Informatica®

10.1

Informatica® PowerCenter

10.1

Referenzhandbuch für die Umwandlungssprache

Informatica PowerCenter Referenzhandbuch für die Umwandlungssprache
10.110.1
Juni 2016

© Copyright Informatica LLC 1998, 2018

Diese Software und die zugehörige Dokumentation enthalten proprietäre Informationen der Informatica LLC, werden unter einem Lizenzvertrag mit Einschränkungen hinsichtlich Verwendung und Veröffentlichung zur Verfügung gestellt und sind urheberrechtlich geschützt. Das Zurückentwickeln (Reverse Engineering) der Software ist untersagt. Ohne ausdrückliche schriftliche Genehmigung der Informatica LLC darf kein Teil dieses Dokuments zu irgendeinem Zweck vervielfältigt oder übertragen werden, unabhängig davon, auf welche Art und Weise oder mit welchen Mitteln (elektronisch, mechanisch, durch Fotokopieren, Aufzeichnen usw.) dies geschieht. Diese Software ist möglicherweise durch US-amerikanische und/oder internationale Patente und weitere angemeldete Patente geschützt.

Die Verwendung, Vervielfältigung oder Veröffentlichung der Software durch die US-Regierung unterliegt den Bestimmungen des jeweiligen Softwarelizenzvertrags sowie ggf. den Bestimmungen in DFARS 227.7202-1(a) und 227.7702-3(a) (1995), DFARS 252.227-7013 © (1)(ii) (OCT. 1988), FAR 12.212(a) (1995), FAR 52.227-19 oder FAR 52.227-14 (ALT III).

Die in diesem Produkt und in dieser Dokumentation enthaltenen Informationen können jederzeit ohne vorherige Ankündigung geändert werden. Sollten Sie mit diesem Produkt oder dieser Dokumentation Probleme haben, teilen Sie uns dies bitte schriftlich mit.

Informatica, Informatica Platform, Informatica Data Services, PowerCenter, PowerCenterRT, PowerCenter Connect, PowerCenter Data Analyzer, PowerExchange, PowerMart, Metadata Manager, Informatica Data Quality, Informatica Data Explorer, Informatica B2B Data Transformation, Informatica B2B Data Exchange Informatica On Demand, Informatica Identity Resolution, Informatica Application Information Lifecycle Management, Informatica Complex Event Processing, Ultra Messaging, Informatica Master Data Management und Live Data Map sind Marken oder eingetragene Marken der Informatica LLC in den USA und anderen Ländern. Alle weiteren Produkt- und Firmennamen sind möglicherweise Markennamen oder Warenzeichen der jeweiligen Eigentümer.

Teile dieser Software und/oder Dokumentation sind durch die Urheberrechte Dritter geschützt, einschließlich und ohne Einschränkung: Copyright DataDirect Technologies. Alle Rechte vorbehalten. Copyright © Sun Microsystems. Alle Rechte vorbehalten. Copyright © RSA Security Inc. Alle Rechte vorbehalten. Copyright © Ordinal Technology Corp. Alle Rechte vorbehalten. Copyright © Aandacht c.v. Alle Rechte vorbehalten. Copyright Genivia, Inc. Alle Rechte vorbehalten. Copyright Isomorphic Software. Alle Rechte vorbehalten. Copyright © Meta Integration Technology, Inc. Alle Rechte vorbehalten. Copyright © Intalio. Alle Rechte vorbehalten. Copyright © Oracle. Alle Rechte vorbehalten. Copyright © Adobe Systems Incorporated. Alle Rechte vorbehalten. Copyright © DataArt, Inc. Alle Rechte vorbehalten. Copyright © ComponentSource. Alle Rechte vorbehalten. Copyright © Microsoft Corporation. Alle Rechte vorbehalten. Copyright © Rouge Wave Software, Inc. Alle Rechte vorbehalten. Copyright © Teradata Corporation. Alle Rechte vorbehalten. Copyright © Yahoo! Inc. Alle Rechte vorbehalten. Copyright © Glyph & Cog, LLC. Alle Rechte vorbehalten. Copyright © Thinkmap, Inc. Alle Rechte vorbehalten. Copyright © Clearpace Software Limited. Alle Rechte vorbehalten. Copyright © Information Builders, Inc. Alle Rechte vorbehalten. Copyright © OSS Nokalva, Inc. Alle Rechte vorbehalten. Copyright Edifecs, Inc. Alle Rechte vorbehalten. Copyright Cleo Communications, Inc. Alle Rechte vorbehalten. Copyright © International Organization for Standardization 1986. Alle Rechte vorbehalten. Copyright © ej-technologies GmbH . Alle Rechte vorbehalten. Copyright © Jaspersoft Corporation. Alle Rechte vorbehalten. Copyright © International Business Machines Corporation. Alle Rechte vorbehalten. Copyright © yWorks GmbH. Alle Rechte vorbehalten. Copyright © Lucent Technologies. Alle Rechte vorbehalten. Copyright © Universität von Toronto. Alle Rechte vorbehalten. Copyright © Daniel Veillard. Alle Rechte vorbehalten. Copyright © Unicode, Inc. Copyright IBM Corp. Alle Rechte vorbehalten. Copyright © MicroQuill Software Publishing, Inc. Alle Rechte vorbehalten. Copyright © PassMark Software Pty Ltd. Alle Rechte vorbehalten. Copyright © LogiXML, Inc. Alle Rechte vorbehalten. Copyright © 2003-2010 Lorenzi Davide. Alle Rechte vorbehalten. Copyright © Red Hat, Inc. Alle Rechte vorbehalten. Copyright © The Board of Trustees of the Leland Stanford Junior University. Alle Rechte vorbehalten. Copyright © EMC Corporation. Alle Rechte vorbehalten. Copyright © Flexera Software. Alle Rechte vorbehalten. Copyright © Jinfonet Software. Alle Rechte vorbehalten. Copyright © Apple Inc. Alle Rechte vorbehalten. Copyright © Telerik Inc. Alle Rechte vorbehalten. Copyright © BEA Systems. Alle Rechte vorbehalten. Copyright © PDFlib GmbH. Alle Rechte vorbehalten. Copyright © Orientation in Objects GmbH. Alle Rechte vorbehalten. Copyright © Tanuki Software, Ltd. Alle Rechte vorbehalten. Copyright © Ricebridge. Alle Rechte vorbehalten. Copyright © Sencha, Inc. Alle Rechte vorbehalten. Copyright © Scalable Systems, Inc. Alle Rechte vorbehalten. Copyright © jQWidgets. Alle Rechte vorbehalten. Copyright © Tableau Software, Inc. Alle Rechte vorbehalten. Copyright© MaxMind, Inc. Alle Rechte vorbehalten. Copyright © TMate Software s.r.o. Alle Rechte vorbehalten. Copyright © MapR Technologies Inc. Alle Rechte vorbehalten. Copyright © Amazon Corporate LLC. Alle Rechte vorbehalten. Copyright © Highsoft. Alle Rechte vorbehalten. Copyright © Python Software Foundation. Alle Rechte vorbehalten. Copyright © BeOpen.com. Alle Rechte vorbehalten. Copyright © CNRI. Alle Rechte vorbehalten.

Dieses Produkt enthält Software, die von der Apache Software Foundation (http://www.apache.org/) entwickelt wurde, und andere Software, die unter den Bedingungen des Apache-Lizenzvertrags lizenziert ist („Lizenz“). Eine Kopie dieser Lizenzen finden Sie unter http://www.apache.org/licenses/. Sofern nicht gesetzlich vorgeschrieben oder schriftlich vereinbart, erfolgt der Vertrieb der Software unter der Lizenz auf der BASIS „WIE BESEHEN” OHNE GARANTIEN ODER KONDITIONEN IRGENDEINER ART, weder ausdrücklich noch impliziert. Berechtigungen und Einschränkungen für bestimmte Sprachen finden Sie in der Lizenz.

Dieses Produkt enthält Software, die von Mozilla (http://www.mozilla.org/) entwickelt wurde, Software Copyright The JBoss Group, LLC. Alle Rechte vorbehalten; Software Copyright © 1999-2006 by Bruno Lowagie und Paulo Soares, und andere Software, die gemäß den verschiedenen Versionen des GNU Lesser General Public License Agreement unter http://www.gnu.org/licenses/lgpl.html lizenziert ist. Die Materialien werden „wie besehen“ kostenlos von Informatica bereitgestellt, ohne ausdrückliche oder stillschweigende Gewährleistung, einschließlich, jedoch nicht beschränkt auf die stillschweigenden Gewährleistungen der Handelsüblichkeit und der Eignung für einen bestimmten Zweck.

Das Produkt enthält ACE(TM) und TAO(TM) Software, Copyright Douglas C. Schmidt und seine Forschungsgruppe an der Washington University, University of California, Irvine und Vanderbilt University, Copyright (©) 1993-2006. Alle Rechte vorbehalten.

Dieses Produkt enthält Software, die von OpenSSL Project zur Verwendung im OpenSSL Toolkit entwickelt wurde (Copyright The OpenSSL Project. Alle Rechte vorbehalten). Die erneute Verteilung dieser Software unterliegt den unter „http://www.openssl.org“ und „http://www.openssl.org/source/license.html“ verfügbaren Bedingungen.

Dieses Produkt enthält urheberrechtlich geschützte Curl-Software (Copyright 1996-2013, Daniel Stenberg, <daniel@haxx.se>). Alle Rechte vorbehalten. Die mit dieser Software verbundenen Berechtigungen und Einschränkungen unterliegen den unter „http://curl.haxx.se/docs/copyright.html“ verfügbaren Bedingungen. Die Erlaubnis, diese Software für jeden beliebigen Zweck gegen Gebühr oder kostenlos zu verwenden, zu kopieren, zu ändern und zu verteilen, wird hiermit erteilt, sofern die oben genannten urheberrechtlichen Hinweise und diese Erlaubnis in allen Exemplaren angegeben werden.

Das Produkt enthält urheberrechtlich geschützte Software, Copyright 2001-2005 (©) MetaStuff, Ltd. Alle Rechte vorbehalten. Die mit dieser Software verbundenen Berechtigungen und Einschränkungen unterliegen den unter „http://www.dom4j.org/ license.html“ verfügbaren Bedingungen.

Das Produkt enthält urheberrechtlich geschützte Software, Copyright © 2004-2007, The Dojo Foundation. Alle Rechte vorbehalten. Die mit dieser Software verbundenen Berechtigungen und Einschränkungen unterliegen den unter „http://dojotoolkit.org/license“ verfügbaren Bedingungen.

Dieses Produkt enthält urheberrechtlich geschützte ICU-Software, Copyright International Business Machines Corporation und andere. Alle Rechte vorbehalten. Die mit dieser Software verbundenen Berechtigungen und Einschränkungen unterliegen den unter „http://source.icu-project.org/repos/icu/icu/trunk/license.html“ verfügbaren Bedingungen.

Dieses Produkt enthält urheberrechtlich geschützte Software, Copyright © 1996-2006 Per Bothner. Alle Rechte vorbehalten. Das Ihnen erteilte Recht, diese Materialien zu verwenden, unterliegt den unter „http:// www.gnu.org/software/ kawa/Software-License.html“ verfügbaren Bedingungen.

Dieses Produkt enthält urheberrechtlich geschützte OSSP UUID-Software (Copyright © 2002 Ralf S. Engelschall, Copyright © 2002 The OSSP Project Copyright © 2002 Cable & Wireless Deutschland). Die mit dieser Software verbundenen Berechtigungen und Einschränkungen unterliegen den unter „http://www.opensource.org/licenses/mit-license.php“ verfügbaren Bedingungen.

Dieses Produkt enthält Software, die von Boost (http://www.boost.org/) oder unter der Softwarelizenz von Boost entwickelt wurde. Die mit dieser Software verbundenen Berechtigungen und Einschränkungen unterliegen den unter „http://www.boost.org/LICENSE_1_0.txt“ verfügbaren Bedingungen.

Dieses Produkt enthält urheberrechtlich geschützte Software, Copyright © 1997-2007 University of Cambridge. Die mit dieser Software verbundenen Berechtigungen und Einschränkungen unterliegen den unter http://www.pcre.org/license.txt einsehbaren Bedingungen.

Dieses Produkt enthält urheberrechtlich geschützte Software, Copyright © 2007 The Eclipse Foundation. Alle Rechte vorbehalten. Die mit dieser Software verbundenen Berechtigungen und Einschränkungen unterliegen den unter „http://www.eclipse.org/org/documents/epl-v10.php“ und „http://www.eclipse.org/org/documents/edl-v10.php“ verfügbaren Bedingungen.

Dieses Produkt enthält Software gemäß den Lizenzbedingungen unter 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/hsqlLicense.html, http://httpunit.sourceforge.net/doc/ license.html, http://jung.sourceforge.net/license.txt, http://www.gzip.org/zlib/zlib_license.html, http://www.openldap.org/software/release/license.html, http://www.libssh2.org, http://slf4j.org/license.html, http://www.sente.ch/software/OpenSourceLicense.html, http://fusesource.com/downloads/license-agreements/fuse-message-broker-v-5-3- license-agreement; http://antlr.org/license.html; http://aopalliance.sourceforge.net/; http://www.bouncycastle.org/licence.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/licence.html, http://www.sqlite.org/copyright.html, http://www.tcl.tk/software/tcltk/license.html, http://www.jaxen.org/faq.html, http://www.jdom.org/docs/faq.html, http://www.slf4j.org/license.html; http://www.iodbc.org/dataspace/iodbc/wiki/iODBC/License; http://www.keplerproject.org/md5/license.html; http://www.toedter.com/en/jcalendar/license.html; http://www.edankert.com/bounce/index.html; http://www.net-snmp.org/about/license.html; http://www.openmdx.org/#FAQ; http://www.php.net/license/3_01.txt; http://srp.stanford.edu/license.txt; http://www.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; https://sourceforge.net/p/xmlunit/code/HEAD/tree/trunk/LICENSE.txt.

Dieses Produkt enthält Software, die unter der Academic Free License (http://www.opensource.org/licenses/afl-3.0.php), der Common Development Distribution License (http://www.opensource.org/licenses/cddl1.php), der Common Public License (http://www.opensource.org/licenses/cpl1.0.php), den Sun Binary Code License Agreement Supplemental License Terms, der BSD License (http:// www.opensource.org/licenses/bsd-license.php), der neuen BSD License (http://opensource.org/licenses/BSD-3-Clause), der MIT License (http://www.opensource.org/licenses/mit-license.php), der Artistic License (http://www.opensource.org/licenses/artistic-license-1.0) und der Initial Developer's Public License Version 1.0 (http://www.firebirdsql.org/en/initial-developer-s-public-license-version-1-0/) lizenziert ist.

Dieses Produkt enthält urheberrechtlich geschützte Software, Copyright © 2003-2006 Joe Walnes, 2006-2007 XStream Committers. Alle Rechte vorbehalten. Die mit dieser Software verbundenen Berechtigungen und Einschränkungen unterliegen den unter „http://xstream.codehaus.org/license.html“ verfügbaren Bedingungen. Dieses Produkt enthält Software, die von der Indiana University Extreme! Lab. entwickelt wurde. Weitere Informationen finden Sie unter http://www.extreme.indiana.edu/.

Dieses Produkt enthält Software, Copyright © 2013 Frank Balluffi und Markus Moeller. Alle Rechte vorbehalten. Die mit dieser Software verbundenen Berechtigungen und Einschränkungen unterliegen den Bedingungen der MIT-Lizenz.

Weitere Informationen über die Patente finden Sie unter https://www.informatica.com/legal/patents.html.

HAFTUNGSAUSSCHLUSS: Informatica LLC stellt diese Dokumentation „wie besehen“ bereit, ohne ausdrückliche oder stillschweigende Gewährleistung, einschließlich, jedoch nicht beschränkt auf die Gewährleistungen der Nichtverletzung der Rechte von Dritten, der Handelsüblichkeit oder Eignung für einen bestimmten Zweck. Informatica LLC garantiert nicht die Fehlerfreiheit dieser Software oder Dokumentation. Die in dieser Software oder Dokumentation bereitgestellten Informationen können technische Ungenauigkeiten oder Druckfehler enthalten. Die in dieser Software und in dieser Dokumentation enthaltenen Informationen können jederzeit ohne vorherige Ankündigung geändert werden.

HINWEISE

Dieses Informatica-Produkt (die „Software“) umfasst bestimmte Treiber (die „DataDirect-Treiber“) von DataDirect Technologies, einem Betreiber von Progress Software Corporation („DataDirect“), die folgenden Bedingungen und Bestimmungen unterliegen:

1. DIE DATADIRECT-TREIBER WERDEN „WIE GESEHEN“ OHNE JEGLICHE GEWÄHRLEISTUNG, WEDER AUSDRÜCKLICH NOCH STILLSCHWEIGEND, BEREITGESTELLT, EINSCHLIESSLICH, JEDOCH NICHT BESCHRÄNKT AUF DIE STILLSCHWEIGENDEN GEWÄHRLEISTUNGEN DER HANDELSÜBLICHKEIT, EIGNUNG FÜR EINEN BESTIMMTEN ZWECK UND DER NICHTVERLETZUNG VON RECHTEN DRITTER.
2. IN KEINEM FALL SIND DATADIRECT ODER DRITTANBIETER DEM ENDBENUTZER GEGENÜBER HAFTBAR FÜR UNMITTELBARE, MITTELBARE, KONKRETE, NEBEN-, FOLGE- ODER ANDERE SCHÄDEN, DIE SICH AUS DER VERWENDUNG DER ODBC-TREIBER ERGEBEN, UNABHÄNGIG DAVON, OB SIE IM VORAUS ÜBER DIE MÖGLICHKEIT SOLCHER SCHÄDEN INFORMIERT WORDEN SIND ODER NICHT. DIESE BESCHRÄNKUNGEN GELTEN FÜR ALLE KLAGEGEGENSTÄNDE, EINSCHLIESSLICH, JEDOCH NICHT BESCHRÄNKT AUF VERTRAGSBRUCH, GEWÄHRLEISTUNGSBRUCH, FAHRLÄSSIGKEIT, KAUSALHAFTUNG, TÄUSCHUNG UND ANDERE UNERLAUBTE HANDLUNGEN.

Publikationsdatum: 2018-07-19

Inhalt

Einleitung 11

Informatica-Ressourcen. 11

Informatica-Netzwerk. 11

Informatica-Wissensdatenbank. 11

Informatica-Dokumentation. 12

Informatica-Produktverfügbarkeitsmatrizen. 12

Informatica Velocity. 12

Informatica Marketplace. 12

Globaler Kundensupport von Informatica. 12

Kapitel 1: Umwandlungssprache. 13

Umwandlungssprache – Übersicht. 13

Komponenten der Umwandlungssprache. 13

Internationalisierung und Umwandlungssprache. 14

Ausdruckssyntax. 15

Ausdruckskomponenten. 15

Regeln und Richtlinien für die Ausdruckssyntax. 17

Hinzufügen von Kommentaren zu Ausdrücken. 18

Reservierte Wörter. 19

Kapitel 2: Konstanten. 20

DD_DELETE. 20

Beispiel. 20

DD_INSERT. 21

Beispiele. 21

DD_REJECT. 21

Beispiele. 22

DD_UPDATE. 22

Beispiele. 22

FALSE. 23

Beispiel. 23

NULL. 23

Arbeiten mit Nullwerten in Booleschen Ausdrücken. 23

Nullwerte in Vergleichsausdrücken. 23

Nullwerte in Aggregatfunktionen. 24

Nullwerte in Filterbedingungen. 24

Nullen und Operatoren. 25

TRUE. 25

Beispiel. 25

Kapitel 3: Operatoren. 26

Rangordnung von Operatoren. 26
Mathematische Operatoren. 27
String-Operatoren. 28
Nullen. 28
Beispiel. 28
Vergleichsoperatoren. 29
Logische Operatoren. 29
Nullen. 30

Kapitel 4: Variablen. 31

Integrierte Variablen. 31
$PM<SourceName>@TableName, $PM<TargetName>@TableName. 33
$PMFolderName. 34
$PMIntegrationServiceName. 34
$PMMappingName. 34
$PMRepositoryServiceName. 34
$PMRepositoryUserName. 34
$PMSessionName. 34
$PMSessionRunMode. 35
$PMWorkflowName. 35
$PMWorkflowRunId. 35
$PMWorkflowRunInstanceName. 35
SESSSTARTTIME. 35
SYSDATE. 36
WORKFLOWSTARTTIME 36
Transaktionssteuerungsvariablen. 37
Lokale Variablen. 37

Kapitel 5: Datumsangaben. 38

Datumsangaben – Übersicht. 38
Datum/Zeit-Datentyp. 38
Julianisches Datum, Modifiziertes Julianisches Datum und Gregorianischer Kalender. 39
Datumsangaben im Jahr 2000. 39
Datumsangaben in relationalen Datenbanken. 41
Datumsangaben in Einfachdateien. 41
Standarddatumsformat. 41
Datumsformatstrings. 42
TO_CHAR-Formatstrings. 44
Beispiele. 46
TO_DATE- und IS_DATE-Formatstrings. 47
Regeln und Richtlinien für Datumsformatstrings. 49

Beispiel. 50
Verständnis von Datumsberechnungen. 51

Kapitel 6: Funktionen. 52

Funktionskategorien. 52
Aggregatfunktionen. 52
Aggregatfunktionen und Nullen. 54
Zeichenfunktionen. 54
Konvertierungsfunktionen. 55
Datenbereinigungsfunktionen. 55
Datumsfunktionen. 56
Kodierungsfunktionen. 57
Finanzfunktionen. 57
Numerische Funktionen. 58
Wissenschaftliche Funktionen. 58
Spezialfunktionen. 58
Stringfunktionen. 59
Testfunktionen. 59
Variablenfunktionen. 59
ABORT. 60
ABS. 61
ADD_TO_DATE. 62
AES_DECRYPT. 65
AES_ENCRYPT. 65
ANY. 66
ASCII. 67
AVG. 68
CEIL. 70
CHOOSE. 71
CHR. 72
CHRCODE. 73
COMPRESS. 74
CONCAT. 75
CONVERT_BASE. 76
COS. 77
COSH. 78
COUNT. 79
CRC32. 81
CREATE_TIMESTAMP_TZ. 82
CUME. 82
DATE_COMPARE. 84
DATE_DIFF. 85
DEC_BASE64. 88

DECODE. 89
DECOMPRESS. 91
ENC_BASE64. 92
ERROR. 93
EXP. 94
FIRST. 95
FLOOR. 96
FV 97
GET_DATE_PART. 98
GET_TIMEZONE. 100
GET_TIMESTAMP. 101
GREATEST. 102
IIF. 103
IN. 106
INDEXOF. 108
INITCAP. 109
INSTR. 110
ISNULL. 113
IS_DATE. 114
IS_NUMBER. 116
IS_SPACES. 118
LAST. 119
LAST_DAY. 120
LEAST. 122
LENGTH. 123
LN. 124
LOG. 125
LOOKUP. 126
LOWER. 128
LPAD. 128
LTRIM. 130
MAKE_DATE_TIME. 131
MAX (Datum). 132
MAX (Zahlen). 133
MAX (String). 135
MD5. 136
MEDIAN. 137
METAPHONE. 139
MIN (Datum). 143
MIN (Zahlen). 144
MIN (String). 145
MOD. 147

MOVINGAVG. 148
MOVINGSUM. 150
NPER. 151
PERCENTILE. 152
PMT. 154
POWER. 155
PV. 156
RAND. 157
RATE. 157
REG_EXTRACT. 158
REG_MATCH. 161
REG_REPLACE. 162
REPLACECHR. 163
REPLACESTR. 166
REVERSE. 169
ROUND (Datum). 170
ROUND (Zahlen). 174
RPAD. 177
RTRIM. 178
SETCOUNTVARIABLE. 179
SET_DATE_PART. 181
SETMAXVARIABLE. 184
SETMINVARIABLE. 186
SETVARIABLE. 188
SIGN. 190
SIN. 190
SINH. 191
SOUNDEX. 192
SQL_LIKE. 194
SQRT. 195
STDDEV. 196
SUBSTR. 198
SUM. 200
SYSTIMESTAMP. 201
TAN. 203
TANH. 203
TO_BIGINT. 204
TO_CHAR (Datum). 206
TO_CHAR (Zahlen). 211
TO_DATE. 213
TO_DECIMAL. 216
TO_DECIMAL38. 218

TO_FLOAT. 219
TO_INTEGER. 220
TO_TIMESTAMP_TZ. 222
TRUNC (Datum). 224
TRUNC (Zahlen). 227
UPPER. 229
UUID4. 230
UUID_UNPARSE. 230
VARIANCE. 230

Kapitel 7: Erstellen von benutzerdefinierten Funktionen. 233

Erstellen von benutzerdefinierten Funktionen – Übersicht. 233
Schritte zum Erstellen von benutzerdefinierten Funktionen. 234
Installieren von benutzerdefinierten Funktionen. 234
Schritt 1. Abrufen von Repository-ID-Attributen. 234
Schritt 2. Erstellen einer Header-Datei. 235
Schritt 3. Erstellen einer Implementierungsdatei. 237
Schritt 4. Erstellen der Module. 246
Erstellen der Module in Windows. 247
Erstellen der Module in UNIX. 248
Schritt 5. Erstellen der Repository-Plug-In-Datei. 248
Das Element PLUGIN. 249
Das Element FUNCTION_GROUP. 249
Festlegen eines Namespace. 250
Das Element FUNCTION. 250
Das Element LIBRARY. 251
Plug-In-XML-Beispieldatei. 252
Schritt 6. Testen von benutzerdefinierten Funktionen. 252
Validieren der Repository-Plug-In-Datei. 252
Überprüfen der Funktionsgenauigkeit. 253
Installieren von benutzerdefinierten Funktionen. 253
Schritt 1. Kopieren benutzerdefinierter Funktionsbibliotheken nach PowerCenter. 253
Schritt 2. Registrieren des Plug-In. 254
Erstellen von Ausdrücken mit benutzerdefinierten Funktionen. 254

Kapitel 8: Referenz für die API für benutzerdefinierte Funktionen. 255

Referenz für die API für benutzerdefinierte Funktionen - Übersicht. 255
Allgemeine APIs. 255
Validierungs-Handle. 256
Validierungsfunktion für Schnittstellen. 256
INFA_EXPR_OPD_METADATA-Struktur. 259
Funktion zum Abrufen der Plug-In-Version. 259
Laufzeit-APIs. 260

Funktionen auf Modulebene. 260
Funktionen auf Funktionsebene. 262
Funktionen auf Funktionsinstanzebene. 263

Index. 266

Einleitung

Das *Informatica DeveloperReferenzhandbuch für die Umwandlungssprache* wendet sich an Entwickler, in deren Zuständigkeitsbereich das Erstellen von Mappings fällt. Im *Informatica DeveloperReferenzhandbuch für die Umwandlungssprache* wird davon ausgegangen, dass Sie mit SQL, relationalen Datenbankkonzepten und den Schnittstellenanforderungen für Ihre unterstützenden Anwendungen vertraut sind.

Informatica-Ressourcen

Informatica-Netzwerk

Im Informatica-Netzwerk finden Sie den globalen Kundensupport von Informatica, die Informatica-Wissensdatenbank und andere Produktressourcen. Für den Zugriff auf das Informatica-Netzwerk besuchen Sie https://network.informatica.com.

Als Mitglied können Sie:

- zentral auf alle Ihre Informatica-Ressourcen zugreifen.
- Durchsuchen Sie die Wissensdatenbank nach Produktressourcen, einschließlich Dokumentation, häufig gestellter Fragen und bewährter Methoden.
- Zeigen Sie Informationen zur Produktverfügbarkeit an.
- Ihre Support-Fälle prüfen.
- Ihr lokales Informatica-Netzwerk für Benutzergruppen suchen und mit anderen Benutzern zusammenarbeiten.

Als Mitglied können Sie:

- zentral auf alle Ihre Informatica-Ressourcen zugreifen.
- Durchsuchen Sie die Wissensdatenbank nach Produktressourcen, einschließlich Dokumentation, häufig gestellter Fragen und bewährter Methoden.
- Zeigen Sie Informationen zur Produktverfügbarkeit an.
- Ihr lokales Informatica-Netzwerk für Benutzergruppen suchen und mit anderen Benutzern zusammenarbeiten.

Informatica-Wissensdatenbank

Verwenden Sie die Informatica-Wissensdatenbank, um das Informatica-Netzwerk nach Produktressourcen, wie z. B. Dokumentation, Ratgeberartikeln, bewährten Methoden und PAMs, zu durchsuchen.

Für den Zugriff auf die Wissensdatenbank besuchen Sie https://kb.informatica.com. Wenn Sie Fragen, Kommentare oder Ideen zur Wissensdatenbank haben, wenden Sie sich per E-Mail an das Team der Informatica-Wissensdatenbank unter KB_Feedback@informatica.com.

Informatica-Dokumentation

Navigieren Sie zur Informatica-Wissensdatenbank unter https://kb.informatica.com/_layouts/ProductDocumentation/Page/ProductDocumentSearch.aspx, um die aktuelle Dokumentation für Ihr Produkt abzurufen.

Wenn Sie Fragen, Kommentare oder Ideen zu dieser Dokumentation haben, wenden Sie sich per E-Mail an das Informatica-Dokumentationsteam unter infa_documentation@informatica.com.

Informatica-Produktverfügbarkeitsmatrizen

Produktverfügbarkeitsmatrizen (PAMs) geben die Versionen der Betriebssysteme, Datenbanken und anderen Typen von Datenquellen und Zielen an, die in einer Produktversion unterstützt werden. Als Mitglied des Informatica-Netzwerks können Sie unter https://network.informatica.com/community/informatica-network/product-availability-matrices auf PAMs zugreifen.

Informatica Velocity

Bei Informatica Velocity handelt es sich um eine Sammlung von Tipps und bewährten Methoden, die von den professionellen Informatica-Diensten entwickelt wurden. Informatica Velocity basiert auf der Praxiserfahrung aus Hunderten von Datenmanagementprojekten und umfasst das kollektive Wissen unserer Berater, die mit Unternehmen aus der ganzen Welt an der Planung, Entwicklung, Bereitstellung und Wartung erfolgreicher Datenmanagementlösungen gearbeitet haben.

Als Mitglied des Informatica-Netzwerks können Sie unter http://velocity.informatica.com auf Informatica Velocity-Ressourcen zugreifen.

Wenn Sie Fragen, Anregungen oder Ideen zu Informatica Velocity haben, wenden Sie sich an die professionellen Informatica-Dienste unter ips@informatica.com.

Informatica Marketplace

Informatica Marketplace ist ein Forum, das Lösungen zur Erweiterung und Verbesserung Ihrer Informatica-Implementierungen bereitstellt. Indem Sie die zahlreichen Lösungen von Informatica-Entwicklern und -Partnern nutzen, können Sie Ihre Produktivität steigern und die Implementierungsdauer Ihrer Projekte verkürzen. Zugriff auf den Informatica Marketplace erhalten Sie unter https://marketplace.informatica.com.

Globaler Kundensupport von Informatica

Sie können sich telefonisch oder über den Online-Support mit einem globalen Support-Center im Informatica-Netzwerk in Verbindung setzen.

Die Telefonnummer des globalen Kundensupports von Informatica vor Ort finden Sie auf der Informatica-Website unter folgender Verknüpfung: http://www.informatica.com/us/services-and-training/support-services/global-support-centers.

Als Mitglied des Informatica-Netzwerks können Sie den Online-Support unter http://network.informatica.com verwenden.

KAPITEL 1

Umwandlungssprache

Dieses Kapitel umfasst die folgenden Themen:

- Umwandlungssprache – Übersicht, 13
- Ausdruckssyntax, 15
- Hinzufügen von Kommentaren zu Ausdrücken, 18
- Reservierte Wörter, 19

Umwandlungssprache – Übersicht

PowerCenter stellt Ihnen eine Umwandlungssprache mit SQL-ähnlichen Funktionen zum Umwandeln von Quelldaten bereit. Mit diesen Funktionen können Sie Ausdrücke formulieren und benutzerdefinierte Funktionen erstellen.

Informatica Developer liefert Ihnen eine Umwandlungssprache mit SQL-ähnlichen Funktionen zum Umwandeln von Quelldaten. Mit diesen Funktionen können Sie Ausdrücke formulieren.

Benutzerdefinierte Funktionen greifen auf die Ausdruckslogik zurück und liefern komplexe Ausdrücke. Diese können Sie in andere benutzerdefinierte Funktionen bzw. in Ausdrücke einschließen. Für benutzerdefinierte Funktionen gelten die gleichen Richtlinien wie für Ausdrücke. Sie nutzen dieselbe Syntax und können auch dieselben Umwandlungssprachenkomponenten verwenden.

Ausdrücke ändern Daten oder überprüfen, ob Daten mit Bedingungen übereinstimmen. So können Sie beispielsweise mithilfe der Funktion AVG das Durchschnittsgehalt aller Mitarbeiter oder mithilfe von SUM den Gesamtumsatz einer bestimmten Filiale berechnen.

Möglich sind einfache Ausdrücke mit nur einem Port, z. B. ORDERS, oder einem numerischen Literal, z. B. 10. Sie können aber auch komplexe Ausdrücke mit verschachtelten Funktionen erstellen oder mithilfe der Operatoren der Umwandlungssprache verschiedene Ports kombinieren.

Komponenten der Umwandlungssprache

Die Umwandlungssprache enthält die folgenden Komponenten zum Erstellen von einfachen und komplexen Umwandlungsausdrücken:

- **Funktionen:** Über 100 SQL-ähnliche Funktionen ermöglichen Ihnen, Daten in einem Mapping zu ändern.
- **Operatoren:** Mit den Umwandlungsoperatoren können Sie Umwandlungsausdrücke erstellen, um mathematische Berechnungen auszuführen oder Daten zu kombinieren und zu vergleichen.
- **Konstanten:** Mit den integrierten Konstanten können Sie konstant bleibende Werte referenzieren, etwa TRUE.

- **Mapping-Parameter und -Variablen:** Erstellen Sie Mapping-Parameter, um in einem Mapping oder einem Mapplet Werte zu referenzieren, die innerhalb einer Sitzung konstant bleiben, etwa einen Steuersatz. Mit Mapping-Variablen können Sie in Mapplets oder Mappings Ausdrücke formulieren, um auf Werte zu verweisen, die sich von Sitzung zu Sitzung ändern.
- **Mapping-Parameter:** Erstellen Sie Mapping-Parameter, um in einem Mapping oder einem Mapplet Werte zu referenzieren, die während der Ausführungsdauer eines Mappings bzw. Mapplets konstant bleiben, etwa einen Steuersatz.
- **Arbeitsablaufsvariablen:** Erstellen Sie Arbeitsablaufsvariablen in einem Arbeitsablauf, um Werte zu referenzieren, die von Arbeitsablauf zu Arbeitsablauf unterschiedlich ausfallen.
- **Integrierte und lokale Variablen:** Mit integrierten Variablen können Sie Ausdrücke formulieren, um variable Werte wie das Systemdatum zu referenzieren. Sie können auch lokale Variablen in Umwandlungen erstellen.
- **Rückgabewerte:** Sie können auch Ausdrücke formulieren, die die Lookup-Umwandlungen der Rückgabewerte einschließen.

Internationalisierung und Umwandlungssprache

Die Funktionen der Umwandlungssprache können Zeichendaten der Datenverschiebungsmodi ASCII und Unicode verarbeiten. Verwenden Sie den Unicode-Modus für die Verarbeitung von *Multibyte*-Zeichendaten. Die Rückgabewerte der folgenden Funktionen und Umwandlungen hängen von der Codepage von Data Integration Service und dem Datenverschiebungsmodus ab:

- INITCAP
- LOWER
- UPPER
- MIN (Datum)
- MIN (Zahl)
- MIN (String)
- MAX (Datum)
- MAX (Zahl)
- MAX (String)
- Alle Funktionen mit konditionalen Anweisungen zum Vergleich von Strings, z. B. IIF und DECODE

Die Rückgabewerte von MIN und MAX hängen darüber hinaus auch von der Sortierreihenfolge der Codepage von Data Integration Service ab.

Beim Validieren eines ungültigen Ausdrucks im Ausdrucksseditor wird der Ausdruck im Dialogfeld mit dem Fehlerindikator „>>>>“ versehen. Der Indikator erscheint links neben dem fehlerhaften Teil des Ausdrucks. Beispiel: Wenn der Ausdruck „a = b + c“ einen Fehler bei c enthält, zeigt die Fehlermeldung Folgendes an:

```
a = b +   >>>> c
```

Die Funktionen der Umwandlungssprache zum Auswerten von Zeichendaten orientieren sich an den Zeichen, nicht an den Byte. So gibt etwa die Funktion LENGTH die Anzahl der Zeichen in einem String zurück, nicht die Anzahl der Byte. Die Funktion LOWER gibt einen String in Kleinbuchstaben zurück, abhängig von der Codepage von Data Integration Service.

Ausdruckssyntax

Obwohl die Umwandlungssprache auf Standard-SQL beruht, gibt es einige Unterschiede zwischen den beiden Sprachen. So unterstützt SQL beispielsweise die Schlüsselwörter ALL und DISTINCT für Aggregatfunktionen, die Umwandlungssprache jedoch nicht. Andererseits unterstützt die Umwandlungssprache im Gegensatz zu SQL eine optionale Filterbedingung für Aggregatfunktionen.

Sie können Ausdrücke erstellen, die nichts anderes als einen einfachen Port (z. B. ORDERS), eine einfache vordefinierte Arbeitsablaufsvariable (z. B. $Start.Status) oder ein einfaches numerisches Literal (z. B. 10) enthalten. Sie können aber auch komplexe Ausdrücke formulieren, etwa verschachtelte Funktionen oder Kombinationen aus verschiedenen Spalten mithilfe der Operatoren der Umwandlungssprache.

Sie können Ausdrücke erstellen, die nichts anderes als einen einfachen Port (z. B. ORDERS) oder ein einfaches numerisches Literal (z. B. 10) enthalten. Sie können aber auch komplexe Ausdrücke formulieren, etwa verschachtelte Funktionen oder Kombinationen aus verschiedenen Spalten mithilfe der Operatoren der Umwandlungssprache.

Ausdruckskomponenten

Ausdrücke können aus einer beliebigen Kombination folgender Komponenten bestehen:

- Ports (Eingabe, Eingabe/Ausgabe, Variable)
- String-Literale, numerische Literale
- Konstanten
- Funktionen
- Integrierte und lokale Variablen
- Mapping-Parameter und Mapping-Variablen
- Mapping-Parameter
- Vordefinierte Arbeitsablaufsvariablen
- Benutzerdefinierte Arbeitsablaufsvariablen
- Operatoren
- Rückgabewerte

Ports und Rückgabewerte

Beim Formulieren eines Ausdrucks mit einem Port oder Rückgabewert aus einer nicht verbundenen Umwandlung verwenden Sie die Referenzqualifikatoren aus der folgenden Tabelle:

Referenzqualifikator	Beschreibung
:EXT	Dies ist erforderlich für Ausdrücke mit einem Rückgabewert aus einer Umwandlung einer externen Prozedur. Die allgemeine Syntax lautet: `:EXT.external_procedure_transformation(argument1, argument2, ...)`
:LKP	Dies ist erforderlich für Ausdrücke mit einem Rückgabewert aus einer nicht verbundenen Lookup-Umwandlung. Die allgemeine Syntax lautet: `:LKP.lookup_transformation(argument1, argument2, ...)` Die Argumente sind die lokalen Ports in der Lookup-Bedingung. Die Reihenfolge muss jener der Ports in der Umwandlung entsprechen. Die Datentypen der lokalen Ports müssen dem Datentyp der Lookup-Ports in der Lookup-Bedingung entsprechen.
:SD	Optional (nur PowerMart 3.5-Ausdrücke); qualifiziert einen Quelltabellen-Port in einem Ausdruck. Die allgemeine Syntax lautet: `:SD.source_table.column_name`
:SEQ	Dies ist erforderlich für Ausdrücke mit einem Port in einer Sequenzgenerator-Umwandlung. Die allgemeine Syntax lautet: `:SEQ.sequence_generator_transformation.CURRVAL`
:SP	Dies ist erforderlich für Ausdrücke mit einem Rückgabewert aus einer nicht verbundenen Umwandlung einer gespeicherten Prozedur. Die allgemeine Syntax lautet: `:SP.stored_procedure_transformation( argument1, argument2, [, PROC_RESULT])` Die Argumente müssen den Argumenten in der nicht verbundenen Umwandlung der gespeicherten Prozedur entsprechen.
:TD	Dies ist erforderlich beim Referenzieren einer Zieltabelle in der PowerMart 3.5-Funktion LOOKUP. Die allgemeine Syntax lautet: `LOOKUP(:TD.SALES.ITEM_NAME, :TD.SALES.ITEM_ID, 10, :TD.SALES.PRICE, 15.99)`

String-Literale und numerische Literale

Ausdrücke können numerische oder String-Literale enthalten.

String-Literale müssen stets zwischen einfachen Anführungszeichen stehen. Beispiel:

```
'Alice Davis'
```

String-Literale unterscheiden zwischen Groß- und Kleinschreibung und können alle Zeichen außer das einfache Anführungszeichen enthalten. Der folgende String ist beispielsweise nicht zulässig:

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

Um die Rückgabe eines Strings mit einem einfachen Anführungszeichen zu ermöglichen, verwenden Sie die Funktion CHR:

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

Verwenden Sie keine einfachen Anführungszeichen in Kombination mit numerischen Literalen. Geben Sie einfach die Zahl ein, die Sie aufnehmen möchten. Beispiel:

```
.05
```

oder

```
$$Sales_Tax
```

Regeln und Richtlinien für die Ausdruckssyntax

Berücksichtigen Sie beim Formulieren von Ausdrücken die folgenden Regeln und Richtlinien:

- Eine Aggregator-Umwandlung kann nicht gleichzeitig einfache (mit nur einer Ebene) und verschachtelte Aggregatfunktionen enthalten.
- Wenn Sie sowohl einstufige als auch eingebettete Funktionen erstellen müssen, erstellen Sie separate Aggregator-Umwandlungen.
- Numerische Ausdrücke dürfen keine Strings enthalten.

  Beispiel: Der Ausdruck `1 + '1'` ist ungültig, da Addierung nur bei numerischen Datentypen möglich ist. Ganzzahlen und Strings können nicht addiert werden.
- Strings können nicht als numerische Parameter eingesetzt werden.

  Beispiel: Der Ausdruck `SUBSTR(TEXT_VAL, '1', 10)` ist ungültig, da die Startposition der Funktion SUBSTR ein Ganzzahlwert sein muss und kein String.
- Vergleichsoperatoren können keine gemischten Datentypen enthalten.

  Beispiel: Der Ausdruck `123.4 = '123.4'` ist ungültig, da ein Dezimalwert mit einem String verglichen wird.
- Sie können Werte aus Ports, String-Literalen oder numerischen Literalen, Variablen, Lookup-Umwandlungen, Umwandlungen von gespeicherten bzw. externen Prozeduren oder Ergebnissen anderer Ausdrücke übergeben.
- Sie können Werte aus Ports, String-Literalen oder numerischen Literalen, Lookup-Umwandlungen oder Ergebnissen anderer Ausdrücke übergeben.
- Verwenden Sie die Registerkarte „Ports“ im Ausdruckseditor, um einem Ausdruck einen Port-Namen hinzuzufügen. Wenn Sie einen Port in einer verbundenen Umwandlung umbenennen, überträgt Designer den geänderten Namen auf alle Ausdrücke in der Umwandlung.
- Verwenden Sie die Registerkarte „Ports“ im Ausdruckseditor, um einem Ausdruck einen Port-Namen hinzuzufügen. Wenn Sie einen Port in einer verbundenen Umwandlung umbenennen, überträgt das Developer-Tool den geänderten Namen auf alle Ausdrücke in der Umwandlung.
- Trennen Sie die einzelnen Argumente durch ein Komma.
- Mit Ausnahme der Literale unterscheidet die Umwandlungssprache nicht zwischen Groß- und Kleinschreibung.
- Außer bei den Literalen werden Leerzeichen von Designer und PowerCenter Integration Service nicht berücksichtigt.
- Außer bei den Literalen werden Leerzeichen vom Developer-Tool und Data Integration Service nicht berücksichtigt.
- Doppelpunkt (:), Komma (,) und Punkt (.) haben eine besondere Bedeutung und sollten nur zur Spezifizierung der Syntax verwendet werden.
- Data Integration Service interpretiert den Gedankenstrich (-) als Minuszeichen (Operator).
- Beim Übergeben eines Literalwerts an eine Funktion müssen String-Literale zwischen einfache Anführungszeichen gesetzt werden. Verwenden Sie bei numerischen Literalen keine Anführungszeichen.

