스에서 실행되는지 확인합니다. 기본적으로 저장 프로시저 변환에서는 대상 데이터베이스를 사용하여 저장 프로시저를 실행합니다. 매핑에서 변환을 두 번 클릭하고 속성 탭을 선택한 다음 연결 정보에 선택되어 있는 데이터베이스를 확인합니다.

Microsoft SQL Server 저장 프로시저 사용 시 출력 매개 변수가 반환되지 않습니다.

반환 값을 유지하는 매개 변수가 저장 프로시저에서 **OUTPUT**으로 선언되어 있는지 확인합니다. **Microsoft SQL Server**에서 **OUTPUT**은 입력/출력을 의미합니다. 매핑에서 포트에 대해 **I** 확인란과 **O** 확인란을 모두 선택했을 수 있습니다. 입력 포트를 선택 취소하십시오.

이전에는 세션에 오류가 없었는데 지금은 저장 프로시저에서 세션이 실패합니다.

저장 프로시저 변환과 관련된 문제의 가장 일반적인 원인은 데이터베이스에서 저장 프로시저에 수행한 변경입니다. 저장 프로시저에서 입력/출력 매개 변수 또는 반환 값이 변경되면 저장 프로시저 변환이 유효하지 않게 됩니

다. 저장 프로시저를 다시 가져오거나 저장 프로시저를 수동으로 구성하여 적절한 포트를 추가, 제거 또는 수정해야 합니다.

매핑을 편집한 후 세션이 유효하지 않게 되었습니다. 그 이유가 무엇입니까?

저장 프로시저 변환에 수행한 모든 변경 사항이 세션을 유효하지 않게 할 수 있습니다. 가장 일반적인 원인은 일반 유형을 사후 로드 소스 유형으로 변경하는 것처럼 저장 프로시저의 유형을 변경하는 것입니다.

제 29 장

트랜잭션 제어 변환

이 장에 포함된 항목:

- [트랜잭션 제어 변환 개요, 434](#)
- [트랜잭션 제어 변환 속성, 434](#)
- [매핑에서 트랜잭션 제어 변환 사용, 436](#)
- [매핑 지칭 및 유효성 검사, 438](#)
- [트랜잭션 제어 변환 작성, 439](#)

트랜잭션 제어 변환 개요

PowerCenter에서는 트랜잭션 제어 변환을 통과하는 행 집합을 기준으로 트랜잭션 커밋과 롤백을 제어할 수 있습니다. 트랜잭션 제어 변환은 활성 변환입니다. 트랜잭션은 커밋 또는 롤백 행으로 바인딩된 행 집합입니다. 변경되는 입력 행의 수를 기반으로 트랜잭션을 정의할 수 있습니다. 직원 ID, 주문 입력 날짜 같은 공통 키로 순서가 지정된 행 그룹을 기반으로 트랜잭션을 정의하고 싶을 수 있습니다.

PowerCenter에서는 다음 수준에서 트랜잭션 제어를 정의합니다.

- **매핑 내.** 매핑 내에서 트랜잭션 제어 변환을 사용하여 트랜잭션을 정의합니다. 트랜잭션 제어 변환에서 식을 사용하여 트랜잭션을 정의한 다음, 식의 반환 값을 기준으로 트랜잭션을 커밋하거나 롤백할지, 트랜잭션 변경 없이 계속할지를 선택할 수 있습니다.
- **세션 내.** 세션을 구성할 때 사용자 정의 커밋에 사용하도록 세션을 구성합니다. 통합 서비스가 행을 변환하지 못하거나 대상에 기록하지 못한 경우 트랜잭션을 커밋하거나 롤백하도록 선택할 수 있습니다.

세션을 실행하면 통합 서비스가 변환을 입력하는 각 행에 대해 식을 평가합니다. 커밋 행을 평가하는 경우에는 트랜잭션의 모든 행을 하나 이상의 대상에 커밋합니다. 통합 서비스가 롤백 행을 평가하는 경우에는 하나 이상의 대상에서 트랜잭션의 모든 행을 롤백합니다.

매핑에 플랫폼 파일 대상이 있는 경우 통합 서비스가 새 트랜잭션을 시작할 때마다 출력 파일을 생성할 수 있습니다. 각 대상 플랫폼 파일의 이름을 동적으로 지정할 수 있습니다.

참고: 다른 변환 속성의 변환 범위를 사용하여 트랜잭션을 정의할 수도 있습니다.

트랜잭션 제어 변환 속성

트랜잭션 제어 변환을 사용하여 트랜잭션 대상에서 트랜잭션을 커밋 및 롤백하는 조건을 정의합니다. 트랜잭션 대상에는 관계형, XML 및 동적 MQSeries 대상이 포함됩니다. 속성 탭의 트랜잭션 제어 식에서 이러한 매개 변수

를 정의합니다. 트랜잭션은 커밋 또는 롤백 행으로 바인딩된 하나 이상의 행 집합입니다. 행 수는 트랜잭션마다 다를 수 있습니다.

트랜잭션 제어 변환을 구성할 때 다음 구성 요소를 정의합니다.

- **변환 탭.** 변환 탭에서 변환의 이름을 바꾸고 설명을 추가할 수 있습니다.
- **포트 탭.** 트랜잭션 제어 변환에 입력/출력 포트를 추가할 수 있습니다.
- **속성 탭.** 트랜잭션에 커밋, 롤백 또는 작업 없음 플래그를 지정하는 트랜잭션 제어 식을 정의할 수 있습니다.
- **메타데이터 확장 탭.** 트랜잭션 제어 변환과 정보를 연관시켜 리포지토리에 저장되어 있는 메타데이터를 확장할 수 있습니다.

속성 탭

속성 탭에서 다음 속성을 구성할 수 있습니다.

- 트랜잭션 제어 식
- 추적 수준

트랜잭션 제어 조건 필드에 트랜잭션 제어 식을 입력합니다. 트랜잭션 제어 식은 IIF 함수를 사용하여 각 행을 조건에 대해 테스트합니다. 식에 다음 구문을 사용합니다.

IIF (condition, value1, value2)

식은 조건의 반환 값에 따라 통합 서비스가 수행하는 작업을 나타내는 값을 포함합니다. 통합 서비스는 행별 기준으로 조건을 평가합니다. 반환 값은 통합 서비스가 행을 커밋 또는 롤백하거나 해당 행의 트랜잭션을 변경하지 않을지를 결정합니다. 통합 서비스는 식의 반환 값을 기준으로 커밋 또는 롤백을 실행할 때 새 트랜잭션을 시작합니다. 트랜잭션 제어 식을 작성할 때 식 편집기에서 다음 기본 제공 변수를 사용할 수 있습니다.

- **TC_CONTINUE_TRANSACTION.** 통합 서비스가 해당 행에 대한 트랜잭션 변경을 수행하지 않습니다. 이 값이 식의 기본값입니다.
- **TC_COMMIT_BEFORE.** 통합 서비스가 트랜잭션을 커밋한 다음 새 트랜잭션을 시작하고 현재 행을 대상에 기록합니다. 현재 행은 새 트랜잭션에 있습니다.
- **TC_COMMIT_AFTER.** 통합 서비스가 현재 행을 대상에 기록한 다음 트랜잭션을 커밋하고 새 트랜잭션을 시작합니다. 현재 행은 커밋된 트랜잭션에 있습니다.
- **TC_ROLLBACK_BEFORE.** 통합 서비스가 현재 트랜잭션을 롤백한 다음 새 트랜잭션을 시작하고 현재 행을 대상에 기록합니다. 현재 행은 새 트랜잭션에 있습니다.
- **TC_ROLLBACK_AFTER.** 통합 서비스가 현재 행을 대상에 기록한 다음 트랜잭션을 롤백하고 새 트랜잭션을 시작합니다. 현재 행은 롤백된 트랜잭션에 있습니다.

트랜잭션 제어 식이 커밋, 롤백 또는 계속 이외의 다른 값으로 평가될 경우 통합 서비스가 세션에 실패합니다.

예제

트랜잭션 제어를 통해 주문 입력 날짜에 따라 주문 정보를 기록할 수 있습니다. 특정 날짜에 입력된 모든 주문이 동일한 트랜잭션에서 대상으로 커밋되었는지 확인할 수 있습니다. 이렇게 하려면 다음 변환을 사용하여 매핑을 작성합니다.

- **분류기 변환.** 주문 입력 날짜별로 소스 데이터를 정렬합니다.
- **식 변환.** 로컬 변수를 사용하여 입력된 날짜가 새 날짜인지 확인합니다.

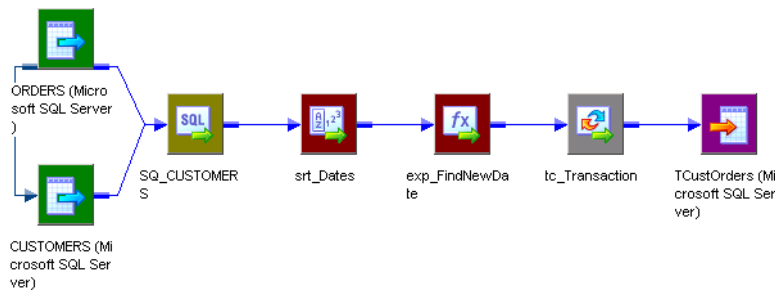
다음 테이블에는 식 변환의 포트가 설명되어 있습니다.

포트 이름	식	설명
DATE_ENTERED	DATE_ENTERED	입력/출력 포트. 입력된 날짜를 받고 전달합니다.
NEW_DATE	IIF(DATE_ENTERED=PREVDATE, 0,1)	변수 포트. DATE_ENTERED의 현재 값을 변수 포트 PREV_DATE에 있는 DATE_ENTERED의 저장된 값에 대해 테스트합니다.
PREV_DATE	DATE_ENTERED	변수 포트. 통합 서비스가 NEW_DATE 포트를 평가한 후 DATE_ENTERED의 값을 받습니다.
DATE_OUT	NEW_DATE	출력 포트. NEW_DATE에서 트랜잭션 제어 변환으로 플래그를 전달합니다.
참고: 통합 서비스는 종속성을 기준으로 포트를 평가합니다. 변환에서 포트가 표시되는 순서는 다음과 같은 평가의 순서와 일치해야 합니다. 즉 입력 포트, 변수 포트, 출력 포트 순서입니다.		

- **트랜잭션 제어 변환.** 통합 서비스가 새 주문 입력 날짜를 발견하면 데이터를 커밋하는 다음과 같은 트랜잭션 제어 식을 작성합니다.

IIF(NEW_DATE = 1, TC_COMMIT_BEFORE, TC_CONTINUE_TRANSACTION)

다음 그림은 트랜잭션 제어 변환을 사용하는 샘플 매핑을 보여 줍니다.



매핑에서 트랜잭션 제어 변환 사용

트랜잭션 제어 변환은 트랜잭션 생성기입니다. 매핑에서 트랜잭션 경계를 정의하고 재정의합니다. 업스트림 활성 소스 또는 트랜잭션 생성기에서 모든 수신 트랜잭션 경계를 삭제하고 새 트랜잭션 경계 다운스트림을 생성합니다.

트랜잭션을 생성하도록 구성된 사용자 지정 변환을 사용하여 트랜잭션 경계를 정의할 수도 있습니다.

트랜잭션 제어 변환은 매핑의 다운스트림 변환 및 대상에 대해 유효하거나 유효하지 않을 수 있습니다. 다운스트림 변환 또는 대상 다음에 수신 트랜잭션 경계를 삭제하는 변환을 배치하면 트랜잭션 제어 변환이 해당 변환 또는 대상에 대해 유효하지 않게 됩니다. 여기에는 다음 활성 소스 또는 변환이 포함됩니다.

- 입력 수준 변환 범위가 모두인 집계 변환

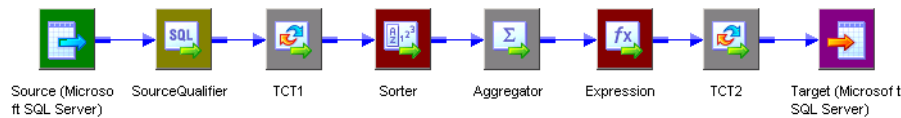
- 입력 수준 변환 범위가 모두인 조이너 변환
- 입력 수준 변환 범위가 모두인 순위 변환
- 입력 수준 변환 범위가 모두인 분류기 변환
- 입력 수준 변환 범위가 모두인 사용자 지정 변환
- 트랜잭션을 생성하도록 구성된 사용자 지정 변환
- 트랜잭션 제어 변환
- 사용자 지정 변환과 같이 여러 업스트림 트랜잭션 제어점에 연결된 여러 입력 그룹 변환

대상에 대해 유효하지 않은 트랜잭션 제어 변환이 포함된 매핑은 올바르게나 올바르게 않을 수 있습니다. 매핑을 저장하거나 매핑의 유효성을 검사할 경우 디자이너에 대상에 대해 유효하지 않은 트랜잭션 제어 변환을 알려 주는 메시지가 표시됩니다.

트랜잭션 제어 변환이 대상에 대해서는 유효하지 않지만 다운스트림 변환에 대해서는 유효할 수 있습니다. 트랜잭션 수준 변환 범위를 갖는 다운스트림 변환은 업스트림 트랜잭션 제어 변환으로 정의된 트랜잭션 경계를 사용할 수 있습니다.

다음 그림에서는 분류기 변환에 대해 유효하지만 대상에 대해 유효하지 않은 트랜잭션 제어 변환이 있는 올바른 매핑을 보여 줍니다.

이 예에서 TCT1 변환은 대상에 대해 유효하지 않지만 분류기 변환에 대해 유효합니다. 분류기 변환의 변환 범위 속성은 트랜잭션입니다. 이 변환은 TCT1로 정의된 트랜잭션 경계를 사용합니다. 집계 변환의 범위 속성은 모든 입력입니다. 이 변환은 TCT1로 정의된 트랜잭션 경계를 삭제합니다. TCT2 변환은 대상에 대한 효과적 트랜잭션 제어 변환입니다.

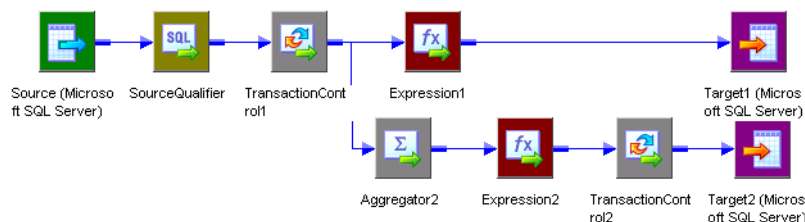


여러 대상이 있는 샘플 트랜잭션 제어 매핑

트랜잭션 제어 변환은 한 대상에 대해 효과적이고 다른 대상에 대해 비효과적일 수 있습니다.

각 대상이 효과적 트랜잭션 제어 변환에 연결되는 경우 매핑이 유효합니다. 매핑의 한 대상이 효과적 트랜잭션 제어 변환에 연결되어 있지 않은 경우 매핑이 올바르게 않습니다.

다음 그림에서는 유효하지 않은 트랜잭션 제어 변환과 효과적 트랜잭션 제어 변환이 모두 있는 올바른 매핑을 보여 줍니다.



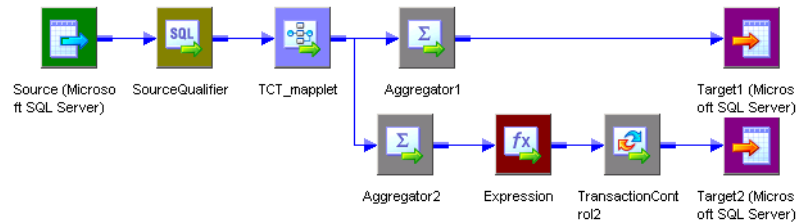
통합 서비스는 TransactionControl1을 처리하고 트랜잭션 제어 식을 평가한 다음 트랜잭션 경계를 작성합니다. 매핑에 TransactionControl1과 Target1 사이의 트랜잭션 경계를 삭제하는 변환이 포함되어 있지 않기 때문에 TransactionControl1이 Target1에 대해 유효하게 됩니다. 통합 서비스는 TransactionControl1로 정의된 트랜잭션 경계를 Target1에 대해 사용합니다.

하지만 매핑에 TransactionControl1과 Target2 사이의 트랜잭션 경계를 삭제하는 변환이 포함되어 있으므로 TransactionControl1이 Target2에 대해 유효하지 않게 됩니다. 통합 서비스는 변환 범위가 모든 입력으로 설정

되어 있는 **Aggregator2**를 처리할 때 **TransactionControl1**로 정의된 트랜잭션 경계를 삭제하고 모든 행을 개방형 트랜잭션으로 출력합니다. 그런 다음, 통합 서비스는 **TransactionControl2**를 평가하고 트랜잭션 경계를 작성한 다음 **Target2**에 대해 해당 경계를 사용합니다.

TransactionControl1에서 롤백이 발생하는 경우 통합 서비스는 **Target1**의 행만 롤백합니다. **Target2**의 어떠한 행도 롤백하지 않습니다.

다음 그림에서는 유효하지 않은 트랜잭션 제어 변환과 효과적 트랜잭션 제어 변환이 모두 있는 올바른지 않은 매핑을 보여 줍니다.



TCT_mapplet 맵렛에는 트랜잭션 제어 변환이 포함되어 있습니다. 이 변환은 **Target1** 및 **Target2**에 대해 유효하지 않습니다. **Aggregator1** 변환의 변환 범위 속성은 모든 입력입니다. 이 변환은 **Target1**의 활성 소스입니다. **Aggregator2** 변환의 변환 범위 속성은 모든 입력입니다. 이 변환은 **Target2**의 활성 소스입니다. **TransactionControl2** 변환은 **Target2**에 대해 유효합니다.

이 매핑은 **Target1**이 효과적 트랜잭션 제어 변환에 연결되어 있지 않으므로 올바르지 않습니다.

매핑 지침 및 유효성 검사

트랜잭션 제어 변환을 포함하는 매핑을 작성할 때 다음 규칙과 지침을 따르십시오.

- 매핑에 XML 대상이 있고 커밋 시 새 문서를 추가하거나 작성하도록 선택하는 경우 입력 그룹이 동일한 트랜잭션 제어점에서 데이터를 수신해야 합니다.
- 관계형, XML 또는 동적 MQSeries 대상 이외의 대상에 연결된 트랜잭션 제어 변환은 해당 대상에서 비효과적입니다.
- 각각의 대상 인스턴스를 트랜잭션 제어 변환에 연결해야 합니다.
- 여러 대상을 단일 트랜잭션 제어 변환에 연결할 수 있습니다.
- 하나의 효과적 트랜잭션 제어 변환만 대상에 연결할 수 있습니다.
- 시퀀스 생성기 변환으로 시작하는 파이프라인 분기에 트랜잭션 제어 변환을 배치할 수 없습니다.
- 동적 조희 변환과 트랜잭션 제어 변환을 동일한 매핑에서 사용하는 경우 롤백된 트랜잭션으로 인해 동기화되지 않은 대상 데이터가 발생할 수 있습니다.
- 트랜잭션 제어 변환은 한 대상에 대해 효과적이고 다른 대상에 대해 비효과적일 수 있습니다. 각 대상이 효과적 트랜잭션 제어 변환에 연결되는 경우 매핑이 유효합니다.
- 매핑의 모든 대상이 효과적 트랜잭션 제어 변환에 연결되어야 하거나 대상이 전혀 연결되지 않아야 합니다.

트랜잭션 제어 변환 작성

다음 절차에 따라 트랜잭션 제어 변환을 매핑에 추가합니다.

1. 매핑 디자이너에서 변환 > 작성을 클릭합니다. 트랜잭션 제어 변환을 선택합니다.

2. 변환 이름을 입력합니다.

트랜잭션 제어 변환의 이름 지정 규칙은 `TC_TransformationName`입니다.

3. 변환의 설명을 입력합니다.

이 설명은 **Repository Manager**에서 변환 세부 정보를 볼 때 표시되므로 변환의 기능을 쉽게 이해할 수 있습니다.

4. 작성을 클릭합니다.

디자이너가 트랜잭션 제어 변환을 작성합니다.

5. 완료를 클릭합니다.

6. 포트를 변환으로 끌어서 놓습니다.

사용자가 포함하는 각 포트에 대해 디자이너가 입력/출력 포트를 작성합니다.

7. 변환 편집 대화 상자를 열고 포트 탭을 선택합니다.

포트를 추가하고, 포트 이름을 편집하고, 포트 설명을 추가하고, 기본값을 입력할 수 있습니다.

8. 속성 탭을 선택합니다. 커밋 및 롤백 동작을 정의하는 트랜잭션 제어 식을 입력합니다.

9. 메타데이터 확장 탭을 선택합니다. 트랜잭션 제어 변환에 대한 메타데이터 확장을 작성하거나 편집합니다.

10. 확인을 클릭합니다.

제 30 장

합집합 변환

이 장에 포함된 항목:

- [합집합 변환 개요, 440](#)
- [그룹 및 포트 작업, 441](#)
- [합집합 변환 작성, 441](#)
- [매핑에서 합집합 변환 사용, 442](#)

합집합 변환 개요

합집합 변환은 다중 입력 그룹 변환이며, 이 변환을 사용하여 여러 파이프라인 또는 파이프라인 분기의 데이터를 단일 파이프라인 분기로 병합할 수 있습니다. 이 변환은 **UNION ALL SQL** 문과 비슷하게 여러 소스의 데이터를 병합하여 두 개 이상의 **SQL** 문 결과를 조합합니다. **UNION ALL** 문과 유사하게 합집합 변환은 중복 행을 제거하지 않습니다. 합집합 변환은 활성 변환입니다.

데이터 통합 서비스는 모든 입력 그룹을 병렬로 처리합니다. 데이터 통합 서비스는 합집합 변환에 연결된 여러 소스를 동시에 읽고 데이터 블록을 해당 변환의 입력 그룹으로 푸시합니다. 합집합 변환은 통합 서비스로부터 데이터 블록을 받은 순서에 따라 데이터 블록을 처리합니다.

다른 유형의 소스를 합집합 변환에 연결할 수 있습니다. 이 변환은 포트가 일치하는 여러 소스를 병합하고 입력 그룹과 포트가 동일한 단일 출력 그룹에서 데이터를 출력합니다.

합집합 변환은 사용자 지정 변환을 사용하여 개발됩니다.

합집합 변환에 대한 규칙 및 지침

합집합 변환 작업을 수행할 때 다음 규칙 및 지침을 따르십시오.

- 입력 그룹을 여러 개 작성할 수 있지만 출력 그룹은 1개만 작성할 수 있습니다.
- 모든 입력 그룹과 출력 그룹은 포트가 일치해야 합니다. 전체 자릿수, 데이터 유형 및 소수 자릿수는 모든 그룹에서 동일해야 합니다.
- 합집합 변환은 중복 행을 제거하지 않습니다. 중복 행을 제거하려면 라우터 또는 필터 변환 등의 다른 변환을 추가해야 합니다.
- 합집합 변환은 트랜잭션을 생성하지 않습니다.

합집합 변환 구성 요소

합집합 변환을 구성할 경우 다음 구성 요소를 정의합니다.

- **변환 탭.** 변환 이름을 변경하고 설명을 추가할 수 있습니다.
- **속성 탭.** 추적 수준을 지정할 수 있습니다.
- **그룹 탭.** 입력 그룹을 작성하고 삭제할 수 있습니다. 포트 탭에서 작성한 그룹이 디자이너에 표시됩니다.
- **그룹 포트 탭.** 입력 그룹의 포트를 작성하고 삭제할 수 있습니다. 포트 탭에서 작성한 포트가 디자이너에 표시됩니다.

합집합 변환에서 포트 탭을 수정할 수 없습니다.

그룹 및 포트 작업

합집합 변환에는 여러 개의 입력 그룹과 1개의 출력 그룹이 있습니다. 입력 그룹은 그룹 탭에서 작성하고 포트는 그룹 포트 탭에서 작성합니다.

그룹 탭에서 하나 이상의 입력 그룹을 작성할 수 있습니다. 디자이너는 기본적으로 출력 그룹 1개를 작성합니다. 사용자가 출력 그룹을 편집하거나 삭제할 수는 없습니다.

변환에서 포트를 복사하여 포트를 작성하거나 수동으로 포트를 작성할 수 있습니다. 그룹 포트 탭에서 포트를 작성하면 디자이너가 각 입력 그룹에 입력 포트를 작성하고 출력 그룹에 출력 포트를 작성합니다. 디자이너는 그룹 탭에서 지정한 포트 이름을 각 입력 및 출력 포트에 대해 사용하고, 번호를 추가하여 변환에서 각 포트 이름이 고유하도록 합니다. 또한 각 포트에 대해 데이터 유형, 전체 자릿수 및 소수 자릿수 등의 동일한 메타데이터를 사용합니다.

포트 탭에는 작성한 그룹 및 포트가 표시됩니다. 포트 탭에서 그룹 및 포트 정보를 편집할 수는 없습니다. 그룹 및 포트를 편집하려면 그룹 탭과 그룹 포트 탭을 사용하십시오.

합집합 변환 작성

다음 절차에 따라 합집합 변환을 작성합니다.

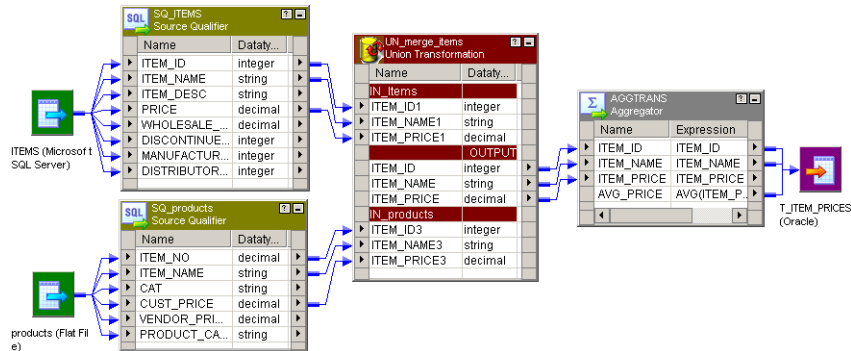
1. 맵핑 디자이너에서 변환 > 작성을 클릭합니다.
2. 합집합 변환을 선택하고 변환 이름을 입력합니다.
합집합 변환의 이름 지정 규칙은 `UN_TransformationName`입니다.
3. 변환의 설명을 입력합니다. 작성을 클릭한 다음 완료를 클릭합니다.
4. 그룹 탭을 클릭합니다.
5. 병합할 각 파이프라인 또는 파이프라인 분기에 대한 입력 그룹을 추가합니다.
디자이너가 각 그룹에 대한 기본 이름을 할당하지만 이름을 바꿀 수 있습니다.
6. 그룹 포트 탭을 클릭합니다.
7. 병합할 각 데이터 행에 대한 새 포트를 추가합니다.
8. 이름 및 데이터 유형 등 포트 속성을 입력합니다.
9. 속성 탭을 클릭하여 추적 수준을 구성합니다.
10. 확인을 클릭합니다.

매핑에서 합집합 변환 사용

합집합 변환은 비차단 다중 입력 그룹 변환입니다. 단일 파이프라인의 여러 분기 또는 여러 소스 파이프라인에 입력 그룹을 연결할 수 있습니다.

합집합 변환을 매핑에 추가할 때는 모든 입력 그룹의 동일한 포트를 연결해야 합니다. 특정 입력 그룹의 모든 포트를 연결하지만 다른 입력 그룹의 포트는 연결하지 않는 경우, 통합 서비스가 NULL을 연결되지 않은 포트로 전달합니다.

다음 그림은 합집합 변환과의 매핑을 보여줍니다.



매핑에 있는 합집합 변환이 단일 트랜잭션 생성기로부터 데이터를 받으면 통합 서비스가 트랜잭션 경계를 전파합니다. 변환이 여러 트랜잭션 생성기로부터 데이터를 받는 경우, 통합 서비스는 수신 모든 트랜잭션 경계를 삭제하고 개방형 트랜잭션에서 행을 출력합니다.

제 31 장

구조화되지 않은 Data Transformation

이 장에 포함된 항목:

- [구조화되지 않은 Data Transformation 개요, 443](#)
- [구조화되지 않은 데이터 옵션 구성, 444](#)
- [Data Transformation 서비스 유형, 445](#)
- [구조화되지 않은 Data Transformation 구성 요소, 445](#)
- [구조화되지 않은 Data Transformation 포트, 448](#)
- [구조화되지 않은 Data Transformation 서비스 이름, 450](#)
- [관계형 계층, 451](#)
- [매핑, 452](#)
- [구조화되지 않은 Data Transformation 작성, 454](#)

구조화되지 않은 Data Transformation 개요

구조화되지 않은 데이터 변환은 메시징 형식, HTML 페이지 및 PDF 문서 등의 구조화되지 않거나 반구조화된 파일 형식을 처리하는 변환입니다. 또한 ACORD, HIPAA, HL7, EDI-X12, EDIFACT 및 SWIFT 등의 구조화된 형식도 변환합니다.

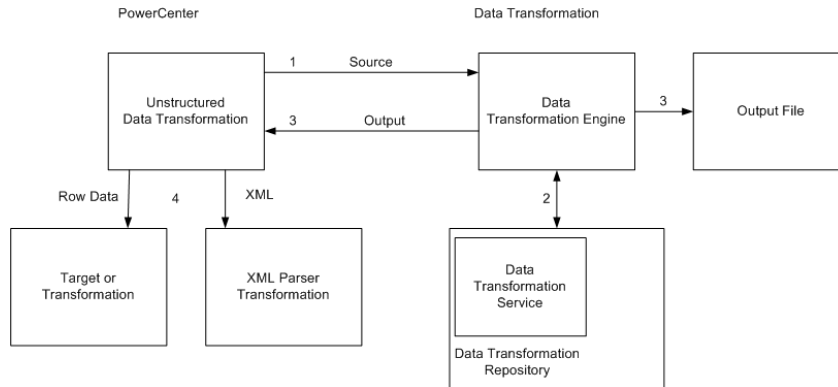
구조화되지 않은 데이터 변환은 PowerCenter 세션에서 Data Transformation 서비스를 호출합니다. Data Transformation은 구조화되지 않거나 반구조화된 파일 형식을 변환하는 응용 프로그램입니다. 구조화되지 않은 데이터 변환에서 Data Transformation 서비스로 데이터를 전달하고 데이터를 변환한 후에 변환된 데이터를 파이프라인으로 반환할 수 있습니다. 구조화되지 않은 데이터 변환은 활성 변환 또는 수동 변환일 수 있습니다.

Data Transformation에는 다음 구성 요소가 있습니다.

- Informatica Developer. 변환 프로젝트를 설계 및 구성할 수 있는 기능을 제공하는 개발 도구입니다.
- Data Transformation 서비스. Data Transformation 리포지토리에 배포되고 바로 실행할 수 있는 Data Transformation 프로젝트입니다.
- Data Transformation 리포지토리. Informatica Developer에서 작성한 실행 가능한 서비스가 저장되는 디렉터리입니다. 테스트용 리포지토리 및 프로덕션 서비스용 리포지토리 등의 여러 리포지토리에 프로젝트를 배포할 수 있습니다.
- Data Transformation Engine. 리포지토리로 배포한 서비스를 실행하는 프로세서입니다.

Data Transformation Engine은 서비스를 실행할 때 출력 데이터를 기록하거나 통합 서비스로 출력 데이터를 반환합니다. **Data Transformation Engine**은 통합 서비스로 출력을 반환할 때 **XML** 데이터를 반환합니다. 구조화되지 않은 데이터 변환이 출력 포트에서 **XML**을 반환하도록 구성하거나 출력 그룹이 행 데이터를 반환하도록 구성할 수 있습니다.

다음 그림은 **PowerCenter** 구조화되지 않은 데이터 변환과 **PowerCenter** 세션의 데이터를 변환하는 **Data Transformation** 서비스 간의 인터페이스를 보여 줍니다.



PowerCenter 세션에서 **Data Transformation** 서비스를 호출하면 다음 이벤트가 발생합니다.

1. 구조화되지 않은 데이터 변환이 소스 데이터를 **Data Transformation Engine**으로 전달합니다.
2. **Data Transformation Engine**이 **Data Transformation** 서비스를 실행하여 데이터를 변환합니다. **Data Transformation** 서비스는 **Data Transformation** 리포지토리 폴더에 있습니다.
3. **Data Transformation Engine**이 변환된 데이터를 출력 파일에 직접 기록하거나 변환된 데이터를 구조화되지 않은 데이터 변환으로 반환합니다.
4. 구조화되지 않은 데이터 변환이 **XML** 데이터 또는 데이터 행을 반환합니다. 구조화되지 않은 데이터 변환이 **XML**을 반환하는 경우, 매핑에서 구조화되지 않은 데이터 변환을 **XML** 파서 변환에 연결하십시오.

구조화되지 않은 데이터 옵션 구성

구조화되지 않은 데이터 변환은 **PowerCenter**와 함께 설치됩니다. **Data Transformation**에는 별도의 설치 프로그램이 있습니다. **PowerCenter**를 설치한 후 **Data Transformation** 서버 및 클라이언트 구성 요소를 설치합니다.

구조화되지 않은 데이터 옵션을 설치하려면 다음 단계를 완료합니다.

1. **PowerCenter**를 설치합니다.
2. **Data Transformation**을 설치합니다.

Data Transformation을 설치할 때 **Java Runtime Environment**를 선택하라는 메시지가 표시됩니다. 프롬프트에서 **PowerCenter**가 사용하는 **JRE**를 찾아봅니다.

기본적으로 설치 프로그램은 프로세스 내에서 실행되도록 **Data Transformation Engine**을 구성합니다. 구조화되지 않은 데이터 변환이 포함된 **Data Transformation**을 사용하려면 프로세서 외부에서 실행되게 **Data Transformation**을 구성하지 마십시오.

3. **Data Transformation** 리포지토리 폴더를 구성합니다.

Data Transformation 리포지토리 디렉터리 구성

Data Transformation 리포지토리에는 실행 가능한 Data Transformation 서비스가 포함되어 있습니다. Data Transformation을 설치하면 설치 프로그램이 다음 폴더에 리포지토리를 작성합니다.

<Data_Transformation_install_dir>\DataTransformation\ServiceDB

다른 리포지토리 폴더 위치를 구성하려면 시작 메뉴에서 Data Transformation 구성을 엽니다. 리포지토리 위치는 Data Transformation 구성의 다음 경로에 있습니다.

CM 구성 > CM 리포지토리 > 파일 시스템 > 기본 경로

Informatica Developer가 원격 파일 시스템에 액세스할 수 있는 경우 Data Transformation 리포지토리를 원격 위치로 변경하고, Informatica Developer에서 통합 서비스를 실행하는 시스템에 직접 서비스를 배포할 수 있습니다.

<Data_Transformation_install_dir>\DataTransformation\autoInclude\user 디렉터리 및

<Data_Transformation_install_dir>\DataTransformation\externLibs\user 디렉터리에서 통합 서비스가 실행되는 노드의 같은 디렉터리로 사용자 지정 파일을 복사합니다.

Data Transformation 서비스 유형

Informatica Developer에서 데이터 프로세서 변환을 작성할 때 개체 또는 구성 요소를 선택하여 변환 서비스 유형을 정의합니다. Data Transformation에는 데이터를 변환하는 다음과 같은 유형의 서비스가 있습니다.

- **파서.** 소스 문서를 XML 또는 JSON으로 변환합니다. 파서의 출력은 항상 XML입니다. 입력은 텍스트, HTML, Word, PDF, HL7 등과 같은 형식일 수 있습니다.
- **직렬 변환기.** XML 또는 JSON 파일을 원하는 형식의 출력 문서로 변환합니다. 직렬 변환기의 출력은 텍스트 문서, HTML 문서, PDF 등과 같은 형식일 수 있습니다.
- **매퍼 및 XMap.** XML 또는 JSON 소스 문서를 다른 XML 또는 JSON 구조로 변환합니다. 매퍼는 직렬 변환기와 유사하게 입력을 처리합니다. 또한 파서와 유사하게 출력을 생성합니다. 입력 및 출력은 완전히 구조화된 XML 또는 JSON입니다.
- **변환기.** 모든 형식의 데이터를 수정합니다. 텍스트를 추가, 제거, 변환 또는 변경합니다. 파서, 매퍼 또는 직렬 변환기와 함께 변환기를 사용하십시오. 변환기를 독립 실행형 구성 요소로 실행할 수도 있습니다.
- **스트리머.** 수 기가바이트 크기의 데이터 스트림과 같은 큰 입력 문서를 세그먼트로 분할합니다. 스트리머는 HIPAA 또는 EDI 파일과 같이 여러 메시지나 레코드를 포함하는 문서를 처리합니다.

구조화되지 않은 Data Transformation 구성 요소

구조화되지 않은 데이터 변환에는 다음 탭이 포함되어 있습니다.

- **변환.** 변환의 이름 및 설명을 입력합니다. 구조화되지 않은 데이터 변환에 대한 명명 규칙은 UD_TransformationName입니다. 구조화되지 않은 데이터 변환을 재사용 가능하도록 할 수도 있습니다.
- **속성.** IsPartitionable 및 출력은 반복 가능합니다. 같은 구조화되지 않은 데이터 변환 일반 속성을 구성합니다.
- **UDT 설정.** 입력 유형, 출력 유형 및 서비스 이름 같은 구조화되지 않은 데이터 변환 설정을 수정합니다.
- **UDT 포트.** 구조화되지 않은 데이터 변환 포트 및 특성을 구성합니다.

- 입력 계층. 입력 그룹 및 포트의 계층을 정의하여 구조화되지 않은 데이터 변환이 데이터 행을 읽을 수 있게 합니다.
- 출력 계층. 출력 그룹 및 포트의 계층을 정의하여 구조화되지 않은 데이터 변환이 관계형 대상에 행을 기록할 수 있게 합니다.

속성 탭

속성 탭에서 구조화되지 않은 데이터 변환 일반 속성을 구성합니다.

다음 테이블에는 속성 탭에서 구성할 수 있는 속성이 설명되어 있습니다.

속성	설명
추적 수준	이 변환을 포함하는 세션을 실행할 때 세션 로그에 포함되는 세부 정보의 양입니다. 기본값은 보통입니다.
IsPartitionable	변환이 둘 이상의 파티션에서 실행될 수 있습니다. 다음 옵션 중 하나를 선택합니다. <ul style="list-style-type: none"> - 아니요. 변환을 분할할 수 없습니다. - 로컬로. 변환을 분할할 수 있지만 통합 서비스가 같은 노드의 파이프라인에서 모든 파티션을 실행해야 합니다. - 그리드에서. 변환을 분할할 수 있고 통합 서비스가 각 파티션을 여러 노드에 분산시킬 수 있습니다. 기본값은 그리드에서입니다.
출력은 반복 가능합니다.	출력 데이터의 순서는 세션 실행 간에 일관됩니다. <ul style="list-style-type: none"> - 사용 안 함 출력 데이터 순서는 세션 실행 간에 일관되지 않습니다. - 입력 순서 기반. 입력 데이터 순서가 세션 실행 간에 일관된 경우 입력 순서는 세션 실행 간에 일관됩니다. - 항상 입력 데이터 순서가 세션 실행 간에 일관되지 않더라도 출력 데이터 순서는 세션 실행 간에 일관됩니다. 활성 변환의 경우 기본값은 사용 안 함입니다. 수동 변환 실행의 경우 기본값은 입력 순서 기반입니다.
확정 출력입니다.	변환이 세션 실행 간에 일치하는 출력 데이터를 생성하는지 여부를 나타냅니다. 이 변환을 사용하는 세션에 대해 복구를 수행하려면 이 속성을 활성화합니다.

경고: 변환을 반복 가능 및 확정으로 구성하는 경우 데이터가 반복 가능 및 확정인지 확인하는 것은 사용자의 책임입니다. 세션과 복구 간에 동일한 데이터를 생성하지 않는 변환으로 세션을 복구하려는 경우 복구 프로세스로 인해 손상된 데이터가 발생할 수 있습니다.

UDT 설정 탭

UDT 설정 탭에서 구조화되지 않은 데이터 변환 특성을 구성합니다.

다음 테이블에는 UDT 설정 탭에 있는 특성이 설명되어 있습니다.

특성	설명
InputType	구조화되지 않은 데이터 변환이 Data Transformation Engine으로 전달하는 입력 데이터의 유형입니다. 다음 입력 유형 중 하나를 선택하십시오. <ul style="list-style-type: none"> - 버퍼. 구조화되지 않은 데이터 변환이 InputBuffer 포트에 소스 데이터를 받고 해당 포트에서 Data Transformation Engine으로 데이터를 전달합니다. - 파일. 구조화되지 않은 데이터 변환이 InputBuffer 포트에 소스 파일 경로를 받고 Data Transformation Engine으로 소스 파일 경로를 전달합니다. Data Transformation Engine은 소스 파일을 엽니다.
OutputType	구조화되지 않은 데이터 변환 또는 Data Transformation Engine이 반환하는 출력 데이터의 유형입니다. 다음 출력 유형 중 하나를 선택합니다. <ul style="list-style-type: none"> - 버퍼. 출력 그룹의 관계형 계층을 구성하지 않은 경우, 구조화되지 않은 데이터 변환이 OutputBuffer 포트를 통해 XML 데이터를 반환합니다. 포트의 관계형 계층을 구성하면 구조화되지 않은 데이터 변환이 OutputBuffer 포트에 기록하지 않습니다. - 파일. Data Transformation Engine이 출력을 파일에 기록합니다. 구조화되지 않은 데이터 변환에서 포트의 관계형 계층을 구성하지 않은 경우에는 구조화되지 않은 데이터 변환으로 데이터를 반환하지 않습니다. - 분할. 구조화되지 않은 데이터 변환이 큰 XML 출력 파일 1개를 OutputBuffer 포트에 맞는 여러 개의 작은 파일로 분할합니다. 분할된 XML 파일을 XML 파서 변환으로 전달해야 합니다.
ServiceName	실행할 Data Transformation 서비스의 이름입니다. 이 서비스는 로컬 Data Transformation 리포지토리에 있어야 합니다.
스트리머 체크 크기	Data Transformation 서비스가 스트리머를 실행할 때 구조화되지 않은 데이터 변환이 Data Transformation Engine으로 전달하는 데이터의 버퍼 크기입니다. 올바른 값은 1 - 1,000,000KB입니다. 기본값은 256KB입니다.
동적 서비스 이름	각 입력 행에 대해 서로 다른 Data Transformation 서비스를 실행합니다. 동적 서비스 이름이 활성화된 경우 구조화되지 않은 데이터 변환은 서비스 이름 입력 포트에서 서비스 이름을 받습니다. 동적 서비스 이름이 비활성화된 경우에는 구조화되지 않은 데이터 변환이 각 입력 행에 대해 동일한 서비스를 실행합니다. UDT 설정에서 서비스 이름 특성은 서비스 이름을 포함해야 합니다. 기본값이 비활성화됩니다.
상태 추적 수준	Data Transformation 서비스로부터 발생하는 상태 메시지의 수준을 설정합니다. <ul style="list-style-type: none"> - 설명만. 상태 코드 및 Data Transformation 서비스가 성공적이었는지 또는 실패했는지 나타내는 짧은 설명을 반환합니다. - 전체 상태. 상태 코드 및 Data Transformation 서비스로부터의 상태 메시지를 XML로 반환합니다. - 없음. Data Transformation 서비스의 상태를 반환하지 않습니다. 기본값은 없음입니다.
입력 플러시 허용	입력 포트 그룹을 구성할 때 입력 플러시 허용을 활성화합니다. 구조화되지 않은 데이터 변환은 그룹에 대한 모든 데이터를 갖고 있으면 XML을 작성합니다. 그룹별로 정렬된 데이터를 변환에서 받도록 매핑을 구성하십시오. 입력 플러시가 활성화되지 않은 경우 구조화되지 않은 데이터 변환은 데이터를 메모리에 저장하며, 모든 그룹에 대한 데이터를 받은 후에 XML을 작성합니다.