Data Integration Service behandelt alle Stringwerte zwischen einfachen Anführungszeichen als Zeichenstring.

- Verwenden Sie bei der Übergabe von Mapping-Parametern bzw. -Variablen oder Arbeitsablauf-Variablen an eine Funktion in einem Ausdruck keine Anführungszeichen zur Bezeichnung der Parameter oder Variablen.
- Verwenden Sie bei der Übergabe von Mapping-Parametern an eine Funktion in einem Ausdruck keine Anführungszeichen zur Bezeichnung der Parameter.
- Verwenden Sie keine Anführungszeichen zum Bezeichnen von Ports.
- In einem Ausdruck können mehrere Ausdrücke verschachtelt werden, mit Ausnahme von Aggregatfunktionen: Verschachtelungen dürfen stets nur eine Aggregatfunktion enthalten. Data Integration Service beginnt die Auswertung mit der innersten Funktion.

Hinzufügen von Kommentaren zu Ausdrücken

Die Umwandlungssprache kennt zwei Kommentarbezeichner, mit deren Hilfe Sie Kommentare in Ausdrücke einfügen können:

- Zwei Gedankenstriche:

```
-- These are comments
```

- Zwei Schrägstriche:

```
// These are comments
```

Data Integration Service ignoriert den gesamten Text in einer Zeile, wenn diese mit einem der beiden Kommentarbezeichner beginnt. Wenn Sie beispielsweise zwei Strings verketten möchten, können Sie folgenden Ausdruck mit Kommentaren in der Mitte des Ausdrucks eingeben:

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

Data Integration Service ignoriert die Kommentare und wertet den Ausdruck wie folgt aus:

```
FIRST_NAME || LAST_NAME
```

Sie können Kommentare nicht auf einer neuen Zeile weiterführen:

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

In diesem Fall können Designer und Workflow Manager den Ausdruck nicht validieren, da die letzte Zeile keinen gültigen Ausdruck darstellt.

In diesem Fall kann das Developer-Tool den Ausdruck nicht validieren, da die letzte Zeile keinen gültigen Ausdruck darstellt.

Wenn Sie die Kommentare nicht direkt einbetten möchten, können Sie sie durch Klicken auf „Kommentar“ im Ausdruckseditor einfügen.

Reservierte Wörter

Einige Schlüsselwörter in der Umwandlungssprache, z. B. Konstanten, Operatoren und integrierte Variablen, sind für bestimmte Funktionen reserviert. Zu diesen gehören:

- :EXT
- :INFA
- :LKP
- :MCR
- :SD
- :SEQ
- :SP
- :TD
- AND
- DD_DELETE
- DD_INSERT
- DD_REJECT
- DD_UPDATE
- FALSE
- NOT
- NULL
- OR
- PROC_RESULT
- SESSSTARTTIME
- SPOUTPUT
- SYSDATE
- TRUE
- WORKFLOWSTARTTIME

Die folgenden Wörter sind für Arbeitsablaufsausdrücke reserviert:

- ABORTED
- DISABLED
- FAILED
- NOTSTARTED
- STARTED
- STOPPED
- SUCCEEDED

Hinweis: Sie können reservierte Wörter nicht dazu verwenden, Ports oder lokale Variablen zu benennen. Sie können nur in Umwandlungs- und Arbeitsablaufsausdrücken zum Einsatz kommen. Reservierte Wörter in Ausdrücken besitzen vordefinierte Bedeutungen.

Hinweis: Sie können reservierte Wörter nicht dazu verwenden, Ports oder lokale Variablen zu benennen. Sie können nur in Umwandlungsausdrücken zum Einsatz kommen. Reservierte Wörter in Ausdrücken besitzen vordefinierte Bedeutungen.

KAPITEL 2

Konstanten

Dieses Kapitel umfasst die folgenden Themen:

- DD_DELETE, 20
- DD_INSERT, 21
- DD_REJECT, 21
- DD_UPDATE, 22
- FALSE, 23
- NULL, 23
- TRUE, 25

DD_DELETE

Kennzeichnet Datensätze in einem Update-Strategie-Ausdruck zur Löschung. DD_DELETE entspricht dem Ganzzahl-Literal 2.

Hinweis: Die Konstante DD_DELETE darf nur in der Update-Strategie-Umwandlung eingesetzt werden. Verwenden Sie DD_DELETE anstelle des Ganzzahl-Literals 2, um die Fehlerbehebung komplexer numerischer Ausdrücke zu vereinfachen.

Beim Ausführen eines Arbeitsablaufs legen Sie in der datengesteuerten Update-Strategie fest, dass Datensätze aus den Zieldaten anhand dieses Flags gelöscht werden.

Beispiel

Beispiel: Folgender Ausdruck markiert die Einträge mit der ID-Nummer 1001 zur Löschung und alle anderen Elemente zur Einfügung:

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

Dieser Update-Strategie-Ausdruck erzielt dasselbe Ergebnis mit numerischen Literalen:

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

Hinweis: Der Ausdruck mit Konstanten ist leichter zu lesen als jener mit numerischen Literalen.

DD_INSERT

Kennzeichnet Datensätze in einem Update-Strategie-Ausdruck zur Einfügung. DD_INSERT entspricht dem Ganzzahl-Literal 0.

Hinweis: Die Konstante DD_INSERT darf nur in der Update-Strategie-Umwandlung eingesetzt werden. Verwenden Sie DD_INSERT anstelle des Ganzzahl-Literals 0, um die Fehlerbehebung komplexer numerischer Ausdrücke zu vereinfachen.

Beim Ausführen eines Arbeitsablaufs legen Sie in der datengesteuerten Update-Strategie fest, dass Datensätze anhand dieses Flags in die Zieldaten aufgenommen werden.

Beispiele

Die folgenden Beispiele ändern ein Mapping zur Berechnung der Monatsumsätze eines Vertriebsmitarbeiters, damit Sie die Umsatzzahlen eines bestimmten Mitarbeiters prüfen können.

Der folgende Update-Strategie-Ausdruck kennzeichnet die Umsatzzahlen eines Mitarbeiters zur Einfügung und lehnt alle übrigen Daten ab:

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

Dieser Update-Strategie-Ausdruck erzielt dasselbe Ergebnis mit numerischen Literalen:

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

Tipp: Der Ausdruck mit Konstanten ist leichter zu lesen als jener mit numerischen Literalen.

Der folgende Update-Strategie-Ausdruck sucht mithilfe von SESSSTARTTIME nur Bestellungen, die in den letzten zwei Tagen ausgeliefert wurden, und markiert sie zur Einfügung. Mit DATE_DIFF zieht der Ausdruck das Lieferdatum DATE_SHIPPED vom Systemdatum ab und gibt den Unterschied zwischen beiden Daten zurück. Da DATE-DIFF einen Double-Wert zurückgibt, wird die Unterschiedsangabe mithilfe von TRUNC abgeschnitten. Anschließend wird das Ergebnis mit dem Ganzzahl-Literal 2 verglichen. Wenn das Ergebnis größer als 2 ist, kennzeichnet der Ausdruck die Datensätze zur Ablehnung. Wenn das Ergebnis 2 oder weniger beträgt, markiert er sie zur Einfügung:

```
IIF( TRUNC( DATE_DIFF( SESSSTARTTIME, ORDERS_DATE_SHIPPED, 'DD' ), 0 ) > 2,DD_REJECT,
DD_INSERT )
```

DD_REJECT

Kennzeichnet Datensätze in einem Update-Strategie-Ausdruck zur Ablehnung. DD_REJECT entspricht dem Ganzzahl-Literal 3.

Hinweis: Die Konstante DD_REJECT darf nur in der Update-Strategie-Umwandlung eingesetzt werden. Verwenden Sie DD_REJECT anstelle des Ganzzahl-Literals 3, um die Fehlerbehebung komplexer numerischer Ausdrücke zu vereinfachen.

Beim Ausführen eines Arbeitsablaufs legen Sie in der datengesteuerten Update-Strategie fest, dass Datensätze aus den Zieldaten anhand dieses Flags abgelehnt werden.

Filtern oder Validieren Sie Daten mit DD_REJECT. Wenn Sie einen Datensatz zur Ablehnung markieren, überspringt Data Integration Service den Datensatz und trägt ihn in die Ablehnungsdatei der Sitzung ein.

Beispiele

Die folgenden Beispiele ändern ein Mapping zur Berechnung des aktuellen Monatsumsatzes und enthalten daher nur positive Werte.

Dieser Update-Strategie-Ausdruck kennzeichnet Datensätze kleiner als 0 zur Ablehnung und alle anderen zur Einfügung:

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

Dieser Ausdruck erzielt dasselbe Ergebnis mit numerischen Literalen:

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

Der Ausdruck mit Konstanten ist leichter zu lesen als jener mit numerischen Literalen.

Die folgenden datengesteuerten Beispiele nutzen DD_REJECT und IS_SPACES, um den Eintrag von Leerzeichen in eine Zeichenspalte einer Zieltabelle zu vermeiden. Dieser Ausdruck kennzeichnet Datensätze, die zur Gänze aus Leerzeichen bestehen, zur Ablehnung und alle anderen zur Einfügung:

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

DD_UPDATE

Kennzeichnet Datensätze in einem Update-Strategie-Ausdruck zur Aktualisierung. DD_UPDATE entspricht dem Ganzzahl-Literal 1.

Hinweis: Die Konstante DD_UPDATE darf nur in der Update-Strategie-Umwandlung eingesetzt werden. Verwenden Sie DD_UPDATE anstelle des Ganzzahl-Literals 1, um die Fehlerbehebung komplexer numerischer Ausdrücke zu vereinfachen.

Beim Ausführen eines Arbeitsablaufs legen Sie in der datengesteuerten Update-Strategie fest, dass Datensätze anhand dieses Flags in die Zieldaten aufgenommen werden.

Beispiele

Die folgenden Beispiele ändern ein Mapping zur Berechnung des aktuellen Monatsumsatzes. Das Mapping lädt die Umsatzzahlen eines Mitarbeiters.

Dieser Ausdruck kennzeichnet die Alex zugeordneten Datensätze als Updates und lehnt alle anderen ab:

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

Dieser Ausdruck erzielt dasselbe Ergebnis mit numerischen Literalen, indem Alex' Umsätze zum Update (1) und alle anderen Umsatzdatensätze zur Ablehnung (3) gekennzeichnet werden:

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

Der Ausdruck mit Konstanten ist leichter zu lesen als jener mit numerischen Literalen.

Der folgende Update-Strategie-Ausdruck sucht mithilfe von SYSDATE nur Bestellungen, die in den letzten zwei Tagen ausgeliefert wurden, und markiert sie zur Einfügung. Mit DATE_DIFF zieht der Ausdruck das Lieferdatum DATE_SHIPPED vom Systemdatum ab und gibt den Unterschied zwischen beiden Daten zurück. Da DATE-DIFF einen Double-Wert zurückgibt, wird die Unterschiedsangabe mithilfe von TRUNC abgeschnitten. Anschließend wird das Ergebnis mit dem Ganzzahl-Literal 2 verglichen. Wenn das Ergebnis größer als 2 ist, kennzeichnet der Ausdruck die Datensätze zur Ablehnung. Wenn das Ergebnis 2 oder weniger beträgt, markiert er die Datensätze für das Update: Andernfalls kennzeichnet er sie zur Ablehnung:

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

FALSE

Bezeichnet einen konditionalen Ausdruck. FALSE entspricht dem Ganzzahlwert 0.

Beispiel

Das folgende Beispiel verwendet FALSE in einem DECODE-Ausdruck, um auf Vergleichsergebnisse basierte Werte zurückzugeben. Dies ist besonders nützlich, wenn Sie mit einem einzigen Suchwert mehrere Suchvorgänge ausführen möchten:

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

NULL

Gibt an, dass ein Wert unbekannt oder nicht definiert ist. NULL ist nicht äquivalent mit einem leeren String (bei Zeichenspalten) oder 0 (bei numerische Spalten).

Obwohl Sie Ausdrücke formulieren können, die Null zurückgeben, akzeptieren Spalten mit den Einschränkungen „Nicht Null“ oder „Primärschlüssel“ keine Nullwerte. Wenn Data Integration Service versucht, einen Nullwert in eine Spalte mit einer dieser Einschränkungen zu schreiben, lehnt die Datenbank die entsprechende Zeile ab und Data Integration Service trägt sie stattdessen in die Ablehnungsdatei ein. Überlegen Sie sich beim Erstellen von Umwandlungen, wie Nullwerte gehandhabt werden sollen.

Verschiedene Funktionen können Nullwerte unterschiedlich verarbeiten. Beim Übergeben eines Nullwerts an eine Funktion kann sie 0 zurückgeben, NULL zurückgeben oder den Nullwert ignorieren.

Verwandte Themen:

- “Funktionen” auf Seite 52

Arbeiten mit Nullwerten in Booleschen Ausdrücken

Ausdrücke, in denen ein Nullwert mit einem Booleschen Ausdruck kombiniert wird, erzielen ANSI-kompatible Ergebnisse. Data Integration Service kommt beispielsweise zu folgendem Ergebnis:

- NULL UND TRUE = NULL
- NULL UND FALSE = FALSE

Nullwerte in Vergleichsausdrücken

Wenn Sie einen Nullwert in einem Ausdruck verwenden, der einen Vergleichsoperator enthält, erzeugt der Data Integration Service einen Nullwert. Um Spalten auf Nullwerte zu überprüfen, müssen Sie ISNULL() in Vergleichsausdrücken verwenden.

Um beispielsweise Zeilen zurückzugeben, die keine Nullwerte enthalten, verwenden Sie die ISNULL-Funktion anstelle der Konstante `!=`. Verwenden Sie beispielsweise `NOT ISNULL(Field_A)`.

Der folgende Ausdruck liefert einen Nullwert und die Filterumwandlung gibt keine Zeilen zurück: `Field_A!=NULL`.

Sie können die Lookup-Umwandlung auch so konfigurieren, dass Nullwerte in Vergleichsoperationen als HIGH oder LOW behandelt werden. Mithilfe der Eigenschaft „Null-Sortierung“ in der Lookup-Quelle können Sie festlegen, wie der Data Integration Service Nullwerte in Vergleichsausdrücken in der Lookup-Umwandlung verarbeitet.

Bei Verwendung eines Nullwerts in einem Ausdruck, der einen Vergleichsoperator enthält, gibt der Nullvergleich für den Datums-/Uhrzeitdatentyp einen Nullwert zurück.

Wenn Sie einen Nullwert in einem Ausdruck verwenden, der einen Vergleichsoperator enthält, erzeugt der Data Integration Service einen Nullwert. Sie können den Data Integration Service jedoch auch so konfigurieren, dass Nullwerte in Vergleichsoperationen als HIGH oder LOW behandelt werden.

Mithilfe der Eigenschaft „Null in Vergleichsoperatoren behandeln als“ können Sie festlegen, wie der Data Integration Service Nullwerte in Vergleichsausdrücken verarbeitet.

Diese Konfigurationseigenschaft des Data Integration Service beeinflusst das Verhalten folgender Vergleichsoperatoren in Ausdrücken:

```
=, !=, ^=, <>, >, >=, <, <=
```

Betrachten Sie beispielsweise folgende Ausdrücke:

```
NULL > 1
NULL = NULL
```

In der folgenden Tabelle wird beschrieben, wie der Data Integration Service die Ausdrücke auswertet:

Ausdruck	**Null in Vergleichsoperatoren behandeln als**		
	NULL	**HIGH**	**LOW**
NULL > 1	NULL	TRUE	FALSE
NULL = NULL	NULL	TRUE	TRUE

Nullwerte in Aggregatfunktionen

Data Integration Service behandelt Nullwerte in Aggregatfunktionen als Nullen. Wenn Sie einen Port oder eine Gruppe mit ausschließlich Nullwerten übergeben, gibt die Funktion NULL zurück.

Data Integration Service behandelt Nullwerte in Aggregatfunktionen als Nullen. Wenn Sie einen Port oder eine Gruppe mit ausschließlich Nullwerten übergeben, gibt die Funktion NULL zurück. Beim Konfigurieren von PowerCenter Integration Service können Sie jedoch festlegen, wie mit Nullwerten in Aggregatfunktionen umgegangen werden soll. Sie können bestimmen, ob Nullwerte in Aggregatfunktionen als 0 oder als NULL verarbeitet werden.

Nullwerte in Filterbedingungen

Wenn die Auswertung durch eine Filterbedingung NULL ergibt, wählt die Funktion den Datensatz nicht aus. Wenn die Auswertung durch die Filterbedingung für alle Datensätze im ausgewählten Port NULL ergibt, gibt die Aggregatfunktion NULL zurück (mit Ausnahme von COUNT, für das 0 zurückgegeben wird). Sie können Filterbedingungen mit Aggregatfunktionen und den Funktionen CUME, MOVINGAVG und MOVINGSUM verwenden.

Nullen und Operatoren

Ein Ausdruck mit Operatoren (mit Ausnahme des Stringoperators ||), der einen Nullwert enthält, ergibt bei der Auswertung immer Null. Beispiel: Die Auswertung des folgenden Ausdrucks ergibt Null:

```
8 * 10 - NULL
```

Um Ausdrücke auf Nullen zu prüfen, verwenden Sie die Funktion ISNULL.

TRUE

Gibt einen auf dem Ergebnis eines Vergleichs basierten Wert aus. TRUE entspricht der Ganzzahl 1.

Beispiel

Das folgende Beispiel nutzt TRUE in einem DECODE-Ausdruck, um auf Vergleichsergebnisse basierte Werte zurückzugeben. Dies ist besonders nützlich, wenn Sie mit einem einzigen Suchwert mehrere Suchvorgänge ausführen möchten:

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

KAPITEL 3

Operatoren

Dieses Kapitel umfasst die folgenden Themen:

- Rangordnung von Operatoren, 26
- Mathematische Operatoren, 27
- String-Operatoren, 28
- Vergleichsoperatoren, 29
- Logische Operatoren, 29

Rangordnung von Operatoren

Die Umwandlungssprache unterstützt die Verwendung mehrerer Operatoren sowie den Einsatz von Operatoren in verschachtelten Ausdrücken.

Bei einem Ausdruck mit mehreren Operatoren wertet Data Integration Service den Ausdruck in der folgenden Reihenfolge aus:

1. Mathematische Operatoren
2. String-Operatoren
3. Vergleichsoperatoren
4. Logische Operatoren

Die Auswertungsreihenfolge der Operatoren durch Data Integration Service wird in der folgenden Tabelle dargestellt. Operatoren der gleichen Rangordnung in einem Ausdruck werden von links nach rechts ausgewertet.

Die folgende Tabelle stellt die Rangordnung aller Operatoren in der Umwandlungssprache dar:

Operator	Bedeutung
( )	Klammern
+, -, NOT	Unäres Plus- und Minuszeichen und logischer Operator NOT (NICHT)
*, /,%	Multiplikation, Division, Modulo
+, -	Addition, Subtraktion
||	Verkettung

Operator	Bedeutung
<, <=, >, >=	Kleiner als, kleiner oder gleich, größer als, größer oder gleich
=, <>, !=, ^=	Gleich, nicht gleich, nicht gleich, nicht gleich
AND	Logischer Operator AND (UND) zur Angabe von Bedingungen
OR	Logischer Operator OR (ODER) zur Angabe von Bedingungen

Die Umwandlungssprache unterstützt auch die Verwendung von Operatoren in verschachtelten Ausdrücken. Wenn Ausdrücke Klammern enthalten, wertet Data Integration Service die Operationen in Klammern vor den Operationen außerhalb der Klammern aus. Die Operationen im innersten Klammerpaar werden als Erstes ausgewertet.

Beispiel: Je nachdem, wie Sie die Operationen verschachteln, ergibt die Gleichung 8 + 5 - 2 * 8 unterschiedliche Werte:

Gleichung	Rückgabewert
8 + 5 - 2 * 8	-3
8 + (5 - 2) * 8	32

Mathematische Operatoren

Mit mathematischen Operatoren können Sie mathematische Berechnungen für numerische Daten ausführen.

Die folgende Tabelle zeigt die mathematischen Operatoren in der Reihenfolge ihrer Rangordnung in der Umwandlungssprache:

Operator	Bedeutung
+, -	Unäres Plus- und Minuszeichen: Unäres Plus bezeichnet positive Werte. Unäres Minus bezeichnet negative Werte.
*, /, %	Multiplikation, Division, Modulo Modulo ist der Rest der Division zweier Ganzzahlen. Beispiel: 13 % 2 = 1, da 13 dividiert durch 2 gleich 6 mit einem Rest von 1
+, -	Addition, Subtraktion Der Additionsoperator (+) kann nicht zum Verketten von Strings verwendet werden. Zum Verketten von Strings dient der String-Operator ||. Für mathematische Berechnungen mit Datumswerten verwenden Sie die Datumsfunktionen.

Bei mathematischen Berechnungen mit Nullwerten gibt die Funktion NULL zurück.

Wenn Sie mathematische Operatoren in einem Ausdruck verwenden, müssen alle Operanden im Ausdruck numerisch sein. Beispiel: Der Ausdruck `1 + '1'` ist ungültig, da eine Ganzzahl mit einem String addiert wird. Der Ausdruck `1.23 + 4 / 2` ist gültig, da alle Operanden numerisch sind.

Hinweis: Die Umwandlungssprache umfasst integrierte Datumsfunktionen, um Ihnen Berechnungen mit Datum/Zeit-Werten zu ermöglichen:

Verwandte Themen:

- "Verständnis von Datumsberechnungen" auf Seite 51

String-Operatoren

Zum Verketten zweier Strings verwenden Sie den String-Operator ||. Der Operator || konvertiert Operanden aller Datentypen (außer Binär) vor der Verkettung in String-Datentypen:

Eingabewert	Rückgabewert
'alpha' || 'betisch'	alphabetisch
'alpha' || 2	alpha2
'alpha' || NULL	alpha

Der Operator || berücksichtigt vor- und nachgestellte Leerzeichen. Mit den Funktionen LTRIM und RTRIM können Sie vor dem Verketten vor- und nachgestellte Leerzeichen entfernen.

Nullen

Der Operator || ignoriert Nullwerte. Wenn jedoch beide Werte NULL sind, gibt der Operator || NULL zurück.

Beispiel

Das folgende Beispiel zeigt einen Ausdruck, in dem die Vor- und Nachnamen von Mitarbeitern aus zwei Spalten verkettet werden. Der Ausdruck entfernt die Leerzeichen am Ende des Vornamens und am Anfang des Nachnamens, verkettet den Vornamen mit einem Leerzeichen und verkettet ihn anschließend mit dem Nachnamen:

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

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

Hinweis: Sie können auch die Funktion CONCAT zum Verketten zweier Stringwerte verwenden. Mit dem Operator || erzielen Sie jedoch dasselbe Ergebnis bei geringerem Zeitaufwand.

Vergleichsoperatoren

Mit Vergleichsoperatoren können Sie Zeichenstrings oder numerische Strings vergleichen, Daten bearbeiten und den Wert TRUE (1) oder FALSE (0) zurückgeben.

Die folgende Tabelle zeigt die Vergleichsoperatoren in der Umwandlungssprache:

Operator	Bedeutung
=	Gleich
>	Größer als
<	Kleiner als
>=	Größer oder gleich
<=	Kleiner oder gleich
<>	Nicht gleich
!=	Nicht gleich
^=	Nicht gleich

Verwenden Sie Größer als (>) und Kleiner als (<), um numerische Werte zu vergleichen oder einen Zeilenbereich zurückzugeben, der auf der Sortierreihenfolge für einen Primärschlüssel in einem bestimmten Port beruht.

Beim Einsatz von Vergleichsoperatoren in einem Ausdruck müssen alle Operanden den gleichen Datentyp aufweisen. Beispiel: Der Ausdruck 123.4 = '123' ist ungültig, da ein Dezimalwert mit einem String verglichen wird. Die Ausdrücke 123.4 > 123 und ‘a’ != ‘b’ sind gültig, da die Operanden dem gleichen Datentyp angehören.

Beim Vergleichen eines Werts mit einem Nullwert lautet das Ergebnis NULL.

Wenn die Auswertung einer Filterbedingung NULL ergibt, gibt Integration Service NULL zurück.

Logische Operatoren

Logische Operatoren dienen zur Verarbeitung numerischer Daten. Ausdrücke, die einen numerischen Wert zurückgeben, werden folgendermaßen ausgewertet: TRUE für alle Werte außer 0, FALSE für 0 und NULL für NULL.

Die folgende Tabelle zeigt die logischen Operatoren in der Umwandlungssprache:

Operator	Bedeutung
NOT	Negiert das Ergebnis eines Ausdrucks. Beispiel: Wenn ein Ausdruck mit TRUE ausgewertet wird, gibt der Operator NOT den Wert FALSE zurück. Bei mit FALSE ausgewerteten Ausdrücken gibt NOT den Wert TRUE zurück.
AND	Verbindet zwei Bedingungen und gibt TRUE zurück, wenn die Auswertung beider Bedingungen TRUE ergibt. Gibt FALSE zurück, wenn eine der Bedingungen nicht TRUE ist.
OR	Verbindet zwei Bedingungen und gibt TRUE zurück, wenn eine der beiden Bedingungen TRUE ergibt. Gibt FALSE zurück, wenn beide Bedingungen FALSE lauten.

Nullen

Ausdrücke, in denen ein Nullwert mit einem Booleschen Ausdruck kombiniert wird, erzielen ANSI-kompatible Ergebnisse. Data Integration Service kommt beispielsweise zu folgendem Ergebnis:

- NULL UND TRUE = NULL
- NULL UND FALSE = FALSE

KAPITEL 4

Variablen

Dieses Kapitel umfasst die folgenden Themen:

- Integrierte Variablen, 31
- Transaktionssteuerungsvariablen, 37
- Lokale Variablen, 37

Integrierte Variablen

Die Umwandlungssprache umfasst integrierte Variablen. Die Rückgabe integrierter Variablen besteht entweder aus Laufzeit- oder aus Systemdaten. Laufzeitvariablen geben Informationen wie den Namen der Quell- und der Zieltabelle, Ordnernamen, Sitzungsausführungsmodus und den Instanznamen der aktuellen Arbeitsablaufausführung zurück. Systemvariablen geben Sitzungsstartzeit, Systemdatum und Startzeit des Arbeitsablaufs zurück.

Die Umwandlungssprache umfasst die integrierte Variable SYSDATE, die das Systemdatum zurückgibt. SYSDATE kann in einem Ausdruck verwendet werden. So können Sie SYSDATE etwa in einer DATE_DIFF-Funktion verwenden.

Integrierte Variablen können in Designer oder Workflow Manager in Ausdrücke eingefügt werden. Sie können die Systemvariable SYSDATE etwa in einer DATE_DIFF-Funktion einsetzen. Laufzeitvariablen können in Ausdrücken und Eingabefeldern eingesetzt werden, die Mapping- oder Arbeitsablaufsvariablen akzeptieren. Beispielsweise können Sie die Laufzeitvariable $PMWorkflowRunInstanceName als Teil eines Zielausgabedateinamens verwenden. Data Integration Service legt den Wert von integrierten Variablen fest. Sie können keine Werte für integrierte Variablen in einer Arbeitsablauf- oder Sitzungsparameterdatei definieren.

Integrierte Variablen können in Ausdrücken verwendet werden. Sie können die Systemvariable SYSDATE etwa in einer DATE_DIFF-Funktion einsetzen.

Die folgenden integrierten Variablen liefern Laufzeitdaten:

- $PM<SourceName>@TableName, $PM<TargetName>@TableName
- $PMFolderName
- $PMIntegrationServiceName
- $PMMappingName
- $PMRepositoryServiceName
- $PMRepositoryUserName
- $PMSessionName

- $PMSessionRunMode
- $PMWorkflowName
- $PMWorkflowRunId
- $PMWorkflowRunInstanceName

Die folgenden integrierten Variablen liefern Systemdaten:

- $$$SessStartTime
- SESSSTARTTIME
- SYSDATE
- WORKFLOWSTARTTIME

Die folgende Tabelle beschreibt den Verwendungsort der integrierten Variablen in Designer und Workflow Manager:

Name der Variablen	Designer	Workflow Manager
$PM<SourceName>@TableName, $PM<TargetName>@TableName,	- Ausdrücke - Eingabefelder, die Mapping-Variablen akzeptieren	- Eingabefelder, die Mapping-Variablen akzeptieren
$PMFolderName	- Ausdrücke - Eingabefelder, die Mapping-Variablen akzeptieren - Eingabefelder, die Arbeitsablaufsvariablen akzeptieren	- Ausdrücke - Eingabefelder, die Mapping-Variablen akzeptieren - Eingabefelder, die Arbeitsablaufsvariablen akzeptieren
$PMIntegrationServiceName	- Ausdrücke - Eingabefelder, die Mapping-Variablen akzeptieren - Eingabefelder, die Arbeitsablaufsvariablen akzeptieren	- Ausdrücke - Eingabefelder, die Mapping-Variablen akzeptieren - Eingabefelder, die Arbeitsablaufsvariablen akzeptieren
$PMMappingName	- Ausdrücke - Eingabefelder, die Mapping-Variablen akzeptieren	- Eingabefelder, die Mapping-Variablen akzeptieren
$PMRepositoryServiceName	- Ausdrücke - Eingabefelder, die Mapping-Variablen akzeptieren - Eingabefelder, die Arbeitsablaufsvariablen akzeptieren	- Ausdrücke - Eingabefelder, die Mapping-Variablen akzeptieren - Eingabefelder, die Arbeitsablaufsvariablen akzeptieren
$PMRepositoryUserName	- Ausdrücke - Eingabefelder, die Mapping-Variablen akzeptieren - Eingabefelder, die Arbeitsablaufsvariablen akzeptieren	- Ausdrücke - Eingabefelder, die Mapping-Variablen akzeptieren - Eingabefelder, die Arbeitsablaufsvariablen akzeptieren
$PMSessionName	- Ausdrücke - Eingabefelder, die Mapping-Variablen akzeptieren	- Eingabefelder, die Mapping-Variablen akzeptieren

Name der Variablen	Designer	Workflow Manager
$PMSessionRunMode	- Ausdrücke - Eingabefelder, die Mapping-Variablen akzeptieren	- Eingabefelder, die Mapping-Variablen akzeptieren
$PMWorkflowName	- Ausdrücke - Eingabefelder, die Mapping-Variablen akzeptieren - Eingabefelder, die Arbeitsablaufsvariablen akzeptieren	- Ausdrücke - Eingabefelder, die Mapping-Variablen akzeptieren - Eingabefelder, die Arbeitsablaufsvariablen akzeptieren
$PMWorkflowRunId	- Ausdrücke - Eingabefelder, die Mapping-Variablen akzeptieren - Eingabefelder, die Arbeitsablaufsvariablen akzeptieren	- Ausdrücke - Eingabefelder, die Mapping-Variablen akzeptieren - Eingabefelder, die Arbeitsablaufsvariablen akzeptieren
$PMWorkflowRunInstanceName	- Ausdrücke - Eingabefelder, die Mapping-Variablen akzeptieren - Eingabefelder, die Arbeitsablaufsvariablen akzeptieren	- Ausdrücke - Eingabefelder, die Mapping-Variablen akzeptieren - Eingabefelder, die Arbeitsablaufsvariablen akzeptieren
$$$SessStartTime	- Mapping- oder Mapplet-Filterbedingungen - Benutzerdefinierte Joins - SQL-Overrides	- Mapping- oder Mapplet-Filterbedingungen - Benutzerdefinierte Joins - SQL-Overrides
SESSSTARTTIME	- Ausdrücke	n/a
SYSDATE	- Ausdrücke	- Ausdrücke
WORKFLOWSTARTTIME	n/a	- Ausdrücke

$PM<SourceName>@TableName, $PM<TargetName>@TableName

$PM<*Quellname*>@TableName und $PM<*Zielname*>@TableName geben den Namen der Quell- bzw. Zieltabelle für relationale Quell- und Zielinstanzen als Stringwerte zurück. Sie können diese Variablen mit allen Funktionen verwenden, die String-Datentypen akzeptieren.

Der Name der Variable hängt vom Instanznamen der Quelle bzw. des Ziels ab. Beispiel: Bei einer Quellinstanz namens „Kunden“ lautet der Name der integrierten Variable $PMKunden@TableName. Wenn relationale Quellen oder Ziele Teil von Mapplets in einem Mapping sind, enthält der Name der integrierten Variable den Mapplet-Namen:

- $PM<*Mappletname*>.<*Quellname*>@TableName
- $PM<*Mappletname*>.<*Zielname*>@TableName

$PM<*Quellname*>@TableName und $PM<*Zielname*>@TableName können in Mappings und Mapplets verwendet werden. Beispiel: In einem Mapping mit mehreren relationalen Quellen können Sie mit $PM<*Quellname*>@TableName im Ausgabeport einer Ausdrucksumwandlung den Quellnamen in alle Zeilen des Ziels schreiben. Sie können diese Variablen auch in Eingabefeldern verwenden, die Mapping-Variablen akzeptieren.

$PMFolderName

$PMFolderName gibt den Namen des Repository-Ordners als Stringwert zurück. Sie können $PMFolderName mit allen Funktionen verwenden, die String-Datentypen akzeptieren.

Verwenden Sie $PMFolderName in Mappings, Mapplets, Arbeitsablauf-Links oder Arbeitsablaufaufgaben wie Zuweisung und Entscheidung. Außerdem können Sie $PMFolderName in Eingabefeldern einsetzen, die Mapping- oder Arbeitsablaufsvariablen akzeptieren.

$PMIntegrationServiceName

$PMIntegrationServiceName gibt den Namen von Data Integration Service zurück, in dem die Sitzung ausgeführt wird. Sie können $PMIntegrationServiceName mit allen Funktionen verwenden, die String-Datentypen akzeptieren. $PMIntegrationServiceName gibt den Namen von Data Integration Service als Stringwert zurück.

Verwenden Sie $PMIntegrationServiceName in Mappings, Mapplets, Arbeitsablauf-Links oder Arbeitsablaufaufgaben wie Zuweisung und Entscheidung. Außerdem können Sie $PMIntegrationServiceName in Eingabefeldern einsetzen, die Mapping- oder Arbeitsablaufsvariablen akzeptieren.

$PMMappingName

$PMMappingName gibt den Mapping-Namen als Stringwert zurück. Sie können $PMMappingName mit allen Funktionen verwenden, die String-Datentypen akzeptieren.

$PMMappingName kann in Mappings oder Mapplets eingesetzt werden. Sie können $PMMappingName auch in Eingabefeldern verwenden, die Mapping-Variablen akzeptieren.

$PMRepositoryServiceName

$PMRepositoryServiceName gibt den Namen von Model Repository Service als Stringwert zurück. Sie können $PMRepositoryServiceName mit allen Funktionen verwenden, die String-Datentypen akzeptieren.

Verwenden Sie $PMRepositoryServiceName in Mappings, Mapplets, Arbeitsablauf-Links oder Arbeitsablaufaufgaben wie Zuweisung und Entscheidung. Außerdem können Sie $PMRepositoryServiceName in Eingabefeldern einsetzen, die Mapping- oder Arbeitsablaufsvariablen akzeptieren.

$PMRepositoryUserName

$PMRepositoryUserName gibt den Namen des Repository-Benutzers zurück, der die Sitzung ausführt. Sie können $PMRepositoryUserName mit allen Funktionen verwenden, die String-Datentypen akzeptieren. $PMRepositoryUserName gibt den Benutzernamen als Stringwert zurück.

Verwenden Sie $PMRepositoryUserName in Mappings, Mapplets, Arbeitsablauf-Links oder Arbeitsablaufaufgaben wie Zuweisung und Entscheidung. Außerdem können Sie $PMRepositoryUserName in Eingabefeldern einsetzen, die Mapping- oder Arbeitsablaufsvariablen akzeptieren.

$PMSessionName

$PMSessionName gibt den Sitzungsnamen als Stringwert zurück. Sie können $PMSessionName mit allen Funktionen verwenden, die String-Datentypen akzeptieren.

$PMSessionName kann in Mappings oder Mapplets eingesetzt werden. Sie können $PMSessionName auch in Eingabefeldern verwenden, die Mapping-Variablen akzeptieren.

$PMSessionRunMode

$PMSessionRunMode gibt den Ausführungsmodus der Sitzung (*Normal* oder *Wiederherstellung*) als Stringwert zurück. Sie können $PMSessionRunMode mit allen Funktionen verwenden, die String-Datentypen akzeptieren.

$PMSessionRunMode kann in Mappings oder Mapplets eingesetzt werden. Sie können $PMSessionRunMode auch in Eingabefeldern verwenden, die Mapping-Variablen akzeptieren.

$PMWorkflowName

$PMWorkflowName gibt den Namen des Arbeitsablaufs als Stringwert zurück. Sie können $PMWorkflowName mit allen Funktionen verwenden, die String-Datentypen akzeptieren.

Verwenden Sie $PMWorkflowName in Mappings, Mapplets, Arbeitsablauf-Links oder Arbeitsablaufaufgaben wie Zuweisung und Entscheidung. Außerdem können Sie $PMWorkflowName in Eingabefeldern einsetzen, die Mapping- oder Arbeitsablaufsvariablen akzeptieren.

$PMWorkflowRunId

Jede Arbeitsablaufausführung weist eine eindeutige Ablaufausführungs-ID auf. $PMWorkflowRunId gibt die Ablaufausführungs-ID als Stringwert zurück. Sie können $PMWorkflowRunId mit allen Funktionen verwenden, die String-Datentypen akzeptieren.

Verwenden Sie $PMWorkflowRunId in Mappings, Mapplets, Arbeitsablauf-Links oder Arbeitsablaufaufgaben wie Zuweisung und Entscheidung. Außerdem können Sie $PMWorkflowRunId in Eingabefeldern einsetzen, die Mapping- oder Arbeitsablaufsvariablen akzeptieren. Beispiel: Sie konfigurieren einen Arbeitsablauf zur gleichzeitigen Ausführung mit demselben Instanznamen und möchten den Status der einzelnen Ausführungen mithilfe einer Anwendung eines Drittanbieters verfolgen. Sie verwenden $PMWorkflowRunId in einem Shell-Befehl nach der Sitzung, um die Ablaufausführungs-ID an die Anwendung zu übergeben.

$PMWorkflowRunInstanceName

$PMWorkflowRunInstanceName gibt den Namen der Arbeitsablaufausführung als Stringwert zurück. Sie können $PMWorkflowRunInstanceName mit allen Funktionen verwenden, die String-Datentypen akzeptieren.

Verwenden Sie $PMWorkflowRunInstanceName in Mappings, Mapplets, Arbeitsablauf-Links oder Arbeitsablaufaufgaben wie Zuweisung und Entscheidung. Außerdem können Sie $PMWorkflowRunInstanceName in Eingabefeldern einsetzen, die Mapping- oder Arbeitsablaufsvariablen akzeptieren. Beispiel: Bei gleichzeitigen Arbeitsabläufen mit eindeutigen Instanznamen können Sie für jede Ausführungsinstanz eindeutige Zieldateien erstellen, indem Sie den Zielausgabedateinamen in den Sitzungseigenschaften auf „OutFile_$PMWorkflowRunInstanceName.txt“ setzen.

Dasselbe gilt, wenn Sie mithilfe eines Shell-Befehls nach der Sitzung eine Indikatordatei zur Nutzung durch eine vordefinierte Event-Wait-Aufgabe erstellen möchten. Im Shell-Befehl, der die Indikatordatei generiert, geben Sie PMWorkflowRunInstanceName im Namen der Indikatordatei an, um sicherzustellen, dass eine Ausführungsinstanz nicht die Indikatordatei einer anderen Arbeitsablauf-Ausführungsinstanz löscht.

SESSSTARTTIME

Nach der Initialisierung der Sitzung durch den Integrationsdienst gibt SESSSTARTTIME die aktuellen Datums- und Zeitwerte auf dem Knoten zurück, auf dem die Sitzung ausgeführt wird. Sie können SESSSTARTTIME mit allen Funktionen verwenden, die Datum/Zeit-Umwandlungsdatentypen akzeptieren. SESSSTARTTIME ist als Wert des Datum/Zeit-Umwandlungsdatentyps gespeichert.

SESSSTARTTIME kann in Mappings oder Mapplets eingesetzt werden. Sie können SESSSTARTTIME nur in der Ausdruckssprache referenzieren.

Beispiel

Der folgende Ausdruck verwendet $$$SESSSTARTTIME in der Quellfilterbedingung eines Quellqualifikators zur Durchführung einer inkrementellen Extraktion. Der Ausdruck spezifiziert einen Datumsbereich aller Tage der Woche vor der Sitzungsinitialisierung durch Data Integration Service. Der Ausdruck sucht mithilfe der Funktion DATE_DIFF den Unterschied in der Anzahl von Tagen zwischen den Werten ORDER_DATE und $$ $SessStartTime. Wenn der Unterschied zwischen beiden Datumsangaben kleiner als oder gleich sieben Tage ist, extrahiert Data Integration Service die betreffende Zeile aus der Quelle:

```
DATE_DIFF(DAY, ORDER_DATE, '$$$SessStartTime') <= 7
```

SYSDATE

SYSDATE gibt für jede Zeile, die die Umwandlung durchläuft, das aktuelle Datum und die aktuelle Uhrzeit (sekundengenau) des Knotens zurück, auf dem die Sitzung ausgeführt wird. SYSDATE ist als Wert des Datentyps „Umwandlungs-Datum/Zeit“ gespeichert.

SYSDATE gibt für jede Zeile, die die Umwandlung durchläuft, das aktuelle Datum und die aktuelle Uhrzeit (sekundengenau) des Knotens zurück, auf dem die Daten verarbeitet werden. SYSDATE ist als Wert des Datentyps „Umwandlungs-Datum/Zeit“ gespeichert.

Zum Erfassen eines statischen Systemdatums ersetzen Sie SYSDATE durch die Variable SESSSTARTTIME.

Beispiel

Der folgende Ausdruck sucht mithilfe von SYSDATE nur Bestellungen, die in den letzten zwei Tagen ausgeliefert wurden, und markiert sie zur Einfügung. Mit DATE_DIFF zieht Data Integration Service den Wert DATE_SHIPPED vom Systemdatum ab und gibt den Unterschied zwischen beiden Datumsangaben zurück. Da DATE_DIFF einen Double-Wert zurückgibt, schneidet der Ausdruck die Differenz ab. Anschließend wird das Ergebnis mit dem Ganzzahl-Literal 2 verglichen. Wenn das Ergebnis größer als 2 ist, markiert der Ausdruck die Zeilen zur Ablehnung. Wenn das Ergebnis 2 oder weniger beträgt, markiert er sie zur Einfügung.

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

WORKFLOWSTARTTIME

WORKFLOWSTARTTIME gibt Datum und Uhrzeit der Startzeit des Arbeitsablaufs auf dem Knoten zurück, auf dem Data Integration Service den Arbeitsablauf hostet. Sie können WORKFLOWSTARTTIME mit allen Funktionen verwenden, die Umwandlungs-Datum/Zeit-Datentypen akzeptieren. WORKFLOWSTARTTIME ist als Wert des Datentyps „Umwandlungs-Datum/Zeit“ gespeichert.

Verwenden Sie WORKFLOWSTARTTIME in Arbeitsablauf-Links oder Arbeitsablaufaufgaben wie Zuweisung und Entscheidung. Sie können WORKFLOWSTARTTIME nur in der Ausdruckssprache referenzieren.

Beispiel

Der folgende Ausdruck verwendet WORKFLOWSTARTTIME zum Anzeigen der Anzahl der Minuten zwischen der Startzeit des Arbeitsablaufs und der Startzeit einer Aufgabe im Arbeitsablauf. Mithilfe der SQL-Funktion DATE_DIFF zieht Data Integration Service den Startzeitpunkt der Aufgabe vom Wert WORKFLOWSTARTTIME ab und gibt das Ergebnis als Anzahl von Tagen zurück:

```
DATE_DIFF(WORKFLOWSTARTTIME, $s_EmployeeData.StartTime, 'MI')
```

Transaktionssteuerungsvariablen

Transaktionssteuerungsvariablen definieren die Bedingungen zum Übernehmen (Commit) oder Rückgängigmachen (Rollback) von Transaktionen bei der Verarbeitung von Datenbankzeilen. Sie verwenden diese Variablen in Transaktionssteuerungsausdrücken, die Sie im Ausdruckseditor erstellen. Transaktionssteuerungsausdrücke prüfen mithilfe der Funktion IIF alle Zeilen anhand einer Bedingung. Je nach Rückgabewert der Bedingung übernimmt Data Integration Service die Zeile, macht sie rückgängig oder nimmt keine Transaktionsänderungen vor.