상태 추적 메시지 보기

Data Transformation 서비스의 상태 메시지를 볼 수 있습니다. 상태 추적 수준을 설명만 또는 전체 상태로 설정합니다. 디자이너는 구조화되지 않은 데이터 변환에 UDT_Status_Code 포트 및 UDT_Status_Message 출력 포트를 작성합니다.

설명만을 선택하면 Data Transformation Engine이 상태 코드 및 다음 상태 메시지 중 하나를 반환합니다.

상태 코드	상태 메시지
1	성공
2	경고
3	실패
4	오류
5	치명적 오류

전체 상태를 선택하면 Data Transformation Engine이 상태 코드 및 Data Transformation 서비스로부터의 오류 메시지를 반환합니다. 메시지는 XML 형식입니다.

구조화되지 않은 Data Transformation 포트

구조화되지 않은 데이터 변환을 작성하면 디자이너가 기본 포트를 작성합니다. 다른 포트는 변환을 어떻게 구성했는지에 따라 디자이너에 의해 작성됩니다. 구조화되지 않은 데이터 변환 입력 및 출력 유형은 구조화되지 않은 데이터 변환이 Data Transformation Engine과 데이터를 주고받는 방법을 결정합니다.

다음 테이블에는 구조화되지 않은 데이터 변환 기본 포트가 설명되어 있습니다.

포트	입력/출력	설명
InputBuffer	입력	입력 유형이 버퍼일 경우 소스 데이터를 수신합니다. 입력 유형이 파일일 경우 소스 파일 이름 및 경로를 수신합니다.
OutputBuffer	출력	출력 유형이 버퍼이면 XML 데이터를 반환합니다. 출력 유형이 파일이면 출력 파일 이름을 반환합니다. 포트의 계층 출력 그룹을 구성한 경우 데이터를 반환하지 않습니다.

다음 테이블에는 변환을 구성할 때 디자이너에서 작성하는 다른 구조화되지 않은 데이터 변환 포트가 설명되어 있습니다.

포트	입력/출력	설명
OutputFileName	입력	출력 유형이 파일이면 출력 파일의 이름을 받습니다.
ServiceName	입력	동적 서비스 이름을 활성화하면 Data Transformation 서비스의 이름을 받습니다.

포트	입력/출력	설명
UDT_Status_Code	출력	상태 추적 수준이 설명만 또는 전체 상태일 경우 Data Transformation Engine의 상태 코드를 반환합니다.
UDT_Status_Message	출력	상태 추적 수준이 설명만 또는 전체 상태일 경우 Data Transformation Engine의 상태 메시지를 반환합니다.

참고: 관계형 대상에 대한 출력 포트 그룹을 **입력** 또는 **출력 계층** 탭에서 추가할 수 있습니다. 포트 그룹을 구성하면 계층적 그룹 및 포트가 다른 탭에 정의되어 있음을 알려 주는 메시지가 **UDT 포트** 탭에 나타납니다.

입력 및 출력 유형

입력 유형은 통합 서비스가 **Data Transformation Engine**으로 전달하는 데이터의 유형을 결정합니다. 입력 유형은 입력이 데이터인지, 아니면 소스 파일 경로인지를 결정합니다.

다음 입력 유형 중 하나를 구성하십시오.

- 버퍼. 구조화되지 않은 데이터 변환이 **InputBuffer** 포트에서 소스 데이터를 받습니다. 통합 서비스는 소스 행을 **InputBuffer** 포트에서 **Data Transformation Engine**으로 전달합니다.
- 파일. 구조화되지 않은 데이터 변환이 **InputBuffer** 포트에서 소스 파일 경로를 받습니다. 통합 서비스는 소스 파일 경로를 **Data Transformation Engine**으로 전달합니다. **Data Transformation Engine**은 소스 파일을 엽니다. Microsoft Excel 또는 Microsoft Word 파일 같은 이진 파일을 구문 분석하려면 파일 입력 유형을 사용합니다.

출력 그룹 및 포트를 정의하지 않으면 구조화되지 않은 데이터 변환은 출력 유형에 따라 데이터를 반환합니다.

다음 출력 유형 중 하나를 구성합니다.

- 버퍼. 구조화되지 않은 데이터 변환이 **Outputbuffer** 포트를 통해 XML을 반환합니다. XML 파서 변환을 **Outputbuffer** 포트에 연결해야 합니다.
- 파일. **Data Transformation Engine**이 통합 서비스로 데이터를 전달하는 대신 출력 파일을 기록합니다. **Data Transformation Engine**은 **OutputFilename** 포트에서 받은 파일 이름에 기반하여 출력 파일 이름을 지정합니다. XML을 PDF 파일 또는 Microsoft Excel 파일 같은 이진 데이터로 변환하려면 파일 출력 유형을 선택합니다.

통합 서비스는 각 소스 행에 대해 출력 파일 이름을 **OutputBuffer** 포트에 반환합니다. 출력 파일 이름이 비어 있으면 통합 서비스가 행 오류를 반환합니다. 오류가 발생하면 통합 서비스는 **OutputBuffer**에 null 값을 기록하고 행 오류를 반환합니다.

- 분할. 구조화되지 않은 데이터 변환이 **Data Transformation Engine**으로부터 받은 XML 데이터를 여러 세그먼트로 분할합니다. 구조화되지 않은 데이터 변환이 **OutputBuffer** 포트에 비해 너무 큰 XML 파일을 반환하는 경우, 분할 출력 유형을 선택합니다. 분할 출력을 구성할 경우 XML 데이터를 XML 파서 변환으로 전달합니다. 여러 XML 행을 단일 XML 파일로 처리하도록 XML 파서 변환을 구성하십시오.

추가적인 구조화되지 않은 Data Transformation 포트

Data Transformation 서비스에는 여러 입력 파일, 파일 이름 및 매개 변수가 필요할 수 있습니다. 이 서비스는 여러 출력 파일을 반환할 수 있습니다. 구조화되지 않은 데이터 변환을 작성하는 경우 디자이너가 하나의 **InputBuffer** 포트와 하나의 **OutputBuffer** 포트를 작성합니다. 구조화되지 않은 데이터 변환 및 **Data Transformation Engine** 간에 추가적인 파일 또는 파일 이름을 전달해야 하는 경우 입력 또는 출력 포트를 추가합니다. 수동으로 또는 **Data Transformation** 서비스에서 포트를 추가할 수 있습니다.

다음 테이블에는 **UDT 포트** 탭에서 작성할 수 있는 포트가 설명되어 있습니다.

포트 유형	입력/출력	설명
추가 입력(버퍼)	입력	Data Transformation Engine에 전달할 입력 데이터를 수신합니다.
추가 입력(파일)	입력	열려는 Data Transformation Engine에 대한 파일 이름 및 경로를 수신합니다.
서비스 매개 변수	입력	Data Transformation 서비스에 대한 입력 매개 변수를 수신합니다.
추가 출력(버퍼)	출력	Data Transformation Engine에서 XML 데이터를 수신합니다.
추가 출력 (파일)	출력	Data Transformation Engine에서 출력 파일 이름을 수신합니다.
통과	입력/ 출력	데이터를 변경하지 않고 구조화되지 않은 데이터 변환을 통해 전달합니다.

Data Transformation 서비스에서 포트 작성

Data Transformation 서비스에 입력 매개 변수, 추가 입력 파일 또는 사용자 정의 변수가 필요할 수 있습니다. 이 서비스는 두 개 이상의 출력 파일을 구조화되지 않은 데이터 변환에 반환할 수 있습니다. 매개 변수, 추가 입력 파일 및 추가 출력 파일을 전달하는 포트를 추가할 수 있습니다. Designer가 Data Transformation 서비스의 포트에 해당하는 포트를 작성합니다.

참고: 서비스의 포트를 채우려면 서비스 이름을 구성해야 합니다.

1. 구조화되지 않은 데이터 변환에서 **포트** 탭을 클릭합니다.
2. **서비스에서 채우기**를 클릭합니다.

디자이너가 Data Transformation 서비스의 서비스 매개 변수, 추가 입력 포트 및 추가 출력 포트 요구 사항을 표시합니다. 서비스 매개 변수에는 Data Transformation 시스템 변수와 사용자 정의 변수가 포함됩니다.

3. 작성할 포트를 선택하고 각 포트를 버퍼 포트 또는 파일 포트로 구성합니다.
4. **채우기**를 클릭하여 선택한 포트를 작성합니다. 표시된 모든 포트를 선택할 수 있습니다.

구조화되지 않은 Data Transformation 서비스 이름

구조화되지 않은 데이터 변환을 작성하는 경우, 디자이너는 Data Transformation 리포지토리에 있는 Data Transformation 서비스의 목록을 표시합니다. 구조화되지 않은 데이터 변환에서 호출하려는 Data Transformation 서비스의 이름을 선택하십시오. 변환을 작성한 후 서비스 이름을 변경할 수 있습니다. 이 서비스 이름은 **UDT 설정** 탭에 표시됩니다.

각 소스 행에 대해 서로 다른 Data Transformation 서비스를 실행하려면 동적 서비스 이름 특성을 활성화합니다. 각 소스 행과 함께 서비스 이름을 전달하십시오. 동적 서비스 이름을 활성화하면 Designer가 ServiceName 입력 포트를 작성합니다.

동적 서비스 이름을 활성화한 경우 Data Transformation 서비스에서 포트를 작성할 수 없습니다.

입력 포트의 관계형 구조를 정의할 경우, 동적 서비스 이름을 활성화할 수 없습니다. 각 Data Transformation 서비스에는 서로 다른 입력 데이터가 필요할 수 있습니다.

관계형 계층

포트 그룹을 정의하고 해당 그룹의 관계형 구조를 정의할 수 있습니다. 입력 포트의 계층을 작성하려면 **입력 계층** 탭에서 포트를 구성합니다. 행 데이터를 관계형 테이블 또는 다른 대상으로 전달하려면 **출력 계층** 탭에서 출력 포트를 구성합니다.

입력 포트의 관계형 구조를 정의하면 구조화되지 않은 데이터 변환이 **Data Transformation** 서비스로 전달할 XML을 생성합니다. 성능을 향상시키려면 구조화되지 않은 데이터 변환으로 XML을 플러시하도록 **PowerCenter** 통합 서비스를 활성화합니다. 입력 플러싱을 활성화할 경우, **PowerCenter** 통합 서비스는 루트 값에 대한 모든 데이터를 받은 후에 각 그룹에서 XML을 플러시합니다. 예를 들어 직원 그룹 및 직원 주소 그룹이 있는 것으로 가정합니다. **PowerCenter** 통합 서비스는 데이터에서 다른 직원이 발생할 때마다 두 그룹에서 구조화되지 않은 데이터 변환으로 데이터를 플러시할 수 있습니다. 각 그룹의 데이터는 루트 그룹의 기본 키를 기준으로 정렬해야 합니다. 그룹에 동일한 키가 없는 경우에는 파이프라인에서 그룹을 조인할 수 있습니다.

출력 그룹을 구성할 때 출력 그룹은 출력 데이터를 전달하려는 관계형 테이블 또는 대상을 나타냅니다. **Data Transformation Engine**은 XML 파일을 **OutputBuffer** 포트에 기록하는 대신 그룹 포트에 행을 반환합니다. 변환은 출력 유형에 따라 행을 기록합니다.

출력 계층 탭의 왼쪽 창에서 그룹의 계층을 작성합니다. 모든 그룹은 루트 그룹 아래에 있습니다. 루트는 삭제할 수 없습니다. 각 그룹마다 포트 및 다른 그룹이 포함될 수 있습니다. 그룹 구조는 대상 테이블간 관계를 나타냅니다. 그룹 내에 그룹을 정의하면 그룹 간에 상위-하위 관계가 정의됩니다. 디자이너는 생성되는 키로 그룹 간에 기본 키-외래 키 관계를 정의합니다.

그룹에 대한 포트를 표시하려면 그룹을 선택하십시오. 해당 그룹에서 포트를 추가하거나 삭제할 수 있습니다. 포트를 추가하면 디자이너가 기본 포트 구성을 작성합니다. 포트 이름, 데이터 유형 및 전체 자릿수를 변경하십시오. 포트에 데이터가 포함되어 하는 경우, **Null**이 아님을 선택합니다. 그렇지 않으면 출력 데이터가 선택 사항이 됩니다.

구조화되지 않은 데이터 변환을 작업 공간에서 보면 변환 그룹의 각 포트에는 그룹 이름을 포함하는 접두사가 있습니다.

그룹을 삭제할 경우 그룹의 포트 및 하위 그룹을 삭제하십시오.

계층 스키마 내보내기

구조화되지 않은 데이터 변환에 계층 출력 그룹을 정의하는 경우 데이터를 변환하기 위해 작성하는 **Data Transformation** 서비스에 동일한 구조를 정의해야 합니다. 구조화되지 않은 데이터 변환에서 XML 스키마 파일로 계층 구조를 내보냅니다. 스키마를 데이터 프로세서 변환에 가져옵니다. 그러면 소스 문서의 콘텐츠를 데이터 프로세서 변환의 XML 요소 및 특성에 매핑할 수 있습니다.

관계형 계층 탭에서 그룹 계층을 내보내려면 **XML 스키마로 내보내기**를 클릭하십시오. XSD 파일의 이름과 위치를 선택하십시오. **Information Developer**를 사용하여 스키마를 가져올 때 액세스할 수 있는 위치를 선택합니다.

디자이너는 다음 네임스페이스를 사용하여 스키마를 작성합니다.

```
"www.informatica.com/UDT/XSD/<mappingName_<Transformation_Name>>"
```

스키마에는 다음 주석이 포함되어 있습니다.

```
<!-- ===== AUTO-GENERATED FILE - DO NOT EDIT ===== -->
<!-- ===== This file has been generated by Informatica PowerCenter ===== -->
```

스키마를 수정하는 경우 **Data Transformation Engine**이 구조화되지 않은 데이터 변환에서 출력 포트와 같은 형식이 아닌 데이터를 반환할 수 있습니다.

스키마의 XML 요소는 계층의 출력 포트를 나타냅니다. **Null** 값을 포함할 수 있는 열에는 **minOccurs=0** 및 **maxOccurs=1** XML 특성이 있습니다.

매핑

매핑을 작성할 때 실행하려는 **Data Transformation** 서비스 유형에 따라 매핑을 디자인합니다. 예를 들어 **Data Transformation** 파서 및 매핑은 XML 데이터를 생성합니다. XML 데이터의 행을 반환하거나 XML 파일을 반환하도록 구조화되지 않은 데이터 변환을 구성할 수 있습니다.

Data Transformation 직렬 변환기 구성 요소는 XML에서 모든 출력을 생성할 수 있습니다. HTML이나 Microsoft Word 또는 Microsoft Excel과 같은 이진 파일을 생성할 수 있습니다. 출력이 이진 데이터인 경우 **Data Transformation Engine**은 이 출력을 구조화되지 않은 데이터 변환에 다시 전달하는 대신 파일에 씁니다.

다음 예제에는 구조화되지 않은 데이터 변환을 사용하여 매핑을 구성하는 방법이 나와 있습니다.

관계형 테이블을 위한 Word 문서 구문 분석

Microsoft Word 문서에서 주문 정보를 추출하여 주문 헤더 테이블 및 주문 세부 테이블에 주문 정보를 기록할 수 있습니다. **Data Transformation** 파서 서비스를 호출하고 구문 분석할 각 Word 문서의 이름을 전달하도록 구조화되지 않은 데이터 변환을 구성하십시오. **Data Transformation Engine**이 Word 문서를 열어 구문 분석한 후, 구조화되지 않은 데이터 변환으로 행을 반환합니다. 구조화되지 않은 데이터 변환은 주문 헤더 및 주문 세부 정보를 관계형 대상으로 전달합니다.

매핑에는 다음과 같은 개체가 포함됩니다.

- 소스 한정자 변환. 각 Microsoft Word 파일 이름을 구조화되지 않은 데이터 변환으로 전달합니다. 소스 파일 이름에는 주문 정보를 포함하는 파일의 전체 경로가 포함됩니다.
- 구조화되지 않은 데이터 변환. 입력 유형은 파일이고, 출력 유형은 버퍼입니다. 변환에는 주문 헤더 출력 그룹 및 주문 세부 정보 출력 그룹이 포함됩니다. 두 그룹은 기본 키-외래 키 관계를 갖고 있습니다.

구조화되지 않은 데이터 변환은 **InputBuffer** 포트에서 소스 파일 이름을 받습니다. 이 **Data Transformation**은 해당 이름을 **Data Transformation Engine**으로 전달합니다. **Data Transformation Engine**은 파서 서비스를 실행하여 Word 문서에서 주문 헤더 및 주문 세부 정보 행을 추출합니다. 그런 다음, **Data Transformation Engine**이 데이터를 구조화되지 않은 데이터 변환으로 반환합니다. 구조화되지 않은 데이터 변환은 주문 헤더 그룹 및 주문 세부 정보 그룹의 데이터를 관계형 대상으로 전달합니다.

- 관계형 대상. 구조화되지 않은 데이터 변환으로부터 행을 받습니다.

XML에서 Excel 시트 작성

XML 파일에서 직원 이름 및 주소를 추출하고 이름 목록이 포함된 Microsoft Excel 시트를 작성할 수 있습니다.

매핑에는 다음 구성 요소가 포함됩니다.

- XML 소스 파일. 직원 이름 및 주소가 포함되어 있습니다.
- 소스 한정자 변환. XML 데이터 및 출력 파일 이름을 구조화되지 않은 데이터 변환에 전달합니다. XML 파일에는 직원 이름이 포함되어 있습니다.
- 구조화되지 않은 데이터 변환. 입력 유형은 버퍼이며 출력 유형은 파일입니다. 구조화되지 않은 데이터 변환에서 **InputBuffer** 포트의 XML 데이터 및 **OutputFileName** 포트의 파일 이름을 수신합니다. XML 데이터 및 파일 이름을 **Data Transformation Engine**에 전달합니다.

Data Transformation Engine은 직렬 변환기 서비스를 실행하여 XML 데이터를 Microsoft Excel 파일로 변환합니다. **OutputFilename** 값에 기반한 파일 이름으로 Excel 파일을 작성합니다.

구조화되지 않은 데이터 변환은 **Data Transformation Engine**에서 출력 파일 이름만 수신합니다. 구조화되지 않은 데이터 변환 **OutputBuffer** 포트는 **OutputFilename** 값을 반환합니다.

- 플랫폼 파일 대상. 출력 파일 이름을 수신합니다.

XML 파일 출력 분할

Data Transformation 파서 및 매핑 구성 요소는 모든 형식의 데이터를 변환하여 **XML** 데이터를 생성할 수 있습니다. **XML** 데이터가 큰 경우, **XML**을 여러 세그먼트로 분할하고 세그먼트를 **XML** 파서 변환으로 전달할 수 있습니다. **XML** 파서 변환은 세그먼트를 받고 **XML** 데이터를 단일 문서로 처리합니다.

XML 출력을 분할하도록 구조화되지 않은 데이터 변환을 구성하면 구조화되지 않은 데이터 변환은 **OutputBuffer** 포트 크기에 맞춰 **XML**을 반환합니다. **XML** 파일 크기가 출력 포트 전체 자릿수보다 큰 경우, 통합 서비스는 포트 크기보다 작거나 같은 여러 개의 파일로 **XML**을 분할합니다. **XML** 파서 변환은 **XML**을 구문 분석하고 행을 관계형 테이블 또는 다른 대상으로 전달합니다.

예를 들어 **Data Transformation** 파서 서비스를 통해 **Microsoft Word** 문서에서 주문 헤더 및 세부 정보를 추출할 수 있습니다.

매핑에는 다음 구성 요소가 포함됩니다.