Das folgende Beispiel nutzt Transaktionssteuerungsvariablen, um zu bestimmen, wo eine Zeile verarbeitet werden soll:

```
IIF (NEWTRAN=1, TC_COMMIT_BEFORE, TC_CONTINUE_TRANSACTION)
```

Wenn NEWTRAN=1, löst die Variable TC_COMMIT_BEFORE einen Commit aus, bevor die aktuelle Zeile verarbeitet wird. Andernfalls erzwingt die Variable TC_CONTINUE_TRANSACTION die Verarbeitung der Zeile in der aktuellen Transaktion.

Verwenden Sie beim Erstellen von Transaktionssteuerungsausdrücken im Ausdruckseditor folgende Variablen:

- **TC_CONTINUE_TRANSACTION:** Data Integration Service nimmt an der aktuellen Zeile keine Transaktionsänderung vor. Dies ist der Standardwert bei den Transaktionssteuerungsvariablen.
- **TC_COMMIT_BEFORE:** Data Integration Service übernimmt die Transaktion (Commit), beginnt eine neue Transaktion und schreibt die aktuelle Zeile in das Ziel. Die aktuelle Zeile befindet sich in der neuen Transaktion.
- **TC_COMMIT_AFTER:** Data Integration Service schreibt die aktuelle Zeile in das Ziel, übernimmt die Transaktion (Commit) und beginnt eine neue Transaktion. Die aktuelle Zeile befindet sich in der übernommenen Transaktion.
- **TC_ROLLBACK_BEFORE:** Data Integration Service macht die aktuelle Transaktion rückgängig (Rollback), beginnt eine neue Transaktion und schreibt die aktuelle Zeile in das Ziel. Die aktuelle Zeile befindet sich in der neuen Transaktion.

Lokale Variablen

Wenn Sie lokale Variablen in einem Mapping verwenden, können Sie sie in allen Umwandlungsausdrücken im Mapping einsetzen. Wenn Sie beispielsweise eine komplexe Steuerberechnung in einem Mapping verwenden, kann es sinnvoll sein, den Ausdruck nur einmal zu erstellen und ihn als Variable festzulegen. Dies erhöht die Leistung, da Data Integration Service die Berechnung nur einmal ausführen muss.

Lokale Variablen sind sinnvoll in Verbindung mit gespeicherten Prozedurausdrücken, um mehrere Rückgabewerte zu erfassen.

KAPITEL 5

Datumsangaben

Dieses Kapitel umfasst die folgenden Themen:

- Datumsangaben – Übersicht, 38
- Datumsformatstrings, 42
- TO_CHAR-Formatstrings, 44
- TO_DATE- und IS_DATE-Formatstrings, 47
- Verständnis von Datumsberechnungen, 51

Datumsangaben – Übersicht

Die Umwandlungssprache enthält eine Reihe von Datumsfunktionen und integrierten Datumsvariablen für die Umwandlung von Datumsangaben. Mit den Datumsfunktionen können Sie Datumswerte runden, abschneiden oder vergleichen, Teile von Datumsangaben extrahieren oder Datumsangaben berechnen. Sie können beliebige Werte mit dem Datum-Datentyp an eine Datumsfunktion übergeben.

Datumsvariablen dienen zum Erfassen des aktuellen Datums oder der Startzeit einer Sitzung auf dem Knoten von Data Integration Service.

Datumsvariablen dienen zum Erfassen des aktuellen Datums auf dem Knoten von Data Integration Service.

Die Umwandlungssprache liefert auch folgende Sätze von Formatstrings:

- **Datumsformatstrings:** Dienen gemeinsam mit Datumsfunktionen zum Festlegen der Teile einer Datumsangabe.
- **TO_CHAR-Formatstrings:** Dienen zum Formatieren des Rückgabestrings.
- **TO_DATE- und IS_DATE-Formatstrings:** Dienen zum Formatieren von Strings, die in Datumsangaben umgewandelt oder getestet werden sollen.

Datum/Zeit-Datentyp

Informatica verwendet generische Datentypen zum Umwandeln von Daten aus unterschiedlichen Quellen. Zu diesen Umwandlungsdatentypen gehört der Datum/Zeit-Datentyp, der Datetime-Werte bis zur Nanosekunde unterstützt. Informatica speichert Datumsangaben intern im Binärformat.

Die Datumsfunktionen akzeptieren nur Datetime-Werte. Um einen String an eine Datumsfunktion zu übergeben, konvertieren Sie ihn zuerst mithilfe von TO_DATE in einen Datetime-Wert. Beispiel: Der folgende Ausdruck konvertiert einen String-Port in Datetime-Werte und fügt dann jedem Datum einen Monat hinzu:

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

Im Gregorianischen Kalendersystem sind Daten zwischen 1 und 9999 zulässig.

Julianisches Datum, Modifiziertes Julianisches Datum und Gregorianischer Kalender

Das einzige gültige Kalendersystem ist das Gregorianische. Datumsangaben aus dem *Julianischen Kalender* werden in Informatica nicht unterstützt. Der Julianische Kalender darf jedoch nicht mit dem *Julianischen Datum* (Julian Day, JD) oder dem Modifizierten Julianischen Datum (Modified Julian Day, MJD) verwechselt werden.

MJD-Formate können mit dem J-Formatstring geändert werden. Das MJD entspricht der Anzahl von Tagen, die seit 00:00:00 Uhr (Mitternacht) am 1. Januar 4713 v. Chr. vergangen sind. Definitionsgemäß besteht das MJD aus einer Zeitkomponente in Dezimalzahlen, die den jeweils verstrichenen Anteil des aktuellen 24-Stunden-Zeitraums angibt. Der J-Formatstring konvertiert diese Zeitkomponente nicht.

Beispiel: Der folgende TO_DATE-Ausdruck wandelt Strings im Port SHIP_DATE_MJD_STRING in Datumswerte im Standarddatumsformat um:

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

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

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

Da der J-Formatstring die Zeitkomponente der Datumsangabe nicht berücksichtigt, wird die Uhrzeit in den Rückgabewerten auf 00: 00: 00.000000000 gesetzt.

Den J-Formatstring können Sie auch in TO_CHAR-Ausdrücken einsetzen. So können Sie beispielsweise den J-Formatstring in einem TO_CHAR-Ausdruck dazu nutzen, Datumswerte in als Strings ausgedrückte MJD-Werte zu konvertieren. Beispiel:

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

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

Hinweis: In TO_CHAR-Ausdrücken ignoriert Data Integration Service die Zeitkomponente des Datums.

Datumsangaben im Jahr 2000

Alle Datumsfunktionen der Umwandlungssprache unterstützen das Jahr 2000. Informatica Developer unterstützt Datumsangaben zwischen 1 und 9999.

RR-Formatstring

Die Umwandlungssprache enthält den RR-Formatstring zum Konvertieren von Strings mit zweistelligen Jahresangaben in Datumsangaben. Zusammen mit TO_DATE können Sie mit dem RR-Formatstring einen String im Format MM/DD/RR in eine Datumsangabe konvertieren. Die Konvertierung mit dem RR-Formatstring fällt je nach aktuellem Jahr unterschiedlich aus.

- **Aktuelles Jahr zwischen 0 und 49:** Wenn das aktuelle Jahr zwischen 0 und 49 liegt (z. B. 2003) und das Jahr aus dem Quellstring ebenfalls zwischen 0 und 49 liegt, gibt Data Integration Service das aktuelle Jahrhundert plus die zweistellige Jahresangabe aus dem Quellstring zurück. Wenn der Quellstring jedoch eine Jahresangabe zwischen 50 und 99 enthält, nimmt die Rückgabe das vorherige Jahrhundert plus das zweistellige Jahr aus dem Quellstring.
- **Aktuelles Jahr zwischen 50 und 99:** Wenn das aktuelle Jahr zwischen 50 und 99 liegt (z. B. 1998), das Jahr aus dem Quellstring jedoch zwischen 0 und 49 liegt, gibt Data Integration Service das darauffolgende Jahrhundert plus die zweistellige Jahresangabe aus dem Quellstring zurück. Wenn der Quellstring eine Jahresangabe zwischen 50 und 99 enthält, gibt Data Integration Service das aktuelle Jahrhundert plus die angegebene zweistellige Jahresangabe zurück.

Die folgende Tabelle bietet einen Überblick über die Konvertierung von Datumsangaben mit dem RR-Formatstring:

Aktuelles Jahr	Quelljahr	Rückgabe durch RR-Formatstring
0-49	0-49	Aktuelles Jahrhundert
0-49	50-99	Vorheriges Jahrhundert
50-99	0-49	Folgendes Jahrhundert
50-99	50-99	Aktuelles Jahrhundert

Beispiel

Der folgende Ausdruck erzielt dieselben Rückgabewerte für alle aktuellen Jahre zwischen 1950 und 2049:

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

ORDER_DATE	RETURN_VALUE
'04/12/98'	04/12/1998 00:00:00.000000000
'11/09/01'	11/09/2001 00:00:00.000000000

Unterschied zwischen YY- und RR-Formatstrings

Informatica Developer enthält auch einen YY-Formatstring. Sowohl der RR- als auch der YY-Formatstring spezifizieren zweistellige Jahresangaben. YY und RR erzielen bei allen Datumsfunktionen identische Ergebnisse – mit Ausnahme von TO_DATE. In TO_DATE-Ausdrücken kommen RR und YY zu unterschiedlichen Ergebnissen.

Die folgende Tabelle zeigt die Ergebnisse pro Formatstring:

String	Aktuelles Jahr	TO_DATE(String, 'MM/DD/RR')	TO_DATE(String, 'MM/DD/YY')
04/12/98	1998	04/12/1998 00:00:00.000000000	04/12/1998 00:00:00.000000000
11/09/01	1998	11/09/2001 00:00:00.000000000	11/09/1901 00:00:00.000000000
04/12/98	2003	04/12/1998 00:00:00.000000000	04/12/2098 00:00:00.000000000
11/09/01	2003	11/09/2001 00:00:00.000000000	11/09/2001 00:00:00.000000000

Bei Datumsangaben im Jahr 2000 und darüber hinaus erzielt der YY-Formatstring weniger sinnvolle Ergebnisse als RR. Verwenden Sie den RR-Formatstring für alle Datumsangaben im 21. Jahrhundert.

Datumsangaben in relationalen Datenbanken

Im Allgemeinen enthalten die Daten in einer relationalen Datenbank einen Datums- und einen Zeitwert. Das Datum enthält Monat, Tag und Jahr, während die Zeit Stunden, Minuten, Sekunden und Subsekunden umfassen kann. Sie können Datetime-Daten an jede beliebige Datumsfunktion übergeben.

Datumsangaben in Einfachdateien

Verwenden Sie die TO_DATE-Funktion, um Strings in Datetime-Werte zu konvertieren. Sie können auch mit IS_DATE prüfen, ob ein String nicht bereits ein gültiges Datum ist, bevor Sie ihn mit TO_DATE konvertieren. Die Datumsfunktionen der Umwandlungssprache akzeptieren nur Datumswerte. Um einen String an eine Datumsfunktion übergeben zu können, müssen Sie ihn zuerst mithilfe der TO_DATE-Funktion in einen Datum/Zeit-Wert (Datetime) konvertieren.

Standarddatumsformat

Data Integration Service nutzt ein Standarddatumsformat zum Speichern und Verändern von Strings, die ein Datum angeben. Um das Standarddatumsformat festzulegen, geben Sie für eine Sitzung oder ein Sitzungskonfigurationsobjekt auf der Registerkarte „Konfigurationsobjekt“ ein Datumsformat im Attribut „Datetime-Formatstring“ ein. Standardmäßig lautet das Datumsformat MM/DD/YYYY HH24:MI:SS.US.

Data Integration Service nutzt ein Standarddatumsformat zum Speichern und Verändern von Strings, die ein Datum angeben. Um das Standarddatumsformat festzulegen, geben Sie in der Daten-Viewer-Konfiguration ein Datumsformat im Attribut „Datetime-Formatstring“ ein. Standardmäßig lautet das Datumsformat MM/DD/YYYY HH24:MI:SS.US.

Da Informatica Daten im Binärformat speichert, verwendet Data Integration Service das Standarddatumsformat, wenn Sie folgende Aktionen ausführen:

- **Konvertieren eines Datums in einen String durch Verbinden eines Datum/Zeit-Ports mit einem String-Port:** Data Integration Service konvertiert das Datum in einen String in dem Datumsformat, das im Sitzungskonfigurationsobjekt definiert wurde.
- **Konvertieren eines Strings in ein Datum durch Verbinden eines String-Ports mit einem Datum/Zeit-Port:** Data Integration Service erwartet Stringwerte in dem Datumsformat, das im Sitzungskonfigurationsobjekt definiert ist. Wenn der Eingabewert nicht mit diesem Format übereinstimmt oder ein ungültiges Datum ist, überspringt Data Integration Service die Zeile. Wenn der String in diesem Format vorliegt, konvertiert Data Integration Service den String in einen Datumswert.

- **Konvertieren von Datumsangaben in Strings mit TO_CHAR(Datum, [Format_String]):** Wenn Sie den Formatstring auslassen, gibt Data Integration Service den String in dem in den Sitzungseigenschaften definierten Datumsformat zurück. Wenn Sie einen Formatstring angeben, gibt Data Integration Service einen String im angegebenen Format zurück.
- **Konvertieren von Strings in Datumsangaben mit TO_DATE(Datum, [Format_String]):** Wenn Sie den Formatstring auslassen, erwartet Data Integration Service einen String in dem in den Sitzungseigenschaften definierten Datumsformat. Wenn Sie einen Formatstring angeben, erwartet Data Integration Service einen String im angegebenen Format.
- **Konvertieren eines Datums in einen String durch Verbinden eines Datum/Zeit-Ports mit einem String-Port:** Data Integration Service konvertiert das Datum in einen String in dem Datumsformat, das in der Daten-Viewer-Konfiguration definiert wurde.
- **Konvertieren eines Strings in ein Datum durch Verbinden eines String-Ports mit einem Datum/Zeit-Port:** Data Integration Service erwartet Stringwerte in dem Datumsformat, das im Daten-Viewer-Konfiguration definiert ist. Wenn der Eingabewert nicht mit diesem Format übereinstimmt oder ein ungültiges Datum ist, überspringt Data Integration Service die Zeile. Wenn der String in diesem Format vorliegt, konvertiert Data Integration Service den String in einen Datumswert.
- **Konvertieren von Datumsangaben in Strings mit TO_CHAR(Datum, [Format_String]):** Wenn Sie den Formatstring auslassen, gibt Data Integration Service den String in dem in der Daten-Viewer-Konfiguration definierten Datumsformat zurück. Wenn Sie einen Formatstring angeben, gibt Data Integration Service einen String im angegebenen Format zurück.
- **Konvertieren von Strings in Datumsangaben mit TO_DATE(Datum, [Format_String]):** Wenn Sie den Formatstring auslassen, erwartet Data Integration Service einen String in dem in der Daten-Viewer-Konfiguration definierten Datumsformat. Wenn Sie einen Formatstring angeben, erwartet Data Integration Service einen String im angegebenen Format.

Das standardmäßige Datumsformat MM/DD/YYYY HH24:MI:SS.US besteht aus folgenden Komponenten:

- Monat (Januar = 01, September = 09)
- Tag (des Monats)
- Jahr (vierstellig, z. B: 1998)
- Stunde (im 24-Stunden-Format, z. B. 00:00:00AM = 0, 01:00:00AM = 1, 12:00:00PM = 12, 11:00:00PM = 23)
- Minuten
- Sekunden
- Mikrosekunden

Datumsformatstrings

Sie können die Daten mit einer Kombination aus Formatstrings und Datumsfunktionen auswerten. Datumsformatstrings sind nicht internationalisiert und müssen genau so wie in der folgenden Tabelle eingegeben werden.

Die folgende Tabelle bietet einen Überblick über die Formatstrings, die einen Teil einer Datumsangabe spezifizieren:

Formatstring	Beschreibung
D, DD, DDD, DAY, DY, J	Tage (01-31): Geben die gesamte Tageskomponente eines Datums an. Beispiel: Wenn Sie „12-APR-1997“ an eine Datumsfunktion übergeben, spezifizieren Sie mit einem dieser Formatstrings die Komponente „12“.
HH, HH12, HH24	Stunde des Tages (0-23): Geben die gesamte Stundenkomponente eines Datums an. Beispiel: Wenn Sie „12-APR-1997 2:01:32 PM“ an eine Datumsfunktion übergeben, spezifizieren Sie mit HH, HH12 oder HH24 die Stundenkomponente des Datums.
MI	Minuten (0-59)
MM, MON, MONTH	Monat (1-12): Geben die gesamte Monatskomponente eines Datums an. Beispiel: Wenn Sie „12-APR-1997“ an eine Datumsfunktion übergeben, spezifizieren Sie mit MM, MON oder MONTH die Komponente „APR“.
MS	Millisekunden (0-999)
NS	Nanosekunden (0-999999999)
SS, SSSS	Sekunden (0-59)
US	Mikrosekunden (0-999999)
Y, YY, YYY, YYYY, RR	Jahreskomponente des Datums (0001 bis 9999): Geben die gesamte Jahreskomponente eines Datums an. Beispiel: Wenn Sie „12-APR-1997“ an eine Datumsfunktion übergeben, spezifizieren Sie mit Y, YY, YYY oder YYYY die Komponente „1997“.

Hinweis: Bei Formatstrings muss nicht auf Groß-/Kleinschreibung geachtet werden. Sie müssen immer zwischen einfachen Anführungszeichen stehen.

Die folgende Tabelle beschreibt Datumsfunktionen, in denen Eingabedaten mithilfe von Datumsformatstrings ausgewertet werden:

Funktion	Beschreibung
ADD_TO_DATE	Der Teil des Datums, der geändert werden soll
DATE_DIFF	Der Teil des Datums, der zur Berechnung des Unterschieds zwischen zwei Datumsangaben verwendet werden soll
GET_DATE_PART	Der Teil des Datums, der zurückgegeben werden soll; diese Funktion gibt einen Ganzzahlwert zurück, der auf dem Standarddatumsformat basiert.
IS_DATE	Das Datum, das überprüft werden soll
ROUND	Der Teil des Datums, der gerundet werden soll
SET_DATE_PART	Der Teil des Datums, der geändert werden soll
SYSTIMESTAMP	Die Genauigkeit des Zeitstempels
TO_CHAR (Datum)	Der Zeichenstring

Funktion	Beschreibung
TO_DATE	Der Zeichenstring
TRUNC (Datum)	Der Teil des Datums, der abgeschnitten werden soll

TO_CHAR-Formatstrings

Die Funktion TO_CHAR konvertiert einen Datum/Zeit-Datentyp in einen String im angegebenen Format. Sie können das gesamte Datum oder nur einen Teil davon in einen String konvertieren. Mit TO_CHAR können Sie beispielsweise das Format von Datumsangaben für Berichtszwecke zu Strings ändern.

TO_CHAR wird generell dann eingesetzt, wenn das Ziel eine Einfachdatei oder eine Datenbank ist, die den Datum/Zeit-Datentyp nicht unterstützt.

Die folgende Tabelle bietet einen Überblick über die Formatstrings für Datumsangaben in der Funktion TO_CHAR:

Formatstring	Beschreibung
AM, A.M., PM, P.M.	Vormittags-/Nachmittagsangabe: Verwenden Sie diese Formatstrings, um Stunden in AM oder PM anzugeben. AM und PM geben die gleichen Werte zurück wie A.M. und P.M.
D	Tag des Woche (1-7), wobei Sonntag gleich 1
TAG	Name des Tages, bis zu neun Zeichen (zum Beispiel Mittwoch);
DD	Tag des Monats (1-31)
DDD	Tag des Jahres (001-366, einschließlich Schaltjahre)
DY	Mit zwei Zeichen abgekürzter Name des Tages (z. B. Mi)
HH, HH12	Stunde des Tages (1-12)
HH24	Stunde des Tages (0-23)
J	Modifiziertes Julianisches Datum: Konvertiert das Kalenderdatum in einen String bestehend aus seinem MJD-Wert (Modifiziertes Julianisches Datum), berechnet ab dem 1. Januar 4713 v. Chr. Die Zeitkomponente des Datums wird ignoriert. Beispiel: Der Ausdruck TO_CHAR(SHIP_DATE, ‘J’) konvertiert „Dec 31 1999 23: 59: 59“ in den String „2451544“.
MI	Minuten (0-59)
MM	Monat (1-12):
MONTH	Name des Monats, bis zu neun Zeichen (z. B. Januar)
MON	Mit drei Zeichen abgekürzter Monat (z. B. Jan)

Formatstring	Beschreibung
MS	Millisekunden (0-999)
NS	Nanosekunden (0-999999999)
Q	Quartal des Jahres (1-4), wobei Januar-März = 1
RR	Die letzten zwei Ziffern einer Jahreszahlenangabe; die Funktion entfernt die beiden ersten Ziffern. Wenn Sie beispielsweise mit RR den Wert 1997 übergeben, gibt TO_CHAR den Wert 97 zurück. Bei Verwendung mit TO_CHAR ist RR mit YY austauschbar und erzielt dasselbe Ergebnis. Bei Verwendung mit TO_DATE berechnet RR jedoch das nächstgelegene geeignete Jahrhundert und liefert die ersten zwei Ziffern des Jahres.
SS	Sekunden (0-59)
SSSSS	Sekunden seit Mitternacht (00000-86399): Wenn Sie SSSSS in einem TO_CHAR-Ausdruck verwenden, wertet Data Integration Service nur die Zeitkomponente einer Datumsangabe aus. Beispiel: Der Ausdruck TO_CHAR(SHIP_DATE, 'MM/DD/YYYY SSSSS') konvertiert „12/31/1999 01:02:03“ in „12/31/1999 03723“.
US	Mikrosekunden (0-999999)
Y	Letzte Ziffer einer Jahresangabe; die Funktion entfernt die beiden ersten Ziffern. Wenn Sie beispielsweise mit Y den Wert 1997 übergeben, gibt TO_CHAR den Wert 7 zurück.
YY	Die letzten zwei Ziffern einer Jahreszahlenangabe; die Funktion entfernt die beiden ersten Ziffern. Wenn Sie beispielsweise mit YY den Wert 1997 übergeben, gibt TO_CHAR den Wert 97 zurück.
YYY	Die letzten drei Ziffern einer Jahreszahlenangabe; die Funktion entfernt die beiden ersten Ziffern. Wenn Sie beispielsweise mit YYY den Wert 1997 übergeben, gibt TO_CHAR den Wert 997 zurück.
YYYY	Die gesamte Jahreskomponente des Datums; wenn Sie beispielsweise mit YYYY den Wert 1997 übergeben, gibt TO_CHAR den Wert 1997 zurück.
W	Woche des Monats (1-5), wobei Woche 1 am ersten Tag des Monats beginnt und mit dem siebten endet, Woche 2 am achten Tag beginnt und am vierzehnten endet usw. Beispiel: Der 1. Februar markiert den Beginn der ersten Februarwoche.
WW	Woche des Jahres (01-53), wobei Woche 01 am 1. Jan beginnt und mit 7. Jan endet, Woche 2 am 8. Jan beginnt und mit 14. Jan endet usw.
- / . ; :	Zeichensetzung in der Ausgabe; mit diesen Symbolen können Sie die einzelnen Teile einer Datumsangabe trennen. Beispiel: Sie erstellen den folgenden Ausdruck, um die einzelnen Datumskomponenten durch einen Punkt zu trennen: TO_CHAR(DATES, 'MM.DD.YYYY').
"Text"	Text, der in der Ausgabe erscheint; Beispiel: Wenn Sie einen Ausgabeport mit dem Ausdruck TO_CHAR(DATES, 'MM/DD/YYYY “Steigende Verkaufszahlen”') erstellen und das Datum Apr 1 1997 übergeben, gibt die Funktion den String '04/01/1997 Steigende Verkaufszahlen' aus. Sie können Multibyte-Zeichen eingeben, die in der Repository-Codepage gültig sind.
“”	Mit doppelten Anführungszeichen trennen Sie mehrdeutige Formatstrings, etwa D“”DDD. Die leeren Anführungszeichenpaare erscheinen nicht in der Ausgabe.

Hinweis: Bei Formatstrings muss nicht auf Groß-/Kleinschreibung geachtet werden. Sie müssen immer zwischen einfachen Anführungszeichen stehen.

Beispiele

Die folgenden Beispiele zeigen die Formatstrings J, SSSSS, RR und YY. Zusätzliche Beispiele finden Sie in den einzelnen Funktionen.

Hinweis: In TO_CHAR-Ausdrücken ignoriert Data Integration Service die Zeitkomponente des Datums.

J-Formatstring

Mit dem J-Formatstring in einem TO_CHAR-Ausdruck können Sie Datumswerte in als Strings ausgedrückte MJD-Werte konvertieren. Beispiel:

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

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

SSSSS-Formatstring

Sie können auch den SSSSS-Formatstring in TO_CHAR-Ausdrücken einsetzen. Beispiel: Der folgende Ausdruck konvertiert die Daten im Port SHIP_DATE in Strings mit Angabe der Gesamtzahl der Sekunden seit Mitternacht:

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

SHIP_DATE	RETURN_VALUE
12/31/1999 01:02:03	3723
09/15/1996 23:59:59	86399

RR-Formatstring

Der folgende Ausdruck konvertiert Daten in Strings im Format „MM/DD/YY“:

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

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

YY-Formatstring

In TO_CHAR-Ausdrücken erzielt der YY-Formatstring dieselben Ergebnisse wie der RR-Formatstring. Der folgende Ausdruck konvertiert Daten in Strings im Format „MM/DD/YY“:

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

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

TO_DATE- und IS_DATE-Formatstrings

Die Funktion TO_DATE konvertiert einen String in einen Datum/Zeit-Wert (Datetime) im angegebenen Format. TO_DATE wird gewöhnlich zur Konvertierung von Strings aus Einfachdateien in Datum/Zeit-Werte eingesetzt. Die Formatstrings für TO_DATE sind nicht internationalisiert und müssen in den vordefinierten Formaten eingegeben werden.

Hinweis: TO_DATE und IS_DATE verwenden den gleichen Satz von Formatstrings.

Beim Erstellen eines TO_DATE-Ausdrucks wenden Sie einen Formatstring für jede Komponente des Datums im Quellstring an. Das Quell-Stringformat und der Formatstring müssen übereinstimmen. Eine Übereinstimmung des Datumstrennzeichens ist für die Datumsvalidierung nicht erforderlich. Wenn eine Komponente nicht übereinstimmt, konvertiert der Data Integration Service den String nicht und überspringt stattdessen die Zeile. Wenn Sie den Formatstring auslassen, muss der Quellstring in dem in der Sitzung definierten Datumsformat vorliegen.

Beim Erstellen eines TO_DATE-Ausdrucks wenden Sie einen Formatstring für jede Komponente des Datums im Quellstring an. Das Quell-Stringformat und der Formatstring müssen übereinstimmen. Eine Übereinstimmung des Datumstrennzeichens ist für die Datumsvalidierung nicht erforderlich. Wenn eine Komponente nicht übereinstimmt, konvertiert der Data Integration Service den String nicht und überspringt stattdessen die Zeile. Wenn Sie den Formatstring auslassen, muss der Quellstring in dem in der Daten-Viewer Konfiguration definierten Datumsformat vorliegen.

IS_DATE gibt an, ob ein Wert ein gültiges Datum ist. Jeder String in dem Datumsformat, das in der Sitzung festgelegt wurde, stellt ein gültiges Datum dar. Wenn die Strings, die Sie testen möchten, nicht im festgelegten Datumsformat vorliegen, verwenden Sie das Format der Strings aus der Tabelle „TO_DATE- und IS_DATE-Formatstrings“. Wenn das Format eines Strings nicht mit dem festgelegten Format übereinstimmt oder wenn der String kein gültiges Datum darstellt, gibt die Funktion FALSE (0) zurück. Wenn das Format des Strings mit dem festgelegten Format übereinstimmt und ein gültiges Datum darstellt, gibt die Funktion TRUE (1) zurück. Die Formatstrings IS_DATE sind nicht internationalisiert und müssen in einem der Formate aus der folgenden Tabelle eingegeben werden.

IS_DATE gibt an, ob ein Wert ein gültiges Datum ist. Jeder String in dem Datumsformat, das in der Daten-Viewer-Konfiguration festgelegt wurde, stellt ein gültiges Datum dar. Wenn die Strings, die Sie testen möchten, nicht im festgelegten Datumsformat vorliegen, verwenden Sie das Format der Strings aus der Tabelle „TO_DATE- und IS_DATE-Formatstrings“. Wenn das Format eines Strings nicht mit dem festgelegten Format übereinstimmt oder wenn der String kein gültiges Datum darstellt, gibt die Funktion FALSE (0) zurück. Wenn das Format des Strings mit dem festgelegten Format übereinstimmt und ein gültiges Datum darstellt,

gibt die Funktion TRUE (1) zurück. Die Formatstrings IS_DATE sind nicht internationalisiert und müssen in einem der Formate aus der folgenden Tabelle eingegeben werden.

Die folgende Tabelle bietet einen Überblick über die Formatstrings für die Funktionen TO_DATE und IS_DATE:

Tabelle 1. TO_DATE- und IS_DATE-Formatstrings

Formatstring	Beschreibung
AM, a.m., PM, p.m.	Längenkreisangabe Verwenden Sie diese Formatstrings, um Stunden in AM oder PM anzugeben. AM und PM geben die gleichen Werte wie a.m. und p.m. zurück.
DAY	Name des Tages, bis zu neun Zeichen (zum Beispiel Mittwoch). Beim DAY-Formatstring muss nicht auf Groß-/Kleinschreibung geachtet werden.
DD	Tag des Monats (1-31)
DDD	Tag des Jahres (001-366, einschließlich Schaltjahre).
DY	Mit zwei Zeichen abgekürzter Name des Tages (zum Beispiel Mi). Beim DY-Formatstring muss nicht auf Groß-/Kleinschreibung geachtet werden.
HH, HH12	Stunde des Tages (1-12).
HH24	Stunde des Tages (0-23), wobei 0 12AM (Mitternacht) ist.
J	Angepasster Julianischer Tag. Konvertiert Strings im MJD-Format in Datumswerte. Die Zeitkomponente des Datums wird ignoriert, alle Datumsangaben erhalten die Uhrzeit „00:00:00.000000000“. Beispiel: Der Ausdruck TO_DATE(‘2451544’, ‘J’) konvertiert „2451544“ zu „Dec 31 1999 00:00:00.000000000“.
MI	Minuten (0-59).
MM	Monat (1-12)
MONTH	Name des Monats, bis zu neun Zeichen (zum Beispiel August). Groß-/Kleinschreibung ist nicht wichtig.
MON	Mit drei Zeichen abgekürzter Monat (zum Beispiel Aug). Groß-/Kleinschreibung ist nicht wichtig.
MS	Millisekunden (0-999).
NS	Nanosekunden (0-999999999).

Formatstring	Beschreibung
RR	Vierstelliges Jahr (zum Beispiel 1998, 2034). Verwenden Sie dies, wenn Quellstrings zweistellige Jahre beinhalten. Mit TO_DATE können zweistellige Jahresangaben in vierstellige Angaben konvertiert werden. - Aktuelles Jahr zwischen 50 und 99**:** Wenn das aktuelle Jahr zwischen 50 und 99 liegt (z. B. 1998), das Jahr aus dem Quellstring jedoch zwischen 0 und 49 liegt, gibt der Data Integration Service das darauffolgende Jahrhundert plus die zweistellige Jahresangabe aus dem Quellstring zurück. Wenn der Quellstring eine Jahresangabe zwischen 50 und 99 enthält, gibt der Data Integration Service das aktuelle Jahrhundert plus die angegebene zweistellige Jahresangabe zurück. - Aktuelles Jahr zwischen 0 und 49**:** Wenn das aktuelle Jahr zwischen 0 und 49 liegt (z. B. 2003) und das Jahr aus dem Quellstring ebenfalls zwischen 0 und 49 liegt, gibt der Data Integration Service das aktuelle Jahrhundert plus die zweistellige Jahresangabe aus dem Quellstring zurück. Wenn der Quellstring jedoch eine Jahresangabe zwischen 50 und 99 enthält, gibt der Data Integration Service das vorherige Jahrhundert plus das zweistellige Jahr aus dem Quellstring zurück.
SS	Sekunden (0-59).
SSSSS	Sekunden seit Mitternacht. Wenn Sie SSSSS in einem TO_DATE-Ausdruck verwenden, wertet der Data Integration Service nur die Zeitkomponente einer Datumsangabe aus. Beispiel: Der Ausdruck TO_DATE(DATE_STR, 'MM/DD/YYYY SSSSS') konvertiert „12/31/1999 3783“ in „12/31/1999 01:02:03“.
US	Mikrosekunden (0-999999).
Y	Das aktuelle Jahr im Knoten, auf dem der Data Integration Service ausgeführt wird, wobei die letzte Ziffer der Jahresangabe durch den Stringwert ersetzt wird.
YY	Das aktuelle Jahr im Knoten, auf dem der Data Integration Service ausgeführt wird, wobei die letzten zwei Ziffern der Jahresangabe durch den Stringwert ersetzt werden.
YYY	Das aktuelle Jahr im Knoten, auf dem der Data Integration Service ausgeführt wird, wobei die letzten drei Ziffern der Jahresangabe durch den Stringwert ersetzt werden.
YYYY	Vier Zahlen eines Jahres. Verwenden Sie diesen Formatstring nicht, wenn Sie zweistellige Jahre übergeben. Verwenden Sie stattdessen den RR- oder YY-Formatstring.

Regeln und Richtlinien für Datumsformatstrings

Befolgen Sie beim Arbeiten mit Datumsformatstrings folgende Regeln und Richtlinien:

- Das Format des Strings TO_DATE muss mit dem Formatstring übereinstimmen. Andernfalls gibt der Data Integration Service möglicherweise ungenaue Werte zurück oder überspringt die Zeile. Beispiel: Bei der Übergabe des Strings 20200512 (12. Mai 2020) an TO_DATE müssen Sie auch den Formatstring YYYYMMDD angeben. Wenn Sie den Formatstring auslassen, erwartet der Data Integration Service einen String in dem in der Sitzung definierten Datumsformat. in der Daten-Viewer-Konfiguration. Wenn Sie einen String übergeben, der nicht mit dem Formatstring übereinstimmt, gibt der Data Integration Service einen Fehler zurück und überspringt die Zeile. Beispiel: Sie übergeben den String 2020120 an TO_DATE und spezifizieren dabei den Formatstring YYYYMMDD. Der Data Integration Service gibt einen Fehler zurück und überspringt die Zeile, weil der String nicht mit dem Formatstring übereinstimmt.
- Der Formatstring muss zwischen einfachen Anführungszeichen stehen.

- Der Data Integration Service verwendet das in der Sitzung festgelegte Standardformat für Datum/Zeit. Standardmäßig ist das MM/DD/YYYY HH24:MI:SS.US. Bei Formatstrings muss nicht auf Groß-/Kleinschreibung geachtet werden.

Beispiel

Die folgenden Beispiele erläutern die Formatstrings J, RR und SSSSS. Zusätzliche Beispiele finden Sie in den einzelnen Funktionen.

J-Formatstring

Der folgende Ausdruck konvertiert
Strings im Port SHIP_DATE_MJD_STRING in Datumswerte im Standarddatumsformat:

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

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

Da der J-Formatstring die Zeitkomponente der Datumsangabe nicht berücksichtigt, wird die Uhrzeit in den Rückgabewerten auf 00: 00: 00.000000000 gesetzt.

RR-Formatstring

Der folgende Ausdruck konvertiert einen String in ein vierstelliges Jahresdatumsformat. Das aktuelle Jahr ist 1998:

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

DATE_STR	RETURN VALUE
04/01/98	04/01/1998 00:00:00.000000000
08/17/05	08/17/2005 00:00:00.000000000

YY-Formatstring

Der folgende Ausdruck konvertiert einen String in ein vierstelliges Jahresdatumsformat. Das aktuelle Jahr ist 1998:

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

DATE_STR	RETURN VALUE
04/01/98	04/01/1998 00:00:00.000000000
08/17/05	08/17/1905 00:00:00.000000000

Hinweis: In der zweiten Zeile gibt RR das Jahr 2005, YY hingegen das Jahr 1905 zurück.

SSSSS-Formatstring

Der folgende Ausdruck konvertiert Strings, die eine Angabe der Sekunden seit Mitternacht enthalten, in Datumswerte:

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

DATE_STR	RETURN_VALUE
12/31/1999 3783	12/31/1999 01:02:03.000000000
09/15/1996 86399	09/15/1996 23:59:59.000000000

Verständnis von Datumsberechnungen

Die Umwandlungssprache umfasst integrierte Datumsfunktionen, um Berechnungen mit Datetime-Werten wie folgt zu ermöglichen:

- **ADD_TO_DATE:** Addieren oder Subtrahieren eines bestimmten Teils eines Datums
- **DATE_DIFF:** Subtrahieren zweier Datumsangaben
- **SET_DATE_PART:** Ändern eines Teils des Datums

Mit mathematischen Operationszeichen (etwa + oder -) können keine Datumsangaben addiert oder subtrahiert werden.

Die Umwandlungssprache erkennt Schaltjahre und akzeptiert Datumsangaben zwischen 1. Januar 0001 00:00:00.000000000 (n. Chr.) und 31. Dezember 9999 23: 59 59.999999999 (n. Chr.).

Hinweis: Die Umwandlungssprache nutzt zur Angabe von Datumswerten den Datum/Zeit-Datentyp der Umwandlung. Die Datumsfunktionen können nur auf Datetime-Werte angewendet werden.

KAPITEL 6

Funktionen

Dieses Kapitel beschreibt die Funktionen der Umwandlungssprache in alphabetischer Reihenfolge. Jede Funktionsbeschreibung enthält:

- Syntax
- Rückgabewert
- Beispiel

Funktionskategorien

Die Umwandlungssprache stellt folgende Typen von Funktionen bereit:

- Aggregat
- Zeichen
- Umwandlung
- Datenbereinigung
- Datum
- Codierung
- Finanz
- Numerisch
- Wissenschaftlich
- Spezial
- String
- Test
- Variable

Aggregatfunktionen

Aggregatfunktionen geben Zusammenfassungswerte für Werte ungleich Null in ausgewählten Ports zurück. Aggregatfunktionen ermöglichen Folgendes:

- Berechnung eines Einzelwerts für alle Zeilen in einer Gruppe
- Rückgabe eines Einzelwerts für jede Gruppe in einer Aggregator-Umwandlung
- Filter zum Berechnen von Werten für bestimmte Zeilen in den ausgewählten Ports

- Operatoren für mathematische Berechnungen innerhalb der Funktion
- Berechnung von zwei oder mehreren, aus denselben Quellspalten in einem Durchgang abgeleiteten Aggregatwerten

Die Umwandlungssprache enthält die folgenden Aggregatfunktionen:

- ANY
- AVG
- COUNT
- FIRST
- LAST
- MAX (Datum)
- MAX (Zahl)
- MAX (String)
- MEDIAN
- MIN (Datum)
- MIN (Zahl)
- MIN (String)
- PERCENTILE
- STDDEV
- SUM
- VARIANCE

Wenn Sie Data Integration Service für die Ausführung im Unicode-Modus konfiguriert haben, geben MIN und MAX Werte entsprechend der Sortierreihenfolge der Codepage zurück, die Sie in den Sitzungseigenschaften angeben.

Wenn Sie Data Integration Service für die Ausführung im Unicode-Modus konfiguriert haben, geben MIN und MAX Werte entsprechend der Sortierreihenfolge der Codepage zurück, die Sie in der Mapping-Konfiguration angeben.

Verwenden Sie Aggregatfunktionen ausschließlich in Aggregator-Umwandlungen. Es kann nur jeweils eine Aggregatfunktion in einer anderen Aggregatfunktion verschachtelt sein. Bei Verschachtelungen wertet Data Integration Service den innersten Aggregatfunktionausdruck aus und verwendet das Ergebnis zum Evaluieren des äußeren. Gehen Sie wie folgt vor, um eine nach IDs gruppierte Aggregator-Umwandlung mit zwei verschachtelten Aggregatfunktionen zu erstellen:

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

wobei der Datensatz die folgenden Werte enthält:

ID	EARNINGS
1	32
1	45
1	100
2	65
2	75

ID	EARNINGS
2	76
3	21
3	45
3	99

Der Rückgabewert lautet 186. Data Integration Service gruppiert die Umwandlung nach IDs, wertet den Ausdruck AVG aus und gibt drei Werte zurück. Anschließend werden die Werte durch die Funktion SUM addiert, um das Ergebnis zu erhalten.

Aggregatfunktionen und Nullen

Beim Konfigurieren von Data Integration Service können Sie bestimmen, wie Nullwerte in Aggregatfunktionen behandelt werden sollen. Sie können bestimmen, ob Data Integration ServiceNullwerte in Aggregatfunktionen als 0 oder als NULL verarbeitet werden.

Standardmäßig behandelt Data Integration Service Nullwerte in Aggregatfunktionen als NULL. Wenn Sie einen Port oder eine Gruppe mit ausschließlich Nullwerten übergeben, gibt die Funktion NULL zurück. Optional können Sie Data Integration Service auch so konfigurieren, dass beim Übergeben eines gesamten Ports aus Nullwerten an eine Aggregatfunktion 0 zurückgegeben wird.

Filterbedingungen

Verwenden Sie Filterbedingungen, um die Anzahl der in einer Suche zurückgegeben Zeilen zu begrenzen.

Mit Filtern begrenzen Sie die Anzahl der Zeilen im Suchergebnis. Filterbedingungen können auf alle Aggregatfunktionen sowie auf CUME, MOVINGAVG und MOVINGSUM angewendet werden. Die Auswertung der Filterbedingung muss TRUE, FALSE oder NULL ergeben. Bei NULL oder FALSE wählt Data Integration Service die Zeile nicht aus.

Sie können einen beliebigen gültigen Umwandlungausdruck eingeben. Beispiel: Der folgende Ausdruck berechnet das durchschnittliche Gehalt aller Mitarbeiter, die mehr als 50000 USD verdienen:

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

Sie können auch andere numerische Werte als Filterbedingung einsetzen. Beispielsweise können Sie Folgendes als komplette Syntax der Funktion MEDIAN angeben, einschließlich numerischen Port:

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

In allen Fällen rundet Data Integration Service Dezimalwerte für die Filterbedingung auf Ganzzahlen (zum Beispiel 1.5 auf 2, 1.2 auf 1, 0.35 auf 0). Wenn der Wert auf 0 abgerundet wird, gibt die Filterbedingung FALSE zurück. Um Rundungen zu vermeiden, verwenden Sie die Funktion TRUNC und schneiden die Dezimalstellen ab:

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

Wenn Sie die Filterbedingung nicht angeben, wählt die Funktion alle Zeilen im Port aus.

Zeichenfunktionen

Die Umwandlungssprache stellt die folgenden Zeichenfunktionen bereit:

- ASCII

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

Die Zeichenfunktionen MAX, MIN, LOWER, UPPER und INITCAP werten Zeichen anhand der Codepage von Data Integration Service aus.

Konvertierungsfunktionen

Die Umwandlungssprache stellt die folgenden Konvertierungsfunktionen bereit:

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

Datenbereinigungsfunktionen

Die Umwandlungssprache enthält eine Gruppe von Funktionen zum Eliminieren von Datenfehlern. Datenbereinigungsfunktionen können folgende Aufgaben ausführen:

- Testen von Eingabewerten
- Konvertieren des Datentyps eines Eingabewerts
- Entfernen von Leerzeichen aus Stringwerten
- Ersetzen von Zeichen in einem String
- Kodieren von Strings
- Matching von Mustern in regulären Ausdrücken

Die Umwandlungssprache stellt folgende Datenbereinigungsfunktionen bereit:

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

Datumsfunktionen

Die Umwandlungssprache enthält eine Gruppe von Datumsfunktionen, mit denen Sie Datumswerte runden, abschneiden oder vergleichen, Teile von Datumsangaben extrahieren oder Datumsangaben berechnen können.