- 소스 한정자 변환. **Word** 문서 파일 이름을 구조화되지 않은 데이터 변환으로 전달합니다. 소스 파일 이름에는 주문 정보를 포함하는 파일의 전체 경로가 포함됩니다.
- 구조화되지 않은 데이터 변환. 입력 유형은 파일이고, 출력 유형은 분할입니다. 구조화되지 않은 데이터 변환은 **InputBuffer** 포트에서 소스 파일 이름을 받습니다. 이 **Data Transformation**은 해당 파일 이름을 **Data Transformation Engine**으로 전달합니다. **Data Transformation Engine**이 소스 파일을 열어 구문 분석한 후, 구조화되지 않은 데이터 변환으로 **XML** 데이터를 반환합니다.

구조화되지 않은 데이터 변환은 **XML** 데이터를 받고 **XML** 파일을 여러 개의 작은 파일로 분할한 후, 세그먼트를 **XML** 파서 변환으로 전달합니다. 구조화되지 않은 데이터 변환은 **OutputBuffer** 포트 크기보다 작은 세그먼트의 데이터를 반환합니다. 이 변환은 여러 세그먼트의 **XML** 데이터를 반환할 때 각 행에 대해 동일한 통과 데이터를 생성합니다. 구조화되지 않은 데이터 변환은 행이 성공적인지 여부와 상관없이 통과 포트에서 데이터를 반환합니다.

- **XML** 파서 변환. 입력 스트리밍 활성화 세션 속성이 활성화됩니다. **XML** 파서 변환은 **DataInput** 포트에서 **XML** 데이터를 받습니다. 입력 데이터는 여러 세그먼트로 분할됩니다. **XML** 파서 변환은 **XML** 데이터를 구문 분석하여 주문 헤더 및 세부 정보 행을 생성합니다. 이 변환은 주문 헤더 및 세부 정보 행을 관계형 대상으로 전달합니다. 통과 데이터는 필터 변환으로 반환합니다.
- 필터 변환. 관계형 대상으로 전달하기 전에 중복되는 통과 데이터를 제거합니다.
- 관계형 대상. **XML** 파서 변환 및 필터 변환의 각 그룹으로부터 데이터를 받습니다.

구조화되지 않은 데이터 매핑에 대한 규칙 및 지침

구조화되지 않은 데이터 매핑을 작성할 때 다음 규칙 및 지침을 따르십시오.

- 출력 포트의 계층적 그룹을 구성할 경우 통합 서비스는 **OutputBuffer** 포트에 기록하는 대신 포트 그룹에 기록합니다. 통합 서비스는 변환에 대해 정의한 출력 유형과 상관없이 포트 그룹에 기록합니다.
- 구조화되지 않은 데이터 변환에 파일 출력 유형이 있고 그룹 출력 포트를 정의하지 않은 경우, **OutputBuffer** 포트를 다운스트림 변환에 연결해야 합니다. 그렇지 않으면 매핑이 올바르지 않습니다. **Data Transformation** 서비스가 출력 파일에 기록하는 경우 **OutputBuffer** 포트는 출력 파일 이름을 포함합니다.
- 구조화되지 않은 데이터 변환으로 서비스 이름 입력 포트에 서비스 이름을 전달하려면 동적 서비스 이름을 활성화합니다. 동적 서비스 이름을 활성화하면 디자이너가 서비스 이름 입력 포트를 작성합니다.
- 구조화되지 않은 데이터 변환을 포함하는 서비스 이름을 구성하거나 동적 서비스 이름 옵션을 활성화해야 합니다. 그렇지 않으면 매핑이 올바르지 않습니다.
- 구조화되지 않은 데이터 변환의 **XML** 출력을 **XML** 파서 변환에 연결합니다.

구조화되지 않은 Data Transformation 작성

PowerCenter 변환 개발자 또는 매핑 디자이너를 사용하여 구조화되지 않은 데이터 변환을 작성할 수 있습니다.

1. 매핑 디자이너 또는 변환 개발자에서 **변환 > 작성**을 클릭합니다.
2. 변환 유형으로 **구조화되지 않은 Data Transformation**을 선택합니다.
3. 변환 이름을 입력합니다.
4. **작성**을 클릭합니다.

구조화되지 않은 **Data Transformation** 대화 상자가 표시됩니다.

5. 다음 속성을 구성합니다.

속성	설명
서비스 이름	사용할 Data Transformation 서비스의 이름입니다. 디자이너가 Data Transformation 리포지토리 폴더의 Data Transformation 서비스를 표시합니다. 동적 서비스 이름을 활성화할 계획인 경우 이름을 선택하지 마십시오. 변환을 작성한 후 UDT 설정 탭에서 서비스 이름을 추가할 수 있습니다.
입력 유형	Data Transformation 엔진이 입력 데이터를 수신하는 방법을 나타냅니다. 기본값은 버퍼입니다.
출력 유형	Data Transformation Engine이 출력 데이터를 반환하는 방법을 설명합니다. 기본값은 버퍼입니다.

6. **확인**을 클릭합니다.
7. **UDT 설정** 탭에서 서비스 이름, 입력 및 출력 유형을 변경할 수 있습니다.
8. **속성** 탭에서 구조화되지 않은 데이터 변환 속성을 구성합니다.
9. Data Transformation 서비스에 두 개 이상의 입력 또는 출력 파일이 있거나 입력 매개 변수가 필요한 경우 **UDT 포트** 탭에서 포트를 추가할 수 있습니다. 또한 **포트** 탭에서 통과 포트를 추가할 수 있습니다.
10. XML 데이터 대신 구조화되지 않은 데이터 변환에서 행 데이터를 반환하려면 **관계형 계층** 탭에서 출력 포트 그룹을 작성합니다.
11. 포트 그룹을 작성하는 경우 **관계형 계층** 탭에서 그룹을 설명하는 스키마를 내보냅니다.
12. 스키마를 Data Transformation 프로젝트로 가져와서 프로젝트 출력을 정의합니다.

제 32 장

업데이트 전략 변환

이 장에 포함된 항목:

- [업데이트 전략 변환 개요, 455](#)
- [매핑 내 행에 플래그 지정, 456](#)
- [세션의 업데이트 전략 설정, 458](#)
- [업데이트 전략 검사 목록, 459](#)

업데이트 전략 변환 개요

업데이트 전략 변환은 활성 변환입니다. 데이터 웨어하우스를 디자인할 때 대상에 저장할 정보의 유형을 결정해야 합니다. 대상 테이블을 디자인하는 과정에서 모든 기록 데이터를 유지할지, 아니면 최근 변경 사항만 유지할지를 결정해야 합니다.

예를 들어 고객 데이터가 포함된 T_CUSTOMERS라는 대상 테이블이 있다고 가정합니다. 고객 주소가 변경되는 경우 고객 행 일부를 업데이트하는 대신 테이블에 원래 주소를 저장하고 싶을 수 있습니다. 이 경우 업데이트된 주소를 포함하는 새 행을 작성하고 이전 고객 주소가 포함된 원래 행을 보존할 수 있습니다. 이 예는 대상 테이블에 기록 정보를 저장하는 방법을 보여 줍니다. 이와 달리, T_CUSTOMERS 테이블이 최신 고객 데이터의 스냅샷이 되게 하려는 경우 기존 고객 행을 업데이트하여 원래 주소가 손실될 수 있습니다.

선택한 모델에 따라 기존 행의 변경 사항을 처리하는 방법이 결정됩니다. PowerCenter에서는 다음 두 수준에서 업데이트 전략을 설정합니다.

- **세션 내.** 세션을 구성할 때 통합 서비스가 모든 행을 동일한 방식으로 처리할지(예: 모든 행을 삽입으로 처리), 아니면 세션 매핑에 코딩된 명령을 사용하여 행에 서로 다른 데이터베이스 작업을 플래그로 지정할지를 결정할 수 있습니다.
- **매핑 내.** 매핑 내에서 업데이트 전략 변환을 사용하여 행에 삽입, 삭제, 업데이트 또는 거부 플래그를 지정합니다.

참고: 사용자 지정 변환을 사용하여 행에 삽입, 삭제, 업데이트 또는 거부 플래그를 지정할 수도 있습니다.

업데이트 전략 설정

업데이트 전략을 정의하려면 다음 단계를 완료합니다.

1. 매핑에서 삽입, 업데이트, 삭제 또는 거부를 위해 행에 플래그를 지정하는 방식을 제어하려면 업데이트 전략 변환을 매핑에 추가합니다. 동일한 대상이 지정된 행에 서로 다른 데이터베이스 작업을 플래그로 지정하거나 행을 거부하려는 경우 업데이트 전략 변환이 필수적입니다.

2. 세션을 구성할 때 행에 플래그를 지정하는 방식을 정의합니다. 모든 행에 삽입, 삭제 또는 업데이트 플래그를 지정하거나, 데이터 구동 옵션을 선택하여 통합 서비스가 세션 매핑 내의 업데이트 전략 변환에 코딩된 명령을 따르게 할 수 있습니다.
3. 세션을 구성할 때 각 대상에 대한 삽입, 업데이트 및 삭제 옵션을 정의합니다. 대상별 기준에서는 삽입 및 업데이트를 허용하거나 허용하지 않을 수 있으며, 업데이트를 처리하는 세 가지 방식 중에서 선택할 수 있습니다.

관련 항목:

- [“세션의 업데이트 전략 설정” 페이지 458](#)

매핑 내 행에 플래그 지정

업데이트 전략을 최대한으로 제어하려면 업데이트 전략 변환을 매핑에 추가합니다. 이 변환의 가장 중요한 기능은 개별 행에 삽입, 삭제, 업데이트 또는 거부 플래그를 지정하는 데 사용되는 업데이트 전략 식입니다.

다음 표에는 각 데이터베이스 작업에 대한 상수 및 해당하는 숫자 값이 나열되어 있습니다.

작업	상수	숫자 값
삽입	DD_INSERT	0
업데이트	DD_UPDATE	1
삭제	DD_DELETE	2
거부	DD_REJECT	3

통합 서비스는 다른 모든 값을 삽입으로 처리합니다.

거부된 행 전달

거부된 행을 다음 변환에 전달하거나 삭제하도록 업데이트 전략 변환을 구성할 수 있습니다. 기본적으로 통합 서비스는 거부된 행을 다음 변환에 전달합니다. 통합 서비스는 행에 거부 플래그를 지정하고 해당 행을 세션 거부 파일에 기록합니다. 거부된 행 전달을 선택하지 않을 경우 통합 서비스는 거부된 행을 삭제하고 해당 행을 세션 로그 파일에 기록합니다.

행 오류 처리를 활성화하면 통합 서비스는 거부된 행과 삭제된 행을 행 오류 로그에 기록하고 거부 파일을 생성하지 않습니다. 삭제된 행을 행 오류 로그뿐 아니라 세션 로그에 기록하려면 자세한 정보 표시 데이터 추적을 활성화합니다.

업데이트 전략 식

업데이트 전략 식에서는 각 행이 특정 조건을 충족하는지 여부를 테스트하기 위해 변환 언어에서 IIF 또는 DECODE 함수를 자주 사용합니다. 조건을 충족하는 경우 각 행에 숫자 코드를 할당하여 특정 데이터베이스 작업을 위해 플래그를 지정할 수 있습니다. 예를 들어 다음 IIF 문은 시작 날짜가 적용 날짜보다 뒤에 오는 경우 거부를 위해 행에 플래그를 지정합니다. 그렇지 않은 경우에는 행에 업데이트 플래그를 지정합니다.

```
IIF( ( ENTRY_DATE > APPLY_DATE), DD_REJECT, DD_UPDATE )
```

업데이트 전략 변환을 작성하려면 다음을 수행하십시오.

1. 매핑 디자이너에서 업데이트 전략 변환을 매핑에 추가합니다.
2. 레이아웃 > 열 연결을 클릭합니다.
3. 업데이트 전략 변환을 통해 전달하려는 데이터를 나타내는 다른 변환에서 모든 포트를 끌어 옵니다.
업데이트 전략 변환에서는 끌어 오는 각 포트의 사본을 디자이너가 작성합니다. 또한 디자이너는 새 포트를 원래 포트에 연결합니다. 업데이트 전략 변환의 각 포트는 조합형 입력/출력 포트입니다.
일반적으로는 특정 대상을 향하는 모든 열을 선택합니다. 해당 열이 업데이트 전략 변환을 통과한 후 이 정보에 업데이트, 삽입, 삭제 또는 거부 플래그가 지정됩니다.
4. 업데이트 전략 변환을 열고 이름을 변경합니다.
업데이트 전략 변환에 대한 명명 규칙은 `UPD_TransformationName`입니다.
5. 속성 탭을 클릭합니다.
6. 업데이트 전략 식 필드에서 단추를 클릭합니다.
식 편집기가 표시됩니다.
7. 업데이트 전략 식을 입력하여 행에 삽입, 삭제, 업데이트 또는 거부 플래그를 지정합니다.
8. 식의 유효성을 검사하고 확인을 클릭합니다.
9. 확인을 클릭합니다.
10. 업데이트 전략 변환의 포트를 다른 변환 또는 대상 인스턴스에 연결합니다.

집계 및 업데이트 전략 변환

집계 및 업데이트 전략 변환을 동일한 파이프라인의 일부로 연결하는 경우 집계 변환을 업데이트 전략 변환 앞에 배치합니다. 이 순서에서는 통합 서비스가 집계 계산을 수행한 후 이 계산 결과를 포함하는 행에 삽입, 업데이트, 삭제 또는 거부 플래그를 지정합니다.

업데이트 전략 변환을 집계 변환 앞에 배치하는 경우 집계 변환이 다른 작업에 대한 플래그가 지정된 행을 처리하는 방식을 고려해야 합니다. 이 순서에서는 통합 서비스가 집계 계산을 수행하기 전에 행에 삽입, 업데이트, 삭제 또는 거부 플래그를 지정합니다. 행에 지정된 플래그에 따라 집계 변환이 계산에 사용된 행의 값을 처리하는 방식이 결정됩니다. 예를 들어 행에 삭제 플래그를 지정한 후 이 행을 사용하여 합계를 계산하면 통합 서비스가 이 행의 값을 뺍니다. 행에 거부 플래그를 지정한 후 이 행을 사용하여 합계를 계산하면 통합 서비스가 이 행의 값을 포함하지 않습니다. 행에 삽입 또는 업데이트 플래그를 지정한 후 이 행을 사용하여 합계를 계산하면 통합 서비스가 이 행의 값을 합계에 추가합니다.

조회 및 업데이트 전략 변환

동적 조회 캐시를 사용하는 조회 변환을 포함하여 매핑을 작성하는 경우 업데이트 전략 변환을 사용하여 대상 테이블의 행에 플래그를 지정해야 합니다. 업데이트 전략 변환 및 동적 조회 캐시를 사용하여 세션을 구성하는 경우 특정한 세션 속성을 정의해야 합니다.

소스 행을 다음으로 처리 옵션을 데이터 구동으로 정의해야 합니다. 세션 속성의 속성 탭에서 이 옵션을 지정합니다.

또한 다음과 같이 업데이트 전략 대상 테이블 옵션도 정의해야 합니다.

- 삽입 선택
- 업데이트 시 업데이트 선택
- 삭제 선택 안 함

이러한 업데이트 전략 대상 테이블 옵션은 통합 서비스가 업데이트로 표시된 행을 업데이트하고 삽입으로 표시된 행을 삽입하도록 합니다.

데이터 구동을 선택하지 않을 경우 통합 서비스는 사용자가 소스 행을 다음으로 처리에서 지정하는 데이터베이스 작업에 대해 모든 행에 플래그를 지정하고 매핑의 업데이트 전략 변환을 사용하여 행에 플래그를 지정하지 않습니다. 통합 서비스는 올바른 행을 삽입하고 업데이트하지 않습니다. 업데이트 시 업데이트를 선택하지 않을 경우 통합 서비스는 대상 테이블에서 업데이트 플래그가 지정된 행을 올바르게 업데이트하지 않습니다. 따라서 조회 캐시와 대상 테이블이 동기화되지 않을 수 있습니다.

세션의 업데이트 전략 설정

세션을 구성할 때 업데이트를 비롯한 데이터베이스 작업을 처리하는 다양한 옵션을 사용할 수 있습니다.

모든 행에 대한 작업 지정

세션을 구성할 때 소스 행을 다음으로 처리 설정을 사용하여 모든 행에 대한 단일 데이터베이스 작업을 선택할 수 있습니다.

다음 테이블에서는 소스 행을 다음으로 처리 설정의 옵션을 보여 줍니다.

설정	설명
삽입	모든 행을 삽입으로 처리합니다. 행 삽입이 데이터베이스의 기본 또는 외래 키 제약 조건을 위반하는 경우 통합 서비스는 행을 거부합니다.
삭제	모든 행을 삭제로 처리합니다. 각 행에 대해 통합 서비스가 대상 테이블에서 대응하는 행을 찾은 경우(기본 키 값 기준) 통합 서비스가 행을 삭제합니다. 리포지토리의 대상 정의에 기본 키 제약 조건이 존재해야 합니다.
업데이트	모든 행을 업데이트로 처리합니다. 각 행에 대해 통합 서비스는 대상 테이블에서 일치하는 기본 키 값을 찾습니다. 값이 있는 경우 통합 서비스는 행을 업데이트합니다. 대상 정의에 기본 키 제약 조건이 존재해야 합니다.
데이터 기반	통합 서비스는 세션 매핑의 업데이트 전략 및 사용자 지정 변환에 코딩된 명령에 따라 행에 삽입, 삭제, 업데이트 또는 거부 플래그를 지정하는 방식을 결정합니다. 세션의 매핑에 업데이트 전략 변환이 포함된 경우 이 필드는 기본적으로 데이터 구동으로 표시됩니다. 매핑이 업데이트 전략 또는 사용자 지정 변환을 포함하는 경우 데이터 구동을 선택하지 않으면 워크플로우 관리자에 경고가 표시됩니다. 세션을 실행하면 통합 서비스가 매핑의 업데이트 전략 또는 사용자 지정 변환의 명령에 따라 행에 플래그를 지정하는 방식을 결정하지 않습니다.

다음 테이블에는 각 설정의 업데이트 전략이 설명되어 있습니다.

설정	용도
삽입	대상 테이블을 처음 채우거나 기록 데이터 웨어하우스를 유지합니다. 후자의 경우 대상 테이블의 선택적 그룹이 아닌 전체 데이터 웨어하우스에 대해 이 전략을 설정해야 합니다.
삭제	대상 테이블을 지웁니다.

설정	용도
업데이트	대상 테이블을 업데이트합니다. 데이터 웨어하우스에 기록 데이터 또는 스냅샷이 포함된 경우 이 설정을 선택할 수 있습니다. 나중에 개별 대상 테이블을 업데이트하는 방법을 구성할 때 업데이트된 행을 새 행으로 삽입할지, 아니면 업데이트된 정보를 사용하여 대상의 기존 행을 수정할지 여부를 결정할 수 있습니다.
데이터 기반	행에 삽입, 삭제, 업데이트 또는 거부 플래그를 지정하는 방법을 세밀하게 제어합니다. 동일한 테이블이 대상으로 지정된 행을 경우에 따라 한 작업(예: 업데이트)이나 다른 작업(예: 거부)의 플래그를 지정해야 한다면 이 설정을 선택하십시오. 또한 이 설정은 행에 거부 플래그를 지정할 수 있는 유일한 방법을 제공합니다.

개별 대상 테이블에 대한 작업 지정

세션에서 모든 행을 처리하는 방식을 결정했으면 개별 대상에 대한 업데이트 전략 옵션도 설정해야 합니다. 세션 속성의 매핑 탭에 있는 변환 보기에서 업데이트 전략 옵션을 정의합니다.

다음 업데이트 전략 옵션을 설정할 수 있습니다.

- **삽입.** 대상 테이블에 행을 삽입하려면 이 옵션을 선택합니다.
- **삭제.** 테이블에서 행을 삭제하려면 이 옵션을 선택합니다.
- **업데이트.** 이 경우 다음 옵션을 사용할 수 있습니다.
 - **업데이트 시 업데이트.** 대상 테이블에 존재하는 경우 업데이트 플래그가 지정된 각 행을 업데이트합니다.
 - **삽입 시 업데이트.** 업데이트 플래그가 지정된 각 행을 삽입합니다.
 - **업데이트 기타 항목 삽입.** 행이 존재하는 경우 업데이트합니다. 그렇지 않으면 행을 삽입합니다.
- **테이블 잘라내기.** 데이터를 로드하기 전에 대상 테이블을 자르려면 이 옵션을 선택합니다.

업데이트 전략 검사 목록

업데이트 전략을 선택하려면 세션 내에서 올바른 옵션을 설정하고 가능한 경우 매핑에 업데이트 전략 변환을 추가해야 합니다. 이 섹션에서는 업데이트 전략의 다양한 버전을 구현하기 위해 필요한 항목을 요약하여 설명합니다.

- 대상 테이블에는 삽입만 수행합니다.

세션을 구성할 때 소스 행을 다음으로 처리 세션 속성에서 삽입을 선택합니다. 또한 세션의 모든 대상 인스턴스에 대해 삽입 옵션을 선택했는지 확인합니다.
- 대상 테이블에서 모든 행을 삭제합니다.

세션을 구성할 때 소스 행을 다음으로 처리 세션 속성에서 삭제를 선택합니다. 또한 세션의 모든 대상 인스턴스에 대해 삭제 옵션을 선택했는지 확인합니다.
- 대상 테이블의 콘텐츠에 대해서만 업데이트를 수행합니다.

세션을 구성할 때 소스 행을 다음으로 처리 세션 속성에서 업데이트를 선택합니다. 각 대상 테이블 인스턴스에 대해 업데이트 옵션을 구성할 때 각 대상 인스턴스에 대한 업데이트 옵션을 선택했는지 확인하십시오.
- 동일한 대상 테이블이 지정된 서로 다른 행에 서로 다른 데이터베이스 작업을 수행합니다.

매핑에 업데이트 전략 변환을 추가합니다. 변환의 업데이트 전략 식을 작성할 때 **DECODE** 또는 **IIF** 함수를 사용하여 행에 서로 다른 작업(삽입, 삭제, 업데이트 또는 거부)의 플래그를 지정합니다. 이 매핑을 사용하는 세

션을 구성할 때 소스 행을 다음으로 처리 세션 속성에서 데이터 구동을 선택합니다. 각 대상 테이블 인스턴스에 대해 삽입, 삭제 또는 업데이트 옵션 중 하나를 선택했는지 확인하십시오.

- 데이터를 거부합니다.

매핑에 업데이트 전략 변환을 추가합니다. 변환의 업데이트 전략 식을 작성할 때 **DECODE** 또는 **IIF**를 사용하여 행을 거부하는 조건을 지정합니다. 이 매핑을 사용하는 세션을 구성할 때 소스 행을 다음으로 처리 세션 속성에서 데이터 구동을 선택합니다.

제 33 장

XML 변환

이 장에 포함된 항목:

- [XML 소스 한정자 변환, 461](#)
- [XML 파서 변환, 461](#)
- [XML 생성기 변환, 462](#)

XML 소스 한정자 변환

XML 소스 정의를 매핑 디자이너 작업 공간으로 끌거나 수동으로 XML 소스 정의를 작성하여 XML 소스 한정자 변환을 매핑에 추가할 수 있습니다. XML 소스 정의를 매핑에 추가하는 경우, XML 소스 한정자 변환에 XML 소스 정의를 연결해야 합니다. XML 소스 한정자 변환은 통합 서비스에서 세션을 실행할 때 읽는 데이터 요소를 정의합니다. 이 변환은 PowerCenter가 소스 데이터를 어떻게 읽는지 결정합니다. XML 소스 한정자 변환은 활성 변환입니다.

XML 소스 한정자 변환에는 항상 XML 소스의 모든 열에 대응되는 입력 또는 출력 포트가 각각 있습니다. 소스 정의를 위해 XML 소스 한정자 변환을 작성하는 경우 디자이너가 XML 소스 정의의 각 포트를 XML 소스 한정자 변환의 포트에 연결합니다. 링크를 제거하거나 편집할 수 없습니다. 매핑에서 XML 소스 정의를 제거하는 경우 디자이너는 해당 XML 소스 한정자 변환도 제거합니다. XML 소스 정의 하나를 특정 XML 소스 한정자 변환에 연결할 수 있습니다.

한 개의 XML 소스 한정자 그룹에 있는 포트를 다른 변환의 포트에 연결하여 별도의 데이터 흐름을 형성할 수 있습니다. 그러나 XML 소스 한정자 변환에서 둘 이상의 그룹에 있는 포트를 동일한 대상 변환의 포트에 연결할 수 없습니다.

일부 속성을 편집하고 메타데이터 확장을 XML 소스 한정자 변환에 추가할 수 있습니다.

XML 파서 변환

파이프라인 내부의 XML을 추출하려면 XML 파서 변환을 사용합니다. XML 파서 변환을 사용하면 TIBCO 또는 MQ Series 등의 메시징 시스템 및 파일 또는 데이터베이스 등의 다른 소스에서 XML 데이터를 추출할 수 있습니다. XML 파서 변환 기능은 XML 소스 기능과 유사하며, 파이프라인의 XML을 구문 분석한다는 점에서 차이가 있습니다. 예를 들어 TIBCO 소스에서 XML 데이터를 추출하여 관계형 대상으로 데이터를 전달할 수 있습니다. XML 파서 변환은 활성 변환입니다.

XML 파서 변환은 단일 입력 포트에서 XML 데이터를 읽고 하나 이상의 출력 포트에 데이터를 기록합니다.

XML 생성기 변환

파이프라인 내부에 XML을 작성하려면 XML 생성기 변환을 사용합니다. XML 생성기 변환을 사용하면 TIBCO 및 MQ Series 등의 메시징 시스템 또는 파일 또는 데이터베이스 등의 다른 소스에서 데이터를 읽을 수 있습니다. XML 생성기 변환 기능은 XML 대상 기능과 유사하며, 파이프라인에 XML을 생성한다는 점에서 차이가 있습니다. 예를 들어 관계형 소스에서 데이터를 추출하여 대상으로 XML 데이터를 전달할 수 있습니다. XML 생성기 변환은 활성 변환입니다.

XML 생성기 변환은 여러 포트에서 데이터를 받아들이고 단일 출력 포트를 통해 XML을 기록합니다.

인덱스

A

ABORT 함수
 사용 [40](#)
address.dic
 데이터 마스킹 [146](#)
API 메서드
 Java 변환 [226](#)
API 함수
 사용자 지정 변환 [89](#)
ASCII
 사용자 지정 변환 [58](#)
 외부 프로시저 변환 [155](#)
ASCII 모드
 조이너 변환에 대해 정렬 순서 구성 [259](#)
AutoCommit
 SQL 설정 탭 [402](#)

B

BankSoft 예제
 Informatica 외부 프로시저 [166](#)
 개요 [157](#)
BigDecimal 데이터 유형
 Java 변환 [223](#)
패키지 가져오기 탭
 Java 변환 [218](#)
 예제 [249](#)
포트
 HTTP 변환 [194](#)
 Java 변환 [214](#)
 구성 [29](#), [30](#)
 구조화되지 않은 데이터 변환 [449](#)
 그룹 기준 [51](#)
 기본값 개요 [38](#)
 라우터 변환 [347](#)
 변수 포트 [36](#)
 사용자 지정 변환 [60](#)
 생성 [30](#)
 소스 한정자 [379](#)
 순위 변환 [339](#)
 시퀀스 생성기 변환 [349](#)
 작성 [29](#)
 정렬된 포트 옵션 [379](#)
 정렬됨 [53](#), [379](#)
 조회 변환 [273](#)
 집계 변환 [49](#)
 평가 순서 [37](#)
 합집합 변환 [441](#)
포트 값
 Java 변환 [214](#)
포트 종속성
 사용자 지정 변환 [61](#)

포트 채우기
 구조화되지 않은 데이터 변환 [450](#)
포트 특성
 개요 [61](#)
 편집 [62](#)
폴의 최대 연결 수(속성)
 SQL 변환 [402](#)
프로그래밍 식별자(속성)
 외부 프로시저 변환 [157](#)
프로세스 변수
 초기화 속성에서 [181](#)
플랫 파일
 데이터 조인 [254](#)
 조회 [268](#)
플랫 파일 조회
 설명 [268](#)
 정렬된 입력 [269](#)
필터 변환
 개발 관련 팁 [189](#)
 개요 [187](#)
 성능 팁 [189](#)
 예제 [187](#)
 작성 [189](#)
 조건 [188](#)
하위 초
 Java 변환에서 처리 [223](#)
함수
 비집계 [51](#)
 사용자 지정 변환 API [89](#)
 집계 [50](#)
합집합 변환
 개요 [440](#)
 구성 요소 [441](#)
 그룹 [441](#)
 작성 [441](#)
 지침 [440](#)
 포트 [441](#)
행들
 사용자 지정 변환 [79](#)
행
 삭제 [459](#)
 업데이트 플래그 지정 [456](#)
행 기반 함수
 데이터 처리 [102](#)
 행 전략 [110](#)
행 내의 SQL 오류에 대해 계속(속성)
 SQL 변환 [402](#)
행 라우팅
 변환 [343](#)
행 변환 범위
 조이너 변환에서의 동작 [264](#)
행 전략 함수
 배열 기반 [117](#)
 행 기반 [110](#)

- 행 필터링
 - 변환 [187](#), [359](#)
 - 소스 한정자를 필터로 [189](#)
- 현재 값
 - 시퀀스 생성기 변환 [355](#)
- 현재 값(속성)
 - 시퀀스 생성기 변환 [353](#)
- 호출 텍스트
 - 저장 프로시저, 입력 [428](#)
- 호출 텍스트(속성)
 - 저장 프로시저 변환 [424](#)
- 확정 출력입니다(속성).
 - Java 변환 [215](#)
 - 외부 프로시저 변환 [157](#)
 - 저장 프로시저 변환 [424](#)
- 환경 변수
 - Java 패키지에 대한 설정 [221](#)
- 활성 변환
 - Java [211](#), [212](#)
 - XML 생성기 [462](#)
 - XML 소스 한정자 [461](#)
 - XML 파서 [461](#)
 - 개요 [25](#)
 - 노멀라이저 [321](#)
 - 라우터 [343](#)
 - 분류기 [359](#)
 - 사용자 지정 [57](#)
 - 소스 한정자 [363](#), [383](#)
 - 순위 [338](#)
 - 업데이트 전략 [455](#)
 - 조이너 [254](#)
 - 집계 [48](#)
 - 트랜잭션 제어 [434](#)
 - 필터 [187](#)
 - 합집합 [440](#)
- 활성 여부(속성)
 - Java 변환 [215](#)
- 효과적 트랜잭션 제어 변환
 - 정의 [436](#)

C

- C/C++
 - 통합 서비스에 연결 [176](#)
- CLASSPATH
 - Java 변환, 구성 [221](#)
- COBOL
 - VSAM 노멀라이저 변환 [325](#)
- COBOL 소스 정의
 - OCCURS 문 [325](#)
 - 노멀라이저 변환 작성 [329](#)
- COM 서버
 - COM 외부 프로시저의 유형 [159](#)
- COM 외부 프로시저
 - Informatica 외부 프로시저와 비교 [156](#)
 - Visual Basic에서 개발 [164](#)
 - Visual C++에서 개발 [159](#), [162](#)
 - 개발 참고 사항 [175](#)
 - 개요 [159](#)
 - 대상 작성 [163](#)
 - 데이터 유형 [175](#)
 - 디버깅 [177](#)
 - 리포지토리에 등록 [162](#)
 - 리포지토리에 추가 [162](#)
 - 메모리 관리 [176](#)
 - 반환 값 [176](#)

COM 외부 프로시저 (계속)

- 배포 [173](#)
- 서버 유형 [159](#)
- 소스 작성 [163](#)
- 연결되지 않음 [178](#)
- 예외 처리 [176](#)
- 작성 [159](#)
- 초기화 중 [179](#)
- 행 수준 프로시저 [175](#)
- company_names.dic
 - 데이터 마스킹 [146](#)
- CURRVAL 포트
 - 시퀀스 생성기 변환 [352](#)

D

- Data Transformation
 - 개요 [443](#)
 - 리포지토리 위치 [445](#)
- DB2
 - IBM DB2 참조 [391](#)
- defineJExpression
 - Java 식 API 메서드 [241](#)
- DLL(동적 연결 라이브러리)
 - 외부 프로시저 컴파일 [171](#)

E

- EDataType 클래스
 - Java 식 [240](#)
- ERROR 함수
 - 사용 [40](#)

F

- failSession 메서드
 - Java 변환 [228](#)
- firstnames.dic
 - 데이터 마스킹 [146](#)

G

- generateRow 메서드
 - Java 변환 [228](#)
- getBytes 메서드
 - Java 변환 [244](#)
- getDouble 메서드
 - Java 변환 [244](#)
- getInRowType 메서드
 - Java 변환 [229](#)
- getInt 메서드
 - Java 변환 [244](#)
- getLong 메서드
 - Java 변환 [244](#)
- getMetadata 메서드
 - Java 변환 [229](#)
- getResultDataType 메서드
 - Java 변환 [245](#)
- getResultMetadata 메서드
 - Java 변환 [245](#)
- getStringBuffer 메서드
 - Java 변환 [245](#)

H

HTTP 변환

- HTTP 탭 구성 [193](#)
- 그룹 [194](#)
- 그룹 및 포트 구성 [194](#)
- 기본 URL [197](#)
- 분할 가능 여부(속성) [192](#)
- 속성 구성 [193](#)
- 스레드별 코드 [192](#)
- 예 [199](#)
- 응답 코드 [191](#)
- 인증 [192](#)
- 작성 [192](#)
- 파티션당 단일 스레드 필요 속성 [192](#)

I

IBM DB2

- 연결 문자열 구문 [391](#)

IDispatch 인터페이스

- 클래스 정의 [159](#)

IDM_Dictionary

- 데이터 마스크 연결 [129](#)

IDM_Storage

- 데이터 마스크 연결 [129](#)

IIF 함수

- 시퀀스 생성기 변환으로 누락된 키 바꾸기 [352](#)

incrementErrorCount 메서드

- Java 변환 [230](#)

Informatica 외부 프로시저

- C++ 코드 생성 [168](#)
- COM과 비교 [156](#)
- 개발 [166](#)
- 개발 참고 사항 [175](#)
- 디버깅 [177](#)
- 메모리 관리 [176](#)
- 반환 값 [176](#)
- 배포 [174](#)
- 연결되지 않음 [178](#)
- 예외 처리 [176](#)
- 초기화 중 [179](#)
- 행 수준 프로시저 [175](#)

Informix

- 저장 프로시저 참고 사항 [421](#)

InputBuffer 포트

- 구조화되지 않은 데이터 변환 [448](#)

InputType

- 구조화되지 않은 데이터 변환 [447](#)

invoke

- Java 식 API 메서드 [245](#)

invokeJExpression

- API 메서드 [238](#)

isNull 메서드

- Java 변환 [231](#)

isResultNull 메서드

- Java 변환 [246](#)

J

Java 기본 데이터 유형

- Java 변환 [212](#)

Java 변환

- API 메서드 [226](#)
- CLASSPATH 설정 [221](#)

Java 변환 (계속)

- failSession 메서드 [228](#)
- generateRow 메서드 [228](#)
- getInRowType 메서드 [229](#)
- getMetadata 메서드 [229](#)
- incrementErrorCount 메서드 [230](#)
- isNull 메서드 [231](#)
- Java 기본 데이터 유형 [212](#)
- Java 코드 작성 [218](#)
- Java 코드 조각 작성 [218](#)
- Java 코드 탭 [213](#)
- logError 메서드 [231](#)
- logInfo 메서드 [232](#)
- null 값 검사 [231](#)
- null 값 설정 [233](#)
- setNull 메서드 [233](#)
- storeMetadata 메서드 [234](#)
- 개요 [211](#)
- 그룹 작성 [214](#)
- 기본 포트 값 [214](#)
- 데이터 끝에서 탭 [220](#)
- 데이터 유형 변환 [212](#)
- 도우미 코드 탭 [219](#)
- 디버깅 [224](#)
- 로그 [231](#)
- 메타데이터 검색 [229](#)
- 메타데이터 저장 [234](#)
- 변환 범위(속성) [215](#), [217](#)
- 변환 수준 클래스 경로 [221](#)
- 분할 가능 여부(속성) [215](#)
- 비사용자 코드 오류 [225](#)
- 사용자 코드 오류 [224](#)
- 세션 로그 [232](#)
- 세션 수준 클래스 경로 [222](#)
- 세션 실패 위치 [228](#)
- 속성 [215](#)
- 수동 [212](#)
- 언어(속성) [215](#)
- 업데이트 전략 설정 [217](#)
- 업데이트 전략(속성) [217](#)
- 예제 [247](#)
- 오류 찾기 [224](#)
- 입력 포트 [214](#)
- 입력 행 유형 가져오기 [229](#)
- 입력 행에서 탭 [220](#)
- 입력을 차단해야 함(속성) [215](#)
- 초 단위 이하 처리 [223](#)
- 추적 수준(속성) [215](#)
- 출력 포트 [214](#)
- 출력 행 유형 설정 [233](#)
- 출력이 정렬됨 속성 [215](#)
- 컴파일 [223](#)
- 컴파일 오류 [224](#)
- 컴파일 오류의 소스 식별 [224](#)
- 클래스 이름(속성) [215](#)
- 트랜잭션 생성 속성 [215](#)
- 트랜잭션 생성(속성) [217](#)
- 트랜잭션 수신 시 탭 [220](#)
- 트랜잭션 제어 [217](#)
- 파티션당 단일 스레드 필요 속성 [215](#)
- 패키지 가져오기 탭 [218](#)
- 포트 작성 [214](#)
- 플랫 파일 구문 분석 [221](#)
- 확정 출력입니다. 속성 [215](#)
- 활성 [212](#)
- Java 변환 API 메서드
- setOutRowType [233](#)

Java 변환 API 메서드 (계속)

로백 [232](#)

커밋 [227](#)

Java 식

EDataType 클래스 [240](#)

invokeJExpression API 메서드 [238](#)

Java 변환 [235](#)

Java 코드 생성 [237](#)

JExpression 클래스 [242](#), [243](#)

JExprParaMetadata 클래스 [241](#)

고급 인터페이스 [239](#)

고급 인터페이스 예제 [243](#)

고급 인터페이스를 사용하여 호출 [240](#)

구성 [236](#)

규칙 및 지침 [238](#), [240](#)

단순 인터페이스 [238](#)

단순 인터페이스 예제 [239](#)

단순 인터페이스를 사용하여 호출 [238](#)

변환 언어 함수 사용 [235](#)

사용자 정의 함수 사용 [235](#)

사용자 지정 함수 사용 [235](#)

생성 [236](#)

식 정의 대화 상자에서 작성 [237](#)

식 함수 유형 [235](#)

작성 [237](#)

함수 구성 [236](#)

Java 식 API 메서드

defineJExpression [241](#)

getBytes [244](#)

getDouble [244](#)

getInt [244](#)

getLong [244](#)

getResultDataType [245](#)

getResultMetadata [245](#)

getStringBuffer [245](#)

invoke [245](#)

isResultNull [246](#)

Java 코드 생성

Java 식 [237](#)

Java 코드 조각

Java 변환에 대해 작성 [218](#)

예제 [249](#)

Java 코드 탭

사용 [213](#)

Java 클래스 경로

세션 속성 [222](#)

Java 패키지

가져오기 [218](#)

JDK

Java 변환 [211](#)

JExpression 클래스

Java 식 [242](#), [243](#)

JExprParaMetadata 클래스

Java 식 [241](#)

JRE

Java 변환 [211](#)

L

logError 메서드

Java 변환 [231](#)

LogicalConnectionObject

SQL 변환 예제 [414](#)

설명 [391](#)

logInfo 메서드

Java 변환 [232](#)

M

MFC AppWizard

개요 [171](#)

Microsoft SQL Server

연결 문자열 구문 [391](#)

저장 프로시저 참고 사항 [421](#)

N

NaN

1.#QNAN으로 변환 [381](#)

NEXTVAL 포트

시퀀스 생성기 [350](#)

null 값

Java 변환에 대한 설정 [233](#)

Java 변환에서 검사 [231](#)

건너뛰기 [41](#)

분류기 변환 [362](#)

상수로 바꾸기 [40](#)

집계 함수 [51](#)

집계 함수를 사용하여 바꾸기 [53](#)

필터링 [189](#)

Null 무시(속성)

선택 [307](#)

NumRowsAffected 포트

SQL 변환 [397](#)

날짜 값

무작위 데이터 마스킹 [132](#)

날짜/시간 값

데이터 마스킹 [126](#)

소스 한정자 변환 [364](#)

노멀라이저 변환

VSAM 노멀라이저 [325](#)

VSAM 노멀라이저 변환 작성 [329](#)

개요 [321](#)

노멀라이저 탭 [323](#)

매핑 예제 [333](#)

문제 해결 [336](#)

발생 수 특성 [323](#)

생성된 열 ID [322](#)

생성된 키 [325](#)

속성 탭 [323](#)

수준 특성 [328](#), [332](#)

예제 [333](#)

키 유형 특성 [328](#)

파이프라인 노멀라이저 [330](#)

파이프라인 노멀라이저 변환 작성 [333](#)

파이프라인 노멀라이저 포트 탭 [331](#)

포트 탭 [322](#)

논리적 데이터베이스 연결

SQL 변환으로 전달 [391](#)

성능 고려 사항 [393](#)

누락된 값

시퀀스 생성기로 바꾸기 [352](#)

다중 그룹

변환 [31](#)

단순 인터페이스

Java 변환 API 메서드 [238](#)

Java 식 [238](#)

예제 [239](#)

단일 스레드 필요(속성)

사용자 지정 변환 [62](#)

대/소문자 구분(속성)

분류기 변환 [361](#)