Sie können beliebige Werte mit dem Datum-Datentyp an eine Datumsfunktion übergeben. Um jedoch einen String an eine Datumsfunktion übergeben zu können, müssen Sie ihn zuerst mithilfe der TO_DATE-Funktion in einen Datum/Zeit-Wert (Datetime) konvertieren.

Die Umwandlungssprache stellt die folgenden Datumsfunktionen bereit:

- ADD_TO_DATE
- DATE_COMPARE
- DATE_DIFF
- GET_DATE_PART
- IS_DATE

- LAST_DAY
- MAKE_DATE_TIME
- MAX
- MIN
- ROUND (Datum)
- SET_DATE_PART
- SYSTIMESTAMP
- TO_CHAR(Date)
- TRUNC (Datum)

Einige der Datumsfunktionen umfassen ein *Formatargument*. Für diese Argumente müssen Sie einen der Formatstrings der Umwandlungssprache angeben. Datumsformatstrings sind nicht internationalisiert.

Der Umwandlungsdatentyp für Datum/Zeit unterstützt Datumsangaben mit einer Präzision bis zur Nanosekunde.

Verwandte Themen:

- "Datumsformatstrings" auf Seite 42

Kodierungsfunktionen

Die Umwandlungssprache stellt folgende Funktionen für Datenverschlüsselung, Kompression, Kodierung und Prüfsumme bereit:

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

Finanzfunktionen

Die Umwandlungssprache stellt folgende Finanzfunktionen bereit:

- FV
- NPER
- PMT
- PV
- RATE

Numerische Funktionen

Die Umwandlungssprache stellt folgende numerische Funktionen bereit:

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

Wissenschaftliche Funktionen

Die Umwandlungssprache stellt folgende wissenschaftliche Funktionen bereit:

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

Spezialfunktionen

Die Umwandlungssprache stellt folgende Spezialfunktionen bereit:

- ABORT
- DECODE
- ERROR
- IIF
- LOOKUP

- UUID4
- UUID_UNPARSE

Im Allgemeinen kommen Spezialfunktionen in Ausdrucks-, Filter- und Update-Strategie-Umwandlungen zum Einsatz. Spezialfunktionen können verschachtelte Funktionen aufnehmen. Sie können Spezialfunktionen auch in Aggregatfunktionen verschachteln.

Stringfunktionen

Die Umwandlungssprache stellt folgende Stringfunktionen bereit:

- CHOOSE
- INDEXOF
- MAX
- MIN
- REVERSE

Testfunktionen

Die Umwandlungssprache stellt folgende Testfunktionen bereit:

- ISNULL
- IS_DATE
- IS_NUMBER
- IS_SPACES

Variablenfunktionen

Die Umwandlungssprache enthält eine Gruppe von Variablenfunktionen zum Aktualisieren des aktuellen Werts einer Mapping-Variablen im Verlauf einer Sitzung. Wenn Sie einen Arbeitsablauf ausführen, wertet PowerCenter Integration Service den Startwert und den aktuellen Wert der Variablen am Anfang der Sitzung aus, basierend auf dem endgültigen Wert der Variablen aus der letzten Ausführung der Sitzung. Verwenden Sie die folgenden Variablenfunktionen:

- SetCountVariable
- SetMaxVariable
- SetMinVariable
- SetVariable

Welche Variablenfunktion Sie verwenden, hängt vom Aggregationstyp der Variable ab.

Bei Mapping-Variablen in Sitzungen mit mehreren Partitionen verwenden Sie Variablenfunktionen zum Bestimmen des endgültigen Werts der Variable für jede Partition. Am Ende der Sitzung führt PowerCenter Integration Service die Aggregatfunktion in allen Partitionen aus, um einen endgültigen Einzelwert zu bestimmen, der im Repository gespeichert wird. Sofern er nicht überschrieben wird, dient der gespeicherte Wert als Startwert der Variable für die nächste Ausführung dieser Sitzung.

Beispiel: Sie setzen mit SetMinVariable eine Variable auf den ausgewerteten Mindestwert. PowerCenter Integration Service berechnet den aktuellen Mindestwert der Variablen in jeder Partition. Am Ende der Sitzung wird dann der aktuelle Mindestwert für alle Partitionen berechnet und im Repository gespeichert.

Verwenden Sie SetVariable nur einmal pro Mapping-Variable in einer Pipeline. Wenn Sie mehrere Partitionen in einer Pipeline erstellen, verarbeitet PowerCenter Integration Service die Pipeline mithilfe mehrerer Threads.

Wenn Sie diese Funktion mehr als einmal für dieselbe Variable einsetzen, kann der aktuelle Wert einer Mapping-Variablen zu indeterministischen Ergebnissen führen.

ABORT

Stoppt die Sitzung und schreibt eine entsprechende Fehlermeldung in die Logdatei der Sitzung. Wenn Data Integration Service auf die Funktion ABORT trifft, wird die Umwandlung an der betreffenden Zeile abgebrochen. Die Verarbeitung und Ladung der bis zu diesem Zeitpunkt gelesenen Zeilen hängt vom quell- oder zielbasierten Commit-Intervall und der für die Sitzung definierten Pufferblockgröße ab. Data Integration Service schreibt bis zur abgebrochenen Ziele Werte in das Ziel und führt dann einen Rollback aller nicht übernommen Daten bis zum letzten Commit-Punkt aus. Nach dem Rollback können Sie die Sitzung wiederherstellen.

Stoppt die Ausführung des Mapping und schreibt eine entsprechende Fehlermeldung in das Protokoll. Wenn Data Integration Service auf die Funktion ABORT trifft, wird die Umwandlung an der betreffenden Zeile abgebrochen. Alle Zeilen bis zum ABORT werden verarbeitet. Data Integration Service schreibt bis zur abgebrochenen Ziele Werte in das Ziel und führt dann einen Rollback aller nicht übernommen Daten bis zum letzten Commit-Punkt aus.

Mit ABORT können Sie Daten validieren. Im Allgemeinen wird ABORT innerhalb einer IIF- oder DECODE-Funktion eingesetzt, um die Regeln für den Sitzungsabbruch festzulegen.

Verwenden Sie ABORT für die Standardwerte sowohl der Eingabe- als auch der Ausgabeports. Sie können ABORT bei Eingabeports dazu nutzen, die Übergabe von Nullwerten in die Umwandlung zu verhindern. Außerdem können Sie mit ABORT alle Arten von Umwandlungsfehlern behandeln, einschließlich der ERROR-Funktionsaufrufe innerhalb eines Ausdrucks. Der Standardwert überschreibt die Funktion ERROR in einem Ausdruck. Wenn Sie sichergehen möchten, dass die Sitzung beendet wird, sobald ein Fehler auftritt, weisen Sie ABORT als Standardwert zu.

Bei ABORT in einem Ausdruck für einen nicht verbundenen Port führt Data Integration Service die ABORT-Funktion nicht aus.

Hinweis: Data Integration Service behandelt die Funktion ABORT und den Befehl „Abbrechen“ aus Workflow Manager unterschiedlich.

Syntax

```
ABORT( string )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
string	Erforderlich	String. Meldung, die bei Abbruch der Sitzung in der Sitzungs-Logdatei angezeigt werden soll. Der String kann beliebig lang sein. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben. String. Meldung, die bei Abbruch des Mapping in der Logdatei angezeigt werden soll. Der String kann beliebig lang sein. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

NULL.

ABS

Gibt den absoluten Wert eines numerischen Werts zurück.

Syntax

```
ABS( numeric_value )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
numeric_value	Erforderlich	Numerischer Datentyp. Übergibt die Werte, für die Sie absolute Werte zurückgeben möchten. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

Positiver numerischer Wert. ABS gibt den gleichen Datentyp zurück wie der numerische Wert, der als Argument übergeben wurde. Wenn Sie einen Double-Wert übergeben, wird ein Double-Wert zurückgegeben. Ebenso werden bei übergebenen Ganzzahlen wiederum Ganzzahlen zurückgegeben.

NULL, wenn Sie der Funktion einen Nullwert übergeben.

Hinweis: Wenn der Rückgabewert eine Dezimalmal mit Präzision höher als 15 ist, können Sie „Hohe Präzision“ aktivieren, um Dezimalgenauigkeit bis zu 28 Stellen zu gewährleisten.

Beispiel

Der folgende Ausdruck gibt den Unterschied zwischen zwei Zahlen als positiven Wert zurück, unabhängig davon, welche Zahl größer ist:

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

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

ADD_TO_DATE

Fügt einem Teil eines Datetime-Werts eine angegebene Menge hinzu und gibt ein Datum in demselben Format zurück wie das Datum, das an die Funktion übergeben wird. ADD_TO_DATE akzeptiert positive und negative Ganzzahlwerte. Verwenden Sie ADD_TO_DATE, um folgende Teile eines Datums zu ändern:

- **Jahr.** Geben Sie eine positive oder negative Ganzzahl in das Argument *amount* ein. Verwenden Sie den Formatstring für Jahr, den Sie möchten: Y Y, YY, YYY oder YYYY. Der folgende Ausdruck fügt allen Datumsangaben im Port SHIP_DATE 10 Jahre hinzu:

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

- **Monat.** Geben Sie eine positive oder negative Ganzzahl in das Argument *amount* ein. Verwenden Sie den Formatstring für Monat, den Sie möchten: MM, MON, MONTH. Der folgende Ausdruck zieht von allen Datumsangaben im Port SHIP_DATE 10 Monate ab:

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

- **Tag.** Geben Sie eine positive oder negative Ganzzahl in das Argument *amount* ein. Verwenden Sie den Formatstring für Tag, den Sie möchten: D, DD, DDD, DY und DAY. Der folgende Ausdruck fügt allen Datumsangaben im Port SHIP_DATE 10 Tage hinzu:

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

- **Stunde.** Geben Sie eine positive oder negative Ganzzahl in das Argument *amount* ein. Verwenden Sie den Formatstring für Stunde, den Sie möchten: HH, HH12, HH24. Der folgende Ausdruck fügt allen Datumsangaben im Port SHIP_DATE 14 Stunden hinzu:

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

- **Minute.** Geben Sie eine positive oder negative Ganzzahl in das Argument *amount* ein. Verwenden Sie zum Festlegen der Minuten den Formatstring MI. Der folgende Ausdruck fügt allen Datumsangaben im Port SHIP_DATE 25 Minuten hinzu:

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

- **Sekunden.** Geben Sie eine positive oder negative Ganzzahl in das Argument *amount* ein. Verwenden Sie zum Festlegen der Sekunden den Formatstring SS. Der folgende Ausdruck fügt allen Datumsangaben im Port SHIP_DATE 59 Sekunden hinzu:

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

- **Millisekunden.** Geben Sie eine positive oder negative Ganzzahl in das Argument *amount* ein. Verwenden Sie zum Festlegen der Millisekunden den Formatstring MS. Der folgende Ausdruck fügt allen Datumsangaben im Port SHIP_DATE 125 Millisekunden hinzu:

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

- **Mikrosekunden.** Geben Sie eine positive oder negative Ganzzahl in das Argument *amount* ein. Verwenden Sie zum Festlegen der Mikrosekunden den Formatstring US. Der folgende Ausdruck fügt allen Datumsangaben im Port SHIP_DATE 2000 Mikrosekunden hinzu:

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

- **Nanosekunden.** Geben Sie eine positive oder negative Ganzzahl in das Argument *amount* ein. Verwenden Sie zum Festlegen der Nanosekunden den Formatstring NS. Der folgende Ausdruck fügt allen Datumsangaben im Port SHIP_DATE 3 000 000 Nanosekunden hinzu:

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

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
Datum	Erforderlich	Datum/Zeit-Datentyp. Übergibt die Werte, die Sie ändern möchten. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.
format	Erforderlich	Formatstring zur Angabe des Teils des Datums, den Sie ändern möchten. Setzen Sie den Formatstring zwischen einfache Anführungszeichen, z. B. 'MM'. Bei Formatstrings muss nicht auf Groß-/Kleinschreibung geachtet werden.
amount	Erforderlich	Ganzzahliger Wert, der die Menge von Jahren, Monate, Tagen, Stunden usw. angibt, um die Sie den Datumswert ändern möchten. Sie können jeden beliebigen Umwandlungsausdruck eingeben, dessen Auswertung eine Ganzzahl ergibt.

Rückgabewert

Das Datum wird in demselben Format zurückgegeben wie das übergebene Datum.

NULL, wenn der Funktion ein Nullwert als Argument übergeben wird.

Beispiele

Alle nachstehenden Ausdrücke fügen jeder Datumsangabe im Port DATE_SHIPPED einen Monat hinzu. Bei Übergabe eines Werts, der einen Tag erstellt, den es in einem bestimmten Monat nicht gibt, gibt Data Integration Service den letzten Tag des betreffenden Monats zurück. Beispiel: Sie fügen dem 31. Januar 1998 einen Monat hinzu. Data Integration Service gibt in diesem Fall den 28. Februar 1998 zurück.

Beachten Sie außerdem, dass ADD_TO_DATE Schaltjahre erkennt und dem 29. Januar 2000 einen Monat hinzufügt:

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

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

Die folgenden Ausdrücke ziehen von allen Datumsangaben im Port DATE_SHIPPED 10 Tage ab:

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

```
ADD_TO_DATE( DATE_SHIPPED, 'DY', -10 )
ADD_TO_DATE( DATE_SHIPPED, 'DAY', -10 )
```

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

Die folgenden Ausdrücke ziehen von allen Datumsangaben im Port DATE_SHIPPED 15 Stunden ab:

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

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

Arbeiten mit Datumsangaben

Folgende Tipps helfen Ihnen bei der Arbeit mit ADD_TO_DATE:

- Zum Hinzufügen oder Abziehen eines Teils eines Datums geben Sie einen Formatstring an und legen für das Argument *amount* eine positive oder negative Ganzzahl fest.
- Bei Übergabe eines Werts, der einen Tag erstellt, den es in einem bestimmten Monat nicht gibt, gibt Data Integration Service den letzten Tag des betreffenden Monats zurück. Beispiel: Sie fügen dem 31. Januar 1998 einen Monat hinzu. Data Integration Service gibt in diesem Fall den 28. Februar 1998 zurück.
- Die Funktionen TRUNC und ROUND können verschachtelt werden, um Datumsangaben zu bearbeiten.
- TO_DATE kann verschachtelt werden, um Strings in Datumsangaben zu konvertieren.
- ADD_TO_DATE ändert nur einen Teil des Datums, und zwar den, den Sie angeben. Wenn Sie ein Datum von Sommer- auf Winterzeit ändern, müssen Sie die Stundenkomponente des Datums ändern.

AES_DECRYPT

Gibt entschlüsselte Daten im Stringformat zurück. Data Integration Service verwendet den AES-Algorithmus (Advanced Encryption Standard) mit 128-Bit-Kodierung. AES ist ein FIPS-konformer kryptographischer Algorithmus.

Syntax

```
AES_DECRYPT ( value, key )
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
value	Erforderlich	Binärer Datentyp. Wert, der entschlüsselt werden soll.
key	Erforderlich	String-Datentyp. Präzision von 16 Zeichen oder weniger. Als Schlüssel können Mapping-Variablen verwendet werden. Verwenden Sie den gleichen Schlüssel zum Entschlüsseln und Verschlüsseln.

Rückgabewert

Entschlüsselter Binärwert.

NULL, wenn der Eingabewert ein Nullwert ist.

Beispiel

Das Beispiel unten gibt entschlüsselte Sozialversicherungsnummern zurück. In diesem Beispiel leitet Data Integration Service den Schlüssel mithilfe der Funktion SUBSTR aus den ersten drei Ziffern der Sozialversicherungsnummer ab:

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

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

AES_ENCRYPT

Gibt Daten im verschlüsselten Format zurück. Data Integration Service verwendet den AES-Algorithmus (Advanced Encryption Standard) mit 128-Bit-Kodierung. AES ist ein FIPS-konformer kryptographischer Algorithmus.

Verwenden Sie diese Funktion, um zu verhindern, dass sensible Daten für jeden einsichtig sind. Beispiel: Schützen Sie die Privatsphäre, indem Sie die Sozialversicherungsnummern beim Speichern im Data Warehouse mithilfe von AES_ENCRYPT verschlüsseln.

Syntax

```
AES_ENCRYPT ( value, key )
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
value	Erforderlich	String-Datentyp. Wert, der verschlüsselt werden soll.
key	Erforderlich	String-Datentyp. Präzision von 16 Zeichen oder weniger. Als Schlüssel können Mapping-Variablen verwendet werden.

Rückgabewert

Verschlüsselter Binärwert.

NULL, wenn der Eingabewert ein Nullwert ist.

Beispiel

Das Beispiel unten gibt verschlüsselte Werte für Sozialversicherungsnummern zurück. In diesem Beispiel leitet Data Integration Service den Schlüssel mithilfe der Funktion SUBSTR aus den ersten drei Ziffern der Sozialversicherungsnummer ab:

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

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

Tipp

Wenn das Ziel keine Binärdaten unterstützt, verwenden Sie AES_ENCRYPT mit der Funktion ENC_BASE64, um Daten in einem datenbankkompatiblen Format zu speichern.

ANY

Gibt eine beliebige Zeile im ausgewählten Port zurück. Optional können Sie einen Filter anwenden, um die Anzahl der Zeilen zu beschränken, die der Datenintegrationsdienst ausliest. Es kann nur eine weitere Aggregatfunktion in ANY verschachtelt sein.

Syntax

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

In der folgenden Tabelle werden die Argumente für diese Funktion beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
value	Erforderlich	Alle Datentypen außer binär. Übergibt die Werte, für die Sie eine beliebige Zeile zurückgeben möchten. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.
filter_condition	Optional	Begrenzt die Zeilen in der Suche. Die Filterbedingung muss ein numerischer Wert sein oder mit TRUE, FALSE oder NULL ausgewertet werden. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

Jede Zeile in einem Port. Gibt jedes Mal eine andere Zeile zurück.

NULL, wenn alle der Funktion übergebenen Werte NULL sind oder keine Zeilen ausgewählt werden. Beispiel: Die Filterbedingung ergibt für alle Zeilen FALSE oder NULL.

Beispiel

Der folgende Ausdruck gibt eine beliebige Zeile im Port ITEM_NAME mit einem höheren Preis als 10,00 USD zurück:

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

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

RETURN VALUE: Flashlight

ASCII

Wenn der Data Integration Service den ASCII-Modus verwendet, gibt die ASCII-Funktion den numerischen ASCII-Wert des ersten Zeichens der Zeichenfolge zurück, die an die Funktion übergeben wurde.

Wenn der Data Integration Service den Unicode-Modus verwendet, gibt die ASCII-Funktion den numerischen Unicode-Wert des ersten Zeichens der Zeichenfolge zurück, die an die Funktion übergeben wurde. Unicode-Werte fallen in den Bereich zwischen 0 und 65535.

Sie können Strings beliebiger Länge an ASCII übergeben, ausgewertet wird jedoch nur das erste Zeichen im String. Vor der Übergabe eines Stringwerts an ASCII können Sie ihn auf bestimmte Zeichen parsen, die in einen ASCII- oder Unicode-Wert konvertiert werden sollen. Dazu können Sie beispielsweise RTRIM oder eine andere Stringbearbeitungsfunktion verwenden. Wenn Sie einen numerischen Wert übergeben, konvertiert ihn ASCII in einen Zeichenstring und gibt den ASCII- oder Unicode-Wert des ersten Zeichens im String zurück.

Diese Funktion verhält sich identisch zur Funktion CHRCODE. Wenn Sie ASCII in bereits bestehende Ausdrücke einfügen, werden sie trotzdem korrekt ausgeführt. Beim Erstellen neuer Ausdrücke sollten Sie jedoch CHRCODE statt ASCII verwenden.

Syntax

```
ASCII ( string )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
string	Erforderlich	Zeichenfolge. Übergibt den Wert, der als ASCII-Wert zurückgegeben werden soll. Sie können einen beliebigen gültigen Umwandlungsausdruck eingeben.

Rückgabewert

Ganzzahl. ASCII oder Unicode-Wert des ersten Zeichens im String.

NULL, falls ein an die Funktion übergebener Wert NULL ist.

Beispiel

Der folgende Ausdruck gibt den ASCII- oder Unicode-Wert des ersten Zeichens aller Werte im Port ITEMS zurück:

```
ASCII( ITEMS )
```

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

AVG

Gibt den Durchschnitt aller Werte in einer Gruppe von Zeilen zurück. Optional können Sie einen Filter anwenden, um die Zeilen zu beschränken, aus denen der Durchschnitt berechnet wird. Sie können eine weitere Aggregatfunktion in AVG schachteln, und die verschachtelte Funktion muss einen numerischen Datentyp zurückgeben.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
numeric_value	Erforderlich	Numerischer Datentyp. Übergibt die Werte, deren Durchschnitt berechnet werden soll. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.
filter_condition	Optional	Begrenzt die Zeilen in der Suche. Die Filterbedingung muss ein numerischer Wert sein oder mit TRUE, FALSE oder NULL ausgewertet werden. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

Numerischer Wert.

NULL, wenn alle der Funktion übergebenen Werte NULL sind oder keine Zeilen ausgewählt werden. Beispiel: Die Filterbedingung ergibt für alle Zeilen FALSE oder NULL.

Hinweis: Wenn der Rückgabewert eine Dezimalmal mit Präzision höher als 15 ist, können Sie „Hohe Präzision“ aktivieren, um Dezimalgenauigkeit bis zu 28 Stellen zu gewährleisten.

Nullen

Wenn nur ein Wert NULL ist, wird die Zeile ignoriert. Wenn jedoch alle vom Port übergebenen Werte NULL ergeben, gibt AVG NULL zurück.

Hinweis: Standardmäßig behandelt Data Integration Service Nullwerte in Aggregatfunktionen als NULL. Wenn Sie einen Port oder eine Gruppe mit ausschließlich Nullwerten übergeben, gibt die Funktion NULL zurück. Beim Konfigurieren von Data Integration Service können Sie jedoch entscheiden, wie mit Nullwerten in Aggregatfunktionen umgegangen werden soll. Sie können Nullwerte in Aggregatfunktionen als 0 oder als NULL verarbeiten.

Gruppieren nach

AVG gruppiert die Werte nach der Einstellung „Gruppieren nach Ports“, die Sie in der Umwandlung festlegen, und gibt pro Gruppe ein Ergebnis zurück.

Wenn „Gruppieren nach Ports“ nicht festgelegt wurde, behandelt AVG alle Zeilen als eine einzige Gruppe und gibt nur einen Wert zurück.

Beispiel

Der folgende Ausdruck gibt den durchschnittlichen Großhandelspreis für Taschenlampen (Flashlight) zurück:

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

ITEM_NAME	WHOLESALE_COST
Flashlight	35.00
Navigation Compass	8.05
Regulator System	150.00

ITEM_NAME	WHOLESALE_COST
Flashlight	29.00
Depth/Pressure Gauge	88.00
Flashlight	31.00

```
RETURN VALUE: 31.66
```

Tipp

Sie können anhand der an AVG übergebenen Werte mathematische Berechnungen durchführen, bevor die Funktion den Durchschnitt berechnet. Beispiel:

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

CEIL

Gibt die kleinste Ganzzahl zurück, die größer oder gleich des numerischen Werts ist, der an diese Funktion übergeben wird. Beispiel: Wenn Sie 3.14 an CEIL übergeben, gibt die Funktion 4 zurück. Wenn Sie 3.98 an CEIL übergeben, gibt die Funktion 4 zurück. Wenn Sie -3.17 übergeben, gibt die Funktion -3 zurück.

Syntax

```
CEIL( numeric_value )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich / Optional	Beschreibung
numeric_value	Erforderlich	Numerischer Datentyp. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

Numerischer Wert.

Double-Wert bei numerischen Werten mit deklarierter Präzision über 38.

Double-Wert bei numerischen Werten mit deklarierter Präzision über 28.

NULL, falls ein an die Funktion übergebener Wert NULL ist.

Beispiel

Der folgende Ausdruck gibt den auf die nächste Ganzzahl gerundeten Preis zurück:

```
CEIL( PRICE )
```

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

Tipp: Sie können anhand der an CEIL übergebenen Werte mathematische Berechnungen durchführen, bevor die Funktion den nächsten Ganzzahlwert berechnet. Beispiel: Wenn Sie einen numerischen Wert mit 10 multiplizieren möchten, bevor die kleinste Ganzzahl, die größer als der bearbeitete Wert ist, berechnet wird, können Sie die Funktion wie folgt formulieren:

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

CHOOSE

Wählt anhand einer bestimmten Position einen String aus einer Liste von Strings aus. Sie geben die Position und den Wert an. Wenn der Wert mit der Position übereinstimmt, gibt Data Integration Service den Wert zurück.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
index	Erforderlich	Numerischer Datentyp. Geben Sie eine Zahl basierend auf der Position des gewünschten Werts ein.
string	Erforderlich	Beliebiger Zeichenwert.

Rückgabewert

Der String an der Position des „index“-Werts.

NULL, wenn kein String mit der Position des „index“-Werts übereinstimmt.

Beispiel

Der folgende Ausdruck gibt anhand des „index“-Werts 2 den String „flashlight“ zurück:

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

Der folgende Ausdruck gibt anhand des „index“-Werts 4 NULL zurück:

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

CHOOSE gibt NULL zurück, da der Ausdruck kein viertes Argument enthält.

CHR

Wenn der Data Integration Service den ASCII-Modus verwendet, gibt CHR das ASCII-Zeichen zurück, das dem numerischen Wert entspricht, den Sie an diese Funktion übergeben. ASCII-Werte fallen in den Bereich zwischen 0 und 255. Sie können CHR zwar beliebige Ganzzahlen übergeben, aber nur ASCII-Codes 32 bis 126 sind druckbare Zeichen.

Wenn der Data Integration Service den Unicode-Modus verwendet, gibt CHR das Unicode-Zeichen zurück, das dem numerischen Wert entspricht, den Sie an diese Funktion übergeben. Unicode-Werte fallen in den Bereich zwischen 0 und 65535.

Syntax

```
CHR( numeric_value )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
numeric_value	Erforderlich	Numerischer Datentyp. Der Wert, der als ASCII- oder Unicode-Zeichen zurückgegeben werden soll. Sie können einen beliebigen gültigen Umwandlungsausdruck eingeben.

Rückgabewert

ASCII- oder Unicode Zeichen. String, bestehend aus einem einzigen Zeichen.

NULL, falls ein an die Funktion übergebener Wert NULL ist.

Beispiel

Der folgende Ausdruck gibt das ASCII- oder Unicode-Zeichen jeden numerischen Werts im Port ITEM_ID zurück:

```
CHR( ITEM_ID )
```

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

Verwenden Sie CHR zum Verketten eines einzelnen Anführungszeichens mit einem String. Das einfache Anführungszeichen ist das einzige Zeichen, das in String-Literalen nicht verwendet werden darf. Betrachten Sie das folgende Beispiel:

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

Der Rückgabewert ist:

```
Joan's car
```

CHRCODE

Wenn der Data Integration Service den ASCII-Modus verwendet, gibt CHRCODE den numerischen ASCII-Wert des ersten Zeichens der Zeichenfolge zurück, die an die Funktion übergeben wurde. ASCII-Werte fallen in den Bereich zwischen 0 und 255.

Wenn der Data Integration Service den Unicode-Modus verwendet, gibt CHRCODE den numerischen Unicode-Wert des ersten Zeichens der Zeichenfolge zurück, die an die Funktion übergeben wurde. Unicode-Werte fallen in den Bereich zwischen 0 und 65535.

In der Regel parsen Sie einen Stringwert vor der Übergabe an ASCII auf bestimmte Zeichen, die in einen ASCII- oder Unicode-Wert konvertiert werden sollen. Dazu können Sie beispielsweise RTRIM oder eine andere Stringbearbeitungsfunktion verwenden. Wenn Sie einen numerischen Wert übergeben, konvertiert ihn CHRCODE in einen Zeichenstring und gibt den ASCII- oder Unicode-Wert des ersten Zeichens im String zurück.

Diese Funktion verhält sich identisch zur Funktion ASCII. Wenn Sie in bereits bestehenden Ausdrücke ASCII verwenden, werden sie trotzdem korrekt ausgeführt. Beim Erstellen neuer Ausdrücke sollten Sie jedoch CHRCODE statt ASCII verwenden.

Syntax

```
CHRCODE ( string )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
string	Erforderlich	Zeichenfolge. Übergibt die Werte, die als ASCII- oder Unicode-Werte zurückgegeben werden sollen. Sie können einen beliebigen gültigen Umwandlungsausdruck eingeben.

Rückgabewert

Ganzzahl.

NULL, falls ein an die Funktion übergebener Wert NULL ist.

Beispiel

Der folgende Ausdruck gibt den ASCII- oder Unicode-Wert des ersten Zeichens aller Werte im Port ITEMS zurück:

```
CHRCODE( ITEMS )
```

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

COMPRESS

Komprimiert Daten mithilfe des Kompressionsalgorithmus zlib 1.2.1. Verwenden Sie COMPRESS, bevor Sie große Datenmengen über ein WAN senden.

Syntax

```
COMPRESS( value )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
value	Erforderlich	String-Datentyp. Daten, die komprimiert werden sollen.

Rückgabewert

Komprimierter Binärwert des Eingabewerts.

NULL, wenn der Eingabewert ein Nullwert ist.

Beispiel

Ihr Unternehmen betreibt einen Online-Bestelldienst. Sie möchten die Bestelldaten Ihrer Kunden über ein WAN übermitteln. Die Quelle enthält eine Zeile mit einer Größe von 10 MB. Sie können die Daten in dieser Zeile mit COMPRESS komprimieren. Beim Komprimieren der Daten reduzieren Sie die Datenmenge, die Data Integration Service über das Netzwerk verschiebt. Damit steigern Sie möglicherweise die Leistung.

CONCAT

Verknüpft zwei Strings. CONCAT konvertiert alle Daten in Text, bevor die Strings verkettet werden. Zum Verketten zweier Strings können Sie auch den String-Operator || verwenden. Mit || anstelle von CONCAT steigern Sie die Leistung von Data Integration Service.

Syntax

```
CONCAT( first_string, second_string )
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
first_string	Erforderlich	Alle Datentypen außer binär. Der erste Teil des Strings, der verkettet werden soll. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.
second_string	Erforderlich	Alle Datentypen außer binär. Der zweite Teil des Strings, der verkettet werden soll. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

String.

NULL, wenn beide Stringwerte NULL ergeben.

Nullen

Wenn einer der beiden Strings NULL ist, ignoriert CONCAT diesen String und gibt den anderen zurück.

Wenn beide Strings NULL sind, gibt CONCAT NULL zurück.

Beispiel

Der folgende Ausdruck verkettet die Namen in den Ports FIRST_NAME und LAST_NAME:

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

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

CONCAT fügt separaten Strings keine Leerzeichen hinzu. Wenn zwischen zwei Strings ein Leerzeichen eingefügt werden soll, können Sie einen Ausdruck mit zwei verschachtelten CONCAT-Funktionen formulieren.

Beispiel: Der folgende Ausdruck verkettet den Vornamen mit einem Leerzeichen und anschließend mit dem Nachnamen:

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

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

Verwenden Sie CHR und CONCAT zum Verketten eines einzelnen Anführungszeichens mit einem String. Das einfache Anführungszeichen ist das einzige Zeichen, das in String-Literalen nicht verwendet werden darf. Betrachten Sie das folgende Beispiel:

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

Der Rückgabewert ist:

```
Joan's car
```

CONVERT_BASE

Wandelt eine nicht-negative numerische Zeichenfolge aus einem Basiswert in einen anderen um.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
value	Erforderlich	Zeichenfolgen-Datentyp. Wert, der von einer Basis in eine andere Basis umgewandelt werden soll. Maximum ist 9.233.372.036.854.775.806.
source_base	Erforderlich	Numerischer Datentyp. Aktueller Basiswert der umzuwandelnden Daten. Mindestbasis ist 2. Maximalbasis ist 36.
dest_base	Erforderlich	Numerischer Datentyp. Basiswert, in den die Daten umgewandelt werden sollen. Mindestbasis ist 2. Maximalbasis ist 36.

Rückgabewert

Numerischer Wert.

Beispiel

Das folgende Beispiel konvertiert 2222 vom Dezimalbasiswert 10 in den binären Basiswert 2:

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

Data Integration Service gibt 100010101110 zurück.

COS

Gibt den Kosinus eines numerischen Werts zurück (ausgedrückt in Radianten).

Syntax

```
COS( numeric_value )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
numeric_value	Erforderlich	Numerischer Datentyp. Numerische Daten, ausgedrückt in Radianten (Grad multipliziert mit Pi durch 180). Übergibt die Werte, deren Kosinus berechnet werden soll. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

Double-Wert

NULL, falls ein an die Funktion übergebener Wert NULL ist.

Beispiel

Der folgende Ausdruck gibt den Kosinus aller Werte im Port DEGREES zurück:

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

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

Tipp: Sie können anhand der an COS übergebenen Werte mathematische Berechnungen durchführen, bevor die Funktion den Kosinus berechnet. Beispiel: Sie können vor der Berechnung des Kosinus die Werte im Port in Radianten konvertieren:

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

COSH

Gibt den hyperbolischen Kosinus eines numerischen Werts zurück (in Radianten).

Syntax

```
COSH( numeric_value )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
numeric_value	Erforderlich	Numerischer Datentyp. Numerische Daten, ausgedrückt in Radianten (Grad multipliziert mit Pi durch 180). Übergibt die Werte, deren hyperbolischer Kosinus berechnet werden soll. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

Double-Wert

NULL, falls ein an die Funktion übergebener Wert NULL ist.

Beispiel

Der folgende Ausdruck gibt den hyperbolischen Kosinus der Werte im Port ANGLES zurück:

```
COSH( ANGLES )
```

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

Tipp: Sie können anhand der an COSH übergebenen Werte mathematische Berechnungen durchführen, bevor die Funktion den hyperbolischen Kosinus berechnet. Beispiel:

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

COUNT

Gibt die Anzahl der Zeilen zurück, die Gruppen mit Werten ungleich Null aufweisen. Optional können Sie mit dem Argument „*“ (Sternchen) alle Eingabewerte in einer Umwandlung zählen. Es kann nur eine weitere Aggregatfunktion in COUNT verschachtelt sein. Sie können eine Bedingung zum Filtern der Zeilen anwenden, bevor ihre Anzahl ermittelt wird.

Syntax

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

oder

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
value	Erforderlich	Alle Datentypen außer binär. Übergibt die zu zählenden Werte. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.
*	Optional	Dient zum Zählen *aller Zeilen* in einer Umwandlung.
filter_condition	Optional	Begrenzt die Zeilen in der Suche. Die Filterbedingung muss ein numerischer Wert sein oder mit TRUE, FALSE oder NULL ausgewertet werden. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

Ganzzahl.

0, wenn alle übergebenen Werte NULL sind (außer beim Sternchen-Argument).

Nullen

Wenn alle Werte Null sind, gibt die Funktion 0 zurück.

Mit dem Sternchen-Argument zählt die Funktion alle Zeilen, auch wenn eine Spalte in einer Zeile einen Nullwert enthält.

Mit dem Argument *value* ignoriert die Funktion die Spalten mit Nullwerten.

Hinweis: Standardmäßig behandelt Data Integration Service Nullwerte in Aggregatfunktionen als NULL. Wenn Sie einen Port oder eine Gruppe mit ausschließlich Nullwerten übergeben, gibt die Funktion NULL zurück. Beim Konfigurieren von Data Integration Service können Sie jedoch entscheiden, wie mit Nullwerten in Aggregatfunktionen umgegangen werden soll. Sie können Nullwerte in Aggregatfunktionen als 0 oder als NULL verarbeiten.

Gruppieren nach

COUNT gruppiert die Werte nach der Einstellung „Gruppieren nach Ports“, die Sie in der Umwandlung festlegen, und gibt pro Gruppe ein Ergebnis zurück. Wenn „Gruppieren nach Ports“ nicht festgelegt wurde, behandelt COUNT alle Zeilen als eine einzige Gruppe und gibt nur einen Wert zurück.

Beispiele

Der folgende Ausdruck zählt Artikel mit weniger als 5 Stück lagernd und berücksichtigt dabei keine Nullwerte:

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

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

RETURN VALUE: 1

In diesem Beispiel zählt die Funktion den Artikel „Halogen Flashlight“, aber nicht den NULL-Eintrag. Die Funktion zählt alle Zeilen in einer Umwandlung, einschließlich Nullwerte, wie im folgenden Beispiel dargestellt:

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

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

RETURN VALUE: 2

In diesem Beispiel zählt die Funktion den Artikel „Halogen Flashlight“ und den NULL-Eintrag. Mit dem Sternchen-Argument ohne Filter zählt die Funktion alle Zeilen, die an die Umwandlung übergeben werden. Beispiel:

```
COUNT( * )
```

ITEM_NAME	QTY
Flashlight	10
NULL	2
Compass	NULL

ITEM_NAME	QTY
Regulator System	5
Safety Knife	8
Halogen Flashlight	1

RETURN VALUE: 6

CRC32

Gibt einen 32-Bit-CRC-Wert (CRC32, Cyclic Redundancy Check) zurück. CRC32 dient zum Ermitteln von Übertragungsfehlern. Sie können CRC32 auch verwenden, um sicherzustellen, dass in einer Datei gespeicherte Daten nicht geändert wurden.

Bei Verwendung von CRC32 zur Durchführung einer Redundanzprüfung bei Daten im ASCII- und Unicode-Modus erzeugt der Data Integration Service möglicherweise unterschiedliche Ergebnisse für denselben Eingabewert. Bei Verwendung von CRC32 zur Durchführung einer Redundanzprüfung bei Daten auf verschiedenen Betriebssystemen erzeugt der Data Integration Service möglicherweise unterschiedliche Ergebnisse für denselben Eingabewert.

Hinweis: CRC32 kann dieselbe Ausgabe für verschiedene Eingabestrings zurückgeben. Wenn Sie Schlüssel in einem Mapping generieren möchten, benutzen Sie eine Sequenzgenerator-Umwandlung. Die Verwendung von CRC32 zum Erzeugen von Schlüsseln in einem Mapping kann zu unerwarteten Ergebnissen führen.

Syntax

```
CRC32( value )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
value	Erforderlich	Zeichenfolge oder binärer Datentyp. Übergibt die Werte, die einer Redundanzprüfung unterzogen werden sollen. Beim Eingabewert wird zwischen Groß- und Kleinschreibung unterschieden. Die Groß- bzw. Kleinschreibung des Eingabewerts wirkt sich auf den Rückgabewert aus. Beispiel: CRC32(informatica) und CRC32 (Informatica) geben unterschiedliche Werte zurück.

Rückgabewert

32-Bit-Ganzzahlwert.

Beispiel

Sie möchten Daten aus einer Quelle über ein WAN auslesen. Dabei möchten Sie sich vergewissern, dass die Daten bei der Übertragung nicht geändert wurden. Sie können die Prüfsumme der Daten in der Datei berechnen und sie zusammen mit der Datei speichern. Beim Lesen der Quelldaten kann Data Integration Service mithilfe von CRC32 deren Prüfsumme berechnen und mit dem gespeicherten Wert vergleichen. Wenn die beiden Werte gleich sind, wurden die Daten nicht geändert.

CREATE_TIMESTAMP_TZ

Erstellt einen „Zeitstempel mit Zeitzone“-Datentyp anhand der Zeitstempel- und Zeitzonenwerte.

Beim Ausgabeport muss es sich um timestampWithTZ for CREATE_TIMESTAMP_TZ-Ausdrücke handeln.

Syntax

```
CREATE_TIMESTAMP_TZ (timestamp_value, timezone_value)
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
timestamp_value	Erforderlich	Datum/Zeit-Datentyp. Sie können einen beliebigen gültigen Umwandlungsausdruck eingeben.
timezone_value	Erforderlich	Muss vom Datentyp „Zeichenfolge“ sein. Die Zeichenfolge muss eine Zeichenfolge sein. Übergibt die Werte, die für die Zeitzone erstellt werden sollen. Sie können einen beliebigen gültigen Umwandlungsausdruck eingeben, der in der Zeitzonendatei im Installationsverzeichnis definiert ist.

Rückgabewert

Gibt einen Zeitstempel mit dem Datentyp „Zeitzone“ zurück.

NULL, wenn die Eingabe ein Nullwert ist.

Beispiel

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

CUME

Gibt einen laufenden Kontostand zurück. Ein laufender Kontostand bedeutet, dass CUME jedes Mal eine Summe zurückgibt, wenn ein Wert hinzugefügt wird. Sie können eine Bedingung hinzufügen, um Zeilen aus dem Zeilensatz herauszufiltern, bevor der laufende Kontostand berechnet wird.

Mit CUME und ähnlichen Funktionen (z. B. MOVINGAVG und MOVINGSUM) können Sie das Berichtswesen vereinfachen, indem Sie laufende Werte berechnen.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
numeric_value	Erforderlich	Numerischer Datentyp. Übergibt den Wert, für den Sie einen laufenden Kontostand berechnen möchten. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben. Sie können auch verschachtelte Ausdrücke formulieren und den laufenden Kontostand für das Ergebnis der Funktion berechnen, solange dieses ein numerischer Wert ist.
filter_condition	Optional	Begrenzt die Zeilen in der Suche. Die Filterbedingung muss ein numerischer Wert sein oder mit TRUE, FALSE oder NULL ausgewertet werden. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

Numerischer Wert.

NULL, wenn alle übergebenen Werte NULL sind oder keine Zeilen ausgewählt wurden (z. B. wenn die Filterbedingung in allen Zeilen FALSE oder NULL ergibt).

Hinweis: Wenn der Rückgabewert eine Dezimalmal mit Präzision höher als 15 ist, können Sie „Hohe Präzision“ aktivieren, um Dezimalgenauigkeit bis zu 28 Stellen zu gewährleisten.

Nullen

Wenn nur ein Wert NULL ist, gibt CUME den laufenden Kontostand für die vorherige Zeile zurück. Wenn jedoch alle Werte im ausgewählten Port NULL ergeben, gibt CUME NULL zurück.

Beispiele

Folgender Zeilensatz könnte beispielsweise das Ergebnis einer CUME-Funktion sein:

```
CUME( PERSONAL_SALES )
```

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

Sie können auch Werte hinzufügen, bevor Sie den laufenden Kontostand berechnen:

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

CA_SALES	OR_SALES	RETURN VALUE
40000	10000	50000

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

DATE_COMPARE

Gibt eine Ganzzahl zurück, die zeigt, welcher von zwei Terminen der frühere ist. DATE_COMPARE gibt keinen Datumswert, sondern einen Ganzzahlwert zurück.

Syntax

```
DATE_COMPARE( date1, date2 )
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
date1	Erforderlich	Datum/Zeit-Datentyp. Das erste Datum im Vergleich. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben, sofern seine Auswertung ein Datum ergibt.
date2	Erforderlich	Datum/Zeit-Datentyp. Das zweite Datum im Vergleich. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben, sofern seine Auswertung ein Datum ergibt.

Rückgabewert

-1, wenn das erste Datum früher liegt.

0, wenn das erste Datum mit dem zweiten identisch ist.

1, wenn das zweite Datum früher liegt.

NULL, wenn einer der Datumswerte NULL ist.

Beispiel

Der folgende Ausdruck vergleicht alle Datumsangaben in den Ports DATE_PROMISED und DATE_SHIPPED und gibt eine Ganzzahl zurück, die angibt, welches Datum früher liegt:

```
DATE_COMPARE( DATE_PROMISED, DATE_SHIPPED )
```

DATE_PROMISED	DATE_SHIPPED	RETURN VALUE
Jan 1 1997	Jan 13 1997	-1
Feb 1 1997	Feb 1 1997	0

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

DATE_DIFF

Gibt den Zeitraum zwischen zwei Datumsangaben zurück. Beim Format können Sie zwischen folgenden Einheiten wählen: Jahre, Monate, Tage, Stunden, Minuten, Sekunden, Millisekunden, Mikrosekunden und Nanosekunden. Data Integration Service zieht das zweite Datum vom ersten ab und gibt die Differenz zurück.

In Data Integration Service basiert die Berechnung der DATE_DIFF-Funktion auf der Anzahl von Monaten und nicht der Anzahl von Tagen. Es berechnet den Datumsunterschied für Teilmonate mit den in jedem Monat ausgewählten Tagen. Zur Berechnung des Datumsunterschieds für den Teilmonat fügt Data Integration Service die in dem Monat bereits vergangenen Tage hinzu. Anschließend wird der Wert durch die Gesamtanzahl von Tagen im ausgewählten Monat dividiert.

Es kann sein, dass Data Integration Service für einen Zeitraum in einem Schaltjahr einen anderen Wert zurückgibt als für den gleichen Zeitraum in einem gewöhnlichen Jahr. Dieser Unterschied tritt dann auf, wenn Februar Teil der DATE_DIFF-Funktion ist. DATE_DIFF dividiert die Februartage in einem Schaltjahr durch 29, in anderen Jahren durch 28.

Beispiel: Sie möchten die Anzahl der Monate zwischen 13. September und 19. Februar berechnen. In einem Schaltjahr berechnet DATE_DIFF den Teilmonat Februar als 19/29 bzw. 0.655 Monate. In einem Nicht-Schaltjahr ergibt Februar 19/28 bzw. 0.678 Monate. Auf diese Weise berechnet Data Integration Service auch den Datumsunterschied für die übrigen Monate und gibt den Gesamtwert für den angegebenen Zeitraum zurück.

Hinweis: Einige Datenbanken verwenden möglicherweise andere Algorithmen zur Berechnung der Differenz zwischen Datumsangaben.

Syntax

```
DATE_DIFF( date1, date2, format )
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
date1	Erforderlich	Datum/Zeit-Datentyp. Übergibt das erste Datum im Vergleich. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.
date2	Erforderlich	Datum/Zeit-Datentyp. Übergibt das zweite Datum im Vergleich. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.
format	Erforderlich	Formatstring mit Einheitenangabe für Datum oder Uhrzeit. Sie können zwischen folgenden Einheiten wählen: Jahre, Monate, Tage, Stunden, Minuten, Sekunden, Millisekunden, Mikrosekunden und Nanosekunden. Sie können auch nur einen Teil des Datums spezifizieren, z. B. 'mm'. Setzen Sie den Formatstring zwischen einfache Anführungszeichen. Bei Formatstrings muss nicht auf Groß-/Kleinschreibung geachtet werden. Der Formatstring 'mm' hat beispielsweise dieselbe Bedeutung wie 'MM', 'Mm' oder 'mM'.

Rückgabewert

Double-Wert Wenn *date1* später als *date2* liegt, ist der Rückgabewert eine positive Zahl. Wenn *date1* früher als *date2* liegt, ist der Rückgabewert eine negative Zahl.

0, wenn das erste Datum mit dem zweiten identisch ist.

NULL, wenn einer (oder beide) der Datumswerte NULL ist.

Beispiele

Die folgenden Ausdrücke geben die Anzahl der Stunden zwischen den Ports DATE_PROMISED und DATE_SHIPPED zurück:

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

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

Die folgenden Ausdrücke geben die Anzahl der Tage zwischen den Ports DATE_PROMISED und DATE_SHIPPED zurück:

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

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

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

Die folgenden Ausdrücke geben die Anzahl der Monate zwischen den Ports DATE_PROMISED und DATE_SHIPPED zurück:

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

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

Die folgenden Ausdrücke geben die Anzahl der Jahre zwischen den Ports DATE_PROMISED und DATE_SHIPPED zurück:

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

DATE_PROMISED	DATE_SHIPPED	RETURN VALUE
Jan 1 1997 12:00:00AM	Mar 29 1997 12:00:00PM	-0.24327956989247
Mar 29 1997 12:00:00PM	Jan 1 1997 12:00:00AM	0.24327956989247
NULL	Dec 10 1997 5:55:10PM	NULL
Dec 10 1997 5:55:10PM	NULL	NULL

DATE_PROMISED	DATE_SHIPPED	RETURN VALUE
Jun 3 1997 1:13:46PM	Aug 23 1996 4:20:16PM	0.77741801697531
Feb 19 2004 12:00:00PM	Feb 19 2005 12:00:00PM	-1

Die folgenden Ausdrücke geben die Anzahl der Monate zwischen den Ports DATE_PROMISED und DATE_SHIPPED zurück:

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

DATE_PROMISED	DATE_SHIPPED	LEAP YEAR VALUE (in Months)	NON-LEAP YEAR VALUE (in Months)
Sept 13	Feb 19	-5.237931034	-5.260714286
NULL	Feb 19	NULL	N/A
Sept 13	NULL	NULL	N/A

DEC_BASE64

Dekodiert einen mit Base 64 kodierten Wert und gibt einen String mit Binärdaten zurück. Wenn Sie mithilfe von ENC_BASE64 kodierte Daten mit DEC_BASE64 dekodieren möchten, müssen Sie die Dekodierungssitzung im gleichen Datenverschiebungsmodus wie die Kodierung ausführen. Andernfalls kann die Ausgabe der dekodierten Daten von den ursprünglichen Daten abweichen.

Dekodiert einen mit Base 64 kodierten Wert und gibt einen String mit Binärdaten zurück. Wenn Sie mithilfe von ENC_BASE64 kodierte Daten mit DEC_BASE64 dekodieren möchten, müssen Sie das Mapping im gleichen Datenverschiebungsmodus wie die Kodierung ausführen. Andernfalls kann die Ausgabe der dekodierten Daten von den ursprünglichen Daten abweichen.

Syntax

```
DEC_BASE64( value )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
value	Erforderlich	Zeichenfolgen-Datentyp. Daten, die dekodiert werden sollen.

Rückgabewert

Dekodierter Binärwert.

NULL, wenn die Eingabe ein Nullwert ist.

Die Rückgabewerte weichen voneinander ab, wenn die Sitzung im Unicode-Modus und nicht im ASCII-Modus oder umgekehrt ausgeführt wird.

Die Rückgabewerte weichen voneinander ab, wenn die Zuordnung den Unicode-Modus anstelle des ASCII-Modus ausführt.

Beispiel

Sie haben WebSphere MQ-Nachrichten-IDs kodiert und während eines Arbeitsablaufs in eine Einfachdatei geschrieben. Sie möchten die Daten aus der Einfachdateiquelle lesen, einschließlich der WebSphere MQ-Nachrichten-IDs. Sie können die IDs mit DEC_BASE64 dekodieren und in den ursprünglichen Binärwert konvertieren.

DECODE

Durchsucht einen Port nach einem angegebenen Wert. Wenn die Funktion den Wert findet, wird der zuvor definierte Ergebniswert zurückgegeben. Sie können beliebig viele Suchvorgänge innerhalb der Funktion DECODE erstellen.

Beim Suchen eines Werts in einem String-Port mit DECODE können Sie entweder nachgestellte Leerzeichen mit der Funktion RTRIM entfernen oder die Leeerzeichen in den Suchstring mit aufnehmen.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
value	Erforderlich	Alle Datentypen außer binär. Übergibt die Werte, nach denen gesucht werden soll. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.
search	Erforderlich	Beliebiger Wert desselben Datentyps wie das „value“-Argument. Übergibt die Werte, in denen gesucht werden soll. Der „search“-Wert muss mit dem „value“-Argument übereinstimmen. Sie können nicht nach einem Teil eines Werts suchen. Der Suchwert unterscheidet zwischen Groß- und Kleinschreibung. Beispiel: Wenn Sie den String „Halogen Flashlight“ in einem bestimmten Port suchen möchten, müssen Sie „Halogen Flashlight“ und nicht nur „Halogen“ angeben. Für „Halogen“ wird die Suche keinen übereinstimmenden Wert finden. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.
result	Erforderlich	Alle Datentypen außer binär. Der Wert, der zurückgegeben werden soll, wenn die Suche einen übereinstimmenden Wert findet: Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.
default	Optional	Alle Datentypen außer binär. Der Wert, der zurückgegeben wird, wenn die Suche keinen übereinstimmenden Wert findet. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

First_result, wenn die Suche einen übereinstimmenden Wert findet.

„default“-Wert, wenn die Suche keinen übereinstimmenden Wert findet.

NULL, wenn Sie kein „default“-Argument angeben und die Suche keinen übereinstimmenden Wert findet.

Auch wenn mehrere Bedingungen erfüllt sind, gibt Data Integration Service das erste übereinstimmende Ergebnis zurück.

Wenn die Daten Multibyte-Zeichen enthalten und der DECODE-Ausdruck Stringdaten vergleicht, hängt der Rückgabewert von der Codepage und dem Datenverschiebungsmodus von Data Integration Service ab.

DECODE und Datentypen

Bei DECODE ist der Datentyp des Rückgabewerts immer der gleiche wie der Datentyp des Ergebnisses mit der höchsten Präzision.

Beispiel: Sie haben folgenden Ausdruck formuliert:

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

Die Rückgabewerte in diesem Ausdruck lauten 5, 1.414213562 und 3.141592654. Der erste Ergebnis ist eine Ganzzahl, die anderen sind Dezimalzahlen. Der Dezimaldatentyp weist eine höhere Präzision auf als die Ganzzahl. Dieser Ausdruck schreibt das Ergebnis immer als Dezimalzahl.

Wenn bei Sitzungen im hohen Präzisionsmodus mindestens ein Ergebnis einen Double-Wert liefert, ist auch der Datentyp des Rückgabewerts Double.

Wenn bei Mappings im hohen Präzisionsmodus mindestens ein Ergebnis einen Double-Wert liefert, ist auch der Datentyp des Rückgabewerts Double.

DECODE-Funktionen mit String- und numerischen Rückgabewerten sind nicht möglich.

Beispiel: Der folgende Ausdruck ist ungültig:

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

Beim Validieren des Ausdrucks oben erhalten Sie folgende Fehlermeldung:

```
Function cannot resolve operands of ambiguously mismatching datatypes.
```

Beispiele

Sie können DECODE in einem Ausdruck verwenden, der eine bestimmte ITEM_ID sucht und den entsprechenden ITEM_NAME zurückgibt:

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

ITEM_ID	RETURN VALUE
10	Flashlight
14	Regulator
17	NONE
20	Knife
25	NONE

ITEM_ID	RETURN VALUE
NULL	NONE
40	Tank

DECODE gibt den Standardwert NONE für Artikel 17 und 25 zurück, weil die Suchwerte der ITEM_ID nicht entsprechen. Auch für die ITEM_ID NULL gibt DECODE NONE zurück.

Der folgende Ausdruck testet mehrere Spalten und Bedingungen, die von oben nach unten mit TRUE und FALSE ausgewertet werden:

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

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

DECOMPRESS

Dekomprimiert Daten mithilfe des Kompressionsalgorithmus zlib 1.2.1. Verwenden Sie die Funktion DECOMPRESS bei Daten, die mit der Funktion COMPRESS oder einem Tool zur Datenkompression, das zlib 1.2.1 verwendet, komprimiert wurden. Wenn bei der Dekompression ein anderer Datenverschiebungsmodus als bei der Kompression verwendet wird, kann die Ausgabe der dekomprimierten Daten von den Originaldaten abweichen.

Dekomprimiert Daten mithilfe des Kompressionsalgorithmus zlib 1.2.1. Verwenden Sie die Funktion DECOMPRESS bei Daten, die mit der Funktion COMPRESS oder einem Tool zur Datenkompression, das zlib 1.2.1 verwendet, komprimiert wurden. Wenn bei der Dekompression ein anderer Datenverschiebungsmodus als bei der Kompression verwendet wird, kann die Ausgabe der dekomprimierten Daten von den Originaldaten abweichen.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich / Optional	Beschreibung
value	Erforderlich	Binärer Datentyp. Die Daten, die dekomprimiert werden sollen.
precision	Optional	Ganzzahl-Datentyp.

Rückgabewert

Dekomprimierter Binärwert des Eingabewerts.

NULL, wenn der Eingabewert ein Nullwert ist.

Beispiel

Ihr Unternehmen betreibt einen Online-Bestelldienst. Sie haben komprimierte Kundenbestelldaten über ein WAN erhalten. Nun möchten Sie die Daten mit PowerCenter auslesen und in ein Data Warehouse einfügen. Dekomprimieren Sie die einzelnen Datenzeilen, indem Sie DECOMPRESS auf alle Zeilen anwenden. Data Integration Service kann die dekomprimierten Daten dann in das Ziel laden.

ENC_BASE64

Kodiert Daten durch Konvertierung von Binärdaten in Stringdaten mithilfe der MIME-Kodierung (Multipurpose Internet Mail Extensions). Kodieren Sie die Daten, wenn Sie sie in einer Datenbank oder einer Datei speichern möchten, die keine Binärdaten erlaubt. Sie können Daten auch kodieren, um binäre Daten durch Umwandlungen im Stringformat zu übergeben. Kodierte Daten sind ca. 33 % länger als die Originaldaten. Ihre Anzeige besteht aus einer Reihe zufälliger Zeichen.

Syntax

```
ENC_BASE64( value )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
value	Erforderlich	Binär- oder String-Datentyp. Daten, die kodiert werden sollen.

Rückgabewert

Kodierter Wert.

NULL, wenn der Eingabewert ein Nullwert ist.

Beispiel

Sie möchten Nachrichten aus WebSphere MQ auslesen und in ein Einfachdateiziel schreiben. Auch die WebSphere MQ-Nachrichten-IDs sollen in die Zieldaten aufgenommen werden. Das Feld MsgID ist binär, die Einfachdateiziel unterstützt jedoch keine Binärdaten. Kodieren Sie MsgID mit ENC_BASE64, bevor Data Integration Service die Daten in das Ziel schreibt.

ERROR

Zwingt Data Integration Service zum Überspringen einer Zeile und Ausgeben einer von Ihnen festgelegten Fehlermeldung. Die Fehlermeldung wird in der Sitzungs-Logdatei angezeigt. Data Integration Service trägt die übersprungenen Zeilen nicht in die Ablehnungsdatei der Sitzung ein.

Zwingt Data Integration Service zum Überspringen einer Zeile und Ausgeben einer von Ihnen festgelegten Fehlermeldung. Die Fehlermeldung wird in der Sitzungs-Logdatei angezeigt. Data Integration Service trägt die übersprungenen Zeilen nicht in die Ablehnungsdatei ein.

Mit ERROR in Ausdrucksumwandlungen können Sie Daten validieren. Im Allgemeinen wird ERROR innerhalb einer IIF- oder DECODE-Funktion eingesetzt, um die Regeln für das Überspringen von Zeilen festzulegen.

Verwenden Sie ERROR für die Standardwerte sowohl der Eingabe- als auch der Ausgabeports. Sie können ERROR bei Eingabeports dazu nutzen, die Übergabe von Nullwerten in die Umwandlung zu verhindern.

Außerdem können Sie mit ERROR in Ausgabeports alle Arten von Umwandlungsfehlern behandeln, einschließlich der ERROR-Funktionsaufrufe innerhalb eines Ausdrucks. Wenn Sie die Funktion ERROR in einem Ausdruck und im Standardwert des Ausgabeports verwenden, überspringt Data Integration Service die Zeile und protokolliert sowohl die Fehlermeldung aus dem Ausdruck als auch jene aus dem Standardwert. Wenn Sie sichergehen möchten, dass Data Integration Service alle Zeilen überspringt, in denen ein Fehler auftritt, ordnen Sie ERROR als Standardwert zu.

Bei anderen Ausgabestandardwerten als ERROR überschreibt der Standardwert die Funktion ERROR in einem Ausdruck. Beispiel: Sie verwenden ERROR in einem Ausdruck, weisen dem Ausgabeport als Standardwert jedoch „1234“ zu. Jedes Mal, wenn Data Integration Service nun im Ausdruck auf die Funktion ERROR trifft, überschreibt es den Fehler mit dem Wert „1234“ und übergibt diesen Wert an die nächste Umwandlung. Die Zeile wird nicht übersprungen und es wird keine Fehlermeldung in das Sitzungsprotokoll eingetragen.

Bei anderen Ausgabestandardwerten als ERROR überschreibt der Standardwert die Funktion ERROR in einem Ausdruck. Beispiel: Sie verwenden ERROR in einem Ausdruck, weisen dem Ausgabeport als Standardwert jedoch „1234“ zu. Jedes Mal, wenn Data Integration Service nun im Ausdruck auf die Funktion ERROR trifft, überschreibt es den Fehler mit dem Wert „1234“ und übergibt diesen Wert an die nächste Umwandlung. Die Zeile wird nicht übersprungen und es wird keine Fehlermeldung in das Protokoll eingetragen.

Syntax

```
ERROR( string )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
string	Erforderlich	Stringwert. Die Meldung, die angezeigt werden soll, wenn Integration Service infolge eines Ausdrucks mit der Funktion ERROR eine Zeile überspringt. Der String kann beliebig lang sein.

Rückgabewert

String.

Beispiel

Das folgende Beispiel zeigt, wie ein Mapping referenziert wird, das das durchschnittliche Gehalt von Mitarbeitern in allen Abteilungen des Unternehmens berechnet, negative Werte aber überspringt. Die Funktion

ERROR wird in einem IIF-Ausdruck verschachtelt, sodass Data Integration Service bei einem negativen Wert im Port SALARY die betreffende Zeile überspringt und einen Fehler anzeigt:

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

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

EXP

Gibt e hoch n zurück, wobei n (Exponent) von Ihnen festgelegt wird und e = 2.71828183. Beispiel: EXP(2) gibt 7.38905609893065 zurück. Diese Funktion dient eher der Analyse von wissenschaftlichen oder technischen Daten als jener von Geschäftsdaten. EXP ist der Kehrwert der Funktion LN, die den natürlichen Logarithmus eines numerischen Werts zurückgibt.

Syntax

```
EXP( exponent )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
exponent	Erforderlich	Numerischer Datentyp. Der Wert, um den e potenziert werden soll. Der Exponent in der Gleichung e^*Wert*. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

Double-Wert

NULL, wenn ein als Argument übergebener Wert NULL ist.

Beispiel

Der folgende Ausdruck verwendet die im Port NUMBERS gespeicherten Werte als Exponentwert:

```
EXP( NUMBERS )
```

NUMBERS	RETURN VALUE
10	22026.4657948067
-2	0.135335283236613

NUMBERS	RETURN VALUE
8.55	5166.754427176
NULL	NULL

FIRST

Gibt den ersten Wert zurück, der in einem Port oder einer Gruppe gefunden wurde. Optional können Sie einen Filter anwenden, um die Anzahl der Zeilen zu beschränken, die Data Integration Service ausliest. Es kann nur eine weitere Aggregatfunktion in FIRST verschachtelt sein.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
value	Erforderlich	Alle Datentypen außer binär. Übergibt die Werte, für die Sie den ersten Wert zurückgeben möchten. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.
filter_condition	Optional	Begrenzt die Zeilen in der Suche. Die Filterbedingung muss ein numerischer Wert sein oder mit TRUE, FALSE oder NULL ausgewertet werden. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

Erster Wert in einer Gruppe.

NULL, wenn alle übergebenen Werte NULL sind oder keine Zeilen ausgewählt wurden (z. B. wenn die Filterbedingung in allen Zeilen FALSE oder NULL ergibt).

Nullen

Wenn nur ein Wert NULL ist, ignoriert FIRST die Zeile. Wenn jedoch alle vom Port übergebenen Werte NULL ergeben, gibt FIRST NULL zurück.

Hinweis: Standardmäßig behandelt Data Integration Service Nullwerte in Aggregatfunktionen als NULL. Wenn Sie einen Port oder eine Gruppe mit ausschließlich Nullwerten übergeben, gibt die Funktion NULL zurück. Beim Konfigurieren von Data Integration Service können Sie jedoch entscheiden, wie mit Nullwerten in Aggregatfunktionen umgegangen werden soll. Sie können Nullwerte in Aggregatfunktionen als 0 oder als NULL verarbeiten.

Gruppieren nach

FIRST gruppiert die Werte nach der Einstellung „Gruppieren nach Ports“, die Sie in der Umwandlung festlegen, und gibt pro Gruppe ein Ergebnis zurück.

Wenn „Gruppieren nach Ports“ nicht festgelegt wurde, behandelt FIRST alle Zeilen als eine einzige Gruppe und gibt nur einen Wert zurück.

Beispiele

Der folgende Ausdruck gibt den ersten Wert im Port ITEM_NAME mit einem höheren Preis als 10.00 USD zurück:

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

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

RETURN VALUE: Flashlight

Der folgende Ausdruck gibt den ersten Wert im Port ITEM_NAME mit einem höheren Preis als $40.00 USD zurück:

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

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

RETURN VALUE: Regulator System

FLOOR

Gibt die größte Ganzzahl zurück, die kleiner oder gleich des numerischen Werts ist, der an diese Funktion übergeben wird. Beispiel: Wenn Sie 3.14 an FLOOR übergeben, gibt die Funktion 3 zurück. Wenn Sie 3.98 an FLOOR übergeben, gibt die Funktion 3 zurück. Wenn Sie -3.17 übergeben, gibt die Funktion -4 zurück.

Syntax

```
FLOOR( numeric_value )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
numeric_value	Erforderlich	Numerischer Datentyp. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben, sofern seine Auswertung numerische Daten ergibt.

Rückgabewert

Ganzzahl, wenn Sie einen numerischen Wert mit deklarierter Präzision zwischen 0 und 28 übergeben.

Double-Wert, wenn Sie einen numerischen Wert mit deklarierter Präzision über 28 übergeben.

NULL, falls ein an die Funktion übergebener Wert NULL ist.

Beispiel

Der folgende Ausdruck gibt die größte Ganzzahl zurück, die kleiner oder gleich den Werten im Port PRICE ist:

```
FLOOR( PRICE )
```

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

Tipp: Sie können anhand der an FLOOR übergebenen Werte mathematische Berechnungen durchführen. Beispiel: Wenn Sie einen ganzzahligen Wert mit 10 multiplizieren und anschließend die größte Ganzzahl, die kleiner als das Produkt ist, berechnen möchten, können Sie die Funktion wie folgt formulieren:

```
FLOOR( UNIT_PRICE * 10 )
```

FV

Gibt den Endwert (Future Value) einer Investition zurück, wobei Sie periodische konstante Zahlungen leisten und die Investition konstante Zinsen erwirtschaftet.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
rate	Erforderlich	Numerisch. Zinsen, die in jeder Periode erwirtschaftet werden. Ausgedrückt als Dezimalzahl. Dividieren Sie den Zinsfuß in % durch 100, um eine Dezimalzahl zu erhalten. Muss größer oder gleich 0 sein.
terms	Erforderlich	Numerisch. Anzahl der Perioden oder Zahlungen. Muss größer als 0 sein.
payment	Erforderlich	Numerisch. Zahlungsbetrag, der pro Periode fällig ist. Muss eine negative Zahl sein.
present value	Optional	Numerisch. Aktueller Wert der Investition. Wenn Sie diesen Wert nicht angeben, verwendet FV 0.
type	Optional	Ganzzahl. Zeitplan der Zahlung. Geben Sie 1 ein, wenn die Zahlung am Anfang der Periode erfolgt. Geben Sie 0 ein, wenn die Zahlung am Ende der Periode erfolgt. Standardwert ist 0. Andere Werte als 0 oder 1 werden von Data Integration Service als 1 behandelt.

Rückgabewert

Numerisch.

Beispiel

Sie legen 2000 USD auf Konto ein, das bei monatlicher Kapitalisierung mit jährlich 9 % verzinst wird (monatliche Zinsen = 9 % / 12 = 0.75 %). Sie möchten am Anfang jedes Monats für die Dauer von 12 Monaten 250 USD einlegen. Der folgende Ausdruck gibt für den Kontostand am Ende der 12 Monate 5337.96 USD zurück:

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

Hinweise

Zur Berechnung der in jeder Periode erwirtschafteten Zinsen dividieren Sie den jährlichen Zinssatz durch die Anzahl der Zahlungen im Jahresverlauf. Die Werte für „payment“ (Zahlung) und „present value“ (Zeitwert) sind negativ, da sie zahlbare Beträge darstellen.

GET_DATE_PART

Gibt den angegebenen Teil eines Datums als Ganzzahlwert zurück. Wenn Sie also in einem Ausdruck die Monatskomponente eines Datums angeben und beispielsweise den 1. April 1997 00:00:00 übergeben, gibt GET_DATE_PART den Wert 4 zurück.

Syntax

```
GET_DATE_PART( date, format )
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
Datum	Erforderlich	Datum/Zeit-Datentyp. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.
format	Erforderlich	Ein Formatstring, der den Teil des Datums angibt, der zurückgegeben werden soll. Setzen Sie Formatstrings zwischen einfache Anführungszeichen, z. B. 'MM'. Bei Formatstrings muss nicht auf Groß-/Kleinschreibung geachtet werden. Jeder Formatstring gibt je nach festgelegtem Datumsformat in der Sitzung den gesamten Teil des Datums zurück. Ein Formatstring, der den Teil des Datums angibt, der zurückgegeben werden soll. Setzen Sie Formatstrings zwischen einfache Anführungszeichen, z. B. 'MM'. Bei Formatstrings muss nicht auf Groß-/Kleinschreibung geachtet werden. Jeder Formatstring gibt je nach festgelegtem Datumsformat im Mapping den gesamten Teil des Datums zurück. Beispiel: Wenn Sie das Datum 1. April 1997 an GET_DATE_PART übergeben, geben die Formatstrings 'Y', 'YY', 'YYY' und 'YYYY' alle 1997 zurück.

Rückgabewert

Ganzzahl, die den angegebenen Teil des Datums darstellt.

NULL, falls ein an die Funktion übergebener Wert NULL ist.

Beispiele

Die folgenden Ausdrücke geben die Stundenkomponente aller Datumsangaben im Port DATE_SHIPPED wieder: 12:00:00AM gibt 0 zurück, da das Standarddatumsformatbasiert auf dem 24-Stunden-Intervall basiert:

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

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

Die folgenden Ausdrücke geben die Tageskomponente aller Datumsangaben im Port DATE_SHIPPED wieder:

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

```
GET_DATE_PART( DATE_SHIPPED, 'DY' )
GET_DATE_PART( DATE_SHIPPED, 'DAY' )
```

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

Die folgenden Ausdrücke geben die Monatskomponente aller Datumsangaben im Port DATE_SHIPPED wieder:

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

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

Die folgenden Ausdrücke geben die Jahreskomponente aller Datumsangaben im Port DATE_SHIPPED wieder:

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

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

GET_TIMEZONE

Gibt den Zeitzonenwert aus einem bestimmten Zeitstempel mit der Zeitzonenspalte zurück.

Beispiel:

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

Der Ausgabeport muss den Datentyp „Zeichenfolge“ für die GET_TIMEZONE-Ausdrücke aufweisen.

Syntax

```
GET_TIMEZONE (timestamp_with_timezone_value)
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
timestamp_with_timezone_value	Erforderlich	Muss ein Zeitstempel mit dem Datentyp „Zeitzone“ sein. Sie können einen beliebigen gültigen Umwandlungsausdruck eingeben.

Rückgabewert

Gibt einen Zeichenfolgendatentyp zurück, der den Regionsnamen der Zeitzone oder ein Zeitzonen-Offset enthält.

NULL, wenn die Eingabe ein Nullwert ist.

Beispiel

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

GET_TIMESTAMP

Gibt den Datums-/Uhrzeitwert für einen timestampWithTZ-Eingabetyp zurück. Das zurückgegebene Datum und die zurückgegebene Uhrzeit liegen in der angeforderten Zeitzone, die als zweites Argument bereitgestellt werden kann. Wenn der Zeitzonenwert nicht im zweiten Argument angegeben wird, gibt die Funktion den Zeitstempelteil des timestampWithTZ-Eingabewerts zurück.

Beispiel:

```
GET_TIMESTAMP(Timestamp with Time Zone,“+08:30”)
```

Das erste Argument, Zeitstempel mit Zeitzone, weist (+05:30) als Zeitzonenwert auf. Die Funktion gibt den Zeitstempel in der als zweites Argument festgelegten Zeitzone (+08:30) zurück.

Der Ausgabeport muss Datum/Uhrzeit für GET_TIMESTAMP-Ausdrücke sein.

Syntax

```
GET_TIMESTAMP (timestamp_with_timezone_value, [timezone_value])
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
timestamp_with_timezone_value	Erforderlich	Muss ein Zeitstempel mit dem Datentyp „Zeitzone“ sein. Sie können einen beliebigen gültigen Umwandlungsausdruck eingeben.
timezone_value	Optional	Muss vom Datentyp „Zeichenfolge“ sein. Die Zeichenfolge muss eine Zeichenfolge sein. Übergibt die für die Zeitzone anzuzeigenden Werte basierend auf der Funktion, die den Zeitstempel zurückgeben kann. Sie können einen beliebigen gültigen Umwandlungsausdruck eingeben. Wenn Sie die Zeitzone nicht angeben, gibt die Funktion den Zeitstempelteil des ersten Arguments zurück.

Rückgabewert

Gibt den Zeitstempelwert im angegebenen Zeitzonen-Offset oder der angegebenen Region zurück.

Wenn der Zeitzonenwert nicht übergeben wird, gibt die Funktion den Zeitstempelteil des ersten Arguments zurück.

NULL, wenn die Eingabe ein Nullwert ist.

Beispiel

```
INPUT VALUE                          RETURN VALUE

'1996-01-05 10:45:00.221111111 AM Returns the timestamp value in time zone offset of
America/Los_Angeles’, '+05:30'    '+05:30':
                                  ’1996-01-06 12:15:00.221111111 AM’

'1996-01-05 10:45:00.221111111 AM Returns the timestamp value in the ‘GMT’ time zone:
America/Los_Angeles’, 'GMT’       ‘1996-01-05 06:45:00.221111111 PM’

'1996-01-05 10:45:00.221111111 AM As the time zone value is not passed as the second input
America/Los_Angeles’              parameter, the timestamp is returned:
                                  ‘1996-01-05 10:45:00.221111111 AM’
```

GREATEST

Gibt den größten Wert aus einer Liste von Eingabewerten zurück. Nutzen Sie diese Funktion, um den größten String, das größte Datum oder die größte Zahl zurückzugeben. Standardgemäß werden Groß-/Kleinschreibung unterschieden.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
value	Erforderlich	Alle Datentypen außer binär. Der Datentyp muss mit anderen Werten kompatibel sein. Wert, den Sie mit anderen Werten vergleichen möchten. Sie müssen mindestens ein Wertargument eingeben. Wenn der Wert und andere Eingabewerte numerisch sind, verwenden alle Werte die höchstmögliche Genauigkeit. Wenn beispielsweise bestimmte Werte den Datentyp „Ganzzahl“ und andere Werte den Datentyp „Doppelt“ aufweisen, wandelt der Data Integration Service die Werte in den Datentyp „Doppelt“ um.
CaseFlag	Optional	Muss eine Ganzzahl sein. Geben Sie einen Wert an, wenn das Eingabewertargument ein Zeichenfolgenwert ist. Legt fest, ob für die Argumente in dieser Funktion zwischen Groß- und Kleinschreibung unterschieden wird. Sie können einen beliebigen gültigen Umwandlungsausdruck eingeben. Wenn CaseFlag eine Zahl ungleich 0 ist, wird bei der Funktion zwischen Groß- und Kleinschreibung unterschieden. Wenn CaseFlag gleich 0 ist, wird bei der Funktion nicht zwischen Groß- und Kleinschreibung unterschieden. Standardwert ist „Groß-/Kleinschreibung beachten“.

Rückgabewert

value1, wenn dieser der größte der Eingabewerte ist, *value2*, wenn dieser der größte der Eingabewerte ist usw.

NULL, wenn eines der Argumente Null ist.

Beispiel

Der folgende Ausdruck gibt die größte Artikelbestellmenge zurück:

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

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

IIF

Gibt einen von zwei angegebenen Werten zurück, abhängig vom Ergebnis einer Bedingung.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
condition	Erforderlich	Die Bedingung, die ausgewertet werden soll. Sie können jeden beliebigen Umwandlungsausdruck eingeben, dessen Auswertung TRUE oder FALSE ergibt.
value1	Erforderlich	Alle Datentypen außer binär. Der Wert, der zurückgegeben werden soll, wenn die Bedingung TRUE ist. Der Rückgabewert weist immer den von diesem Argument angegebenen Datentyp auf. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben, auch einen weiteren IIF-Ausdruck.
value2	Optional	Alle Datentypen außer binär. Der Wert, der zurückgegeben werden soll, wenn die Bedingung FALSE ist. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben, auch einen weiteren IIF-Ausdruck.

Im Unterschied zu konditionalen Funktionen in manchen Systemen ist die Bedingung FALSE (*value2*) in der IIF-Funktion nicht erforderlich. Wenn Sie *value2* nicht angeben, gibt die Funktion bei FALSE Folgendes zurück:

- 0, wenn *value1* ein numerischer Datentyp ist.
- Leerer String, wenn *value1* ein String-Datentyp ist.
- NULL, wenn *value1* ein Datum/Zeit-Datentyp ist.

Beispiel: Der folgende Ausdruck enthält keine FALSE-Bedingung und *value1* weist einen String-Datentyp auf; daher gibt Data Integration Service einen leeren String für jede Zeile zurück, die mit FALSE ausgewertet wird:

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

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

Rückgabewert

value1, wenn die Bedingung TRUE lautet.

value2, wenn die Bedingung FALSE lautet.

Beispiel: Der folgende Ausdruck enthält die FALSE-Bedingung NULL,; daher gibt Data Integration Service Null für jede Zeile zurück, die mit FALSE ausgewertet wird:

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

SALES	EMP_NAME	RETURN VALUE
150	John Smith	John Smith
50	Pierre Bleu	NULL

SALES	EMP_NAME	RETURN VALUE
120	Sally Green	Sally Green
NULL	Greg Jones	NULL

Wenn die Daten Multibyte-Zeichen enthalten und das Bedingungsargument Stringdaten vergleicht, hängt der Rückgabewert von der Codepage und dem Datenverschiebungsmodus von Data Integration Service ab.

IIF und Datentypen

Bei IIF ist der Datentyp des Rückgabewerts der gleiche wie der Datentyp des Ergebnisses mit der höchsten Präzision.

Beispiel: Sie haben folgenden Ausdruck formuliert:

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

Die Ergebnis für TRUE (1) ist eine Ganzzahl und jenes für FALSE (.3333) eine Dezimalzahl. Der Dezimaldatentyp weist die höhere Präzision als die Ganzzahl auf, sodass der Datentyp des Rückgabewerts immer eine Dezimalzahl ist.

Wenn bei Sitzungen im hohen Präzisionsmodus mindestens ein Ergebnis einen Double-Wert liefert, ist auch der Datentyp des Rückgabewerts Double.

Wenn bei Mappings im hohen Präzisionsmodus mindestens ein Ergebnis einen Double-Wert liefert, ist auch der Datentyp des Rückgabewerts Double.

Besondere Verwendungszwecke von IIF

Mit verschachtelten IIF-Anweisungen können Sie mehrere Bedingungen testen. Das folgende Beispiel testet verschiedene Bedingungen und gibt 0 zurück, wenn der Umsatz 0 oder negativ ist:

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

Mithilfe von Kommentaren können Sie diesen logischen Ausdruck leichter lesbar machen:

```
IIF( SALES > 0,
--then test to see if sales is between 1 and 49:
   IIF( SALES < 50,

   --then return SALARY1
      SALARY1,

      --else test to see if sales is between 50 and 99:
         IIF( SALES < 100,

          --then return
             SALARY2,

             --else test to see if sales is between 100 and 199:
                IIF( SALES < 200,

                --then return
                   SALARY3,

                  --else for sales over 199, return
                     BONUS)
              )
           ),

--else for sales less than or equal to zero, return
      0)
```

Verwenden Sie IIF in Update-Strategien. Beispiel:

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

Alternative zu IIF

In vielen Fällen können Sie statt IIF "DECODE" auf Seite 89 verwenden. DECODE kann die Lesbarkeit verbessern. Im Folgenden sehen Sie die Verwendung von DECODE anstelle von IIF anhand des ersten Beispiels aus dem vorherigen Abschnitt:

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

Häufig können Sie mit einer Filterumwandlung anstelle von IIF die Sitzungsleistung maximieren.

Häufig können Sie mit einer Filterumwandlung anstelle von IIF die Leistung maximieren.

IN

Prüft die Übereinstimmung von Eingabedaten mit einer Werteliste. Standardgemäß werden Groß-/Kleinschreibung unterschieden.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
valueToSearch	Erforderlich	Kann eine Zeichenfolge, ein Datum oder ein numerischer Wert sein. Eingabewert, der mit einer durch Kommas getrennten Liste von Werten abgeglichen werden soll.
value	Erforderlich	Kann je nach Typ, der für das Argument valueToSearch angegeben ist, eine Zeichenfolge, ein Datum oder ein numerischer Wert sein. Durch Kommas getrennte Liste von Werten, nach denen gesucht werden soll. Werte können Ports in einer Umwandlung sein. Die Anzahl der Werte, die aufgelistet werden können, ist unbegrenzt.
CaseFlag	Optional	Muss eine Ganzzahl sein. Muss eine Ganzzahl oder NULL sein. Geben Sie einen Wert an, wenn das Argument valueToSearch ein Zeichenfolgenwert ist. Legt fest, ob für die Argumente in dieser Funktion zwischen Groß- und Kleinschreibung unterschieden wird. Sie können einen beliebigen gültigen Umwandlungsausdruck eingeben. Wenn CaseFlag eine Zahl ungleich 0 ist, wird bei der Funktion zwischen Groß- und Kleinschreibung unterschieden. Wenn CaseFlag gleich 0 ist, wird bei der Funktion nicht zwischen Groß- und Kleinschreibung unterschieden. Wenn CaseFlag gleich 0 ist, wird bei der Funktion nicht zwischen Groß- und Kleinschreibung unterschieden. Wenn CaseFlag ein Nullwert ist, gibt die Funktion NULL zurück, wenn das Argument nicht mit den Argumenten in der Funktion übereinstimmt. Andernfalls gibt CaseFlag den Wert 1 zurück, wenn eine Übereinstimmung mit dem Argument in der Funktion vorliegt. Standardwert ist „Groß-/Kleinschreibung beachten“.

Rückgabewert

TRUE (1), wenn der Eingabewert mit der Werteliste übereinstimmt.

FALSE (0), wenn der Eingabewert nicht mit der Werteliste übereinstimmt.

NULL, wenn die Eingabe ein Nullwert ist.

Beispiel

Der folgende Ausdruck stellt fest, ob der Eingabewert ein „Safety Knife“, „Chisel Point Knife“ oder „Medium Titanium Knife“ ist. Die Eingabewerte müssen nicht mit der Groß- und Kleinschreibung der Werte in der durch Kommas getrennten Liste übereinstimmen:

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

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

ITEM_NAME	RETURN VALUE
Medium Titanium knife	1 (TRUE)
	NULL

INDEXOF

Findet den Index eines Werts in einer Werteliste. Standardgemäß werden Groß-/Kleinschreibung unterschieden.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
valueToSearch	Erforderlich	Zeichenfolgen-Datentyp. Wert, nach dem Sie in der Liste der Zeichenfolgen suchen möchten.
string	Erforderlich	Zeichenfolgen-Datentyp. Kommagetrennte Liste von Werten, in der gesucht werden soll. Werte können im Zeichenfolgenformat vorliegen. Die Anzahl der Werte, die aufgelistet werden können, ist unbegrenzt. Der Wert unterscheidet zwischen Groß- und Kleinschreibung, es sei denn, Sie setzen CaseFlag auf 0.
CaseFlag	Optional	Muss eine Ganzzahl sein. Geben Sie einen Wert an, wenn das Argument valueToSearch ein Zeichenfolgenwert ist. Legt fest, ob für die Argumente in dieser Funktion zwischen Groß- und Kleinschreibung unterschieden wird. Sie können einen beliebigen gültigen Umwandlungsausdruck eingeben. Wenn CaseFlag eine Zahl ungleich 0 ist, wird bei der Funktion zwischen Groß- und Kleinschreibung unterschieden. Wenn CaseFlag gleich 0 ist, wird bei der Funktion nicht zwischen Groß- und Kleinschreibung unterschieden.

Rückgabewert

1, wenn der Eingabewert mit *string1* übereinstimmt; 2, wenn der Eingabewert mit *string2* übereinstimmt usw.

0, wenn der Eingabewert nicht gefunden wurde.

NULL, wenn die Eingabe ein Nullwert ist.

Beispiel

Der folgende Ausdruck stellt fest, ob die Werte aus dem Port ITEM_NAME mit dem ersten, zweiten oder dritten String übereinstimmen:

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

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

„Safety Knife“ ergibt 0, da die Großschreibung nicht mit dem kleingeschriebenen Eingabewert übereinstimmt.

INITCAP

Schreibt den ersten Buchstaben in jedem Wort einer Zeichenfolge groß und alle anderen Buchstaben klein. Wörter werden durch Leerräume (Leerzeichen, Seitenvorschub, Zeilenumbruch, Wagenrücklauf, Tabulator, vertikaler Tabulator) und nicht-alphanumerische Zeichen getrennt. Beispiel: Für den String „… THOMAS“ gibt die Funktion „Thomas“ zurück.

Syntax

```
INITCAP( string )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
string	Erforderlich	Alle Datentypen außer binär. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

String. Wenn die Daten Multibyte-Zeichen enthalten, hängt der Rückgabewert von der Codepage und dem Datenverschiebungsmodus von Data Integration Service ab.

NULL, falls ein an die Funktion übergebener Wert NULL ist.

Beispiel

Der folgende Ausdruck schreibt alle Namen im Port FIRST_NAME groß:

```
INITCAP( FIRST_NAME )
```

FIRST_NAME	RETURN VALUE
ramona	Ramona
18-albert	18-Albert
NULL	NULL
?!SAM	?!Sam
THOMAS	Thomas
PierRe	Pierre

INSTR

Gibt die Position eines Zeichens in einem String zurück, wobei von links nach rechts gezählt wird.

Syntax

```
INSTR( string, search_value [,start [,occurrence [,comparison_type ]]] )
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
string	Erforderlich	Der String muss ein Zeichenstring sein. Übergibt den auszuwertenden Wert. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben. Das Ergebnis des Ausdrucks muss ein Zeichenstring sein. Andernfalls konvertiert INSTR den Wert vor der Auswertung in einen String.
search_value	Erforderlich	Beliebiger Wert. Der Suchwert unterscheidet zwischen Groß- und Kleinschreibung. Übergibt den Zeichensatz, nach dem gesucht werden soll. Der Suchwert „search_value“ muss mit einem Teil des Strings übereinstimmen. Beispiel: Für den Ausdruck `INSTR('Alfred Pope', 'Alfred Smith')` gibt die Funktion 0 zurück. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben. Wenn Sie nach einem Zeichenstring suchen möchten, setzen Sie die gewünschten Zeichen zwischen einfache Anführungszeichen: 'abc'.

Argument	Erforderlich/ Optional	Beschreibung
start	Optional	Muss ein ganzzahliger Wert sein. Die Position im String, an der die Suche gestartet werden soll. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben. Der Standard ist 1. Das bedeutet, dass INSTR die Suche beim ersten Zeichen des Strings beginnt. Bei Startposition 0 beginnt die Suche mit dem ersten Zeichen im String. Wenn die Startposition eine positive Zahl ist, findet INSTR die Startposition durch Zählen der Positionen vom Stringanfang. Wenn die Startposition eine negative Zahl ist, wird die Startposition vom Stringende aus ermittelt. Wenn Sie dieses Argument nicht angeben, verwendet die Funktion den Standardwert 1.
occurrence	Optional	Eine positive Ganzzahl größer als 0. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben. Wenn der Wert mehr als einmal im String vorkommt, können Sie angeben, welches Vorkommen gesucht werden soll. Beispiel: Sie geben 2 ein, um das zweite Vorkommen von der Startposition aus zu suchen. Wenn Sie dieses Argument auslassen, wird der Standardwert 1 verwendet, was bedeutet, dass INSTR nach dem ersten Vorkommen des Suchwerts sucht. Bei der Übergabe von Dezimalzahlen rundet Data Integration Service auf den nächsten Ganzzahlwert . Wenn Sie eine negative Ganzzahl oder 0 übergeben, schlägt die Sitzung fehl.
comparison_type	Optional	Der Stringvergleichstyp, entweder linguistisch oder binär, wenn Data Integration Service im Unicode-Modus ausgeführt wird. Wenn Data Integration Service im ASCII-Modus ausgeführt wird, ist der Vergleichstyp immer binär. Bei linguistischen Vergleichen werden sprachspezifische Vergleichsregeln berücksichtigt, während bei binären Vergleichen bitweises Matching durchgeführt wird. Beispiel: Das deutsche scharfe ß stimmt beim linguistischen Vergleich mit „ss" überein, beim binären Vergleich nicht. Binäre Vergleiche werden schneller ausgeführt als linguistische. Muss ein Ganzzahlwert sein, entweder 0 oder 1: - 0: INSTR führt einen linguistischen Stringvergleich aus. - 1: INSTR führt einen binären Stringvergleich aus. Standardwert ist 0. Wenn Sie 0 eingeben, darf die Sortierreihenfolge für die Sitzung nicht binär sein.