- 대상
 - 업데이트 [455](#)
- \$Target
 - 조회 변환 [274](#)
 - 저장 프로시저 변환 [424](#)
- 대상 로드 순서
 - 소스 한정자 [364](#)
- 대상 테이블
 - 삽입 [459](#)
 - 업데이트 전략 설정 대상 [459](#)
 - 행 삭제 [459](#)
- 대체 마스크
 - 관계형 사전 구성 [128](#)
 - 데이터 마스크 규칙 및 지침 [129](#)
 - 마스크 속성 [127](#)
 - 설명 [126](#)
- 데이터
 - 고유 항목 선택 [379](#)
 - 미리 정렬 [54](#)
 - 업데이트 전략 변환을 통한 거부 [459](#)
 - 임시 저장 [36](#)
 - 조인 [254](#)
- 데이터 기반
 - 개요 [458](#)
- 데이터 끝에서 탭
 - Java 변환 [220](#)
- 데이터 마스크
 - 샘플 성 [146](#)
 - 샘플 이름 [146](#)
 - 샘플 조회 데이터 [146](#)
 - 샘플 주소 [146](#)
 - 샘플 회사 이름 [146](#)
 - 식 변환으로 매핑 [149](#)
 - 조회 변환을 포함하는 매핑 [146](#)
- 데이터 마스크 변환
 - IP 주소 마스크 [143](#)
 - URL 마스크 [143](#)
 - 고유한 출력 [144](#)
 - 공유 저장소 테이블 [144](#)
 - 관계형 사전 구성 [128](#)
 - 규칙 및 지침 [145](#)
 - 기본값 파일 [144](#)
 - 날짜 값 마스크 [135](#)
 - 대체 마스크 [126](#)
 - 대체 마스크 속성 [127, 128](#)
 - 대체 마스크에 대한 사전 [126](#)
 - 데이터 마스크 변환 [144](#)
 - 마스크 형식 [133](#)
 - 마스크 속성 [122](#)
 - 매핑 매개 변수 사용 [123](#)
 - 무작위 마스크 [131](#)
 - 반복 가능 SIN 번호 [143](#)
 - 반복 가능 SSN [140](#)
 - 반복 가능 종속 마스크 [131](#)
 - 반복 가능한 식 마스크 [136](#)
 - 범위 [135](#)
 - 블러링 [135](#)
 - 사전 이름 식 마스크 [137](#)
 - 사회 보장 번호 마스크 [139](#)
 - 사회 보험 번호 마스크 [143](#)
 - 세션 속성 [144](#)
 - 소스 문자열 문자 [134](#)
 - 식 마스크 [136](#)
 - 식 마스크 지침 [138](#)
 - 연결 요구 사항 [129](#)
 - 저장소 커밋 간격 [144](#)
 - 저장소 테이블 [127, 137](#)
- 데이터 마스크 변환 (계속)
 - 전자 메일 주소 마스크 [141](#)
 - 전화 번호 마스크 [141](#)
 - 종속 데이터 마스크 [129](#)
 - 캐시 디렉터리 [144](#)
 - 캐시 크기 [144](#)
 - 데이터 액세스 모드 함수
 - 사용자 지정 변환 함수 [90](#)
 - 데이터 유형
 - COM [175](#)
 - Java 변환 [212](#)
 - 변환 [175](#)
 - 소스 한정자 [364](#)
 - 데이터 유형 재바인딩 함수
 - 사용자 지정 변환 함수 [100](#)
 - 데이터 차단
 - 사용자 지정 변환 [66](#)
 - 사용자 지정 변환 함수 [108](#)
 - 조이너 변환 [263](#)
 - 데이터 처리 함수
 - 배열 기반 [115](#)
 - 행 기반 [102](#)
 - 데이터 코드 페이지 설정 함수
 - 사용자 지정 변환 함수 [110](#)
 - 데이터베이스
 - 다른 항목의 데이터 조인 [254](#)
 - 지원되는 옵션 [430](#)
 - 데이터베이스 복원력
 - SQL 변환 [394](#)
 - 데이터베이스 연결
 - SQL 변환 [390](#)
 - 데이터베이스에 연결
 - SQL 변환 [390](#)
 - 데이터베이스에서 다시 캐시
 - 개요 [294](#)
 - 명명되지 않은 캐시 [295](#)
 - 명명된 캐시 [295](#)
 - 도우미 코드 탭
 - Java 변환 [219](#)
 - 예제 [249](#)
 - 동시 캐시
 - 참조 캐시 [294](#)
 - 동적 SQL 쿼리
 - SQL 변환 [387](#)
 - SQL 변환 예제 [406](#)
 - 동적 서비스 이름
 - 구조화되지 않은 데이터 변환 [447](#)
 - 동적 연결
 - SQL 변환 [391](#)
 - SQL 변환 예제 [411](#)
 - 성능 고려 사항 [393](#)
 - 동적 조회 캐시
 - 대상과 동기화 [314](#)
 - 로드 거부 [314](#)
 - 설명 [304](#)
 - 오류 임계값 [314](#)
 - 조회 SQL 재정의 [308](#)
 - 플랫 파일 소스 사용 [304](#)
 - 동적 조회 캐시(속성)
 - 설명 [274](#)
 - 등록
 - COM 프로시저를 리포지토리에 [162](#)
 - 디버깅
 - Java 변환 [224](#)
 - 외부 프로시저 [177](#)
 - 디스패치 함수
 - 설명 [181](#)

- 따옴표 붙은 식별자
 - 예약어 [366](#)
- 라우터 변환
 - 개요 [343](#)
 - 그룹 [344](#)
 - 그룹 필터 조건 [345](#)
 - 노멀라이저 데이터 필터링 [333](#)
 - 매핑에서 연결 [347](#)
 - 예제 [345](#)
 - 작성 [347](#)
 - 포트 [347](#)
- 라이브러리
 - C++ 외부 프로시저용 [161](#)
 - Informatica 외부 프로시저용 [171](#)
 - VB 외부 프로시저용 [165](#)
- 래퍼 클래스
 - 기존 라이브러리 또는 함수용 [176](#)
- 런타임 위치(속성)
 - 외부 프로시저 변환 [157](#)
- 로그
 - Java 변환 [231](#)
- 로드 순서
 - 소스 한정자 [364](#)
- 로드 유형
 - 저장 프로시저 [428](#)
- 로컬 변수
 - 개요 [36](#)
- 롤백
 - Java 변환 API 에서드 [232](#)
- 롤백 행 생성
 - Java 변환 [232](#)
- 리포지토리
 - COM 외부 프로시저 [162](#)
 - COM 프로시저 등록 [162](#)
- 마법사
 - ATL COM AppWizard [159](#)
 - MFC AppWizard [171](#)
 - Visual Basic 응용 프로그램 설치 마법사 [174](#)
- 마스크 형식
 - 문자열 값 마스크 [133](#)
- 마스킹 규칙
 - 결과 문자열 대체 문자 [134](#)
 - 마스크 형식 [133](#)
 - 범위 [135](#)
 - 블러링 [135](#)
 - 소스 문자열 문자 [134](#)
 - 적용 [132](#)
- 마스킹 기술
 - 암호화 [138](#)
 - 형식 유지 암호화 [138](#)
- 마스킹 매개 변수
 - 데이터 마스킹 [123](#)
- 마스킹 속성 탭
 - 데이터 마스킹 변환 [122](#)
- 마스터 Null 순서(속성)
 - 조이너 변환 [255](#)
- 마스터 외부 조인
 - 설명 [258](#)
 - 트랜잭션 경계 유지 [264](#)
- 마스터 정렬 순서(속성)
 - 조이너 변환 [255](#)
- 마스터 행
 - 정렬되지 않은 조이너 변환에서 처리 [263](#)
 - 정렬된 조이너 변환 처리 [263](#)
 - 캐싱 [263](#)
- 많은 전체 자릿수
 - Java 변환에 대한 활성화 [223](#)

- 매개 변수 바인딩
 - SQL 변환 쿼리 [385](#)
- 매개 변수 액세스 함수
 - 64비트 [183](#)
 - 설명 [183](#)
- 매퍼
 - Data Transformation [445](#)
- 매핑
 - COBOL 소스 추가 [325](#)
 - 라우터 변환 사용 [347](#)
 - 변환 추가 [29](#)
 - 연결되지 않은 저장 프로시저 변환 구성 [426](#)
 - 연결된 저장 프로시저 변환 구성 [426](#)
 - 외부 프로시저 변환 사용 [163](#)
 - 재사용 가능 변환 수정 [47](#)
 - 재사용 가능 변환 추가 [46](#)
 - 저장 프로시저의 영향을 받음 [418](#)
 - 조회 구성 요소 [272](#)
 - 행에 업데이트 플래그 지정 [456](#)
- 매핑 디자이너
 - 재사용 가능 변환 추가 [46](#)
- 매핑 매개 변수
 - 소스 한정자 변환에서 [365](#)
 - 조회 SQL 재정의 [280](#)
- 매핑 변수
 - 소스 한정자 변환에서 [365](#)
 - 재사용 가능 변환 [45](#)
 - 조회 SQL 재정의 [280](#)
- 메모리 관리
 - 외부 프로시저용 [176](#)
- 메서드
 - Java 변환 [220](#)
 - Java 변환 API [226](#)
- 메타데이터 확장
 - 사용자 지정 변환 [68](#)
- 명명되지 않은 캐시
 - 공유 [297](#)
 - 데이터베이스에서 다시 캐시 [295](#)
 - 지속형 [295](#)
- 명명된 지속형 조회 캐시
 - 개요 [297](#)
 - 공유 [299](#)
- 명명된 캐시
 - 공유 [299](#)
 - 데이터베이스에서 다시 캐시 [295](#)
 - 지속형 [295](#)
- 모듈(속성)
 - 외부 프로시저 변환 [157](#)
- 모든 입력 변환 범위
 - 조이너 변환에서의 동작 [264](#)
- 무작위 마스킹
 - 날짜 값 마스킹 [132](#)
 - 문자열 값 마스킹 [131](#)
 - 숫자 값 [131](#)
- 문자열
 - 순위 [339](#)
- 문자열 값
 - 사용자 지정 데이터 마스킹 [131](#)
 - 키 데이터 마스킹 [124](#)
- 문자열 대체
 - SQL 변환 쿼리 [387](#)
- 문제 해결
 - Java 변환 [224](#)
 - 노멀라이저 변환 [336](#)
 - 소스 한정자 변환 [381](#)
 - 저장 프로시저 변환 [432](#)
 - 집계 변환 [56](#)

- 반복 가능 종속 마스크
 - 설명 [131](#)
- 반복 가능 출력
 - 데이터 마스크 변환 [123](#)
- 반환 값
 - 외부 프로시저에서 [176](#)
 - 저장 프로시저 변환 [417](#)
 - 조회 변환 [287](#)
- 반환 포트
 - 조회 변환 [273](#), [287](#)
- 발생 수 특성
 - 노멀라이저 변환 [323](#)
- 배열 기반 함수
 - 개요 [112](#)
 - 데이터 처리 [115](#)
 - 올바른 행임 [114](#)
 - 입력 오류 행 설정 [118](#)
 - 최대 행 수 [112](#)
 - 행 수 [114](#)
 - 행 전략 [117](#)
- 배정밀도 데이터 유형
 - Java 변환 [223](#)
- 배포
 - 사용자 지정 변환 프로시저 [59](#)
 - 외부 프로시저 [173](#)
- 범위
 - 숫자 값 마스크 [135](#)
- 변수
 - Java 변환 [219](#), [220](#)
 - 개요 [36](#)
 - 저장 프로시저 결과, 램처 [37](#)
 - 초기화 [38](#)
 - 포트 평가 순서 [37](#)
- 변수 포트
 - 개요 [36](#)
- 변환
 - Java [211](#)
 - SQL [383](#)
 - XML 생성기 [462](#)
 - XML 소스 한정자 [461](#)
 - XML 파서 [461](#)
 - 개요 [25](#)
 - 노멀라이저 [321](#)
 - 다중 그룹 [31](#)
 - 라우터 [343](#)
 - 매핑에 추가 [29](#)
 - 사용자 지정 [57](#)
 - 설명 [26](#)
 - 소스 한정자 [363](#)
 - 순위 [338](#)
 - 시퀀스 생성기 [349](#)
 - 식 [152](#)
 - 식에 허용되는 유형 [31](#)
 - 업데이트 전략 [455](#)
 - 연결 끊김 [26](#)
 - 연결됨 [26](#)
 - 오류 처리 [42](#)
 - 원시 및 비원시 [26](#)
 - 유형 [25](#)
 - 작성 [29](#)
 - 재사용 가능 변환 [44](#)
 - 재사용 가능으로 승격 [46](#)
 - 재사용 가능하게 만들기 [46](#)
 - 저장 프로시저 [416](#)
 - 정의 [25](#)
 - 조이너 [254](#)
 - 조회 [267](#)

- 변환 (계속)
 - 집계 [48](#)
 - 추적 수준 [44](#)
 - 필터 [187](#)
 - 합집합 [440](#)
 - 활성 및 수동 [25](#)
- 변환 개발자
 - 재사용 가능 변환 [44](#)
- 변환 교환(TX)
 - 정의 [155](#)
- 변환 범위
 - Java 변환 [217](#)
 - 모든 입력 변환 범위와 조이너 변환 [264](#)
 - 분류기 변환 [362](#)
 - 사용자 지정 변환 [65](#)
 - 조이너 변환 속성 [255](#)
 - 조이너 변환 정의 [264](#)
 - 트랜잭션 변환 범위와 조이너 변환 [264](#)
 - 행 변환 범위와 조이너 변환 [264](#)
- 변환 범위(속성)
 - Java 변환 [215](#)
- 변환 언어
 - Java 식 사용 [235](#)
 - 집계 함수 [50](#)
- 변환기 구성 요소
 - Data Transformation [445](#)
- 복원력
 - SQL 변환 데이터베이스 [396](#)
- 복합 키
 - 시퀀스 생성기 변환을 사용하여 작성 [351](#)
- 부분 쿼리
 - 대체 [388](#)
- 분류기 변환
 - 개요 [359](#)
 - 구성 [360](#)
 - 분류기 캐시 크기 구성 [361](#)
 - 속성 [360](#)
 - 작성 [362](#)
 - 작업 디렉터리 [361](#)
 - 조이너 변환과 함께 사용 [259](#)
- 분할 가능 여부(속성)
 - HTTP 변환 [192](#)
 - Java 변환 [215](#)
 - 사용자 지정 변환 [62](#)
 - 외부 프로시저 변환 [157](#)
- 분할된 파이프라인
 - 정렬된 데이터 조인 [259](#)
- 블러링
 - 날짜 값 [136](#)
 - 숫자 값 [135](#)
- 비교 시 무시(속성)
 - 설명 [308](#)
- 비사용자 코드 오류
 - Java 변환 [225](#)
- 비원시 변환
 - 개요 [26](#)
- 비집계 식
 - 개요 [53](#)
- 비집계 함수
 - 예제 [51](#)
- 비효과적 트랜잭션 제어 변환
 - 정의 [436](#)
- 사용자 정의 그룹
 - 라우터 변환 [344](#)
- 사용자 정의 메서드
 - Java 변환 [219](#), [220](#)

사용자 정의 조인

입력 [370](#)

사용자 정의 함수

Java 식 사용 [235](#)

사용자 지정 변환

개요 [57](#)

구성 요소 [60](#)

규칙 및 지침 [59](#)

그룹 작성 [60](#)

단일 스레드 필요 속성 [62](#)

데이터 차단 [66](#)

메타데이터 확장 [68](#)

모듈 작성 [76](#)

배포 [59](#)

변환 범위 속성 [65](#)

분할 가능 여부(속성) [62](#)

속성 [62](#)

속성 ID [94](#)

스레드 [64](#)

스레드별 코드 [62](#)

업데이트 전략 설정 [64](#)

업데이트 전략 속성 [64](#)

입력이 차단할 수 있음 속성 [66](#)

작성 [59, 69](#)

초기화 속성 [68](#)

코드 파일 생성 [59, 71](#)

코드 페이지 [58](#)

트랜잭션 경계 [65](#)

트랜잭션 생성 속성 [65](#)

트랜잭션 제어 [64](#)

포트 관계 정의 [61](#)

포트 작성 [60](#)

포트 특성 [61](#)

프로시저 속성 [68](#)

프로시저 작성 [68](#)

프로시저 컴파일 [76](#)

프로시저에 행 전달 [83](#)

함수 [79](#)

사용자 지정 변환 프로시저

스레드별 [62](#)

예 [71](#)

작성 [68](#)

코드 파일 생성 [71](#)

행 작업 [83](#)

사용자 지정 변환 함수

API [89](#)

기본 행 전략 변경 [111](#)

데이터 경계 출력 [105](#)

데이터 액세스 모드 설정 [90](#)

데이터 유형 재바인딩 [100](#)

데이터 처리(배열 기반) [115](#)

데이터 처리(행 기반) [102](#)

데이터 코드 페이지 설정 [110](#)

문자열 모드 변경 [109](#)

배열 기반 [112](#)

생성됨 [84](#)

세션 로그 [106](#)

속성 [93](#)

알림 [86](#)

오류 [106](#)

오류 개수 증분 [107](#)

올바른 행임 [114](#)

입력 오류 행 설정 [118](#)

종료됨 [107](#)

차단 논리 [108](#)

초기화 [85](#)

초기화 취소 [88](#)

사용자 지정 변환 함수 (계속)

최대 행 수 [112](#)

출력 알림 [105](#)

탐색 [91](#)

통과 포트 설정 [104](#)

포인터 [109](#)

행들 작업 [79, 91](#)

행 수 [114](#)

행 전략(배열 기반) [117](#)

행 전략(행 기반) [110](#)

사용자 지정 함수

Java 식 사용 [235](#)

사용자 코드 오류

Java 변환 [224](#)

사전

대체 데이터 마스킹 [126](#)

반복 가능한 식 마스킹 [137](#)

사전 빌드 조회 캐시

조회 속성 [274](#)

사전 세션

오류 [430](#)

저장 프로시저 [428](#)

사전 정보

데이터 마스킹 변환 [128](#)

사회 보장 번호

반복 가능 데이터 마스킹 [140](#)

지역 번호 마스킹 [140](#)

사후 세션

오류 [430](#)

저장 프로시저 [428](#)

삼입 기타 항목 업데이트(속성)

설명 [312](#)

삼입으로 업데이트(속성)

설명 [459](#)

상세 Null 순서(속성)

조이너 변환 [255](#)

상수

null 값을 바꾸기 [40](#)

상태 추적 수준

구조화되지 않은 데이터 변환 [448](#)

상태 코드

저장 프로시저 변환 [417](#)

생성

포트 [30](#)

생성된 열 ID

노멀라이저 변환 [322](#)

생성된 키

노멀라이저 변환 [325](#)

생성된 함수

사용자 지정 변환 [84](#)

설명

식에 추가 [33](#)

성능

논리적 데이터베이스 연결 [393](#)

변수를 사용하여 성능 개선 [36](#)

정적 데이터베이스 연결 [393](#)

조이너 변환 [266](#)

조회 변환 [290](#)

집계 변환 [53](#)

필터 항상 [189](#)

세부 외부 조인

설명 [258](#)

세부 정보 행

정렬되지 않은 조이너 변환에서 처리 [263](#)

정렬된 조이너 변환 처리 [263](#)

조이너 변환에서 차단 [263](#)

세션

- \$\$\$SessStartTime [365](#)
- 고유 항목 선택 재정의 [380](#)
- 사전 저장 프로시저 및 사후 저장 프로시저, 실행 [428](#)
- 업데이트 전략 설정 [458](#)
- 외부 프로시저 변환 [164](#)
- 저장 프로시저 변환 [418](#)
- 저장 프로시저 오류 처리를 위한 구성 [430](#)
- 증분 집계 [48](#)

세션 로그

- Java 변환 [232](#)

세션 실패

- Java 변환 [228](#)

세션 이전 및 세션 이후 SQL

- 소스 한정자 변환 [380](#)

\$Source

- 조회 변환 [274](#)
- 저장 프로시저 변환 [424](#)

소스

- 동일한 소스의 데이터 조인 [261](#)
- 병합 [440](#)
- 여러 항목 조인 [254](#)
- 조인 [254](#)

소스 문자열 문자

- 데이터 마스킹 변환 [134](#)

소스 분석기

- 키 관계 작성 [369](#)

소스 필터

- 소스 한정자에 추가 [378](#)

소스 한정자 변환

- \$\$\$SessStartTime [365](#)
 - SQL 재정의 [369](#)
 - XML 소스 한정자 [461](#)
 - 개요 [363](#)
 - 고유 항목 선택 옵션 [379](#)
 - 구성 [381](#)
 - 기본 조인 [367](#)
 - 기본 쿼리 [366](#)
 - 기본 쿼리 보기 [367](#)
 - 기본 쿼리 재정의 [367](#), [369](#)
 - 대상 로드 순서 [364](#)
 - 데이터 유형 [364](#)
 - 매핑 매개 변수 및 변수 [365](#)
 - 문제 해결 [381](#)
 - 사용자 정의 조인 입력 [370](#)
 - 사용자 지정 조인 [368](#)
 - 세션 이전 및 세션 이후 SQL [380](#)
 - 소스 데이터 조인 [367](#)
 - 소스 필터 입력 [378](#)
 - 속성 [381](#)
 - 외부 조인 지원 [371](#)
 - 정렬 순서와 집계 [54](#)
 - 정렬된 포트 수 옵션 [379](#)
 - 조인 [369](#)
 - 조회 소스로 사용 [269](#)
 - 키 관계 작성 [369](#)
- ## 소스 행 필터링
- 조회 변환 [282](#), [283](#)
- ## 소스 행을 다음으로 처리
- 업데이트 전략 [458](#)
- ## 속성 ID
- 사용자 지정 변환 [94](#)
- ## 속성 액세스 함수
- 설명 [182](#)
- ## 속성 함수
- 사용자 지정 변환 [93](#)

수동 모드

- SQL 변환 [390](#)

수동 변환

- Java [211](#), [212](#)
- SQL 변환 구성 [389](#)
- 개요 [26](#)
- 시퀀스 생성기 [349](#)
- 식 [152](#)
- 저장 프로시저 [416](#)
- 조회 [267](#)

수준 특성

- VSAM 노멀라이저 변환 [328](#)
- 파이프라인 노멀라이저 변환 [332](#)

순위

- 데이터 그룹 [340](#)
- 문자열 값 [339](#)