Rückgabewert

Ganzzahl, wenn die Suche erfolgreich ist. Die Ganzzahl stellt die Position des ersten Zeichens im *search_value* dar, wobei von links nach rechts gezählt wird.

0, wenn die Suche nicht erfolgreich ist.

NULL, falls ein an die Funktion übergebener Wert NULL ist.

Beispiele

Der folgende Ausdruck gibt die Position des ersten Vorkommens des Buchstabens „a" zurück, und zwar beginnend mit jedem Firmennamen. Da das Argument *search_value* zwischen Groß- und Kleinschreibung

unterscheidet, überspringt es in „Blue Fin Aqua Center“ das große „A“ in „Aqua“ und gibt die Position des kleinen „a“ zurück:

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

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

Der folgende Ausdruck gibt die Position des zweiten Vorkommens des Buchstabens „a“ zurück, und zwar beginnend mit jedem Firmennamen. Da das Argument *search_value* zwischen Groß- und Kleinschreibung unterscheidet, überspringt es in „Blue Fin Aqua Center“ das große „A“ in „Aqua“ und gibt 0 zurück:

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

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

Der folgende Ausdruck gibt die Position des zweiten Vorkommens des Buchstabens „a“ in allen Firmennamen zurück, und zwar beginnend mit dem letzten Zeichen der Firmennamen. Da das Argument *search_value* zwischen Groß- und Kleinschreibung unterscheidet, überspringt es in „Blue Fin Aqua Center“ das große „A“ in „Aqua“ und gibt 0 zurück:

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

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

Der folgende Ausdruck gibt die Position des ersten Zeichens im String „Blue Fin Aqua Center“, und zwar beginnend mit dem letzten Zeichen des Firmennamens.

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

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

Verschachtelte INSTR-Funktionen

Für komplexere Aufgaben können Sie die Funktion INSTR in anderen Funktionen verschachteln.

Der folgende Ausdruck wertet einen String aus, beginnend beim Stringende. Der Ausdruck sucht das letzte (rechteste) Leerzeichen im String und gibt alle Zeichen links davon zurück:

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

CUST_NAME	RETURN VALUE
PATRICIA JONES	PATRICIA
MARY ELLEN SHAH	MARY ELLEN

Der folgende Ausdruck entfernt das Zeichen # aus einem String:

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

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

ISNULL

Gibt zurück, ob ein Wert NULL ist. ISNULL evaluiert einen leeren String als FALSE.

Hinweis: Zum Suchen von leeren Strings verwenden Sie LENGTH.

Syntax

```
ISNULL( value )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
value	Erforderlich	Alle Datentypen außer binär. Übergibt die auszuwertenden Zeilen. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

TRUE (1), wenn der Wert NULL ist.

FALSE (0), wenn der Wert nicht NULL ist.

Beispiel

Das folgende Beispiel überprüft die Artikeltabelle auf Nullwerte:

```
ISNULL( ITEM_NAME )
```

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

IS_DATE

Gibt zurück, ob ein Stringwert ein gültiges Datum ist. Ein gültiges Datum ist jeder String in der Datumskomponente des Datum/Zeit-Formats, das in der Sitzung festgelegt wurde. Wenn der String, den Sie testen möchten, nicht in diesem Datumsformat vorliegt, verwenden Sie den Formatstring TO_DATE zum Festlegen des Datumsformats. Wenn die an IS_DATE übergebenen Strings nicht mit dem angegebenen Formatstring übereinstimmen, gibt die Funktion FALSE (0) zurück. Wenn die Strings mit dem angegebenen Formatstring übereinstimmen, gibt die Funktion TRUE (1) zurück.

IS_DATE wertet Strings aus und gibt einen Ganzzahlwert zurück.

Der Ausgabeport für einen IS_DATE-Ausdruck muss einen String- oder numerischen Datentyp aufweisen.

Mit IS_DATE können Sie Daten in einer Einfachdatei testen oder filtern, bevor sie in ein Ziel geschrieben werden.

Mit IS_DATE sollten Sie den RR-Formatstring und nicht YY verwenden. In vielen Fällen führen beiden Formatstrings zu gleichen Werten, aber es gibt einige Situationen, in denen YY inkorrekte Ergebnisse ergibt. Beispiel: Der Ausdruck IS_DATE('02/29/00', 'YY') wird intern als IS_DATE(02/29/1900 00:00:00) interpretiert, was mit FALSE ausgewertet wird. Data Integration Service interpretiert IS_DATE('02/29/00', 'RR') hingegen als IS_DATE(02/29/2000 00:00:00) und gibt TRUE zurück. 1900 war kein Schaltjahr – es gab also keinen 29. Februar.

Hinweis: Für IS_DATE gelten dieselben Formatstrings wie für TO_DATE.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
value	Erforderlich	Muss ein String-Datentyp sein. Übergibt die auszuwertenden Zeilen. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.
format	Optional	Geben Sie einen gültigen TO_DATE-Formatstring ein. Der Formatstring muss den Teilen des Arguments *string* entsprechen. Beispiel: Beim Übergeben des Strings „Mar 15 1997 12:43:10AM“ müssen Sie den Formatstring DD MON YYYY HH12:MI:SSAM angeben. Wenn Sie den Formatstring auslassen, muss der Stringwert in dem Datumsformat vorliegen, das in der Sitzung definiert wurde. Geben Sie einen gültigen TO_DATE-Formatstring ein. Der Formatstring muss den Teilen des Arguments *string* entsprechen. Beispiel: Beim Übergeben des Strings „Mar 15 1997 12:43:10AM“ müssen Sie den Formatstring DD MON YYYY HH12:MI:SSAM angeben. Wenn Sie den Formatstring auslassen, muss der Stringwert in dem Datumsformat vorliegen, das in der Mapping-Konfiguration definiert wurde.

Rückgabewert

TRUE (1), wenn die Zeile ein gültiges Datum ist.

FALSE (0), wenn die Zeile kein gültiges Datum ist.

NULL, wenn ein Wert im Ausdruck oder der Formatstring NULL ist.

Warnung: Das Format des IS_DATE-Strings muss mit dem Formatstring übereinstimmen; dazu gehören auch die Datumstrennzeichen. Andernfalls kann Data Integration Service ungenaue Werte zurückgeben oder den Datensatz überspringen.

Beispiele

Der folgende Ausdruck prüft den Port INVOICE_DATE auf gültige Datumsangaben:

```
IS_DATE( INVOICE_DATE )
```

Der Ausdruck gibt Datumsangaben wie die folgenden zurück:

INVOICE_DATE	RETURN VALUE
NULL	NULL
'180'	0 (FALSE)
'04/01/98'	0 (FALSE)
'04/01/1998 00:12:15.7008'	1 (TRUE)
'02/31/1998 12:13:55.9204'	0 (FALSE) (*February does not have 31 days*)
'John Smith'	0 (FALSE)

Der folgende IS_DATE-Ausdruck spezifiziert den Formatstring YYYY/MM/DD:

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

Wenn der Stringwert nicht mit diesem Format übereinstimmt, gibt IS_DATE FALSE zurück:

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

Das folgende Beispiel zeigt, wie Sie mithilfe von IS_DATE Daten testen, bevor Sie diese Daten mit TO_DATE in Datumsangaben konvertieren. Dieser Ausdruck prüft die Werte im Port INVOICE_DATE und konvertiert jedes gültige Datum in einen Datumswert. Wenn der Wert kein gültiges Datum ist, gibt Data Integration Service einen Fehler zurück und überspringt die Zeile.

Dieses Beispiel gibt einen Datum/Zeit-Wert zurück. Daher muss der Datentyp des Ausgabeports für den Ausdruck Datum/Zeit lauten:

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

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

IS_NUMBER

Gibt zurück, ob ein String eine gültige Zahl ist. Eine gültige Zahl besteht aus den folgenden Teilen:

- Optionales Leerzeichen vor der Zahl
- Optionales Zeichen (+/-)
- Eine oder mehrere Ziffern mit einem Dezimaltrennzeichen

- Optionale wissenschaftliche Schreibweise wie die Buchstaben „e“ oder „E“ (und „d“ oder „D“ in Windows), gefolgt von einem optionalen Zeichen (+/-), gefolgt von einer oder mehreren Ziffern
- Optionaler Leerraum nach der Zahl

Folgende Zahlen sind gültig:

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

Der Ausgabeport für einen IS_NUMBER-Ausdruck muss einen String- oder numerischen Datentyp aufweisen.

Mit IS_NUMBER können Sie Daten in einer Einfachdatei testen oder filtern, bevor sie in ein Ziel geschrieben werden.

Syntax

```
IS_NUMBER( value )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
value	Erforderlich	Muss ein String-Datentyp sein. Übergibt die auszuwertenden Zeilen. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

TRUE (1), wenn die Zeile eine gültige Zahl ist.

FALSE (0), wenn die Zeile keine gültige Zahl ist.

NULL, falls ein Wert in der Funktion NULL ist.

Beispiele

Der folgende Ausdruck prüft den Port ITEM_PRICE auf gültige Zahlen:

```
IS_NUMBER( ITEM_PRICE )
```

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

ITEM_PRICE	RETURN VALUE	
'+123abc'	0 (False)	
' 123'	1 (True)	*Leading white blanks*
'123 '	1 (True)	*Trailing white blanks*
'ABC'	0 (False)	
'-ABC'	0 (False)	
NULL	NULL	

Mit IS_NUMBER können Sie Daten prüfen, bevor Sie eine der numerischen Konvertierungsfunktionen wie TO_FLOAT verwenden. Beispiel: Der folgende Ausdruck prüft die Werte im Port ITEM_PRICE und konvertiert alle gültigen Zahlen in Gleitkommawerte mit Double-Präzision. Wenn der Wert keine gültige Zahl ist, gibt Data Integration Service 0.00 zurück:

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

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

IS_SPACES

Gibt zurück, ob ein Stringwert vollständig aus Leerzeichen besteht. Als Leerzeichen gilt hierbei ein Leerraum (Leerzeichen, Seitenvorschub, Zeilenumbruch, Wagenrücklauf, Tabulator, vertikaler Tabulator).

IS_SPACES wertet einen leeren String mit FALSE aus, da er keine Leerzeichen enthält. Zum Suchen von leeren Strings verwenden Sie LENGTH.

Syntax

```
IS_SPACES( value )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
value	Erforderlich	Muss ein String-Datentyp sein. Übergibt die auszuwertenden Zeilen. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

TRUE (1), wenn die Zeile zur Gänze aus Leerzeichen besteht.

FALSE (0), wenn die Zeile Daten enthält.

NULL, falls ein Wert in der Funktion NULL ist.

Beispiel

Der folgende Ausdruck prüft den Port ITEM_NAME auf Zeilen, die zur Gänze aus Leerzeichen bestehen:

```
IS_SPACES( ITEM_NAME )
```

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

Tipp: Mit IS_SPACES können Sie vermeiden, dass Leerzeichen in eine Zeichenspalte der Zieltabelle eingetragen werden. Beispiel: Bei einer Umwandlung, die Kundennamen in eine CHAR(5)-Spalte mit fester Länge in eine Zieltabelle schreibt, können Sie statt der Leerzeichen „00000“ eintragen. Sie würden in diesem Fall einen Ausdruck wie den folgenden erstellen:

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

LAST

Gibt die letzte Zeile im ausgewählten Port zurück. Optional können Sie einen Filter anwenden, um die Anzahl der Zeilen zu beschränken, die Data Integration Service ausliest. Es kann nur eine weitere Aggregatfunktion in LAST verschachtelt sein.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
value	Erforderlich	Alle Datentypen außer binär. Übergibt die Werte, für die Sie die letzte Zeile zurückgeben möchten. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.
filter_condition	Optional	Begrenzt die Zeilen in der Suche. Die Filterbedingung muss ein numerischer Wert sein oder mit TRUE, FALSE oder NULL ausgewertet werden. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

Letzte Zeile in einem Port.

NULL, wenn alle übergebenen Werte NULL sind oder keine Zeilen ausgewählt wurden (z. B. wenn die Filterbedingung in allen Zeilen FALSE oder NULL ergibt).

Hinweis: Standardmäßig behandelt Data Integration Service Nullwerte in Aggregatfunktionen als NULL. Wenn Sie einen Port oder eine Gruppe mit ausschließlich Nullwerten übergeben, gibt die Funktion NULL zurück. Beim Konfigurieren von Data Integration Service können Sie jedoch entscheiden, wie mit Nullwerten in Aggregatfunktionen umgegangen werden soll. Sie können Nullwerte in Aggregatfunktionen als 0 oder als NULL verarbeiten.

Beispiel

Der folgende Ausdruck gibt die letzte Zeile im Port ITEMS_NAME mit einem höheren Preis als 10.00 USD zurück:

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

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

RETURN VALUE: Vest

LAST_DAY

Gibt für jedes Datum in einem Port das Datum des letzten Tages im betreffenden Monat zurück.

Syntax

```
LAST_DAY( date )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
Datum	Erforderlich	Datum/Zeit-Datentyp. Übergibt die Datumsangaben, für die Sie den letzten Tag des Monats zurückgeben möchten. Sie können jeden beliebigen Umwandlungsausdruck eingeben, dessen Auswertung ein Datum ergibt.

Rückgabewert

Datum. Der letzte Tag des Monats für den Datumswert, den Sie der Funktion übergeben.

NULL, falls ein Wert im ausgewählten Port NULL ist.

Null

Wenn nur ein Wert NULL ist, wird die Zeile ignoriert. Wenn jedoch alle vom Port übergebenen Werte NULL ergeben, gibt LAST_DAY NULL zurück.

Gruppieren nach

LAST_DAY gruppiert die Werte nach der Einstellung „Gruppieren nach Ports“, die Sie in der Umwandlung festlegen, und gibt pro Gruppe ein Ergebnis zurück. Wenn „Gruppieren nach Ports“ nicht festgelegt wurde, behandelt LAST_DAY alle Zeilen als eine einzige Gruppe und gibt nur einen Wert zurück.

Beispiele

Der folgende Ausdruck gibt für jedes Datum im Port ORDER_DATE das Datum des letzten Tages im betreffenden Monat zurück:

```
LAST_DAY( ORDER_DATE )
```

ORDER_DATE	RETURN VALUE	
Apr 1 1998 12:00:00AM	Apr 30 1998 12:00:00AM	
Jan 6 1998 12:00:00AM	Jan 31 1998 12:00:00AM	
Feb 2 1996 12:00:00AM	Feb 29 1996 12:00:00AM	*(Leap year)*
NULL	NULL	
Jul 31 1998 12:00:00AM	Jul 31 1998 12:00:00AM	

TO_DATE kann verschachtelt werden, um Stringwerte in Datumsangaben zu konvertieren. TO_DATE enthält immer einen Zeitwert. Wenn Sie einen String ohne Zeitwert übergeben, gibt das Rückgabedatum die Uhrzeit immer mit 00:00:00 an.

Der folgende Ausdruck gibt für jedes Bestelldatum den letzten Tag des betreffenden Monats im selben Format wie der String zurück:

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

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

LEAST

Gibt den kleinsten Wert aus einer Liste von Eingabewerten zurück. Standardgemäß werden Groß-/Kleinschreibung unterschieden.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
value	Erforderlich	Alle Datentypen außer binär. Der Datentyp muss mit anderen Werten kompatibel sein. Wert, den Sie mit anderen Werten vergleichen möchten. Sie müssen mindestens ein Wertargument eingeben. Wenn der Wert numerisch ist und andere Eingabewerte andere numerische Datentypen aufweisen, verwenden alle Werte die höchstmögliche Genauigkeit. Wenn beispielsweise bestimmte Werte den Datentyp „Ganzzahl“ und andere Werte den Datentyp „Doppelt“ aufweisen, wandelt der Data Integration Service die Werte in den Datentyp „Doppelt“ um.
CaseFlag	Optional	Muss eine Ganzzahl sein. Geben Sie einen Wert an, wenn das Eingabewertargument ein Zeichenfolgenwert ist. Legt fest, ob für die Argumente in dieser Funktion zwischen Groß- und Kleinschreibung unterschieden wird. Sie können einen beliebigen gültigen Umwandlungsausdruck eingeben. Wenn CaseFlag eine Zahl ungleich 0 ist, wird bei der Funktion zwischen Groß- und Kleinschreibung unterschieden. Wenn CaseFlag gleich 0 ist, wird bei der Funktion nicht zwischen Groß- und Kleinschreibung unterschieden. Standardwert ist „Groß-/Kleinschreibung beachten“.

Rückgabewert

value1, wenn dieser der kleinste der Eingabewerte ist, *value2*, wenn dieser der kleinste der Eingabewerte ist usw.

NULL, wenn eines der Argumente Null ist.

Beispiel

Der folgende Ausdruck gibt die kleinste Artikelbestellmenge zurück:

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

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

LENGTH

Gibt die Anzahl der Zeichen in einem String zurück, nachgestellte Leerzeichen eingeschlossen.

Syntax

```
LENGTH( string )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
string	Erforderlich	String-Datentyp. Die Strings, die ausgewertet werden sollen. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

Ganzzahl zur Angabe der Stringlänge.

NULL, falls ein an die Funktion übergebener Wert NULL ist.

Beispiel

Der folgende Ausdruck gibt die Länge der einzelnen Kundennamen zurück:

```
LENGTH( CUSTOMER_NAME )
```

CUSTOMER_NAME	RETURN VALUE
Bernice Davis	13
NULL	NULL

CUSTOMER_NAME	RETURN VALUE
John Baer	9
Greg Brown	10

Tipps für die Arbeit mit LENGTH

Mit LENGTH können Sie leere Strings ermitteln. Um Felder zu finden, in denen der Kundenname leer ist, verwenden Sie einen Ausdruck wie den folgenden:

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

Zum Auffinden von Null-Feldern dient ISNULL. Um Ausdrücke auf Leerzeichen zu überprüfen, verwenden Sie IS_SPACES.

LN

Gibt den natürlichen Logarithmus eines numerischen Werts zurück. Beispiel: `LN(3)` gibt `1.098612` zurück. Diese Funktion dient eher der Analyse von wissenschaftlichen Daten als jener von Geschäftsdaten.

Diese Funktion ist der Kehrwert der Funktion EXP.

Syntax

```
LN( numeric_value )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
numeric_value	Erforderlich	Numerischer Datentyp. Muss eine positive Zahl größer als 0 sein. Übergibt die Werte, für die Sie den natürlichen Logarithmus berechnen möchten. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

Double-Wert

NULL, falls ein an die Funktion übergebener Wert NULL ist.

Beispiel

Der folgende Ausdruck gibt den natürlichen Logarithmus für alle Werte im Port NUMBERS zurück:

```
LN( NUMBERS )
```

NUMBERS	RETURN VALUE
10	2.302585092994
125	4.828313737302

NUMBERS	RETURN VALUE
0.96	-0.04082199452026
NULL	NULL
-90	*Error. (The Integration Service does not write row.)*
0	*Error. (The Integration Service does not write row.)*

Hinweis: Wenn Sie eine negative Zahl oder 0 übergeben, zeigt Data Integration Service eine Fehlermeldung an schreibt die Zeile nicht. Das Argument *numeric_value* muss eine positive Zahl größer als 0 sein.

LOG

Gibt den Logarithmus eines numerischen Werts zurück. Diese Funktion dient eher der Analyse von wissenschaftlichen Daten als jener von Geschäftsdaten.

Syntax

```
LOG( base, exponent )
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
base	Erforderlich	Die Basis des Logarithmus. Muss ein positiver numerischer Wert, aber nicht 0 oder 1, sein. Jeder gültige Umwandlungsausdruck, dessen Auswertung eine andere positive Zahl als 0 oder 1 ergibt.
exponent	Erforderlich	Der Exponent des Logarithmus. Muss ein positiver numerischer Wert größer als 0 sein. Jeder gültige Umwandlungsausdruck, dessen Auswertung eine positive Zahl größer als 0 ergibt.

Rückgabewert

Double-Wert

NULL, falls ein an die Funktion übergebener Wert NULL ist.

Beispiel

Der folgende Ausdruck gibt den Logarithmus für alle Werte im Port NUMBERS zurück:

```
LOG( BASE, EXPONENT )
```

BASE	EXPONENT	RETURN VALUE
15	1	0
.09	10	-0.956244644696599

BASE	EXPONENT	RETURN VALUE
NULL	18	NULL
35.78	NULL	NULL
-9	18	*Error. (Data Integration Service does not write the row.)*
0	5	*Error. (Data Integration Service does not write the row.)*
10	-2	*Error. (Data Integration Service does not write the row.)*

Wenn Sie als Basiswert eine negative Zahl, 0 oder 1 oder als Exponent einen negativen Wert übergeben, zeigt Data Integration Service einen Fehler an und schreibt die Zeile nicht.

LOOKUP

Sucht einen Wert in einer Lookup-Quellspalte.

Das Funktion LOOKUP vergleicht Daten in einer Lookup-Quelle mit einem angegebenen Wert. Wenn der Suchwert in der Lookup-Tabelle gefunden wird, gibt Data Integration Service einen Wert aus einer angegebenen Spalte in der gleichen Zeile der Lookup-Tabelle zurück.

Beim Erstellen einer Sitzung auf Grundlage eines Mappings, das mit der Funktion LOOKUP arbeitet, müssen Sie die Datenbankverbindungen für den $Source- und den $Target-Verbindungswert in den Sitzungseigenschaften angeben. Zum Validieren einer LOOKUP-Funktion in einer Ausdrucksumwandlung stellen Sie sicher, dass die Lookup-Definition im Mapping vorhanden ist.

Hinweis: Diese Funktion wird in Mapplets nicht unterstützt.

Verwendung der Lookup-Umwandlung oder der Funktion LOOKUP

Zum Nachschlagen von Werten in PowerCenter-Mappings sollten Sie die *Lookup-Umwandlung* anstelle der *Funktion LOOKUP* verwenden. Für den Einsatz der Funktion LOOKUP in einem Mapping müssen Sie in den Sitzungseigenschaften die Lookup-Zwischenspeicherung für 3.5-Kompatibilität aktivieren. Diese Option gibt es eigens für Benutzer von PowerMart 3.5, die anstatt Lookup-Umwandlungen zu erstellen auch weiterhin die Funktion LOOKUP verwenden möchten. Weitere Informationen finden Sie unter „Lookup-Umwandlung ” im *PowerCenter-Umwandlungshandbuch*.

Sie können mehrere Suchvorgänge für eine einzelne Lookup-Tabelle innerhalb einer LOOKUP-Funktion definieren. Bei jeder Suche muss jedoch ein übereinstimmender Wert gefunden werden, damit der Lookup-Wert zurückgegeben wird.

Syntax

```
LOOKUP( result, search1, value1 [, search2, value2]... )
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
result	Erforderlich	Alle Datentypen außer binär. Muss ein Ausgabeport in derselben Lookup-Tabelle wie „search“ sein. Legt den Ausgabewert zurück, wenn die Suche mit dem Wert übereinstimmt. Stellen Sie diesem Argument stets den Referenzqualifikator „:TD“ voran.
search1	Erforderlich	Übereinstimmender Datentyp mit *value1*. Muss ein Ausgabeport in derselben Lookup-Tabelle wie „result“ sein. Legt die Werte fest, die mit „value“ übereinstimmen sollen. Stellen Sie diesem Argument stets den Referenzqualifikator „:TD“ voran.
value1	Erforderlich	Alle Datentypen außer binär. Muss dem Datentyp von *search1* entsprechen. Die Werte, die in der unter *search1* angegebenen Lookup-Quellspalte gesucht werden sollen. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

Das *Ergebnis* aus „result“, wenn alle Suchanfragen übereinstimmende Werte finden. Wenn Data Integration Service übereinstimmende Werte findet, wird das Ergebnis aus derselben Zeile wie das Argument *search1* zurückgegeben

NULL, wenn die Suche keine übereinstimmenden Werte ermittelt.

Fehler, wenn die Suche mehr als einen übereinstimmenden Wert ermittelt.

Beispiel

Der folgende Ausdruck durchsucht die Lookup-Quelle :TD.SALES nach einer bestimmten Artikel-ID und einem bestimmten Preis und gibt den Namen des Artikels zurück, wenn beide Suchwerte eine Übereinstimmung ergeben:

```
LOOKUP( :TD.SALES.ITEM_NAME, :TD.SALES.ITEM_ID, 10, :TD.SALES.PRICE, 15.99 )
```

ITEM_NAME	ITEM_ID	PRICE
Regulator	5	100.00
Flashlight	10	15.99
Halogen Flashlight	15	15.99
NULL	20	15.99

RETURN VALUE: Flashlight

Tipps für die Arbeit mit LOOKUP

Beim Vergleichen von Char- und Varchar-Werten gibt LOOKUP nur dann ein Ergebnis zurück, wenn beide Zeilen übereinstimmen. Das bedeutet, dass sowohl der Wert als auch die Länge jeder Zeile übereinstimmen müssen. Wenn die Lookup-Quelle ein mit Leerzeichen aufgefüllter Char-Wert mit angegebener Länge und die Lookup-Suche ein Varchar-Wert ist, müssen Sie mithilfe der Funktion RTRIM die nachgestellten Leerzeichen aus der Lookup-Quelle entfernen, damit die Werte der Lookup-Suche entsprechen:

```
LOOKUP(:TD.ORDERS.PRICE, :TD.ORDERS.ITEM, RTRIM( ORDERS.ITEM, '   '))
```

Verwenden Sie den Referenzqualifikator „:TD“ in den Argumenten *result* und *search* der Funktion LOOKUP:

```
LOOKUP(:TD.ORDERS.ITEM, :TD.ORDERS.PRICE, ORDERS.PRICE, :TD.ORDERS.QTY, ORDERS.QTY)
```

LOWER

Konvertiert Großbuchstaben in einem String in Kleinbuchstaben.

Syntax

```
LOWER( string )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
string	Erforderlich	Beliebiger Stringwert. Das Argument übergibt die gewünschten Stringwerte in Kleinbuchstaben. Sie können jeden beliebigen Umwandlungsausdruck eingeben, dessen Auswertung einen String ergibt.

Rückgabewert

Zeichenstring in Kleinbuchstaben. Wenn die Daten Multibyte-Zeichen enthalten, hängt der Rückgabewert von der Codepage und dem Datenverschiebungsmodus von Integration Service ab.

NULL, falls ein Wert im ausgewählten Port NULL ist.

Beispiel

Der folgende Ausdruck gibt alle Vornamen in Kleinbuchstaben zurück:

```
LOWER( FIRST_NAME )
```

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

LPAD

Fügt am Anfang eines Strings eine Reihe von Zeichen oder Leerzeichen hinzu, damit der String eine bestimmte Länge erreicht.

Syntax

```
LPAD( first_string, length [,second_string] )
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
first_string	Erforderlich	Kann eine Zeichenfolge sein. Die Zeichenfolgen, die Sie ändern möchten. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.
length	Erforderlich	Muss ein positives Ganzzahl-Literal sein. Dieses Argument gibt die Länge an, die alle Zeichenfolgen erreichen sollen.
second_string	Optional	Kann ein beliebiger Zeichenfolgenwert sein. Die Zeichen, die links an die Werte der ersten Zeichenfolge *first_string* angehängt werden sollen. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben. Sie können ein bestimmtes Zeichenfolgen-Literal eingeben. Setzen Sie die Zeichen, die Sie am Anfang der Zeichenfolge hinzufügen möchten, jedoch zwischen einfache Anführungszeichen: 'abc'. Bei diesem Argument wird zwischen Groß- und Kleinschreibung unterschieden. Wenn Sie das Argument *second_string* nicht angeben, füllt die Funktion den Anfang der ersten Zeichenfolge mit Leerzeichen auf.

Rückgabewert

String der angegebenen Länge.

NULL, wenn ein der Funktion übergebener Wert NULL oder *length* eine negative Zahl ist.

Beispiele

Der folgende Ausdruck standardisiert die Länge von Zahlen auf sechs Ziffern, indem sie Nullen voranstellt:

```
LPAD( PART_NUM, 6, '0')
```

PART_NUM	RETURN VALUE
702	000702
1	000001
0553	000553
484834	484834

LPAD zählt die Länge von links nach rechts. Wenn der angegebene erste String länger ist als die Länge in „length“, schneidet LPAD den String von rechts nach links ab. Beispiel: LPAD(‘alphabetisch’, 5, ‘x’) gibt den String „alpha“ zurück.

Wenn der zweite String länger ist als die gesamte Zeichenanzahl, die zur Rückgabe der angegebenen Länge erforderlich ist, verwendet LPAD einen Teil des zweiten Strings:

```
LPAD( ITEM_NAME, 16, '*..*' )
```

ITEM_NAME	RETURN VALUE
Flashlight	*..**.Flashlight

ITEM_NAME	RETURN VALUE
Compass	*..**..**Compass
Regulator System	Regulator System
Safety Knife	*..*Safety Knife

LTRIM

Entfernt Zeichen oder Leerzeichen am Anfang eines Strings. Mit LTRIM und IIF oder DECODE in einem Ausdruck oder einer Update-Strategie-Umwandlung können Sie vermeiden, dass Leerzeichen in eine Zieltabelle geschrieben werden.

Wenn Sie den Parameter *trim_set* im Ausdruck nicht angeben:

- Unicode-Modus: LTRIM entfernt Single- und Double-Byte-Leerzeichen am Stringanfang.
- ASCII-Modus: LTRIM entfernt nur Single-Byte-Leerzeichen.

Beim Entfernen von Zeichen aus einem String mit LTRIM vergleicht die Funktion den Wert *trim_set* einzeln von links nach rechts mit allen Zeichen im Argument *string*. Wenn eine Übereinstimmung zwischen einem Zeichen im String und einem Zeichen in *trim_set* gefunden wird, wird das betreffende Zeichen entfernt. LTRIM vergleicht und entfernt so lange Zeichen, bis keine übereinstimmenden Zeichen in *trim_set* mehr gefunden werden. Dann wird der String zurückgegeben, für den keine übereinstimmenden Zeichen ermittelt wurden.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Arguments	Erforderlich/ Optional	Beschreibung
string	Erforderlich	Beliebiger Stringwert. Übergibt die Strings, die Sie ändern möchten. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben. Verwenden Sie Operatoren, um Strings vor dem Entfernen von Zeichen am Stringanfang zu vergleichen oder zu verketten.
trim_set	Optional	Beliebiger Stringwert. Übergibt die Zeichen, die Sie am Anfang des ersten Strings entfernen möchten. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben. Sie können auch einen Zeichenstring eingeben. Sie müssen die Zeichen, die Sie am Stringanfang entfernen möchten, jedoch zwischen einfache Anführungszeichen setzen: 'abc'. Wenn Sie den zweiten String nicht angeben, entfernt die Funktion keine Leerzeichen am Stringanfang. LTRIM unterscheidet zwischen Groß- und Kleinschreibung. Beispiel: Wenn Sie den Buchstaben „A“ aus dem String „Alfredo“ entfernen möchten, müssen Sie „A“ angeben und nicht „a“.

Rückgabewert

String. Die Stringwerte ohne die im Argument *trim_set* angegebenen Zeichen.

NULL, falls ein an die Funktion übergebener Wert NULL ist. Wenn *trim_set* NULL ist, gibt die Funktion NULL zurück.

Beispiel

Der folgende Ausdruck entfernt die Zeichen „S“ und „.“ aus den Strings im Port LAST_NAME:

```
LTRIM( LAST_NAME, 'S.')
```

LAST_NAME	RETURN VALUE
Nelson	Nelson
Osborne	Osborne
NULL	NULL
S. MacDonald	MacDonald
Sawyer	awyer
H. Bender	H. Bender
Steadman	teadman

LTRIM entfernt „S.“ aus „S. MacDonald“ und „S“ aus „Sawyer“ und „Steadman“, aber nicht den Punkt aus „H. Bender“. Das liegt daran, dass LTRIM Zeichen um Zeichen nach dem Zeichensatz absucht, den Sie im Argument *trim_set* angegeben haben. Falls das erste Zeichen im String dem ersten Zeichen in *trim_set* entspricht, wird es von LTRIM entfernt. Anschließend analysiert LTRIM das zweite Zeichen im String. Wenn es mit dem zweiten Zeichen in *trim_set* übereinstimmt, wird es entfernt usw. Wenn aber das erste Zeichen im String nicht mit dem entsprechenden Zeichen in *trim_set* übereinstimmt, gibt LTRIM den String zurück und fährt mit der Auswertung der nächsten Zeile fort.

Im vorherigen Beispiel entspricht „H“ keinem der Zeichen in *trim_set*, sodass LTRIM den String im Port LAST_NAME zurückgibt und zur nächsten Zeile übergeht.

Tipps für die Arbeit mit LTRIM

Verwenden Sie RTRIM und LTRIM mit || oder CONCAT zum Entfernen von vor- und nachgestellten Leerzeichen nach dem Verketten von zwei Strings.

Sie können auch mehrere Zeichensätze entfernen, indem Sie LTRIM verschachteln. Beispiel: Zum Entfernen von Leerzeichen am Anfang des Strings und des Buchstabens „T“ aus einer Spalte von Namen können Sie folgende Funktion formulieren:

```
LTRIM( LTRIM( NAMES ), 'T' )
```

MAKE_DATE_TIME

Gibt Datum und Uhrzeit auf Basis der Eingabewerte zurück.

Syntax

```
MAKE_DATE_TIME( year, month, day, hour, minute, second, nanosecond )
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
Jahr	Erforderlich	Numerischer Datentyp. Positive vierstellige Ganzzahl. Wenn Sie dieser Funktion eine zweistellige Jahreszahl übergeben, gibt Data Integration Service für die ersten beiden Stellen der Jahresangabe „00" zurück.
Monat	Erforderlich	Numerischer Datentyp. Positive Ganzzahl zwischen 1 und 12 (Januar = 1, Dezember = 12).
Tag	Erforderlich	Numerischer Datentyp. Positive Ganzzahl zwischen 1 und 31 (außer für die Monate, die weniger als 31 Tage haben: Februar, April, Juni, September und November).
Stunde	Optional	Numerischer Datentyp. Positive Ganzzahl zwischen 0 und 24 (wobei 0 = 12AM, 12 = 12PM und 24 = 12AM).
Minute	Optional	Numerischer Datentyp. Positive Ganzzahl zwischen 0 und 59.
Sekunde	Optional	Numerischer Datentyp. Positive Ganzzahl zwischen 0 und 59.
Nanosekunde	Optional	Numerischer Datentyp. Positive Ganzzahl zwischen 0 und 999,999,999.

Rückgabewert

Datum als MM/DD/YYYY HH24:MI:SS. Gibt einen Nullwert zurück, wenn Sie der Funktion keine Angabe eines Jahres, Monats oder Tags übergeben.

Beispiel

Der folgende Ausdruck erstellt ein Datum und eine Uhrzeit aus den Eingabeports:

```
MAKE_DATE_TIME( SALE_YEAR, SALE_MONTH, SALE_DAY, SALE_HOUR, SALE_MIN, SALE_SEC )
```

SALE_YR	SALE_MTH	SALE_DAY	SALE_HR	SALE_MIN	SALE_SEC	RETURN VALUE
2002	10	27	8	36	22	10/27/2002 08:36:22
2000	6	15	15	17		06/15/2000 15:17:00
2003	1	3		22	45	01/03/2003 00:22:45
04	3	30	12	5	10	03/30/0004 12:05:10
99	12	12	5		16	12/12/0099 05:00:16

MAX (Datum)

Gibt das späteste Datum zurück, das in einem Port oder einer Gruppe gefunden wurde. Optional können Sie einen Filter anwenden, um die Anzahl der Zeilen in der Suche zu beschränken. Es kann nur eine weitere Aggregatfunktion in MAX verschachtelt sein.

Sie können MAX auch dazu verwenden, den größten numerischen Wert oder den höchsten Stringwert in einem Port oder einer Gruppe zurückzugeben.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
Datum	Erforderlich	Datum/Zeit-Datentyp. Übergibt das Datum, für das Sie das maximale Datum zurückgeben möchten. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.
filter_condition	Optional	Begrenzt die Zeilen in der Suche. Die Filterbedingung muss ein numerischer Wert sein oder mit TRUE, FALSE oder NULL ausgewertet werden. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

Datum.

NULL, wenn alle übergebenen Werte NULL sind oder keine Zeilen ausgewählt wurden (z. B. wenn die Filterbedingung in allen Zeilen FALSE oder NULL ergibt).

Beispiel

Sie können das maximale Datum für einen Port oder eine Gruppe zurückgeben. Der folgende Ausdruck gibt das maximale Bestelldatum für den Artikel „Flashlight“ zurück:

```
MAX( ORDERDATE, ITEM_NAME='Flashlight' )
```

ITEM_NAME	ORDER_DATE
Flashlight	Apr 20 1998
Regulator System	May 15 1998
Flashlight	Sep 21 1998
Diving Hood	Aug 18 1998
Flashlight	NULL

MAX (Zahlen)

Gibt den maximalen numerischen Wert zurück, der in einem Port oder einer Gruppe gefunden wurde. Optional können Sie einen Filter anwenden, um die Anzahl der Zeilen in der Suche zu beschränken. Es kann nur eine weitere Aggregatfunktion in MAX verschachtelt sein. Sie können MAX auch dazu verwenden, das jüngste Datum oder den höchsten Stringwert in einem Port oder einer Gruppe zurückzugeben.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
numeric_value	Erforderlich	Numerischer Datentyp. Übergibt die numerischen Werte, für die Sie einen maximalen numerischen Wert zurückgeben möchten. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.
filter_condition	Optional	Begrenzt die Zeilen in der Suche. Die Filterbedingung muss ein numerischer Wert sein oder mit TRUE, FALSE oder NULL ausgewertet werden. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

Numerischer Wert.

NULL, wenn alle übergebenen Werte NULL sind oder keine Zeilen ausgewählt wurden (z. B. wenn die Filterbedingung in allen Zeilen FALSE oder NULL ergibt).

Hinweis: Wenn der Rückgabewert eine Dezimalmal mit Präzision höher als 15 ist, können Sie „Hohe Präzision“ aktivieren, um Dezimalgenauigkeit bis zu 28 Stellen zu gewährleisten.

Nullen

Wenn nur ein Wert NULL ist, wird er ignoriert. Wenn jedoch alle vom Port übergebenen Werte NULL ergeben, gibt MAX NULL zurück.

Hinweis: Standardmäßig behandelt Data Integration Service Nullwerte in Aggregatfunktionen als NULL. Wenn Sie einen Port oder eine Gruppe mit ausschließlich Nullwerten übergeben, gibt die Funktion NULL zurück. Beim Konfigurieren von Data Integration Service können Sie jedoch entscheiden, wie mit Nullwerten in Aggregatfunktionen umgegangen werden soll. Sie können Nullwerte in Aggregatfunktionen als 0 oder als NULL verarbeiten.

Gruppieren nach

MAX gruppiert die Werte nach der Einstellung „Gruppieren nach Ports“, die Sie in der Umwandlung festlegen, und gibt pro Gruppe ein Ergebnis zurück.

Wenn „Gruppieren nach Ports“ nicht festgelegt wurde, behandelt MAX alle Zeilen als eine einzige Gruppe und gibt nur einen Wert zurück.

Beispiel

Der folgende Ausdruck gibt den maximalen Preis für den Artikel „Flashlight“ zurück:

```
MAX( PRICE, ITEM_NAME='Flashlight' )
```

ITEM_NAME	PRICE
Flashlight	10.00
Regulator System	360.00
Flashlight	55.00
Diving Hood	79.00
Halogen Flashlight	162.00

ITEM_NAME	PRICE
Flashlight	85.00
Flashlight	NULL

RETURN VALUE: 85.00

MAX (String)

Gibt den höchsten Stringwert in einem Port oder einer Gruppe zurück. Optional können Sie einen Filter anwenden, um die Anzahl der Zeilen in der Suche zu beschränken. Es kann nur eine weitere Aggregatfunktion in MAX verschachtelt sein.

Hinweis: Die Funktion MAX verwendet die gleiche Sortierreihenfolge wie die Sortier-Umwandlung. MAX unterscheidet jedoch zwischen Groß- und Kleinschreibung, die Sortier-Umwandlung nicht unbedingt.

Sie können MAX auch dazu verwenden, das jüngste Datum oder den höchsten numerischen Wert in einem Port oder einer Gruppe zurückzugeben.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
string	Erforderlich	String-Datentyp. Übergibt die Stringwerte, für die Sie einen maximalen Stringwert zurückgeben möchten. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.
filter_condition	Optional	Begrenzt die Zeilen in der Suche. Die Filterbedingung muss ein numerischer Wert sein oder mit TRUE, FALSE oder NULL ausgewertet werden. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

String.

NULL, wenn alle übergebenen Werte NULL sind oder keine Zeilen ausgewählt wurden (z. B. wenn die Filterbedingung in allen Zeilen FALSE oder NULL ergibt).

Nullen

Wenn nur ein Wert NULL ist, wird er ignoriert. Wenn jedoch alle vom Port übergebenen Werte NULL ergeben, gibt MAX NULL zurück.

Hinweis: Standardmäßig behandelt Data Integration Service Nullwerte in Aggregatfunktionen als NULL. Wenn Sie einen Port oder eine Gruppe mit ausschließlich Nullwerten übergeben, gibt die Funktion NULL zurück. Beim Konfigurieren von Data Integration Service können Sie jedoch entscheiden, wie mit Nullwerten in

Aggregatfunktionen umgegangen werden soll. Sie können Nullwerte in Aggregatfunktionen als 0 oder als NULL verarbeiten.

Gruppieren nach

MAX gruppiert die Werte nach der Einstellung „Gruppieren nach Ports“, die Sie in der Umwandlung festlegen, und gibt pro Gruppe ein Ergebnis zurück.

Wenn „Gruppieren nach Ports“ nicht festgelegt wurde, behandelt MAX alle Zeilen als eine einzige Gruppe und gibt nur einen Wert zurück.

Beispiel

Der folgende Ausdruck gibt den maximalen Artikelnamen für die Hersteller-ID 104 zurück:

```
MAX( ITEM_NAME, MANUFACTURER_ID='104' )
```

MANUFACTURER_ID	ITEM_NAME
101	First Stage Regulator
102	Electronic Console
104	Flashlight
104	Battery (9 volt)
104	Rope (20 ft)
104	60.6 cu ft Tank
107	75.4 cu ft Tank
108	Wristband Thermometer

RETURN VALUE: Rope (20 ft)

MD5

Berechnet die Prüfsumme des Eingabewerts. Die Funktion nutzt den Message-Digest-Algorithmus 5 (MD5). MD5 ist eine unidirektionale kryptographische Hashfunktion mit einem 128-Bit-Hashwert. Wenn die Prüfsummen der Eingabewerte unterschiedlich ausfallen, können Sie davon ausgehen, dass auch die Eingabewerte unterschiedlich sind. Verwenden Sie MD5 zur Prüfung der Datenintegrität.

Syntax

```
MD5( value )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
value	Erforderlich	String oder binärer Datentyp. Wert, für den die Prüfsumme berechnet werden soll. Die Groß- bzw. Kleinschreibung des Eingabewerts wirkt sich auf den Rückgabewert aus. Beispiel: MD5(informatica) und MD5(Informatica) ergeben unterschiedliche Werte.

Rückgabewert

Eindeutiger 32-Zeichen-String aus hexadezimalen Ziffern 0-9 und Buchstaben a-f.

NULL, wenn der Eingabewert ein Nullwert ist.

Beispiel

Sie möchten geänderte Daten in einen Datenbank aufnehmen. Sie generieren mit MD5 Prüfsummenwerte für die Zeilen der Daten, die aus einer Quelle ausgelesen werden. Beim Ausführen einer Sitzung vergleichen Sie die zuvor erstellte Prüfsumme mit den neuen Prüfsummenwerten. Anschließend schreiben Sie die Zeilen mit den aktualisierten Prüfsummenwerten in das Ziel. Eine geänderte aktuelle Prüfsumme bedeutet, dass sich auch die Daten geändert haben.

Sie möchten geänderte Daten in einen Datenbank aufnehmen. Sie generieren mit MD5 Prüfsummenwerte für die Zeilen der Daten, die aus einer Quelle ausgelesen werden. Beim Ausführen eines Mapping vergleichen Sie die zuvor erstellte Prüfsumme mit den neuen Prüfsummenwerten. Anschließend schreiben Sie die Zeilen mit den aktualisierten Prüfsummenwerten in das Ziel. Eine geänderte aktuelle Prüfsumme bedeutet, dass sich auch die Daten geändert haben.

Tipp

Sie können den Rückgabewert als Hashschlüssel nutzen.

MEDIAN

Gibt den Median aller Werte in einem ausgewählten Port zurück.

In Ports mit einer geraden Anzahl von Werten ist der Median der Durchschnitt der beiden mittleren Werte, wenn sie der Größe nach geordnet sind. Bei einer ungeraden Anzahl ist der Median die Zahl in der Mitte.

Es kann nur eine weitere Aggregatfunktion in MEDIAN verschachtelt werden, und diese muss einen numerischen Datentyp zurückgeben.

Data Integration Service liest alle Datenzeilen aus, um den Median zu berechnen. Dieser Vorgang kann sich auf die Leistung auswirken. Optional können Sie einen Filter anwenden, um die Anzahl der Zeilen zu beschränken, aus denen der Median berechnet wird.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
numeric_value	Erforderlich	Numerischer Datentyp. Übergibt die Werte, für die ein Median berechnet werden soll. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.
filter_condition	Optional	Begrenzt die Zeilen in der Suche. Die Filterbedingung muss ein numerischer Wert sein oder mit TRUE, FALSE oder NULL ausgewertet werden. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

Numerischer Wert.

NULL, wenn alle der Funktion übergebenen Werte NULL sind oder keine Zeilen ausgewählt werden. Beispiel: Die Filterbedingung ergibt für alle Zeilen FALSE oder NULL.

Hinweis: Wenn der Rückgabewert eine Dezimalmal mit Präzision höher als 15 ist, können Sie „Hohe Präzision“ aktivieren, um Dezimalgenauigkeit bis zu 28 Stellen zu gewährleisten.

Nullen

Wenn nur ein Wert NULL ist, wird die Zeile ignoriert. Wenn jedoch alle vom Port übergebenen Werte NULL ergeben, gibt MEDIAN NULL zurück.

Hinweis: Standardmäßig behandelt Data Integration Service Nullwerte in Aggregatfunktionen als NULL. Wenn Sie einen Port oder eine Gruppe mit ausschließlich Nullwerten übergeben, gibt die Funktion NULL zurück. Beim Konfigurieren von Data Integration Service können Sie jedoch entscheiden, wie mit Nullwerten in Aggregatfunktionen umgegangen werden soll. Sie können Nullwerte in Aggregatfunktionen als 0 oder als NULL verarbeiten.

Gruppieren nach

MEDIAN gruppiert die Werte nach der Einstellung „Gruppieren nach Ports“, die Sie in der Umwandlung festlegen, und gibt pro Gruppe ein Ergebnis zurück.

Wenn „Gruppieren nach Ports“ nicht festgelegt wurde, behandelt MEDIAN alle Zeilen als eine einzige Gruppe und gibt nur einen Wert zurück.

Beispiel

Um den Median der Gehälter in allen Unternehmensabteilungen zu ermitteln, erstellen Sie eine nach Abteilungen gruppierte Aggregator-Umwandlung und geben den folgenden Ausdruck an:

```
MEDIAN( SALARY )
```

Der folgende Ausdruck gibt den Medianwert von Bestellungen des Artikels „Stabilizing Vest“ zurück:

```
MEDIAN( SALES, ITEM = 'Stabilizing Vest' )
```

ITEM	SALES
Flashlight	85
Stabilizing Vest	504
Stabilizing Vest	36

ITEM	SALES
Safety Knife	5
Medium Titanium Knife	150
Tank	NULL
Stabilizing Vest	441
Chisel Point Knife	60
Stabilizing Vest	NULL
Stabilizing Vest	1044
Wrist Band Thermometer	110

RETURN VALUE: 472.5

METAPHONE

Kodiert Stringwerte. Sie können die Länge des Strings angeben, der kodiert werden soll.

METAPHONE kodiert Zeichen des englischen Alphabets (A-Z). Dabei werden Groß- und Kleinbuchstaben als Großbuchstaben kodiert.

Die Kodierung durch METAPHONE erfolgt nach den folgenden Regeln:

- Vokale (A, E, I, O und U) werden übersprungen, es sei denn, es handelt sich dabei um das erste Zeichen eines Eingabestrings. METAPHONE('CAR') gibt 'KR' zurück, METAPHONE('AAR') gibt 'AR' zurück.
- Verwendet spezielle Kodierungsrichtlinien.

Die folgende Tabelle beschreibt die Kodierungsrichtlinien für METAPHONE:

Eingabe	Rückgabe	Bedingung	Beispiel
B	- n/a	- Wenn folgt auf M	- METAPHONE ('Lamb') gibt LM zurück.
B	- B	- In allen anderen Fällen	- METAPHONE ('Box') gibt BKS zurück.
C	- X	- Wenn gefolgt von IA oder H	- METAPHONE ('Facial') gibt FXL zurück.
C	- S	- Wenn gefolgt von I, E oder Y	- METAPHONE ('Fence') gibt FNS zurück.
C	- n/a	- Wenn folgt auf S und gefolgt von I, E oder Y	- METAPHONE ('Scene') gibt SN zurück.
C	- K	- In allen anderen Fällen	- METAPHONE ('Cool') gibt KL zurück.

Eingabe	Rückgabe	Bedingung	Beispiel
D	- J	- Wenn gefolgt von GE, GY oder GI	- METAPHONE (‘Dodge’) gibt TJ zurück.
D	- T	- In allen anderen Fällen	- METAPHONE (‘David’) gibt TFT zurück.
F	- F	- In allen Fällen	- METAPHONE (‘FOX’) gibt FKS zurück.
G	- F	- Wenn gefolgt von H und das erste Zeichen im Eingabestring nicht B D oder H ist	- METAPHONE (‘Tough’) gibt TF zurück.
G	- n/a	- Wenn gefolgt von H und das erste Zeichen im Eingabestring B, D oder H ist	- METAPHONE (‘Hugh’) gibt HF zurück.
G	- J	- Wenn gefolgt von I, E oder Y und nicht wiederholt	- METAPHONE (‘Magic’) gibt MJK zurück.
G	- K	- In allen anderen Fällen	- METAPHONE(‘GUN’) gibt KN zurück.
H	- H	- Wenn nicht folgt auf C, G, P, S oder T und nicht gefolgt von A, E, I oder U	- METAPHONE (‘DHAT’) gibt THT zurück.
H	- n/a	- In allen anderen Fällen	- METAPHONE (‘Chain’) gibt XN zurück.
J	- J	- In allen Fällen	- METAPHONE (‘Jen’) gibt JN zurück.
K	- n/a - K	- Wenn folgt auf C - In allen anderen Fällen	- METAPHONE (‘Ckim’) gibt KM zurück. - METAPHONE (‘Kim’) gibt KM zurück.
L	- L	- In allen Fällen	- METAPHONE (‘Laura’) gibt LR zurück.
M	- M	- In allen Fällen	- METAPHONE (‘Maggi’) gibt MK zurück.
N	- N	- In allen Fällen	- METAPHONE (‘Nancy’) gibt NNS zurück.
P	- F	- Wenn gefolgt von H	- METAPHONE (‘Phone’) gibt FN zurück.
P	- P	- In allen anderen Fällen	- METAPHONE (‘Pip’) gibt PP zurück.
Q	- K	- In allen Fällen	- METAPHONE (‘Queen’) gibt KN zurück.
R	- R	- In allen Fällen	- METAPHONE (‘Ray’) gibt R zurück.
S	- X	- Wenn gefolgt von H, IO, IA oder CHW	- METAPHONE (‘Cash’) gibt KX zurück.
S	- S	- In allen anderen Fällen	- METAPHONE (‘Sing’) gibt SNK zurück.
T	- X	- Wenn gefolgt von IA oder IO	- METAPHONE (‘Patio’) gibt PX zurück.
T	- 0[1]	- Wenn gefolgt von H	- METAPHONE (‘Thor’) gibt 0R zurück.

Eingabe	Rückgabe	Bedingung	Beispiel
T	- n/a	- Wenn gefolgt von CH	- METAPHONE ('Glitch') gibt KLTX zurück.
T	- T	- In allen anderen Fällen	- METAPHONE ('Tim') gibt TM zurück.
V	- F	- In allen Fällen	- METAPHONE ('Vin') gibt FN zurück.
W	- W	- Wenn gefolgt von A, E, I, O oder U	- METAPHONE ('Wang') gibt WNK zurück.
W	- n/a	- In allen anderen Fällen	- METAPHONE ('When') gibt HN zurück.
X	- KS	- In allen Fällen	- METAPHONE ('Six') gibt SKS zurück.
Y	- Y	- Wenn gefolgt von A, E, I, O oder U	- METAPHONE ('Yang') gibt YNK zurück.
Y	- n/a	- In allen anderen Fällen	- METAPHONE ('Bobby') gibt BB zurück.
Z	- S	- In allen Fällen	- METAPHONE ('Zack') gibt SK zurück.

1. Ganzzahl 0

- Das erste Zeichen wird übersprungen und der restliche String kodiert, wenn die ersten beiden Zeichen des Eingabestrings aus folgenden Werten bestehen:
  - **KN:** METAPHONE('KNOT') gibt beispielsweise 'NT' zurück.
  - **GN:** METAPHONE('GNOB') gibt beispielsweise 'NB' zurück.
  - **PN:** METAPHONE('PNRX') gibt beispielsweise 'NRKS' zurück.
  - **AE:** METAPHONE('AERL') gibt beispielsweise 'ERL' zurück.
- Wenn ein anderes Zeichen als „C" mehr als einmal im Eingabestring vorkommt, wird nur das erste Vorkommen kodiert. Beispiel: METAPHONE('BBOX') gibt 'BKS' zurück, METAPHONE('CCOX') gibt 'KKKS' zurück.

Syntax

```
METAPHONE( string [,length] )
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
string	Erforderlich	Muss ein Zeichenstring sein. Übergibt den Wert, der kodiert werden soll. Das erste Zeichen muss ein Zeichen aus dem englischen Alphabet (A-Z) sein. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben. Alle nicht-alphabetischen Zeichen in *string* werden übersprungen.
length	Optional	Muss eine Ganzzahl größer als Null sein. Gibt die Anzahl der Zeichen in *string* an, die kodiert werden sollen. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben. Wenn *length* 0 ist oder einen größeren Wert als die Länge von *string* enthält, wird der gesamt Eingabestring kodiert. Standardwert ist 0.

Rückgabewert

String.

NULL, wenn eine der folgenden Bedingungen eintritt:

- Alle der Funktionen übergebenen Werte sind NULL.
- Kein Zeichen in *string* ist ein Buchstabe des englischen Alphabets.
- Das Argument *string* ist leer.

Beispiele

Der folgende Ausdruck kodiert die ersten beiden Zeichen im Port EMPLOYEE_NAME zu einem String:

```
METAPHONE( EMPLOYEE_NAME, 2 )
```

Employee_Name	Return Value
John	JH
*@#$	NULL
P$%%oc&&KMNL	PK

Der folgende Ausdruck kodiert die ersten vier Zeichen im Port EMPLOYEE_NAME zu einem String:

```
METAPHONE( EMPLOYEE_NAME, 4 )
```

Employee_Name	Return Value
John	JHN
1ABC	ABK
*@#$	NULL
P$%%oc&&KMNL	PKKM

MIN (Datum)

Gibt das früheste Datum in einem Port oder einer Gruppe zurück. Optional können Sie einen Filter anwenden, um die Anzahl der Zeilen in der Suche zu beschränken. Es kann nur eine weitere Aggregatfunktion in MIN verschachtelt werden, und diese muss einen Datum-Datentyp zurückgeben.

Sie können MIN auch dazu verwenden, den kleinsten numerischen Wert oder den geringsten Stringwert in einem Port oder einer Gruppe zurückzugeben.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
Datum	Erforderlich	Datum/Zeit-Datentyp. Übergibt die Werte, für die Sie den Mindestwert zurückgeben möchten. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.
filter_condition	Optional	Begrenzt die Zeilen in der Suche. Die Filterbedingung muss ein numerischer Wert sein oder mit TRUE, FALSE oder NULL ausgewertet werden. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

Datum, wenn das Argument *value* ein Datum ist.

NULL, wenn alle übergebenen Werte NULL sind oder keine Zeilen ausgewählt wurden (z. B. wenn die Filterbedingung in allen Zeilen FALSE oder NULL ergibt).

Nullen

Wenn nur ein Wert NULL ist, wird er ignoriert. Wenn jedoch alle vom Port übergebenen Werte NULL ergeben, gibt MIN NULL zurück.

Gruppieren nach

MIN gruppiert die Werte nach der Einstellung „Gruppieren nach Ports“, die Sie in der Umwandlung festlegen, und gibt pro Gruppe ein Ergebnis zurück.

Wenn „Gruppieren nach Ports“ nicht festgelegt wurde, behandelt MIN alle Zeilen als eine einzige Gruppe und gibt nur einen Wert zurück.

Beispiel

Der folgende Ausdruck gibt das älteste Bestelldatum für den Artikel „Flashlight“ zurück:

```
MIN( ORDER_DATE, ITEM_NAME='Flashlight' )
```

ITEM_NAME	ORDER_DATE
Flashlight	Apr 20 1998
Regulator System	May 15 1998
Flashlight	Sep 21 1998

ITEM_NAME	ORDER_DATE
Diving Hood	Aug 18 1998
Halogen Flashlight	Feb 1 1998
Flashlight	Oct 10 1998
Flashlight	NULL

RETURN VALUE: Feb 1 1998

MIN (Zahlen)

Gibt den kleinsten numerischen Wert in einem Port oder einer Gruppe zurück. Optional können Sie einen Filter anwenden, um die Anzahl der Zeilen in der Suche zu beschränken. Es kann nur eine weitere Aggregatfunktion in MIN verschachtelt werden, und diese muss einen numerischen Datentyp zurückgeben.

Sie können MIN auch dazu verwenden, das jüngste Datum oder den kleinsten Stringwert in einem Port oder einer Gruppe zurückzugeben.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
numeric_value	Erforderlich	Numerische Datentypen. Übergibt die Werte, für die Sie den Mindestwert zurückgeben möchten. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.
filter_condition	Optional	Begrenzt die Zeilen in der Suche. Die Filterbedingung muss ein numerischer Wert sein oder mit TRUE, FALSE oder NULL ausgewertet werden. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

Numerischer Wert.

NULL, wenn alle übergebenen Werte NULL sind oder keine Zeilen ausgewählt wurden (z. B. wenn die Filterbedingung in allen Zeilen FALSE oder NULL ergibt).

Hinweis: Wenn der Rückgabewert eine Dezimalmal mit Präzision höher als 15 ist, können Sie „Hohe Präzision“ aktivieren, um Dezimalgenauigkeit bis zu 28 Stellen zu gewährleisten.

Nullen

Wenn nur ein Wert NULL ist, wird er ignoriert. Wenn jedoch alle vom Port übergebenen Werte NULL ergeben, gibt MIN NULL zurück.

Hinweis: Standardmäßig behandelt Data Integration Service Nullwerte in Aggregatfunktionen als NULL. Wenn Sie einen Port oder eine Gruppe mit ausschließlich Nullwerten übergeben, gibt die Funktion NULL zurück. Beim Konfigurieren von Data Integration Service können Sie jedoch entscheiden, wie mit Nullwerten in Aggregatfunktionen umgegangen werden soll. Sie können Nullwerte in Aggregatfunktionen als 0 oder als NULL verarbeiten.

Gruppieren nach

MIN gruppiert die Werte nach der Einstellung „Gruppieren nach Ports“, die Sie in der Umwandlung festlegen, und gibt pro Gruppe ein Ergebnis zurück.

Wenn „Gruppieren nach Ports“ nicht festgelegt wurde, behandelt MIN alle Zeilen als eine einzige Gruppe und gibt nur einen Wert zurück.

Beispiel

Der folgende Ausdruck gibt den Mindestpreis für den Artikel „Flashlight“ zurück:

```
MIN ( PRICE, ITEM_NAME='Flashlight' )
```

ITEM_NAME	PRICE
Flashlight	10.00
Regulator System	360.00
Flashlight	55.00
Diving Hood	79.00
Halogen Flashlight	162.00
Flashlight	85.00
Flashlight	NULL

RETURN VALUE: 10.00

MIN (String)

Gibt den kleinsten Stringwert in einem Port oder einer Gruppe zurück. Optional können Sie einen Filter anwenden, um die Anzahl der Zeilen in der Suche zu beschränken. Es kann nur eine weitere Aggregatfunktion in MIN verschachtelt werden, und diese muss einen String-Datentyp zurückgeben.

Hinweis: Die Funktion MIN verwendet die gleiche Sortierreihenfolge wie die Sortier-Umwandlung. MIN unterscheidet jedoch zwischen Groß- und Kleinschreibung, die Sortier-Umwandlung nicht unbedingt.

Sie können MIN auch dazu verwenden, das jüngste Datum oder den niedrigsten numerischen Wert in einem Port oder einer Gruppe zurückzugeben.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
string	Erforderlich	String-Datentyp. Übergibt die Werte, für die Sie den Mindestwert zurückgeben möchten. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.
filter_condition	Optional	Begrenzt die Zeilen in der Suche. Die Filterbedingung muss ein numerischer Wert sein oder mit TRUE, FALSE oder NULL ausgewertet werden. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

Stringwert.

NULL, wenn alle übergebenen Werte NULL sind oder keine Zeilen ausgewählt wurden (z. B. wenn die Filterbedingung in allen Zeilen FALSE oder NULL ergibt).

Nullen

Wenn nur ein Wert NULL ist, wird er ignoriert. Wenn jedoch alle vom Port übergebenen Werte NULL ergeben, gibt MIN NULL zurück.

Hinweis: Standardmäßig behandelt Data Integration Service Nullwerte in Aggregatfunktionen als NULL. Wenn Sie einen Port oder eine Gruppe mit ausschließlich Nullwerten übergeben, gibt die Funktion NULL zurück. Beim Konfigurieren von Data Integration Service können Sie jedoch entscheiden, wie mit Nullwerten in Aggregatfunktionen umgegangen werden soll. Sie können Nullwerte in Aggregatfunktionen als 0 oder als NULL verarbeiten.

Gruppieren nach

MIN gruppiert die Werte nach der Einstellung „Gruppieren nach Ports“, die Sie in der Umwandlung festlegen, und gibt pro Gruppe ein Ergebnis zurück.

Wenn „Gruppieren nach Ports“ nicht festgelegt wurde, behandelt MIN alle Zeilen als eine einzige Gruppe und gibt nur einen Wert zurück.

Beispiel

Der folgende Ausdruck gibt den minimalen Artikelnamen für die Hersteller-ID 104 zurück:

```
MIN ( ITEM_NAME, MANUFACTURER_ID='104' )
```

MANUFACTURER_ID	ITEM_NAME
101	First Stage Regulator
102	Electronic Console
104	Flashlight
104	Battery (9 volt)
104	Rope (20 ft)
104	60.6 cu ft Tank
107	75.4 cu ft Tank

MANUFACTURER_ID	ITEM_NAME
108	Wristband Thermometer

RETURN VALUE: 60.6 cu ft Tank

MOD

Gibt bei einer Division den berechneten Rest zurück. Beispiel: `MOD(8,5)` gibt 3 zurück.

Syntax

```
MOD( numeric_value, divisor )
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
numeric_value	Erforderlich	Numerischer Datentyp. Die Werte, die dividiert werden sollen. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.
divisor	Erforderlich	Der numerische Wert, durch den dividiert werden soll. Der Divisor kann nicht 0 sein.

Rückgabewert

Numerischer Wert des Datentyps, den Sie an die Funktion übergeben. Der Rest der Division „numeric_value“ durch „divisor“.

NULL, falls ein an die Funktion übergebener Wert NULL ist.

Beispiele

Der folgende Ausdruck gibt den Modulo aus der Division der Werte im Port PRICE durch die Werte im Port QTY zurück:

```
MOD( PRICE, QTY )
```

PRICE	QTY	RETURN VALUE
10.00	2	0
12.00	5	2
9.00	2	1
15.00	3	0

PRICE	QTY	RETURN VALUE
NULL	3	NULL
20.00	NULL	NULL
25.00	0	*Error. Integration Service does not write row.*

Die letzte Zeile (25, 0) ergibt einen Fehler, da nicht durch 0 dividiert werden kann. Um die Division durch 0 zu vermeiden, können Sie einen Ausdruck wie den folgenden erstellen, der nur dann den Modulo von Preis durch Menge zurückgibt, wenn die Menge nicht 0 ist. Wenn die Menge 0 ist, gibt die Funktion NULL zurück:

```
MOD( PRICE, IIF( QTY = 0, NULL, QTY ))
```

PRICE	QTY	RETURN VALUE
10.00	2	0
12.00	5	2
9.00	2	1
15.00	3	0
NULL	3	NULL
20.00	NULL	NULL
25.00	0	NULL

Die letzte Zeile (25, 0) ergibt NULL und keinen Fehler, da die IIF-Funktion die 0 im Port QTY durch NULL ersetzt.

MOVINGAVG

Gibt den Durchschnitt (Zeile für Zeile) eines angegebenen Zeilensatzes zurück. Optional können Sie eine Bedingung zum Filtern der Zeilen anwenden, bevor der gleitende Durchschnitt berechnet wird.

Syntax

```
MOVINGAVG( numeric_value, rowset [, filter_condition] )
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
numeric_value	Erforderlich	Numerischer Datentyp. Die Werte, deren gleitenden Durchschnitt Sie berechnen möchten. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.
rowset	Erforderlich	Muss ein positives Ganzzahl-Literal größer als 0 sein. Definiert den Zeilensatz, für den Sie den gleitenden Durchschnitt berechnen möchten. Beispiel: Wenn Sie den gleitenden Durchschnitt einer Datenspalte berechnen möchten, und zwar von jeweils fünf Zeilen, formulieren Sie einen Ausdruck wie `MOVINGAVG(SALES, 5)`.
filter_condition	Optional	Begrenzt die Zeilen in der Suche. Die Filterbedingung muss ein numerischer Wert sein oder mit TRUE, FALSE oder NULL ausgewertet werden. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

Numerischer Wert.

NULL, wenn alle übergebenen Werte NULL sind oder keine Zeilen ausgewählt wurden (z. B. wenn die Filterbedingung in allen Zeilen FALSE oder NULL ergibt).

Hinweis: Wenn der Rückgabewert eine Dezimalmal mit Präzision höher als 15 ist, können Sie „Hohe Präzision“ aktivieren, um Dezimalgenauigkeit bis zu 28 Stellen zu gewährleisten.

Nullen

MOVINGAVG ignoriert bei der Berechnung des gleitenden Durchschnitts die Nullwerte. Wenn jedoch alle Werte NULL sind, gibt die Funktion NULL zurück.

Beispiel

Der folgende Ausdruck gibt anhand der ersten fünf Zeilen im Port SALES die durchschnittlichen Bestellmengen für den Artikel „Stabilizing Vest“ und anschließend den Durchschnitt der letzten fünf gelesenen Zeilen zurück:

```
MOVINGAVG( SALES, 5 )
```

ROW_NO	SALES	RETURN VALUE
1	600	NULL
2	504	NULL
3	36	NULL
4	100	NULL
5	550	358
6	39	245.8
7	490	243

Die Funktion gibt jeweils den Durchschnitt für einen Satz aus fünf Zeilen zurück: 358 für Zeilen 1 bis 5, 245,8 für Zeilen 2 bis 6 und 243 für Zeilen 3 bis 7.

MOVINGSUM

Gibt die Summe (Zeile für Zeile) eines angegebenen Zeilensatzes zurück.

Optional können Sie eine Bedingung zum Filtern der Zeilen anwenden, bevor die gleitende Summe berechnet wird.

Syntax

```
MOVINGSUM( numeric_value, rowset [, filter_condition] )
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
numeric_value	Erforderlich	Numerischer Datentyp. Die Werte, aus denen Sie die gleitende Summe berechnen möchten. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.
rowset	Erforderlich	Muss ein positives Ganzzahl-Literal größer als 0 sein. Definiert den Zeilensatz, für den Sie die gleitenden Summe berechnen möchten. Beispiel: Wenn Sie die gleitenden Summe einer Datenspalte berechnen möchten, und zwar von jeweils fünf Zeilen, formulieren Sie einen Ausdruck wie `MOVINGSUM( SALES, 5 )`.
filter_condition	Optional	Begrenzt die Zeilen in der Suche. Die Filterbedingung muss ein numerischer Wert sein oder mit TRUE, FALSE oder NULL ausgewertet werden. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

Numerischer Wert.

NULL, wenn alle übergebenen Werte NULL sind oder keine Zeilen ausgewählt wurden (z. B. wenn die Filterbedingung in allen Zeilen FALSE oder NULL ergibt).

Hinweis: Wenn der Rückgabewert eine Dezimalmal mit Präzision höher als 15 ist, können Sie „Hohe Präzision“ aktivieren, um Dezimalgenauigkeit bis zu 28 Stellen zu gewährleisten.

Nullen

MOVINGSUM ignoriert bei der Berechnung der gleitenden Summe die Nullwerte. Wenn jedoch alle Werte NULL sind, gibt die Funktion NULL zurück.

Beispiel

Der folgende Ausdruck gibt anhand der ersten fünf Zeilen im Port SALES die Gesamtbestellmengen für den Artikel „Stabilizing Vest“ und anschließend den Durchschnitt der letzten fünf gelesenen Zeilen zurück:

```
MOVINGSUM( SALES, 5 )
```

ROW_NO	SALES	RETURN VALUE
1	600	NULL
2	504	NULL
3	36	NULL
4	100	NULL

ROW_NO	SALES	RETURN VALUE
5	550	1790
6	39	1229
7	490	1215

Die Funktion gibt jeweils die Summe eines Satzes aus fünf Zeilen zurück: 1790 für Zeilen 1 bis 5, 1229 für Zeilen 2 bis 6 und 1215 für Zeilen 3 bis 7.

NPER

Gibt die Anzahl der Perioden einer Investition basierend auf konstanten Zinssatz und periodischen konstanten Zahlungen zurück.

Syntax

```
NPER( rate, present value, payment [, future value, type] )
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
rate	Erforderlich	Numerisch. Zinsen, die in jeder Periode erwirtschaftet werden. Ausgedrückt als Dezimalzahl. Dividieren Sie den Zinssatz durch 100, um eine Dezimalzahl zu erhalten. Muss größer oder gleich 0 sein.
present value	Erforderlich	Numerisch. Zeitwert: Pauschalbetrag, den eine Reihe zukünftiger Zahlungen heute wert ist.
payment	Erforderlich	Numerisch. Zahlungsbetrag, der pro Periode fällig ist. Muss eine negative Zahl sein.
future value	Optional	Numerisch. Endwert: Barbestand, den Sie nach der letzten Zahlung erreicht haben möchten. Wenn Sie diesen Wert nicht angeben, verwendet NPER 0.
type	Optional	Boolescher Wert. Zeitplan der Zahlung. Geben Sie 1 ein, wenn die Zahlung am Anfang der Periode erfolgt. Geben Sie 0 ein, wenn die Zahlung am Ende der Periode erfolgt. Standardwert ist 0. Andere Werte als 0 oder 1 werden von Data Integration Service als 1 behandelt.

Rückgabewert

Numerisch.

Beispiel

Der aktuelle Wert einer Investition ist 500 USD. Die einzelnen Zahlungen belaufen sich auf jeweils 2000 USD, der Endwert der Investition liegt bei 20.000 USD. Der folgende Ausdruck gibt 9 als die Anzahl der Perioden zurück, in denen Sie die Zahlungen leisten müssen:

```
NPER ( 0.015, -500, -2000, 20000, TRUE )
```

Hinweise

Um die in jeder Periode erwirtschafteten Zinsen zu errechnen, dividieren Sie den jährlichen Zinssatz durch die Anzahl der Zahlungen im Jahresverlauf. Beispiel: Wenn Sie bei jährlichen Zinsen von 15 % monatliche Zahlungen leisten, beträgt der Wert des Arguments „rate“ 15 % dividiert durch 12. Wenn Sie einmal im Jahr zahlen, beträgt „rate“ 15 %.

Die Werte für „payment“ (Zahlung) und „present value“ (Zeitwert) sind negativ, da sie zahlbare Beträge darstellen.

PERCENTILE

Berechnet den Wert, der bei einer gegebenen Perzentile in eine Gruppe von Zahlen fällt. Sie können eine weitere Aggregatfunktion in PERCENTILE schachteln, und die verschachtelte Funktion muss einen numerischen Datentyp zurückgeben.

Data Integration Service liest alle Datenzeilen aus, um die Perzentile zu berechnen. Dieser Vorgang kann sich auf die Leistung auswirken. Optional können Sie einen Filter anwenden, um die Anzahl der Zeilen zu beschränken, aus denen die Perzentile berechnet wird.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
numeric_value	Erforderlich	Numerischer Datentyp. Übergibt die Werte, für die eine Perzentile berechnet werden soll. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.
percentile	Erforderlich	Ganzzahl zwischen 0 und 100, jeweils inklusive. Übergibt die Perzentile, die berechnet werden soll. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben. Wenn Sie eine Zahl außerhalb des Bereichs zwischen 0 und 100 übergeben, zeigt Data Integration Service eine Fehlermeldung an und schreibt die Zeile nicht.
filter_condition	Optional	Begrenzt die Zeilen in der Suche. Die Filterbedingung muss ein numerischer Wert sein oder mit TRUE, FALSE oder NULL ausgewertet werden. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

Numerischer Wert.

NULL, wenn alle übergebenen Werte NULL sind oder keine Zeilen ausgewählt wurden (z. B. wenn die Filterbedingung in allen Zeilen FALSE oder NULL ergibt).

Hinweis: Wenn der Rückgabewert eine Dezimalmal mit Präzision höher als 15 ist, können Sie „Hohe Präzision“ aktivieren, um Dezimalgenauigkeit bis zu 28 Stellen zu gewährleisten.

Nullen

Wenn nur ein Wert NULL ist, wird die Zeile ignoriert. Wenn jedoch alle Werte in einer Gruppe NULL ergeben, gibt PERCENTILE NULL zurück.

Hinweis: Standardmäßig behandelt Data Integration Service Nullwerte in Aggregatfunktionen als NULL. Wenn Sie einen Port oder eine Gruppe mit ausschließlich Nullwerten übergeben, gibt die Funktion NULL zurück. Beim Konfigurieren von Data Integration Service können Sie jedoch entscheiden, wie mit Nullwerten in Aggregatfunktionen umgegangen werden soll. Sie können Nullwerte in Aggregatfunktionen als 0 oder als NULL verarbeiten.

Gruppieren nach

PERCENTILE gruppiert die Werte nach der Einstellung „Gruppieren nach Ports“, die Sie in der Umwandlung festlegen, und gibt pro Gruppe ein Ergebnis zurück.

Wenn „Gruppieren nach Ports“ nicht festgelegt wurde, behandelt PERCENTILE alle Zeilen als eine einzige Gruppe und gibt nur einen Wert zurück.

Beispiel

Data Integration Service berechnet die Perzentile folgendermaßen:

$$i = \frac{(x + 1) \times \text{percentile}}{100}$$

Befolgen Sie folgende Richtlinien:

- *x* ist die Anzahl der Elemente in der Gruppe von Werten, für die Sie die Perzentile berechnen.
- Wenn *i* < 1, gibt PERCENTILE den Wert des ersten Elements in der Liste zurück.
- Wenn *i* ein Ganzzahlwert ist, gibt PERCENTILE des *i*ten Elements in der Liste zurück.
- Andernfalls gibt PERCENTILE den Wert von *n* zurück:

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

Der folgende Ausdruck gibt das Gehalt zurück, das in die 75te Perzentile der Gehälter über 50000 fällt:

```
PERCENTILE( SALARY, 75, SALARY > 50000 )
```

SALARY
125000.0
27900.0
100000.0
NULL
55000.0
9000.0

```
SALARY
85000.0
86000.0
48000.0
99000.0
RETURN VALUE:  106250.0
```

PMT

Gibt die Zahlung für ein Darlehen basierend auf konstanten Zahlungen und konstanten Zinssatz zurück.

Syntax

```
PMT( rate, terms, present value[, future value, type] )
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
rate	Erforderlich	Numerisch. Zinssatz des Darlehens für jede Periode. Ausgedrückt als Dezimalzahl. Dividieren Sie den Zinssatz durch 100, um eine Dezimalzahl zu erhalten. Muss größer oder gleich 0 sein.
terms	Erforderlich	Numerisch. Anzahl der Perioden oder Zahlungen. Muss größer als 0 sein.
present value	Erforderlich	Numerisch. Richtlinie für das Darlehen.
future value	Optional	Numerisch. Endwert: Barbestand, den Sie nach der letzten Zahlung erreicht haben möchten. Wenn Sie diesen Wert nicht angeben, verwendet PMT 0.
type	Optional	Boolescher Wert. Zeitplan der Zahlung. Geben Sie 1 ein, wenn die Zahlung am Anfang der Periode erfolgt. Geben Sie 0 ein, wenn die Zahlung am Ende der Periode erfolgt. Standardwert ist 0. Andere Werte als 0 oder 1 werden von Data Integration Service als 1 behandelt.

Rückgabewert

Numerisch.

Beispiel

Der folgende Ausdruck gibt -2111.64 als monatlichen Zahlungsbetrag für ein Darlehen zurück:

```
PMT( 0.01, 10, 20000 )
```

Hinweise

Zur Berechnung der in jeder Periode erwirtschafteten Zinsen dividieren Sie den jährlichen Zinssatz durch die Anzahl der Zahlungen im Jahresverlauf. Beispiel: Wenn Sie bei jährlichen Zinsen von 15 % monatliche Zahlungen leisten, beträgt „rate“ 15 % / 12. Wenn Sie einmal im Jahr zahlen, beträgt „rate“ 15 %.

Der Wert für „payment“ ist negativ, da er einen zahlbaren Betrag darstellt.

POWER

Gibt einen Wert zurück, der in die Potenz des angegebenen Exponenten der Funktion erhoben wurde.

Syntax

```
POWER( base, exponent )
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
base	Erforderlich	Numerischer Wert. Dieses Argument ist der Basiswert. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben. Wenn der Basiswert negativ ist, muss der Exponent eine Ganzzahl sein.
exponent	Erforderlich	Numerischer Wert. Dieses Argument ist der Exponentwert. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben. Wenn der Basiswert negativ ist, muss der Exponent eine Ganzzahl sein. In diesem Fall rundet die Funktion alle Dezimalwerte auf die nächste Ganzzahl, bevor ein Wert zurückgegeben wird.

Rückgabewert

Double-Wert

NULL, wenn Sie der Funktion einen Nullwert übergeben.

Beispiel

Der folgende Ausdruck gibt die Werte im Port NUMBERS zur Potenz der Werte im Port EXPONENT zurück:

```
POWER( NUMBERS, EXPONENT )
```

NUMBERS	EXPONENT	RETURN VALUE
10.0	2.0	100
3.5	6.0	1838.265625
3.5	5.5	982.594307804838
NULL	2.0	NULL
10.0	NULL	NULL
-3.0	-6.0	0.00137174211248285

NUMBERS	EXPONENT	RETURN VALUE
3.0	-6.0	0.00137174211248285
-3.0	6.0	729.0
-3.0	5.5	729.0

Der Wert -3.0 hoch 6 ergibt dasselbe Ergebnis wie -3.0 hoch 5.5. Wenn die Basis negativ ist, muss der Exponent eine Ganzzahl sein. Andernfalls rundet Data Integration Service den Exponenten auf die nächste Ganzzahl.

PV

Gibt den Zeitwert einer Investition zurück.

Syntax

```
PV( rate, terms, payment [, future value, type] )
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
rate	Erforderlich	Numerisch. Zinsen, die in jeder Periode erwirtschaftet werden. Ausgedrückt als Dezimalzahl. Dividieren Sie den Zinssatz durch 100, um eine Dezimalzahl zu erhalten. Muss größer oder gleich 0 sein.
terms	Erforderlich	Numerisch. Anzahl der Perioden oder Zahlungen. Muss größer als 0 sein.
payments	Erforderlich	Numerisch. Zahlungsbetrag, der pro Periode fällig ist. Muss eine negative Zahl sein.
future value	Optional	Numerisch. Barguthaben nach der letzten Zahlung. Wenn Sie diesen Wert nicht angeben, verwendet PV 0.
types	Optional	Boolescher Wert. Zeitplan der Zahlung. Geben Sie 1 ein, wenn die Zahlung am Anfang der Periode erfolgt. Geben Sie 0 ein, wenn die Zahlung am Ende der Periode erfolgt. Standardwert ist 0. Andere Werte als 0 oder 1 werden von Data Integration Service als 1 behandelt.

Rückgabewert

Numerisch.

Beispiel

Der folgende Ausdruck gibt 12524.43 als den Betrag zurück, den Sie zum heutigen Tag auf das Konto einlegen müssen, um in einem Jahr den Endwert von 20000 USD zu erreichen, wenn Sie auch gleichzeitig am Anfang jeder Periode 500 USD einzahlen:

```
PV( 0.0075, 12, -500, 20000, TRUE )
```

RAND

Gibt eine Zufallszahl zwischen 0 und 1 zurück. Dies ist nützlich für Wahrscheinlichkeitsszenarien.

Syntax

```
RAND( seed )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
seed	Optional	Numerisch. Startwert für Integration Service zum Generieren der Zufallszahl. Wert muss eine Konstante sein. Wenn Sie keinen Startwert (seed) angeben, verwendet Data Integration Service die aktuelle Systemzeit, um die Anzahl von Sekunden seit dem 1. Januar 1971 abzuleiten. Dieser Wert wird als Startwert herangezogen.

Rückgabewert

Numerisch.

Für denselben Startwert generiert Data Integration Service dieselbe Zahlensequenz.

Beispiel

Der folgende Ausdruck kann den Wert 0.417022004702574 zurückgeben:

```
RAND (1)
```

RATE

Gibt die Zinsen zurück, die eine Investition pro Periode erwirtschaftet.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
terms	Erforderlich	Numerisch. Anzahl der Perioden oder Zahlungen. Muss größer als 0 sein.
payments	Erforderlich	Numerisch. Zahlungsbetrag, der pro Periode fällig ist. Muss eine negative Zahl sein.
present value	Erforderlich	Numerisch. Zeitwert: Pauschalbetrag, den eine Reihe zukünftiger Zahlungen heute wert ist.

Argument	Erforderlich/ Optional	Beschreibung
future value	Optional	Numerisch. Endwert: Barbestand, den Sie nach der letzten Zahlung erreicht haben möchten. Der Endwert eines Darlehens ist beispielsweise 0. Wenn Sie diesen Wert nicht angeben, verwendet RATE 0.
types	Optional	Boolescher Wert. Zeitplan der Zahlung. Geben Sie 1 ein, wenn die Zahlung am Anfang der Periode erfolgt. Geben Sie 0 ein, wenn die Zahlung am Ende der Periode erfolgt. Standardwert ist 0. Andere Werte als 0 oder 1 werden von Data Integration Service als 1 behandelt.

Rückgabewert

Numerisch.

Beispiel

Der folgende Ausdruck gibt 0.0077 als monatlichen Zinssatz für ein Darlehen zurück:

```
RATE( 48, -500, 20000 )
```

Zur Berechnung des jährlichen Zinssatzes multiplizieren Sie 0.0077 mit 12. Der jährliche Zinssatz beträgt demnach 0.0924 bzw. 9.24 %.

REG_EXTRACT

Extrahiert Untermuster eines regulären Ausdrucks in einem Eingabewert. Beispiel: Aus einem Muster eines regulären Ausdrucks für einen vollständigen Namen können Sie den Vor- oder Nachnamen extrahieren.

Hinweis: Mit REG_REPLACE können Sie ein Zeichenmuster in einem String durch ein anderes Zeichenmuster ersetzen.

Syntax

```
REG_EXTRACT( subject, 'pattern', subPatternNum, match_from_start )
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
subject	Erforderlich	Zeichenfolgen-Datentyp. Übergibt den Wert, der mit dem Muster des regulären Ausdrucks verglichen werden soll.
pattern	Erforderlich	Zeichenfolgen-Datentyp. Das Muster des regulären Ausdrucks, das abgeglichen werden soll. Sie müssen dafür die Perl-kompatible Syntax für reguläre Ausdrücke verwenden. Setzen Sie das Muster zwischen einfache Anführungszeichen. Setzen Sie alle Untermuster in Klammern.

Argument	Erforderlich/ Optional	Beschreibung
subPatternNum	optional	Ganzzahlwert. Nummer des Untermusters des regulären Ausdrucks, der abgeglichen werden soll. Beachten Sie beim Ermitteln der Untermusternummer folgende Richtlinien: - Kein Wert oder 1: Extrahiert das erste Untermuster des regulären Ausdrucks. - 2: Extrahiert das zweite Untermuster des regulären Ausdrucks. - n: Extrahiert das *n*te Untermuster des regulären Ausdrucks. Standard ist 1.
match_from_start	optional	Numerischer Wert. Gibt die untergeordnete Zeichenfolge zurück, wenn ab dem Start der Zeichenfolge ein Match gefunden wird. Halten Sie sich an die folgenden Richtlinien, um das Match vom Startwert zu ermitteln: - 0. Bringt das Muster mit dem Betreff-String ab dem Startindex oder einem beliebigen Index in Übereinstimmung. - Von null verschieden. Bringt das Muster mit dem Betreff-String ab dem Startindex in Übereinstimmung.

Verwendung der Perl-kompatiblen Syntax für reguläre Ausdrücke

Für die Funktionen REG_EXTRACT, REG_MATCH und REG_REPLACE müssen Sie die Perl-kompatible Syntax für reguläre Ausdrücke verwenden.

Die folgende Tabelle enthält Richtlinien für die Perl-kompatible Syntax für reguläre Ausdrücke:

Syntax	Beschreibung
. (Punkt)	Findet eine Instanz eines beliebigen Zeichens.
[a-z]	Findet eine Instanz eines Zeichens in Kleinbuchstaben. Beispiel: [a-z] findet „ab“. Zum Finden von Übereinstimmungen mit Zeichen in Großbuchstaben verwenden Sie [A-Z].
\d	Findet eine Instanz einer Ziffer zwischen 0 und 9.
\s	Findet ein Leerraumzeichen.
\w	Findet eine Instanz eines alphanumerischen Zeichens, einschließlich Unterstrich (_)
()	Gruppiert einen Ausdruck. Beispiel: Die Klammern in „(\d-\d-\d\d)“ gruppieren den Ausdruck „\d \d-\d\d“, der zwei beliebige Ziffern gefolgt von einem Bindestrich und zwei weiteren beliebigen Ziffern findet, etwa „12-34“.
{}	Findet Zeichen anhand ihrer Anzahl. Beispiel: „\d{3}“ findet drei beliebige Ziffern, etwa 650 oder 510. „[a-z]{2}“ findet zwei beliebige Buchstaben, z. B. CA oder NY.
?	Findet eine oder keine Instanz des vorherigen Zeichens bzw. der vorherigen Zeichengruppe. Beispiel: \d{3}(-{d{4})? findet drei beliebige Ziffern, möglicherweise gefolgt von einem Bindestrich und vier beliebigen Ziffern.

Syntax	Beschreibung
* (Sternchen)	Findet keine oder mehrere Instanzen der Werte, die auf das Sternchen folgen. Beispiel: „*0“ findet jeden Wert, vor einer 0 steht.
+	Findet eine oder mehrere Instanzen der Werte, die auf ein Pluszeichen folgen. Beispiel: „\w+“ findet jeden Wert, der auf ein alphanumerisches Zeichen folgt.

Beispiel: Der folgende reguläre Ausdruck findet fünfstellige US-Postleitzahlen, z. B. 93930, sowie neunstellige Postleitzahlen wie 93930-5407:

```
\d{5}(-\d{4})?
```

„\d{5}“ bezeichnet fünf beliebige Ziffern, etwa 93930. Die Klammern rund um „-\d{4}“ gruppieren dieses Segment des Ausdrucks. Der Bindestrich bezeichnet den Bindestrich in einer neunstelligen Postleitzahl, z. B. 93930-5407. „\d{4}“ bezeichnet vier beliebige Ziffern, etwa 5407. Das Fragezeichen gibt an, dass der Bindestrich und die letzten vier Ziffern entweder gar nicht oder nur einmal auftreten dürfen.