순위 변환

- RANKINDEX 포트 [340](#)
- 개요 [338](#)
- 그룹 정의 대상 [340](#)
- 변수 사용 [36](#)
- 옵션 [339](#)
- 작성 [341](#)
- 포트 [339](#)

순차 캐시

- 참조 캐시 [294](#)
- 캐시 참조 [294](#)

숫자 값

- 무작위 마스킹 [131](#)
- 키 마스킹 [125](#)

스레드

- 사용자 지정 변환 [64](#)

스레드별 작업

- HTTP 변환 [192](#)
- 사용자 지정 변환 [62](#)
- 쓰기 [64](#)

스크립트 로컬 옵션

- SQL 변환 [402](#)

스크립트 모드

- SQL 변환 [384](#)
- 규칙 및 지침 [385](#)

스트리머 구성 요소

- Data Transformation [445](#)

스트리머 청크 크기

- 구조화되지 않은 데이터 변환 [447](#)

승격

- 재사용 불가능 변환 [46](#)

시드 값

- 데이터 마스킹 변환 [123](#)

시작 값

- 시퀀스 생성기 변환 [354](#)

시작 값(속성)

- 시퀀스 생성기 변환 [353](#)

시작 자릿수

- 사회 보험 번호 [143](#)

시퀀스 ID

- 조회 변환 [306](#)

시퀀스 생성기 변환

- Blaze 엔진 [358](#)
- CURRVAL 포트 [352](#)
- IIF 함수를 사용하여 누락된 키 바꾸기 [352](#)
- NEXTVAL 포트 [350](#)
- Spark 엔진 [358](#)
- 값 범위 [355](#)
- 개요 [349](#)
- 고급 속성 [357](#)
- 복합 키 작성 [351](#)

시퀀스 생성기 변환 (계속)

비원시 환경 [358](#)
속성 [353](#), [357](#)
시작 값 [354](#)
작성 [357](#)
재사용 가능 [356](#)
재사용 불가능 [356](#)
재설정 [357](#)
주기 [354](#), [355](#)
증분 범위 속성 [354](#)
총 개시된 값 [355](#)
포트 [349](#)
현재 값 [355](#)

식

Java 변환 [235](#)
다음에서 저장 프로시저 호출 [426](#)
단순화 [36](#)
반환 값 [31](#)
비집계 [53](#)
업데이트 전략 [456](#)
유효성 검사 [33](#)
입력 [31](#), [33](#)
저장 프로시저 변환에 대한 규칙 [431](#)
조회 호출 [288](#)
집계 변환 [50](#)
필터 조건 [188](#)

식 마스킹

규칙 및 지침 [138](#)
반복 가능 마스킹 [136](#)
반복 가능한 마스킹 예 [137](#)
설명 [136](#)

식 변환

개요 [152](#)
데이터 라우팅 [153](#)
변수 사용 [36](#)
작성 [153](#)

식 편집기

Java 식 사용 [237](#)
개요 [33](#)
구문 색상 [33](#)
다음을 사용하여 식 유효성 검사 [33](#)

실시간 데이터

조이너 변환 사용 [264](#)
조이너 변환을 사용하여 처리 [265](#)

실행 순서(속성)

저장 프로시저 변환 [424](#)

알림 함수

사용자 지정 변환 [86](#)

언어(속성)

Java 변환 [215](#)
업데이트 기타 항목 삽입(속성)
설명 [312](#), [459](#)
업데이트 시 이전 값 출력(속성)
설명 [274](#)

업데이트 전략

Java 변환을 사용하여 설정 [217](#)
사용자 지정 변환을 사용하여 설정 [64](#)
행 전략 함수(행 기반) [110](#)

업데이트 전략 변환

개요 [455](#)
거부된 행 전달 [456](#)
검사 목록 [459](#)
구성 단계 [455](#)
세션에 대한 옵션 설정 [458](#), [459](#)
식 입력 [456](#)
작성 [456](#)
조회 결합 [457](#)

업데이트 전략 변환 (계속)

집계 조합 [457](#)
업데이트 전략 변환(속성)
Java 변환 [215](#)
업데이트로 업데이트(속성)
설명 [459](#)
여러 개의 일치 항목
조회 변환 [284](#)
조회 정책 속성 [274](#)
여러 행 반환
조회 변환 [286](#)
연결
SQL 변환 [390](#)
연결 문자열 예 [391](#)
연결 개체
저장 프로시저 변환에서 구성 [424](#)
조회 변환에서 구성 [274](#)
연결 문자열
구문 [391](#)
연결 변수
저장 프로시저 변환에서 사용 [424](#)
조회 변환에서 사용 [274](#)
연결 설정
SQL 변환 [391](#)
연결 정보(속성)
저장 프로시저 변환 [424](#)
조회 변환 [274](#)
연결 풀링 사용(속성)
SQL 변환 [402](#)
연결되지 않은 변환
외부 프로시저 변환 [178](#)
저장 프로시저 변환 [416](#)
조회 [267](#)
조회 변환 [272](#), [286](#)
연결되지 않은 조회
개요 [272](#), [286](#)
반환 값 지정 [287](#)
설명 [271](#)
식을 통한 호출 [288](#)
조회 조건 추가 [287](#)
연결되지 않은 조회 변환
반환 포트 [287](#)
입력 포트 [287](#)
연결된 변환
Java [211](#)
SQL [383](#)
XML 생성기 [462](#)
XML 소스 한정자 [461](#)
XML 파서 [461](#)
노멀라이저 [321](#)
라우터 [343](#)
사용자 지정 [57](#)
소스 한정자 [363](#)
순위 [338](#)
시퀀스 생성기 [349](#)
식 [152](#)
업데이트 전략 [455](#)
저장 프로시저 [416](#)
조이너 [254](#)
조회 [267](#)
집계 [48](#)
필터 [187](#)
합집합 변환 [440](#)
연결된 조회
개요 [271](#)
설명 [271](#)
작성 [289](#)

- 연결된 포트
 - 시퀀스 ID [306](#)
 - 조회 변환 [306](#)
- 연산자
 - 조회 조건 [284](#)
- 예약어
 - resword.txt [366](#)
 - SQL 생성 [366](#)
 - 조회 쿼리 [281](#)
- 예외
 - 외부 프로시저에서 [176](#)
- 오류
 - Java 변환의 임계값 증가 [230](#)
 - 대상:동적 조회 캐시 [314](#)
 - 식 편집기에서 유효성 검사 [33](#)
 - 처리 [42](#)
- 오류 메시지
 - 외부 프로시저에 대한 추적 [177](#)
 - 외부 프로시저용 [177](#)
- 오류 수
 - Java 변환에 대해 증분 [230](#)
- 오류 처리
 - 대상:동적 조회 캐시 [314](#)
 - 저장 프로시저용 [430](#)
- 오류 행
 - SQL 변환 [398](#)
- 오른쪽 외부 조인
 - 구문 [375](#)
 - 작성 [375](#)
- 외래 키
 - 시퀀스 생성기 변환을 사용하여 작성 [351](#)
- 외부 조인
 - 작성 [376](#)
 - 조인 재정의로 작성 [376](#)
 - 참조 조인 유형[외부 조인 aab] [376](#)
 - 추출 재정의로 작성 [377](#)
 - 통합 서비스에서 지원하는 유형 [371](#)
- 외부 프로시저
 - Informatica 외부 프로시저 배포 [174](#)
 - 개발 참고 사항 [175](#)
 - 디버깅 [177](#)
 - 배포 [173](#)
 - 인터페이스 함수 [181](#)
- 외부 프로시저 변환
 - 64비트[외부 프로시저 변환 64] [183](#)
 - ATL 개체 [160](#)
 - BankSoft 예제 [157](#)
 - BankSoft 예제를 사용하는 Informatica 외부 프로시저 [166](#)
 - C++ 외부 프로시저용 라이브러리 작성 [161](#)
 - COM 데이터 유형 [175](#)
 - COM 및 Informatica 유형 [156](#)
 - COM 외부 프로시저 [159](#)
 - IDispatch 인터페이스 [159](#)
 - Informatica 외부 프로시저 [166](#)
 - Informatica 외부 프로시저용 라이브러리 작성 [171](#)
 - MFC AppWizard [171](#)
 - Visual Basic [164](#)
 - Visual Basic 외부 프로시저용 라이브러리 작성 [165](#)
 - Visual C++ [159](#)
 - 개발 참고 사항 [175](#)
 - 개요 [155](#)
 - 다중 스레드 코드 [155](#)
 - 디버깅 [177](#)
 - 디스패치 함수 [181](#)
 - 디자인너에서 작성 [166](#)
- 외부 프로시저 변환 (계속)
 - 래퍼 클래스 [176](#)
 - 런타임 위치(속성) [157](#)
 - 매개 변수 액세스 함수 [183](#)
 - 매핑에서 사용 [163](#)
 - 메모리 관리 [176](#)
 - 멤버 변수 [183](#)
 - 모듈(속성) [157](#)
 - 반환 값 [176](#)
 - 분할 가능 여부(속성) [157](#)
 - 설명 [156](#)
 - 세션 [164](#)
 - 속성 [156, 157](#)
 - 속성 액세스 함수 [182](#)
 - 연결되지 않음 [178](#)
 - 예외 처리 [176](#)
 - 외부 프로시저 함수 [182](#)
 - 인터페이스 함수 [181](#)
 - 초기화 중 [179](#)
 - 추적 수준 함수 [186](#)
 - 추적 수준(속성) [157](#)
 - 출력은 반복 가능합니다(속성). [157](#)
 - 코드 페이지 액세스 함수 [185](#)
 - 파티션 관련 함수 [186](#)
 - 프로그래밍 식별자(속성) [157](#)
 - 프로세스 변수 지원 [181](#)
 - 필요한 파일 [180](#)
 - 행 수준 프로시저 [175](#)
 - 확정 출력입니다(속성). [157](#)
- 왼쪽 외부 조인
 - 구문 [373](#)
 - 작성 [373](#)
- 원시 데이터 유형
 - SQL 변환 [386](#)
- 원시 변환
 - 개요 [26](#)
- 웹 링크
 - 식에 추가 [33](#)
- 유니코드 모드
 - 사용자 지정 변환 [58](#)
 - 외부 프로시저 변환 [155](#)
 - 조이너 변환에 대해 정렬 순서 구성 [259](#)
- 유효성 검사
 - 기본값 [44](#)
 - 식 [33](#)
- 유효성 검사 단추
 - 변환 [44](#)
- 인덱스
 - 조회 조건 [290](#)
 - 조회 테이블 [273, 290](#)
- 인스턴스
 - 재사용 가능 변환 작성 [45](#)
- 인스턴스 변수
 - Java 변환 [219, 220](#)
- 일반 조인
 - 구문 [372](#)
 - 작성 [372](#)
 - 정의 [257](#)
 - 트랜잭션 경계 유지 [264](#)
- 일반 추적 수준
 - 개요 [44](#)
- 일치 항목이 여러 개인 경우의 조회 정책(속성)
 - 설명 [274](#)
- 입력
 - SQL 쿼리 재정의 [369](#)
 - 사용자 정의 조인 [370](#)
 - 소스 필터 [378](#)

입력 (계속)

식 [31](#), [33](#)

입력 계층 탭

구조화되지 않은 데이터 변환 [451](#)

입력 매개 변수

저장 프로시저 [417](#)

입력 포트

Java 변환 [214](#)

개요 [30](#)

기본값 [38](#)

변수로 사용 [220](#)

입력 행

행 유형 가져오기 [229](#)

입력 행에서 탭

Java 변환 [220](#)

예제 [250](#)

입력/출력 포트

개요 [30](#)

입력을 차단해야 함(속성)

Java 변환 [215](#)

자동 커밋

SQL 변환으로 구성 [394](#)

설명 [402](#)

자세한 정보 표시 데이터 추적 수준

SQL 변환 [400](#)

개요 [44](#)

자세한 정보 표시 초기화 추적 수준

개요 [44](#)

작성

COM 외부 프로시저 [159](#)

Informatica 외부 프로시저 [166](#)

라우터 변환 [347](#)

변환 [29](#)

사용자 지정 변환 [59](#), [69](#)

순위 변환 [341](#)

시퀀스 생성기 변환 [357](#)

식 변환 [153](#)

업데이트 전략 변환 [456](#)

재사용 가능 변환 [45](#)

재사용 가능 변환의 재사용 불가능 인스턴스 [46](#)

저장 프로시저 변환 [422](#)

조이너 변환 [265](#)

집계 변환 [55](#)

필터 변환 [189](#)

합집합 변환 [441](#)

작업 디렉터리

분류기 변환, 지정 [361](#)

재사용 가능 변환

개요 [44](#)

매핑 변수 [45](#)

매핑에 추가 [46](#)

변경 [47](#)

작성 [45](#)

재사용 불가능 인스턴스 작성 [46](#)

재설정

시퀀스 생성기 변환 [357](#)

재설정(속성)

시퀀스 생성기 변환 [357](#)

재정의

기본 소스 한정자 SQL 쿼리 [369](#)

저장 프로시저

IBM DB2 예제 [421](#)

Informix 예제 [421](#)

Microsoft 예제 [421](#)

Oracle 예제 [421](#)

Sybase 예제 [421](#)

Teradata 예제 [422](#)

저장 프로시저 (계속)

가져오기 [422](#)

데이터베이스별 구문 참고 사항 [421](#)

로드 유형 [428](#)

매개 변수 변경 [425](#)

변수에 기록 [37](#)

사전 세션 또는 사후 세션 실행에 대한 세션 작성 [428](#)

사전 세션 오류 [430](#)

사후 세션 오류 [430](#)

세션 오류 [430](#)

쓰기 [420](#)

오류 처리 [430](#)

유형 설정 [424](#)

정의 [416](#)

지원되는 데이터베이스 [430](#)

처리 순서, 지정 [418](#)

저장 프로시저 변환

가져와서 작성 [422](#)

개요 [416](#)

구성 [419](#)

날짜/시간 포트에 대한 초 단위 이하 정밀도 [424](#)

문제 해결 [432](#)

반환 값 [417](#)

사전 세션 또는 사후 세션 실행 [428](#)

사전 세션 및 사후 세션 [428](#)

상태 코드 [417](#)

설정 옵션 [424](#)

성능 팁 [432](#)

세션 런타임, 지정 [418](#)

속성 [424](#)

수동으로 작성 [423](#), [424](#)

수정 [425](#)

식 규칙 [431](#)

실행 순서(속성) [424](#)

실행 시에 지정 [418](#)

연결 정보(속성) [424](#)

연결되지 않은 저장 프로시저 구성 [426](#)

연결되지 않음 [418](#), [426](#)

연결된 저장 프로시저 구성 [426](#)

연결됨 [418](#)

입력 데이터 [417](#)

입력/출력 매개 변수 [417](#)

저장 프로시저 가져오기 [422](#)

저장 프로시저 유형(속성) [424](#)

추적 수준(속성) [424](#)

출력 데이터 [417](#)

출력은 반복 가능합니다(속성). [424](#)

호출 텍스트(속성) [424](#)

확정 출력입니다(속성). [424](#)

저장 프로시저 유형(속성)

저장 프로시저 변환 [424](#)

저장소 암호화

데이터 마스킹 변환 [144](#)

저장소 암호화 키

데이터 마스킹 변환 [144](#)

저장소 커밋 간격

데이터 마스킹 변환 [144](#)

저장소 테이블

대체 데이터 마스킹 [127](#)

식 마스킹 [137](#)

전체 데이터베이스 연결

SQL 변환으로 전달 [391](#)

전체 외부 조인

정의 [259](#)

전체 코드 창

Java 컴파일 오류 [224](#)

- 정렬 순서
 - 소스 한정자 변환 [379](#)
 - 조이너 변환에 대한 구성 [259](#)
 - 집계 변환 [54](#)
- 정렬 원본
 - 사용할 조인 조건 구성 [259](#)
 - 정의 [259](#)
- 정렬되지 않은 조이너 변환
 - 마스터 행 처리 [263](#)
 - 세부 행 처리 [263](#)
- 정렬된 관계형 데이터
 - 조이너 변환에서 사용 [259](#)
- 정렬된 데이터
 - 분할된 파이프라인에서 조인 [259](#)
 - 조이너 변환에서 사용 [259](#)
- 정렬된 데이터 조인
 - 분류기 변환 사용 [259](#)
 - 정렬된 관계형 데이터 사용 [259](#)
 - 정렬된 플랫 파일 사용 [259](#)
 - 조인 성능을 최적화하기 위해 구성 [259](#)
- 정렬된 입력
 - 플랫 파일 조회 [269](#)
- 정렬된 입력(속성)
 - 조이너 변환 [255](#)
 - 집계 변환 [53](#)
- 정렬된 포트
 - 데이터 미리 정렬 [54](#)
 - 사용하지 않아야 하는 이유 [54](#)
 - 소스 한정자 [379](#)
 - 정렬 순서 [379](#)
 - 집계 변환 [53](#)
 - 캐싱 요구 사항 [50](#)
- 정렬된 플랫 파일
 - 조이너 변환에서 사용 [259](#)
- 정의
 - 사용자 지정 변환의 포트 종속성 [61](#)
- 정적 SQL 쿼리
 - SQL 변환 [385](#)
 - 포트 구성 [386](#)
- 정적 데이터베이스 연결
 - 설명 [390](#)
 - 성능 고려 사항 [393](#)
- 정적 변수
 - Java 변환 [219](#)
- 정적 조회 캐시
 - 개요 [297](#)
- 정적 코드
 - Java 변환 [219](#)
- 정확히 한 번 배달
 - SQL 변환 [395](#)
- 조건
 - 라우터 변환 [345](#)
 - 조이너 변환 [256](#)
 - 조회 변환 [283](#), [287](#)
 - 필터 변환 [188](#)
- 조이너 데이터 캐시 크기
 - 조이너 변환 속성 [255](#)
- 조이너 변환
 - ASCII 모드에서의 정렬 순서 구성 [259](#)
 - 개요 [254](#)
 - 동일한 소스의 데이터 조인 [261](#)
 - 모든 입력 변환 범위 [264](#)
 - 모든 입력 변환 범위를 사용할 경우의 동작 [264](#)
 - 변환 범위 [264](#)
 - 분류기 변환과 함께 사용 [259](#)
 - 성능 팁 [266](#)
 - 소스 데이터 차단 [263](#)

- 조이너 변환 (계속)
 - 속성 [255](#)
 - 실시간 데이터 [264](#)
 - 실시간 데이터 처리 [265](#)
 - 여러 데이터베이스 조인 [254](#)
 - 유니코드 모드에서의 정렬 순서 구성 [259](#)
 - 입력에 대한 규칙 [254](#)
 - 작성 [265](#)
 - 정렬 순서 구성 [259](#)
 - 정렬 원본 포트를 사용하도록 조인 조건 구성 [259](#)
 - 조건 [256](#)
 - 조인 유형 [257](#)
 - 캐시 [263](#)
 - 트랜잭션 [264](#)
 - 트랜잭션 경계 삭제 [265](#)
 - 트랜잭션 경계 유지 [264](#)
 - 트랜잭션 변환 범위 [264](#)
 - 트랜잭션 변환 범위를 사용할 경우의 동작 [264](#)
 - 행 변환 범위 [264](#)
 - 행 변환 범위를 사용할 경우의 동작 [264](#)
- 조이너 인덱스 캐시 크기
 - 조이너 변환 속성 [255](#)
- 조이너 캐시
 - 조이너 변환 [263](#)
- 조인
 - Informatica 구문 [371](#)
 - 사용자 정의 [370](#)
 - 사용자 지정 [368](#)
 - 소스 한정자의 기본값 [367](#)
 - 키 관계 작성 [369](#)
- 조인 구문
 - 오른쪽 외부 조인 [375](#)
 - 왼쪽 외부 조인 [373](#)
 - 일반 조인 [372](#)
- 조인 유형
 - 마스터 외부 조인 [258](#)
 - 세부 외부 조인 [258](#)
 - 소스 한정자 변환 [371](#)
 - 오른쪽 외부 조인 [371](#)
 - 왼쪽 외부 조인 [371](#)
 - 일반 조인 [257](#)
 - 전체 외부 조인 [259](#)
 - 조이너 속성 [257](#)
- 조인 유형(속성)
 - 조이너 변환 [255](#)
- 조인 재정의
 - 오른쪽 외부 조인 구문 [375](#)
 - 왼쪽 외부 조인 구문 [373](#)
 - 일반 조인 구문 [372](#)
- 조인 조건
 - 개요 [256](#)
 - 정렬 원본 포트 사용 [259](#)
 - 정의 [260](#)
- 조회 SQL 재정의
 - 동적 캐시 [308](#)
- 조회 SQL 재정의 옵션
 - 매핑 매개 변수 및 변수 [280](#)
 - 설명 [274](#)
 - 캐시 크기 줄이기 [280](#)
- 조회 변환
 - Null 무시 [307](#)
 - 개요 [267](#)
 - 구성 요소 [272](#)
 - 기본 쿼리 [280](#)
 - 기본 쿼리 재정의 [280](#)
 - 날짜/시간 포트에 대한 초 단위 이하 정밀도 [274](#)
 - 데이터베이스에서 다시 캐시 [294](#)

조회 변환 (계속)

- 동적 캐시 [304](#)
- 동적 캐시를 조회 소스에 동기화 [317](#)
- 동적 캐시와 대상 동기화 [314](#)
- 로드 거부 [314](#)
- 매핑 매개 변수 및 변수 [280](#)
- 매핑에서 파이프라인 사용 [270](#)
- 명명된 지속형 캐시 [297](#)
- 반환 값 [287](#)
- 비교 시 무시 [308](#)
- 사용자 지정 쿼리 입력 [282](#)
- 성능 팁 [290](#)
- 속성 [274](#)
- 시퀀스 ID [306](#)
- 식 [288](#)
- 업데이트 전략 조합 [457](#)
- 여러 개의 일치 항목 [284](#)
- 여러 행 반환 [286](#)
- 연결 끊김 [272](#)
- 연결되지 않음 [271](#), [286](#)
- 연결된 입력 포트 [306](#)
- 연결된 조회 작성 [289](#)
- 연결됨 [271](#)
- 오류 임계값 [314](#)
- 입력 포트 [273](#)
- 재사용 가능 파이프라인 [290](#)
- 재사용 불가능 파이프라인 [290](#)
- 조건 [283](#), [287](#)
- 조회 소스 [267](#)
- 조회 포트 [273](#)
- 지속형 캐시 [295](#)
- 출력 포트 [273](#)
- 캐시 [292](#)
- 캐시 공유 [297](#)
- 파이프라인 조회 설명 [267](#)
- 파이프라인 조회 세션 속성 구성 [279](#)
- 파이프라인 조회 예제 [269](#), [270](#)
- 포트 [273](#)
- 플랫 파일 조회 [267](#), [268](#)

조회 소스 필터

- 설명 [274](#)
- 조회 제한 [282](#), [283](#)

조회 속성

- 세션에서 구성 [278](#)

조회 조건

- 개요 [284](#)
- 구성 [283](#)
- 데이터 마스킹 변환 [128](#)
- 정의 [274](#)

조회 캐시

- ORDER BY 재정의 [280](#)
- 개요 [292](#)
- 공유 [297](#)
- 데이터베이스에서 다시 캐시 [294](#)
- 동적 [304](#)
- 동적, 대상과 동기화 [314](#)
- 동적, 오류 임계값 [314](#)
- 동적, 조회 소스에 동기화 [317](#)
- 로드 거부 [314](#)
- 명명되지 않은 조회 공유 [297](#)
- 명명되지 않은 캐시를 사용하는 분할 지침 [297](#)
- 명명된 지속형 캐시 [297](#)
- 정의 [292](#)
- 정적 [297](#)
- 조회 데이터 캐시 크기 속성 [274](#)
- 지속형 [295](#)
- 지속형 조회 캐시 속성 [274](#)

조회 캐시 (계속)

- 첫 번째 값 및 마지막 값 처리 [284](#)
- 캐시 사전 빌드 [274](#)
- 캐싱 활성화 [274](#)

조회 캐시 다시 초기화

- 참조 데이터베이스에서 다시 캐시 [294](#)

조회 캐시 디렉터리 이름(속성)

- 설명 [274](#)

조회 캐싱 설정(속성)

- 설명 [274](#)

조회 쿼리

- ORDER BY [280](#)
- Sybase ORDER BY 제한 사항 [280](#)
- 기본 쿼리 [280](#)
- 설명 [280](#)
- 예약어 [281](#)
- 재정의 [280](#)

조회 테이블

- 이름 [274](#)
- 인덱스 [273](#), [290](#)

조회 포트

- 설명 [273](#)

종속 마스킹

- 반복 가능 마스킹 [131](#)
- 설명 [129](#)

종속 열

- 데이터 마스킹 [129](#)

종속성

- 사용자 지정 변환의 포트 [61](#)

주기

- 시퀀스 생성기 변환 속성 [354](#)

주기(속성)

- 시퀀스 생성기 변환 속성 [353](#)

증분

- 시퀀스 간격 설정 [354](#)

증분 범위(속성)

- 시퀀스 생성기 변환 속성 [353](#)

지속형 조회 캐시

- 개요 [295](#)
- 공유 [297](#)
- 데이터베이스에서 다시 캐시 [294](#)
- 명명된 파일 [297](#)
- 명명됨 및 명명되지 않음 [295](#)

지속형 조회 캐시(속성)

- 설명 [274](#)

직렬 변환기

- Data Transformation [445](#)

집계 변환

- null 값 [51](#)
- STDDEV(표준 편차) 함수 [50](#)
- VARIANCE 함수 [50](#)
- 개요 [48](#)
- 구성 요소 [48](#)
- 그룹 기준 포트 [51](#)
- 문제 해결 [56](#)
- 변수 사용 [36](#)
- 비집계 함수 예제 [51](#)
- 성능 최적화 [55](#)
- 식 변환과 비교 [48](#)
- 업데이트 전략 조합 [457](#)
- 작성 [55](#)
- 정렬된 포트 [53](#)
- 조건절 예제 [51](#)
- 조이너 변환과 함께 사용 [260](#)
- 중점 집계 [51](#)
- 추적 수준 [49](#)
- 포트 [49](#)

집계 변환 (계속)

함수 목록 [50](#)

집계 함수

null 값 [51](#)

개요 [50](#)

목록 [50](#)

식에 사용 [50](#)

차단

조이너 변환에서의 세부 행 [263](#)

차단 변환

설명 [31](#)

초 단위 이하 정밀도

저장 프로시저 변환 [424](#)

조회 변환 [274](#)

초기화 중

변수 [38](#)

사용자 지정 변환 프로시저 [68](#)

외부 프로시저 [179](#)

통합 서비스 변수 지원 [181](#)

초기화 취소 함수

사용자 지정 변환 [88](#)

초기화 함수

사용자 지정 변환 [85](#)

총 캐시된 값

시퀀스 생성기 변환 속성 [355](#)

시퀀스 생성기 속성 값 [353](#)

최대 출력 행 개수(속성)

SQL 변환 [402](#)

추가

그룹 [346](#)

식에 대한 설명 [33](#)

추적 메시지

외부 프로시저용 [177](#)

추적 수준

Java 변환 속성 [215](#)

간단 [44](#)

개요 [44](#)

보통 [44](#)

분류기 변환 속성 [361](#)

세션 속성 [49](#)

시퀀스 생성기 변환 속성 [357](#)

외부 프로시저 변환 속성 [157](#)

자세한 정보 표시 데이터 [44](#)

자세한 정보 표시 초기화 [44](#)

재정의 [44](#)

저장 프로시저 변환 속성 [424](#)

조이너 변환 속성 [255](#)

추적 수준 함수

설명 [186](#)

출력 계층 탭

구조화되지 않은 데이터 변환 [451](#)

출력 매개 변수

연결되지 않은 저장 프로시저 변환 [426](#)

저장 프로시저 [417](#)

출력 포트

Java 변환 [214](#)

개요 [30](#)

기본값 [38](#)

변수로 사용 [220](#)

식 변환에 필요 [153](#)

오류 처리 [38](#)

출력 행

Java 변환에서 행 유형 설정 [233](#)

출력 행 생성

Java 변환 [228](#)

출력은 반복 가능합니다(속성).

외부 프로시저 변환 [157](#)

출력은 반복 가능합니다(속성). (계속)

저장 프로시저 변환 [424](#)

출력이 정렬됨(속성)

Java 변환 [215](#)

캐시

동시 [294](#), [295](#)

동적 조회 캐시 [304](#)

명명된 지속형 조회 [297](#)

순차 [294](#)

정적 조회 캐시 [297](#)

조이너 변환 [263](#)

조회 공유 [297](#)

조회 변환 [292](#)

캐시 디렉터리

데이터 마스크 변환 [144](#)

조회 캐시 디렉터리 이름 구성 [274](#)

캐시 디렉터리(속성)

조이너 변환 [255](#)

캐시 크기

데이터 마스크 변환 [144](#)

캐시 파일 이름

지속형 조회 캐시에 대해 지정 [274](#)

캐시 파일 이름 접두사

개요 [297](#)

캐시 파일 이름 접두사(속성)

설명 [274](#)

캐싱

조이너 변환의 마스터 행 [263](#)

커밋

Java 변환 API 에서도 [227](#)

컴파일

Java 변환 [223](#)

Windows 시스템의 DLL [171](#)

사용자 지정 변환 프로시저 [76](#)

컴파일 오류

Java 변환의 소스 식별 [224](#)

코드 조각

Java 변환에 대해 작성 [218](#)

코드 페이지

사용자 지정 변환 [58](#)

액세스 함수 [185](#)

외부 프로시저 변환 [155](#)

코드 페이지 ID

사용자 지정 변환, 변경 [110](#)

쿼리

소스 한정자 변환 [366](#), [369](#)

조회 변환 [280](#)

조회 재정의 [280](#)

쿼리 모드

SQL 변환 [385](#)

규칙 및 지침 [390](#)

클래스 이름(속성)

Java 변환 [215](#)

키

소스 정의 [369](#)

시퀀스 생성기 변환을 사용하여 작성 [351](#)

조인에 대해 작성 [369](#)

키 마스크

날짜/시간 값 마스크 [126](#)

문자열 값 마스크 [124](#)

설명 [124](#)

숫자 값 [124](#)

숫자 값 마스크 [125](#)

키 유형 특성

노멀라이저 변환 [328](#)

테이블

키 관계 작성 [369](#)

통계 출력 포트 추가(속성)

SQL 변환 [402](#)

통과 포트

SQL 변환에 추가 [389](#)

기본값 [38](#)

통합 서비스

데이터 유형 [175](#)

데이터 집계 [51](#)

디버그 모드에서 실행 [177](#)

변수 지원 [181](#)

저장 프로시저의 오류 처리 [430](#)

트랜잭션 경계 [65](#)

트랜잭션

생성 [65](#), [217](#), [227](#)

정의 [434](#)

조이너 변환 작업 [264](#)

트랜잭션 경계

사용자 지정 변환 [65](#)

조이너 변환 삭제 [265](#)

조이너 변환에서 유지 [264](#)

트랜잭션 변환 범위

조이너 변환에서의 동작 [264](#)

트랜잭션 생성

Java 변환 [217](#), [227](#)

사용자 지정 변환 [65](#)

트랜잭션 생성(속성)

Java 변환 [215](#)

트랜잭션 수신 시 탭

Java 변환 [220](#)

트랜잭션 제어

Java 변환 [217](#)

SQL 변환 [394](#)

개요 [434](#)

변환 [434](#)

사용자 지정 변환 [64](#)

식 [435](#)

예제 [435](#)

트랜잭션 제어 변환

개요 [434](#)

매핑 내 [436](#)

매핑 유효성 검사 [438](#)

비효과적 [436](#)

속성 [434](#)

유효 [436](#)

작성 [439](#)

특수 형식 마스킹

IP 주소 [143](#)

URL [143](#)

반복 가능 SIN 번호 [143](#)

사회 보장 번호 [139](#)

사회 보험 번호 [143](#)

전자 메일 주소 [141](#)

전화 번호 [141](#)

파서

Data Transformation [445](#)

Data Transformation 출력 [453](#)

파이프라인

합집합 변환을 사용하여 병합 [440](#)

파이프라인 노멀라이저 변환

노멀라이저 탭 [332](#)

설명 [330](#)

작성 [333](#)

포트 탭 [331](#)

파이프라인 분할

HTTP 변환 [192](#)

사용자 지정 변환 [62](#)

파이프라인 조회

매핑 [270](#)

매핑 예제 [270](#)

설명 [267](#)

세션 속성 구성 [279](#)

재사용 가능 변환 [290](#)

재사용 불가능 변환 [290](#)

조회 변환 속성 [269](#)

조회 변환 예제 [269](#)

파일 입력 유형

구조화되지 않은 데이터 변환 [449](#)

파일 출력 유형

구조화되지 않은 데이터 변환 [452](#)

파티션 관련 함수

설명 [186](#)

파티션당 단일 스레드 필요(속성)

HTTP 변환 [192](#)

Java 변환 [215](#)

O

OCCURS 문

COBOL 예제 [325](#)

Oracle

연결 문자열 구문 [391](#)

저장 프로시저 참고 사항 [421](#)

ORDER BY

재정의 [280](#)

조회 쿼리 [280](#)

OutputStream 포트

구조화되지 않은 데이터 변환 [448](#)

OutputFileName 포트

구조화되지 않은 데이터 변환 [448](#)

OutputType

구조화되지 않은 데이터 변환 [447](#)

S

ScriptError 포트

설명 [384](#)

ScriptName 포트

SQL 변환 [384](#)

ScriptResults 포트

SQL 변환 [384](#)

setNull 메서드

Java 변환 [233](#)

setOutRowType

Java 변환 API 메서드 [233](#)

SIN 번호

반복 가능 데이터 마스킹 [143](#)

사회 보험 번호 마스킹 [143](#)

SQL

기본 쿼리 보기 [367](#)

기본 쿼리 재정의 [367](#), [369](#)

사용자 지정 쿼리 추가 [369](#)

SQL 문

SQL 변환에 의해 지원됨 [403](#)

SQL 변환

HA 복구 지침 [395](#)

NumRowsAffected [397](#)

PM_REC_STATE 테이블

SQL 변환 복구 [395](#)

ScriptError 포트 [384](#)

ScriptName 포트 [384](#)

ScriptResults 포트 [384](#)

SQL 변환 (계속)

- SELECT 쿼리 [386](#)
- SQL 특성 설정 [402](#)
- SQL 포트 설명 [402](#)
- 고급 옵션 [392](#)
- 데이터베이스 복구 [396](#)
- 데이터베이스 복원력 [394](#)
- 동적 SQL 쿼리 [387](#)
- 동적 연결 예제 [411](#)
- 동적 쿼리 예제 [406](#)
- 문자열 대체 사용 [387](#)
- 설명 [383](#)
- 속성 [400](#)
- 수동 모드 [389](#), [390](#)
- 스크립트 모드 [384](#)
- 연결 구성 [390](#)
- 원시 데이터 유형 열 [386](#)
- 자세한 정보 표시 데이터 설정 [400](#)
- 전체 연결 정보 전달 [391](#)
- 정적 SQL 쿼리 [385](#)
- 정적 쿼리 포트 [386](#)
- 정확히 한 번 메시지 배달 [395](#)
- 지원되는 SQL 문 [403](#)
- 쿼리 모드 [385](#)
- 통과 포트 [389](#)
- 트랜잭션 제어 [394](#)
- SQL 설정 탭
 - SQL 변환 [402](#)
- SQL 재정의
 - 기본 쿼리 [367](#)
- SQL 쿼리
 - 기본 쿼리 보기 [367](#)
 - 기본 쿼리 재정의 [367](#), [369](#)
 - 동적 업데이트 예제 [406](#)
 - 동적 연결 예제 [411](#)
 - 사용자 지정 쿼리 추가 [369](#)
- SQL 포트 탭
 - SQL 변환 [402](#)
- storeMetadata 메서드
 - Java 변환 [234](#)
- surnames.dic
 - 데이터 마스킹 [146](#)
- Sybase ASE
 - ORDER BY 제한 사항 [280](#)
 - 연결 문자열 구문 [391](#)
 - 저장 프로시저 참고 사항 [421](#)

T

- TC_COMMIT_AFTER 상수
 - 설명 [435](#)
- TC_COMMIT_BEFORE 상수
 - 설명 [435](#)
- TC_CONTINUE_TRANSACTION 상수
 - 설명 [435](#)
- TC_ROLLBACK_AFTER 상수
 - 설명 [435](#)
- TC_ROLLBACK_BEFORE 상수
 - 설명 [435](#)
- Teradata
 - 연결 문자열 구문 [391](#)
- TINFParm 매개 변수 유형
 - 정의 [178](#)
- TX 접두사가 있는 파일
 - 외부 프로시저 [168](#)

U

- UDT 설정
 - 구조화되지 않은 데이터 변환 [447](#)
- URL
 - 비즈니스 설명서 링크를 통해 추가 [33](#)

V

- Vertica
 - 연결 문자열 구문 [391](#)
- Visual Basic
 - COM 데이터 유형 [175](#)
 - COM 외부 프로시저 개발 [164](#)
 - 래퍼 클래스 대상 [176](#)
 - 수동으로 프로시저 배포 [174](#)
 - 외부 프로시저용 코드 [156](#)
 - 응용 프로그램 설치 마법사 [174](#)
 - 통합 서비스에 함수 추가 [176](#)
- Visual C++
 - COM 데이터 유형 [175](#)
 - COM 외부 프로시저 개발 [159](#)
 - 래퍼 클래스 대상 [176](#)
 - 수동으로 프로시저 배포 [174](#)
 - 통합 서비스에 라이브러리 추가 [176](#)
- VSAM 노멀라이저 변환
 - 노멀라이저 탭 [328](#)
 - 설명 [325](#)
 - 작성 [329](#)
 - 포트 탭 [327](#)

W

- Windows 시스템
 - DLL 컴파일 [171](#)

X

- XML
 - 큰 출력 분할 [453](#)
- XML 변환
 - XML 생성기 [462](#)
 - XML 파서 [461](#)
 - 소스 한정자 [461](#)
- XML 분할
 - 구조화되지 않은 데이터 변환 [453](#)
- XML 생성기 변환
 - 개요 [462](#)
- XML 스키마로 내보내기
 - 구조화되지 않은 데이터 변환 [451](#)
- XML 입력 스트리밍 활성화
 - XML 파서 세션 속성 [453](#)
- XML 파서 변환
 - 개요 [461](#)
 - 입력 분할 활성화 [453](#)

ㄱ

- 간단 추적 수준
 - 정의됨 [44](#)
- 값
 - 식 변환으로 계산 [153](#)

- 개발
 - COM 외부 프로시저 [159](#)
 - Informatica 외부 프로시저 [166](#)
- 거부 파일
 - 업데이트 전략 [456](#)
- 거부된 행 전달
 - 구성 [456](#)
 - 옵션 [456](#)
- 결과 문자열 대체 문자
 - 데이터 마스크 변환 [134](#)
- 계산
 - 변수 사용 [36](#)
 - 식 변환 사용 [152](#)
 - 집계 [48](#)
- 고급 옵션
 - SQL 변환 [392](#)
- 고급 인터페이스
 - EDatatype 클래스 [240](#)
 - Java 식 [239](#)
 - Java 식 호출 [240](#)
 - JExpression 클래스 [242](#), [243](#)
 - JExprParaMetadata 클래스 [241](#)
 - 예제 [243](#)
- 고유 출력 행
 - 분류기 변환 [361](#)
- 고유 항목 선택
 - 세션에서 재정의 [380](#)
 - 소스 한정자 옵션 [379](#)
- 고유한 출력
 - 데이터 마스크 변환 [128](#)
- 공유
 - 명명되지 않은 조회 캐시 [297](#)
 - 명명된 조회 캐시 [299](#)
- 공유 저장소 테이블
 - 데이터 마스크 변환 [144](#)
- 관계형 계층
 - 구조화되지 않은 데이터 변환 [451](#)
- 관계형 데이터베이스
 - 조인 [254](#)
- 구문
 - 오른쪽 외부 조인 작성 [375](#)
 - 왼쪽 외부 조인 작성 [373](#)
 - 일반 데이터베이스 제한 사항 [377](#)
 - 일반 조인 작성 [372](#)
- 구성
 - 포트 [29](#), [30](#)
- 구조화되지 않은 데이터 변환
 - Data Transformation에서 포트 채우기 [450](#)
 - InputBuffer 포트 [448](#)
 - OutputBuffer 포트 [448](#)
 - OutputFileName 포트 [448](#)
 - UDT 설정 탭 [447](#)
 - XML 스키마로 내보내기 [451](#)
 - XML 출력 분할 [453](#)
 - XML 파일에 쓰기 [453](#)
 - 개요 [443](#)

- 구조화되지 않은 데이터 변환 (계속)
 - 관계형 대상에 쓰기 [451](#), [452](#)
 - 구성 [454](#)
 - 구성 요소 [445](#)
 - 상태 추적 수준 설정 [448](#)
 - 입력 데이터 플러시 [451](#)
 - 파일 출력 유형 예제 [452](#)
 - 포트 [449](#)
- 규칙
 - 기본값 [43](#)
- 그룹
 - HTTP 변환 [194](#)
 - Java 변환 [214](#)
 - 라우터 변환 [344](#)
 - 사용자 정의 [344](#)
 - 사용자 지정 변환 [60](#)
 - 사용자 지정 변환 규칙 [60](#)
 - 추가 [346](#)
 - 합집합 변환 [441](#)
- 그룹 기준 포트
 - 기본값 사용 [53](#)
 - 비집계 식 [53](#)
 - 집계 변환 [51](#)
- 그룹 필터 조건
 - 라우터 변환 [345](#)
- 기본 URL
 - 구성 [197](#)
- 기본 그룹
 - 라우터 변환 [344](#)
- 기본 조인
 - 소스 한정자 [367](#)
- 기본 캐리
 - 개요 [366](#)
 - 보기 [366](#)
 - 소스 한정자를 사용하여 재정의 [369](#)
 - 재정의 방법 [367](#)
- 기본 키
 - 시퀀스 생성기 변환을 사용하여 작성 [351](#)
- 기본값
 - 개요 [38](#)
 - 규칙 [43](#)
 - 데이터 마스크 [144](#)
 - 사용자 정의 [39](#)
 - 유효성 검사 [44](#)
 - 입력 [44](#)
 - 입력 포트 [38](#), [39](#)
 - 집계 그룹 기준 포트 [53](#)
 - 출력 포트 [38](#), [39](#)
 - 통과 포트 [38](#), [39](#)

ㄱ

- 끝 값(속성)
 - 시퀀스 생성기 변환 [353](#)