Konvertieren der COBOL-Syntax in Perl-kompatible Syntax für reguläre Ausdrücke

Wenn Sie mit der COBOL-Syntax vertraut sind, können Sie anhand der folgenden Informationen Perl-kompatible reguläre Ausdrücke formulieren.

Die folgende Tabelle zeigt Beispiele der COBOL-Syntax und deren Entsprechungen in Perl:

COBOL-Syntax	Perl-Syntax	Beschreibung
9	\d	Findet eine Instanz einer Ziffer zwischen 0 und 9.
9999	\d\d\d\d oder \d{4}	Findet vier beliebige Ziffern zwischen 0 und 9, etwa 1234 oder 5936.
x	[a-z]	Findet eine Instanz eines Buchstabens.
9xx9	\d[a-z][a-z]\d	Findet eine beliebige Ziffer gefolgt von zwei Buchstaben und einer weiteren Ziffer, etwa „1ab2“.

Konvertieren der SQL-Syntax in Perl-kompatible Syntax für reguläre Ausdrücke

Wenn Sie mit der SQL-Syntax vertraut sind, können Sie anhand der folgenden Informationen Perl-kompatible reguläre Ausdrücke formulieren.

Die folgende Tabelle zeigt Beispiele der SQL-Syntax und deren Entsprechungen in Perl:

SQL-Syntax	Perl-Syntax	Beschreibung
%	. *	Findet jede beliebige Zeichenfolge.
A%	A.*	Findet den Buchstaben „A“ gefolgt von einer Zeichenfolge, etwa „Arena“.
_	. (Punkt)	Findet eine Instanz eines beliebigen Zeichens.
A_	A.	Findet „A“ gefolgt von einem beliebigen Zeichen, etwa AZ.

Rückgabewert

Gibt den Wert des *n*ten Untermusters zurück, das Teil des Eingabewerts ist. Das *n*te Untermuster basiert auf dem für subPatternNum angegebenen Wert.

NULL, wenn die Eingabe ein Nullwert ist oder das Muster NULL ergibt.

Beispiel

Mit REG_EXTRACT in einem Ausdruck können Sie den zweiten Vornamen aus einem regulären Ausdruck extrahieren, mit dem nach Vornamen, zweiten Vornamen und Nachnamen gesucht wird. Beispiel: Der folgende Ausdruck gibt den zweiten Vornamen in einem regulären Ausdruck zurück:

```
REG_EXTRACT( Employee_Name, '(\w+)\s+(\w+)\s+(\w+)',2)
```

Employee_Name	Return Value
Stephen Graham Smith	Graham
Juan Carlos Fernando	Carlos

REG_MATCH

Gibt zurück, ob ein Wert mit dem Muster eines regulären Ausdrucks übereinstimmt. Damit können Sie Datenmuster wie IDs, Telefonnummern oder Postleitzahlen validieren.

Hinweis: Mit REG_REPLACE können Sie ein Zeichenmuster in einem String durch ein anderes Zeichenmuster ersetzen.

Syntax

```
REG_MATCH( subject, pattern )
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
subject	Erforderlich	String-Datentyp. Übergibt den Wert, der mit dem regulären Ausdrucksmuster abgeglichen werden soll.
pattern	Erforderlich	String-Datentyp. Das Muster des regulären Ausdrucks, das abgeglichen werden soll. Sie müssen dafür die Perl-kompatible Syntax für reguläre Ausdrücke verwenden. Setzen Sie das Muster zwischen einfache Anführungszeichen. Weitere Informationen hierzu finden Sie unter "REG_EXTRACT" auf Seite 158.

Rückgabewert

TRUE, wenn die Daten mit dem Muster übereinstimmen.

FALSE, wenn die Daten mit dem Muster nicht übereinstimmen.

NULL, wenn die Eingabe ein Nullwert ist oder das Muster NULL ergibt.

Beispiel

Angenommen, Sie verwenden REG_MATCH in einem Ausdruck zum Validieren von Telefonnummern. Der folgende Ausdruck gleicht beispielsweise zehnstellige Telefonnummern mit dem Muster ab und gibt je nach Match einen Booleschen Wert zurück:

```
REG_MATCH (Phone_Number, '(\d\d\d-\d\d\d-\d\d\d\d)' )
```

Phone_Number	Return Value
408-555-1212	TRUE
	NULL
510-555-1212	TRUE
92 555 51212	FALSE
650-555-1212	TRUE
415-555-1212	TRUE
831 555 12123	FALSE

Tipp

Sie können REG_MATCH auch für folgende Aufgaben einsetzen:

- Um sicherzugehen, dass ein Wert mit einem Muster übereinstimmt: Dieser Verwendungszweck ähnelt dem der Funktion SQL LIKE.
- Um sicherzugehen, dass alle Werte Zeichen sind: Dieser Verwendungszweck ähnelt dem der Funktion SQL IS_CHAR.

Um sich zu vergewissern, dass ein Wert mit einem Muster übereinstimmt, verwenden Sie REG_MATCH zusammen mit einem Punkt (.). Ein Punkt findet eine Instanz eines beliebigen Zeichens. Ein Sternchen findet keine oder mehrere Instanzen der Werte, die darauf folgen.

Verwenden Sie beispielsweise den folgenden Ausdruck zum Suchen von Kontonummern, die mit 1835 beginnen:

```
REG_MATCH(ACCOUNT_NUMBER, '1835.*')
```

Um sicherzustellen, dass es sich bei Werten um Zeichen handelt, verwenden Sie eine REG_MATCH-Funktion mit „[a-zA-Z]+“. „a-z“ findet alle kleingeschriebenen Zeichen. „A-Z“ findet alle großgeschriebenen Zeichen. Das Pluszeichen (+) gibt an, dass mindestens ein Zeichen vorkommen muss.

Beispiel: Mit dem folgenden Ausdruck überprüfen Sie, ob eine Liste von Nachnamen ausschließlich Zeichen enthält:

```
REG_MATCH(LAST_NAME, '[a-zA-Z]+')
```

REG_REPLACE

Ersetzt Zeichen in einem String durch ein anderes Zeichenmuster. Standardmäßig durchsucht REG_REPLACE den Eingabestring nach dem angegebenen Zeichenmuster und ersetzt jedes Vorkommen davon durch das

neue Muster. Sie können auch die Anzahl der Vorkommen des Musters angeben, die im String ersetzt werden sollen.

Syntax

```
REG_REPLACE( subject, pattern, replace, numReplacements )
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
subject	Erforderlich	String-Datentyp. Übergibt den String, der durchsucht werden soll.
pattern	Erforderlich	String-Datentyp. Übergibt den zu ersetzenden Zeichenstring. Sie müssen dafür die Perl-kompatible Syntax für reguläre Ausdrücke verwenden. Setzen Sie das Muster zwischen einfache Anführungszeichen. Weitere Informationen hierzu finden Sie unter "REG_EXTRACT" auf Seite 158.
replace	Erforderlich	String-Datentyp. Übergibt den neuen Zeichenstring.
numReplacements	Optional	Numerischer Datentyp. Gibt die Anzahl der Vorkommen an, die ersetzt werden sollen. Wenn Sie diese Option nicht angeben, ersetzt REG_REPLACE alle Vorkommen des Zeichenstrings.

Rückgabewert

String

Beispiel

Der folgende Ausdruck entfernt zusätzliche Leerzeichen aus den Mitarbeiternamen in jeder Zeile des Ports „Employee_Name“:

```
REG_REPLACE( Employee_Name, '\s+', ' ')
```

Employee_Name	RETURN VALUE
Adam Smith	Adam Smith
Greg Sanders	Greg Sanders
Sarah Fe	Sarah Fe
Sam Cooper	Sam Cooper

REPLACECHR

Ersetzt Zeichen in einer Zeichenfolge durch ein einzelnes Zeichen oder kein Zeichen. REPLACECHR durchsucht die Eingabezeichenfolge nach den angegebenen Zeichen und ersetzt jedes Vorkommen davon durch das neue Zeichen, das Sie angeben.

Syntax

```
REPLACECHR( CaseFlag, InputString, OldCharSet, NewChar )
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
CaseFlag	Erforderlich	Muss eine Ganzzahl sein. Legt fest, ob für die Argumente in dieser Funktion zwischen Groß- und Kleinschreibung unterschieden wird. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben. Wenn *CaseFlag* eine andere Zahl als 0 enthält, wird bei der Funktion zwischen Groß- und Kleinschreibung unterschieden. Wenn *CaseFlag* einen Nullwert oder 0 enthält, wird bei der Funktion nicht zwischen Groß- und Kleinschreibung unterschieden.
InputString	Erforderlich	Muss eine Zeichenfolge sein. Übergibt die Zeichenfolge, die durchsucht werden soll. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben. Wenn Sie einen numerischen Wert übergeben, konvertiert ihn die Funktion in eine Zeichenfolge. Wenn *InputString* NULL ist, gibt REPLACECHR NULL zurück.
OldCharSet	Erforderlich	Muss eine Zeichenfolge sein. Die Zeichen, die ersetzt werden sollen. Sie können eines oder mehrere Zeichen eingeben. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben. Sie können auch ein Text-Literal zwischen einfachen Anführungszeichen angeben, z. B. 'abc'. Wenn Sie einen numerischen Wert übergeben, konvertiert ihn die Funktion in eine Zeichenfolge. Wenn *OldCharSet* NULL oder leer ist, gibt REPLACECHR *InputString* zurück.
NewChar	Erforderlich	Muss eine Zeichenfolge sein. Sie können ein Zeichen, eine leere Zeichenfolge oder NULL angeben. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben. Wenn *NewChar* NULL oder leer ist, entfernt REPLACECHR alle Vorkommen aller Zeichen in *OldCharSet* aus *InputString*. Wenn *NewChar* mehr als ein Zeichen enthält, ersetzt REPLACECHR *OldCharSet* durch das erste Zeichen.

Rückgabewert

String.

Leerer String, wenn REPLACECHR alle Zeichen aus *InputString* entfernt.

NULL, wenn *InputString* NULL ist.

InputString, wenn *OldCharSet* NULL oder leer ist.

Beispiele

Der folgende Ausdruck entfernt die doppelten Anführungszeichen aus den Webprotokolldaten im Port WEBLOG:

```
REPLACECHR( 0, WEBLOG, '"', NULL )
```

WEBLOG	RETURN VALUE
"GET /news/index.html HTTP/1.1"	GET /news/index.html HTTP/1.1
"GET /companyinfo/index.html HTTP/1.1"	GET /companyinfo/index.html HTTP/1.1

WEBLOG	RETURN VALUE
GET /companyinfo/index.html HTTP/1.1	GET /companyinfo/index.html HTTP/1.1
NULL	NULL

Der folgende Ausdruck entfernt mehrere Zeichen aus jeder Zeile im Port WEBLOG:

```
REPLACECHR ( 1, WEBLOG, ']["', NULL )
```

WEBLOG	RETURN VALUE
[29/Oct/2001:14:13:50 -0700]	29/Oct/2001:14:13:50 -0700
[31/Oct/2000:19:45:46 -0700] "GET /news/index.html HTTP/1.1"	31/Oct/2000:19:45:46 -0700 GET /news/index.html HTTP/1.1
[01/Nov/2000:10:51:31 -0700] "GET /news/index.html HTTP/1.1"	01/Nov/2000:10:51:31 -0700 GET /news/index.html HTTP/1.1
NULL	NULL

Der folgende Ausdruck ändert einen Teil des Werts des Kundencodes im Port CUSTOMER_CODE:

```
REPLACECHR ( 1, CUSTOMER_CODE, 'A', 'M' )
```

CUSTOMER_CODE	RETURN VALUE
ABA	MBM
abA	abM
BBC	BBC
ACC	MCC
NULL	NULL

Der folgende Ausdruck ändert einen Teil des Werts des Kundencodes im Port CUSTOMER_CODE:

```
REPLACECHR ( 0, CUSTOMER_CODE, 'A', 'M' )
```

CUSTOMER_CODE	RETURN VALUE
ABA	MBM
abA	MbM
BBC	BBC
ACC	MCC

Der folgende Ausdruck ändert einen Teil des Werts des Kundencodes im Port CUSTOMER_CODE:

```
REPLACECHR ( 1, CUSTOMER_CODE, 'A', NULL )
```

CUSTOMER_CODE	RETURN VALUE
ABA	B
BBC	BBC
ACC	CC
AAA	*[empty string]*
aaa	aaa
NULL	NULL

Der folgende Ausdruck entfernt mehrere Zahlen aus jeder Zeile im Port INPUT:

```
REPLACECHR ( 1, INPUT, '14', NULL )
```

INPUT	RETURN VALUE
12345	235
4141	NULL
111115	5
NULL	NULL

Zur Angabe eines einfachen Anführungszeichens (') in *OldCharSet* oder *NewChar* benötigen Sie die Funktion CHR. Das einfache Anführungszeichen ist das einzige Zeichen, das in String-Literalen nicht verwendet werden darf.

Der folgende Ausdruck entfernt mehrere Zeichen, darunter auch das einfache Ausführungszeichen, aus jeder Zeile im Port INPUT:

```
REPLACECHR (1, INPUT, CHR(39), NULL )
```

INPUT	RETURN VALUE
'Tom Smith' 'Laura Jones'	Tom Smith Laura Jones
Tom's	Toms
NULL	NULL

REPLACESTR

Ersetzt Zeichen in einem String durch ein einzelnes Zeichen, mehrere Zeichen oder kein Zeichen. REPLACESTR durchsucht den Eingabestring nach allen angegebenen Strings und ersetzt sie durch den neuen String, den Sie angeben.

Syntax

```
REPLACESTR ( CaseFlag, InputString, OldString1, [OldString2, ... OldStringN,] NewString )
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
CaseFlag	Erforderlich	Muss eine Ganzzahl sein. Legt fest, ob für die Argumente in dieser Funktion zwischen Groß- und Kleinschreibung unterschieden wird. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben. Wenn *CaseFlag* eine andere Zahl als 0 enthält, wird bei der Funktion zwischen Groß- und Kleinschreibung unterschieden. Wenn *CaseFlag* einen Nullwert oder 0 enthält, wird bei der Funktion nicht zwischen Groß- und Kleinschreibung unterschieden.
InputString	Erforderlich	Muss ein Zeichenstring sein. Übergibt die Strings, die durchsucht werden sollen. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben. Wenn Sie einen numerischen Wert übergeben, konvertiert ihn die Funktion in einen Zeichenstring. Wenn *InputString* NULL ist, gibt REPLACESTR NULL zurück.
OldString	Erforderlich	Muss ein Zeichenstring sein. Der String, der ersetzt werden soll. Sie müssen mindestens ein *OldString*-Argument eingeben. Pro *OldString*-Argument können Sie ein oder mehrere Zeichen angeben. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben. Sie können auch ein Text-Literal zwischen einfachen Anführungszeichen angeben, z. B. 'abc'. Wenn Sie einen numerischen Wert übergeben, konvertiert ihn die Funktion in einen Zeichenstring. Wenn REPLACESTR mehrere *OldString*-Argumente enthält und mindestens ein *OldString*-Argument NULL oder leer ist, ignoriert REPLACESTR das *OldString*-Argument. Sind alle *OldString*-Argumente NULL oder leer, gibt REPLACESTR *InputString* zurück. Die Zeichen in den *OldString*-Argumenten werden in der Reihenfolge ersetzt, in der sie in der Funktion angegeben sind. Beispiel: Wenn Sie mehrere *OldString*-Argumente angeben, hat das erste *OldString*-Argument Vorrang vor dem zweiten und das zweite *OldString*-Argument Vorrang vor dem dritten.. Beim Ersetzen eines Strings platziert REPLACESTR den Cursor nach dem ersetzten Zeichen in *InputString*, bevor es nach der nächsten Übereinstimmung sucht.
NewString	Erforderlich	Muss ein Zeichenstring sein. Sie können ein Zeichen, mehrere Zeichen, einen leeren String oder NULL angeben. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben. Wenn *NewString* NULL oder leer ist, entfernt REPLACESTR alle Vorkommen von *OldString* aus *InputString*.

Rückgabewert

String.

Leerer String, wenn REPLACESTR alle Zeichen aus *InputString* entfernt.

NULL, wenn *InputString* NULL ist.

InputString, wenn alle *OldString*-Argumente NULL oder leer sind.

Beispiele

Der folgende Ausdruck entfernt die doppelten Anführungszeichen und zwei verschiedene Textstrings aus den Webprotokolldaten im Port WEBLOG:

```
REPLACESTR( 1, WEBLOG, '"', 'GET ', ' HTTP/1.1', NULL )
```

WEBLOG	RETURN VALUE
"GET /news/index.html HTTP/1.1"	/news/index.html
"GET /companyinfo/index.html HTTP/1.1"	/companyinfo/index.html
GET /companyinfo/index.html	/companyinfo/index.html
GET	*[empty string]*
NULL	NULL

Der folgende Ausdruck ändert die Anrede für bestimmte Werte aus jeder Zeile im Port TITLE:

```
REPLACESTR ( 1, TITLE, 'rs.', 'iss', 's.' )
```

TITLE	RETURN VALUE
Mrs.	Ms.
Miss	Ms.
Mr.	Mr.
MRS.	MRS.

Der folgende Ausdruck ändert die Anrede für bestimmte Werte aus jeder Zeile im Port TITLE:

```
REPLACESTR ( 0, TITLE, 'rs.', 'iss', 's.' )
```

TITLE	RETURN VALUE
Mrs.	Ms.
MRS.	Ms.

Der folgende Ausdruck zeigt, wie REPLACESTR mehrere `OldString`-Argumente in jeder Zeile im Port INPUT ersetzt:

```
REPLACESTR ( 1, INPUT, 'ab', 'bc', '*' )
```

INPUT	RETURN VALUE
abc	*c
abbc	**
abbbbc	*bb*
bc	*

Der folgende Ausdruck zeigt, wie REPLACESTR mehrere `OldString`-Argumente in jeder Zeile im Port INPUT ersetzt:

```
REPLACESTR ( 1, INPUT, 'ab', 'bc', 'b' )
```

INPUT	RETURN VALUE
ab	b
bc	b
abc	bc
abbc	bb
abbcc	bbc

Zur Angabe eines einfachen Anführungszeichens (') in *OldString* oder *NewString* benötigen Sie die Funktion CHR. Verwenden Sie CHR und CONCAT zum Verketten eines einzelnen Anführungszeichens mit einem String. Das einfache Anführungszeichen ist das einzige Zeichen, das in String-Literalen nicht verwendet werden darf. Betrachten Sie das folgende Beispiel:

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

Der Rückgabewert ist:

```
Joan's car
```

Der folgende Ausdruck entfernt einen String mit einem einfachen Ausführungszeichen aus jeder Zeile des Ports INPUT:

```
REPLACESTR ( 1, INPUT, CONCAT('it', CONCAT(CHR(39), 's' )), 'its' )
```

INPUT	RETURN VALUE
it's	its
mit's	mits
mits	mits
mits'	mits'

REVERSE

Kehrt den Eingabestring um.

Syntax

```
REVERSE( string )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
string	Erforderlich	Beliebiger Zeichenwert. Wert, der umgekehrt werden soll.

Rückgabewert

String. Umgekehrter Eingabewert.

Beispiel

Der folgende Ausdruck kehrt die Zahlen des Kundencode um:

```
REVERSE( CUSTOMER_CODE )
```

CUSTOMER_CODE	RETURN VALUE
0001	1000
0002	2000
0003	3000
0004	4000

ROUND (Datum)

Rundet einen Teil des Datums. ROUND kann auch zum Runden von Zahlen eingesetzt werden.

Die Funktion kann folgende Teile eines Datums runden:

Jahr

Rundet die Jahreskomponente eines Datums anhand des Monats.

Monat

Rundet die Monatskomponente eines Datums anhand des Tages des Monats.

Tag

Rundet die Tageskomponente eines Datums anhand der Uhrzeit.

Stunde

Rundet die Stundenkomponente eines Datums anhand der Minuten.

Minute

Rundet die Minutenkomponente eines Datums anhand der Sekunden.

Sekunde

Rundet die Sekundenkomponente eines Datums anhand der Millisekunden.

Millisekunde

Rundet die Millisekundenkomponente eines Datums anhand der Mikrosekunden.

Mikrosekunde

Rundet die Mikrosekundenkomponente eines Datums anhand der Nanosekunden.

Die folgende Tabelle zeigt die Bedingungen des ROUND-Ausdrucks und die entsprechenden Rückgabewerte:

Bedingung	Ausdruck	Rückgabewert
Für die Monate von Januar bis Juni gibt die Funktion den 1. Januar desselben Jahres zurück und setzt die Uhrzeit auf 00:00:00.000000000.	`ROUND(TO_DATE('04/16/1998 8:24:19','MM/DD/YYYY HH24:MI:SS'),'YY')`	01/01/1998 00:00:00.000000000
Für die Monate von Juli bis Dezember gibt die Funktion den 1. Januar des nächsten Jahres zurück und setzt die Uhrzeit auf 00:00:00.000000000.	`ROUND(TO_DATE('07/30/1998 2:30:55','MM/DD/YYYY HH24:MI:SS'),'YY')`	01/01/1999 00:00:00.000000000
Für den Tag des Monats zwischen dem 1. und dem 15. gibt die Funktion den ersten Tag des Eingabemonats zurück und setzt die Uhrzeit auf 00:00:00.000000000.	`ROUND(TO_DATE('04/15/1998 8:24:19','MM/DD/YYYY HH24:MI:SS'),'MM')`	04/01/1998 00:00:00.000000000
Für den 16. bis letzten Tag des Monats gibt die Funktion den ersten Tag des nächsten Monats zurück und setzt die Uhrzeit auf 00:00:00.000000000.	`ROUND(TO_DATE('05/22/1998 10:15:29','MM/DD/YYYY HH24:MI:SS'),'MM')`	06/01/1998 00:00:00.000000000
Für die Uhrzeit zwischen 00:00:00 (12 a.m.) und 11:59:59 (a.m.) gibt die Funktion das aktuelle Datum zurück und setzt die Uhrzeit auf 00:00:00.000000000 (12 a.m.).	`ROUND(TO_DATE('06/13/1998 2:30:45','MM/DD/YYYY HH24:MI:SS'),'DD')`	06/13/1998 00:00:00.000000000
Ab 12:00:00 (12 p.m.) gibt die Funktion das Datum des nächsten Tages zurück und setzt die Uhrzeit auf 00:00:00.000000000 (12 a.m.).	`ROUND(TO_DATE('06/13/1998 22:30:45','MM/DD/YYYY HH24:MI:SS'),'DD')`	06/14/1998 00:00:00.000000000
Für die Minuten zwischen 0 und 29 gibt die Funktion die aktuelle Stunde zurück und setzt die Minuten, Sekunden, Millisekunden und Nanosekunden auf 0.	`ROUND(TO_DATE('04/01/1998 11:29:35','MM/DD/YYYY HH24:MI:SS'),'HH')`	04/01/1998 11:00:00.000000000
Ab 30 Minuten nach der vollen Stunde gibt die Funktion die nächste volle Stunde zurück und setzt die Minuten, Sekunden, Millisekunden und Nanosekunden auf 0.	`ROUND(TO_DATE('04/01/1998 13:39:00','MM/DD/YYYY HH24:MI:SS'),'HH')`	04/01/1998 14:00:00.000000000
Für die Zeit zwischen 0 und 29 Sekunden gibt die Funktion die aktuelle Minute zurück und setzt die Sekunden, Millisekunden und Nanosekunden auf 0.	`ROUND(TO_DATE('05/22/1998 10:15:29','MM/DD/YYYY HH24:MI:SS'),'MI')`	05/22/1998 10:15:00.000000000
Für die Zeit zwischen 30 und 59 Sekunden gibt die Funktion die nächste volle Minute zurück und setzt Sekunden, Millisekunden und Nanosekunden auf 0.	`ROUND(TO_DATE('05/22/1998 10:15:30','MM/DD/YYYY HH24:MI:SS'),'MI')`	05/22/1998 10:16:00.000000000
Für die Zeit zwischen 0 und 499 Millisekunden gibt die Funktion die aktuelle Sekunde zurück und setzt die Millisekunden auf 0.	`ROUND(TO_DATE('05/22/1998 10:15:29.499','MM/DD/YYYY HH24:MI:SS.MS'),'SS')`	05/22/1998 10:15:29.000000000

Bedingung	Ausdruck	Rückgabewert
Für die Zeit zwischen 500 und 999 Millisekunden gibt die Funktion die nächste volle Sekunde zurück und setzt die Millisekunden auf 0.	`ROUND(TO_DATE('05/22/1998 10:15:29.500','MM/DD/YYYY HH24:MI:SS.MS'),'SS')`	05/22/1998 10:15:30.000000000
Für die Zeit zwischen 0 und 499 Mikrosekunden gibt die Funktion die aktuelle Millisekunde zurück und setzt die Mikrosekunden auf 0.	`ROUND(TO_DATE('05/22/1998 10:15:29.498125','MM/DD/YYYY HH24:MI:SS.US'),'MS')`	05/22/1998 10:15:29.498000000
Für die Zeit zwischen 500 und 999 Mikrosekunden gibt die Funktion die nächste volle Millisekunde zurück und setzt die Mikrosekunden auf 0.	`ROUND(TO_DATE('05/22/1998 10:15:29.498785','MM/DD/YYYY HH24:MI:SS.US'),'MS')`	05/22/1998 10:15:29.499000000
Für die Zeit zwischen 0 und 499 Nanosekunden gibt die Funktion die aktuelle Mikrosekunde zurück und setzt die Nanosekunden auf 0.	`ROUND(TO_DATE('05/22/1998 10:15:29.498125345','MM/DD/YYYY HH24:MI:SS.NS'),'US')`	05/22/1998 10:15:29.498125000
Für die Zeit zwischen 500 und 999 Nanosekunden gibt die Funktion die nächste volle Mikrosekunde zurück und setzt die Nanosekunden auf 0.	`ROUND(TO_DATE('05/22/1998 10:15:29.498125876','MM/DD/YYYY HH24:MI:SS.NS'),'US')`	05/22/1998 10:15:29.498126000

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
date	Erforderlich	Datum/Zeit-Datentyp. Vor dem Runden können Sie die Funktion TO_DATE verschachteln, um Zeichenfolgen in Datumsangaben zu konvertieren.
format	optional	Geben Sie eine gültige Formatzeichenfolge ein. Das ist der Teil des Datums, der gerundet werden soll. Es kann nur jeweils ein Teil des Datums gerundet werden. Wenn Sie die Formatzeichenfolge nicht angeben, rundet die Funktion das Datum auf den nächsten Tag.

Rückgabewert

Datum mit gerundetem angegebenen Teil. ROUND gibt ein Datum im selben Format wie das Ausgangsdatum zurück. Sie können die Ergebnisse dieser Funktion mit einem beliebigen Port des Datum/Zeit-Datentyps verknüpfen.

NULL, wenn Sie der Funktion einen Nullwert übergeben.

Beispiele

Die folgenden Ausdrücke runden die Jahreskomponente der Datumsangaben im Port DATE_SHIPPED:

```
ROUND( DATE_SHIPPED, 'Y' )
ROUND( DATE_SHIPPED, 'YY' )
```

```
ROUND( DATE_SHIPPED, 'YYY' )
ROUND( DATE_SHIPPED, 'YYYY' )
```

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:30AM	Jan 1 1998 12:00:00.000000000AM
Apr 19 1998 1:31:20PM	Jan 1 1998 12:00:00.000000000AM
Dec 20 1998 3:29:55PM	Jan 1 1999 12:00:00.000000000AM
NULL	NULL

Die folgenden Ausdrücke runden die Monatskomponente aller Datumsangaben im Port DATE_SHIPPED:

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

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:30AM	Jan 1 1998 12:00:00.000000000AM
Apr 19 1998 1:31:20PM	May 1 1998 12:00:00.000000000AM
Dec 20 1998 3:29:55PM	Jan 1 1999 12:00:00.000000000AM
NULL	NULL

Die folgenden Ausdrücke runden die Tageskomponente aller Datumsangaben im Port DATE_SHIPPED:

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

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:30AM	Jan 15 1998 12:00:00.000000000AM
Apr 19 1998 1:31:20PM	Apr 20 1998 12:00:00.000000000AM
Dec 20 1998 3:29:55PM	Dec 21 1998 12:00:00.000000000AM
Dec 31 1998 11:59:59PM	Jan 1 1999 12:00:00.000000000AM
NULL	NULL

Die folgenden Ausdrücke runden die Stundenkomponente aller Datumsangaben im Port DATE_SHIPPED:

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

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:31AM	Jan 15 1998 2:00:00.000000000AM

DATE_SHIPPED	RETURN VALUE
Apr 19 1998 1:31:20PM	Apr 19 1998 2:00:00.000000000PM
Dec 20 1998 3:29:55PM	Dec 20 1998 3:00:00.000000000PM
Dec 31 1998 11:59:59PM	Jan 1 1999 12:00:00.000000000AM
NULL	NULL

Der folgende Ausdruck rundet die Minutenkomponente aller Datumsangaben im Port DATE_SHIPPED:

```
ROUND( DATE_SHIPPED, 'MI' )
```

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:30AM	Jan 15 1998 2:11:00.000000000AM
Apr 19 1998 1:31:20PM	Apr 19 1998 1:31:00.000000000PM
Dec 20 1998 3:29:55PM	Dec 20 1998 3:30:00.000000000PM
Dec 31 1998 11:59:59PM	Jan 1 1999 12:00:00.000000000AM
NULL	NULL

ROUND (Zahlen)

Rundet Zahlen auf eine angegebene Anzahl von Ziffern oder Dezimalstellen. ROUND kann auch zum Runden von Datumsangaben eingesetzt werden.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
numeric_value	Erforderlich	Numerischer Datentyp. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben. Mithilfe von Operatoren können Sie vor dem Runden mathematische Berechnungen durchführen.
precision	Optional	Positive oder negative Ganzzahl. Wenn Sie für *precision* eine positive Präzision angeben, rundet die Funktion die Zahl auf die entsprechende Anzahl von Dezimalstellen. Beispiel: ROUND(12.99, 1) ergibt 13.0 und ROUND(15.44, 1) ergibt 15.4. Bei einer negativen *precision* rundet die Funktion die Zahl auf die entsprechende Anzahl von Ziffern links vom Dezimalzeichen und gibt eine Ganzzahl zurück. Beispiel: ROUND(12.99, -1) ergibt 10, ROUND(15.99, -1) ergibt 20. Wenn Sie eine dezimale *precision* angeben, wird die Zahl auf die nächste Ganzzahl gerundet, bevor der Ausdruck ausgewertet wird. Beispiel: ROUND(12.99, 0.8) gibt 13.0 zurück, da 0.8 vor der Auswertung des Ausdrucks erst auf 1 gerundet wird. Wenn Sie das Argument *precision* auslassen, wird die Zahl auf die nächste Ganzzahl gerundet. Die Dezimalstellen werden abgeschnitten. Beispiel: ROUND(12.99) ergibt 13.

Rückgabewert

Numerischer Wert.

Wenn eines der Argumente NULL ist, gibt ROUND NULL zurück.

Hinweis: Wenn der Rückgabewert eine Dezimalmal mit Präzision höher als 15 ist, können Sie „Hohe Präzision“ aktivieren, um Dezimalgenauigkeit bis zu 28 Stellen zu gewährleisten.

Beispiele

Der folgende Ausdruck gibt die Werte im Port PRICE auf drei Dezimalstellen gerundet zurück:

```
ROUND( PRICE, 3 )
```

PRICE	RETURN VALUE
12.9936	12.994
15.9949	15.995
-18.8678	-18.868
56.9561	56.956
NULL	NULL

Sie können die Ziffern links vom Dezimaltrennzeichen runden, indem Sie für das Argument *precision* eine negative Zahl angeben:

```
ROUND( PRICE, -2 )
```

PRICE	RETURN VALUE
13242.99	13200.0
1435.99	1400.0
-108.95	-100.0
NULL	NULL

Bei Dezimalwerten im Argument *precision* rundet Data Integration Service die Zahl auf die nächste Ganzzahl und wertet den Ausdruck erst danach aus:

```
ROUND( PRICE, 0.8 )
```

PRICE	RETURN VALUE
12.99	13.0
56.34	56.3
NULL	NULL

Wenn Sie die Präzision in *precision* nicht angeben, wird die Zahl auf die nächste Ganzzahl gerundet:

```
ROUND( PRICE )
```

PRICE	RETURN VALUE
12.99	13.0
-15.99	-16.0
-18.99	-19.0
56.95	57.0
NULL	NULL

Tipp

Sie können ROUND auch dazu nutzen, die Präzision der berechneten Werte genau anzugeben und die erwarteten Resultate erzielen. Wenn Data Integration Service mit geringer Präzision ausgeführt wird, werden Ergebnisse, deren Präzision 15 Stellen überschreitet, abgeschnitten. Beispiel: Angenommen, Sie möchten folgenden Ausdruck mit geringer Präzision verarbeiten:

```
7/3 * 3 = 7
```

In diesem Fall wertet Data Integration Service den linken Teil des Ausdrucks mit 6.999999999999999 aus, da das Ergebnis der ersten Division abgeschnitten wird. Data Integration Service wertet den gesamten Ausdruck entsprechend mit FALSE aus. Das ist möglicherweise nicht das Ergebnis, das Sie erwartet haben.

Um das erwartete Ergebnis zu erzielen, runden Sie das abgeschnittene Ergebnis auf der linken Ausdrucksseite mit ROUND. Data Integration Service wertet folgenden Ausdruck mit TRUE aus:

```
ROUND(7/3 * 3) = 7
```

RPAD

Ändert einen String auf die angegebene Länge, indem Zeichen oder Leerzeichen am Ende des Strings hinzugefügt werden.

Syntax

```
RPAD( first_string, length [,second_string] )
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
first_string	Erforderlich	Beliebiger Stringwert. Die Strings, die Sie ändern möchten. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.
length	Erforderlich	Muss ein positives Ganzzahl-Literal sein. Gibt die Länge an, die alle Strings erreichen sollen.
second_string	Optional	Beliebiger Stringwert. Übergibt den String, der an die rechte Seite der Werte in *first_string* angehängt werden soll. Setzen Sie die Zeichen, die Sie am Ende des Strings hinzufügen möchten, zwischen einfache Anführungszeichen: 'abc'. Bei diesem Argument wird zwischen Groß- und Kleinschreibung unterschieden. Wenn Sie „second_string“ nicht angeben, füllt die Funktion das Ende des ersten Strings mit Leerzeichen auf.

Rückgabewert

String der angegebenen Länge.

NULL, wenn ein der Funktion übergebener Wert NULL oder „length“ eine negative Zahl ist.

Beispiele

Der folgende Ausdruck gibt den Artikelnamen mit einer Länge von 16 Zeichen zurück, indem der String „.“ an das Ende jedes Artikelnamens angefügt wird:

```
RPAD( ITEM_NAME, 16, '.')
```

ITEM_NAME	RETURN VALUE
Flashlight	Flashlight.....
Compass	Compass.....
Regulator System	Regulator System
Safety Knife	Safety Knife....

RPAD zählt die Länge von links nach rechts. Wenn der angegebene erste String also länger ist als die Länge in „length“, schneidet RPAD den String von rechts nach links ab. Beispiel: RPAD(‘alphabetisch’, 5, ‘x’) gibt den String „alpha“ zurück. RPAD verwendet bei Bedarf einen Teil des Arguments *second_string*.

Der folgende Ausdruck gibt den Artikelnamen mit einer Länge von 16 Zeichen zurück, indem der String „*..*“an das Ende jedes Artikelnamens angefügt wird:

```
RPAD( ITEM_NAME, 16, '*..*' )
```

ITEM_NAME	RETURN VALUE
Flashlight	Flashlight*..**.
Compass	Compass*..**..**
Regulator System	Regulator System
Safety Knife	Safety Knife*..*

RTRIM

Entfernt Zeichen oder Leerzeichen am Ende eines Strings.

Wenn Sie den Parameter *trim_set* im Ausdruck nicht angeben:

- Unicode-Modus: RTRIM entfernt Single- und Double-Byte-Leerzeichen am Stringende.
- ASCII-Modus: RTRIM entfernt nur Single-Byte-Leerzeichen.

Beim Entfernen von Zeichen aus einem String mit RTRIM vergleicht die Funktion den Wert *trim_set* einzeln von rechts nach links mit allen Zeichen im Argument *string*. Wenn eine Übereinstimmung zwischen einem Zeichen im String und einem Zeichen in *trim_set* gefunden wird, wird das betreffende Zeichen entfernt. RTRIM vergleicht und entfernt so lange Zeichen, bis keine übereinstimmenden Zeichen in *trim_set* mehr gefunden werden. Dann wird der String ohne die übereinstimmenden Zeichen zurückgegeben.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
string	Erforderlich	Beliebiger Stringwert. Übergibt die Werte, um die der String beschnitten werden soll. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben. Verwenden Sie Operatoren, um Strings vor dem Entfernen von Leerzeichen am Stringende zu vergleichen oder zu verketten.
trim_set	Optional	Beliebiger Stringwert. Übergibt die Zeichen, die am Ende des ersten Strings entfernt werden sollen. Sie können auch ein Text-Literal eingeben. Sie müssen die Zeichen, die Sie am Stringende entfernen möchten, jedoch zwischen einfache Anführungszeichen setzen: 'abc'. Wenn Sie den zweiten String nicht angeben, entfernt die Funktion die Leerzeichen am Ende des ersten Strings. RTRIM unterscheidet zwischen Groß- und Kleinschreibung.

Rückgabewert

String. Die Stringwerte ohne die im Argument *trim_set* angegebenen Zeichen.

NULL, falls ein an die Funktion übergebener Wert NULL ist.

Beispiel

Der folgende Ausdruck entfernt die Zeichen „re“ aus den Strings im Port LAST_NAME:

```
RTRIM( LAST_NAME, 're')
```

LAST_NAME	RETURN VALUE
Nelson	Nelson
Page	Pag
Osborne	Osborn
NULL	NULL
Sawyer	Sawy
H. Bender	H. Bend
Steadman	Steadman

RTRIM entfernt „e“ aus „Page“, obwohl das erste Zeichen im Argument *trim_set* „r“ lautet. Das liegt daran, dass RTRIM Zeichen um Zeichen nach dem Zeichensatz absucht, den Sie im Argument *trim_set* angegeben haben. Falls das letzte Zeichen im String dem ersten Zeichen in *trim_set* entspricht, wird es von RTRIM entfernt. Wenn für das letzte Zeichen im String keine Übereinstimmung gefunden wird, vergleicht RTRIM das zweite Zeichen in *trim_set*. Entspricht nun das vorletzte Zeichen im String dem zweiten Zeichen in *trim_set*, wird es entfernt usw. Wenn das Zeichen im String nicht mit *trim_set* übereinstimmt, gibt RTRIM den String zurück und fährt mit der Auswertung der nächsten Zeile fort.

In vorherigen Beispiel stimmt das letzte Zeichen in „Nelson“ mit keinem Zeichen in *trim_set* überein, sodass RTRIM den String zurückgibt und die nächste Zeile auswertet.

Tipps für die Arbeit mit RTRIM

Verwenden Sie RTRIM und LTRIM mit || oder CONCAT zum Entfernen von vor- und nachgestellten Leerzeichen nach dem Verketten von zwei Strings.

Sie können auch mehrere Zeichensätze entfernen, indem Sie RTRIM verschachteln. Beispiel: Zum Entfernen von nachgestellten Leerzeichen und des Buchstabens „t“ am Ende aller Strings in einer Spalte von Namen können Sie folgende Funktion formulieren:

```
RTRIM( RTRIM( NAMES ), 't' )
```

SETCOUNTVARIABLE

Zählt die Zeilen, die von der Funktion ausgewertet werden, und erhöht den aktuellen Wert einer Mapping-Variable ausgehend von der Anzahl. Erhöht für jede Zeile, die zum Einfügen markiert ist, den aktuellen Wert um eins. Verringert für jede Zeile, die zum Löschen markiert ist, den aktuellen Wert um eins. Für jede Zeile,

die zum Aktualisieren oder Ablehnen markiert ist, wird der aktuelle Wert beibehalten. Gibt den neuen aktuellen Wert zurück.

Am Ende einer erfolgreichen Sitzung speichert Data Integration Service den letzten aktuellen Wert im Repository. In Sitzungen mit mehreren Partitionen generiert Data Integration Service einen eigenen aktuellen Wert für jede Partition. Am Ende der Sitzung wird die Gesamtzahl für alle Partitionen bestimmt und im Repository gespeichert. Sofern er nicht überschrieben wird, dient der gespeicherte Wert als Startwert der Variable für die nächste Ausführung dieser Sitzung.

Verwenden Sie SETCOUNTVARIABLE nur einmal pro Mapping-Variable in einer Pipeline. Data Integration Service verarbeitet Variablenfunktionen in der Reihenfolge, in der sie angetroffen werden. Diese Reihenfolge Data Integration Serviceim Mapping kann von Sitzung zu Sitzung unterschiedlich ausfallen. Dies kann zu inkonsistenten Ergebnissen führen, wenn Sie die gleiche Variablenfunktion in einem Mapping mehrfach verwenden.

Verwenden Sie SETCOUNTVARIABLE in Mapping-Variablen mit dem Zähler-Aggregationstyp. Verwenden Sie SETCOUNTVARIABLE in folgenden Umwandlungen:

- Ausdruck
- Filter
- Router
- Update-Strategie

Data Integration Service speichert den endgültigen Wert von Mapping-Variablen nicht im Repository, wenn eine der folgenden Situationen eintritt:

- Die Sitzung kann nicht abgeschlossen werden.
- Die Sitzung ist für eine Testladung konfiguriert.
- Die Sitzung ist eine Debug-Sitzung.
- Die Sitzung wird im Debug-Modus ausgeführt, bei dem die Sitzungsausgabe verworfen wird.

Syntax

```
SETCOUNTVARIABLE( $$Variable )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
$$*Variable*	Erforderlich	Name der Mapping-Variable, die festgelegt werden soll. Verwenden Sie Mapping-Variablen mit dem Zähler-Aggregationstyp.

Rückgabewert

Der aktuelle Wert der Variable.

Beispiel

Angenommen, in Ihrem Mapping wird eine sich schrittweise ändernde Dimensionstabelle mit Händlerdaten aktualisiert. Der folgende Ausdruck zählt mithilfe der Mapping-Variable $$CurrentDistributors die Anzahl der aktuellen Händler und gibt den aktuellen Wert im Port CUR_DIST zurück. Der Zähler wird für jede eingefügte Zeile erhöht und für jede gelöschte Zeile verringert und bleibt für alle aktualisierten oder abgelehnten Zeilen

unverändert. Der Anfangswert von $$CurrentDistributors aus der vorherigen Ausführung der Sitzung beträgt 23.

```
SETCOUNTVARIABLE ($$CurrentDistributors)
```

(row marked for...)	DIST_ID	DISTRIBUTOR	CUR_DIST
(update)	000015	MSD Inc.	23
(insert)	000024	Darkroom Co.	24
(insert)	000025	Howard's Supply	25
(update)	000003	JNR Ltd.	25
(delete)	000024	Darkroom Co.	24
(insert)	000026	Supply.com	25

Am Ende der Sitzung speichert Data Integration Service den letzten aktuellen Wert von $$CurrentDistributors im Repository: in diesem Fall 25. Bei der nächsten Ausführung der Sitzung wertet Integration Service den Anfangswert für $$CurrentDistributors mit 25 aus.

Data Integration Service speichert für Sitzungen mit mehreren Partitionen den gleichen Wert für $ $CurrentDistributors im Repository wie für Sitzungen mit einer einzigen Partition.

SET_DATE_PART

Legt für einen Teil eines Datum/Zeit-Werts einen angegebenen Wert fest. Mit SET_DATE_PART können Sie folgende Teile eines Datums ändern:

- **Jahr.** Ändern Sie das Jahr, indem Sie für das Argument *value* eine positive Ganzzahl eingeben. Verwenden Sie den gewünschten Formatstring (Y, YY, YYY oder YYYY), um das Jahr festzulegen. Beispiel: Der folgende Ausdruck ändert die Jahreskomponente in allen Datumsangaben im Port SHIP_DATE zu 2001:

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

- **Monat.** Ändern Sie den Monat, indem Sie für das Argument *value* eine positive Ganzzahl zwischen 1 und 12 (Januar = 1, Dezember = 12) eingeben. Verwenden Sie den gewünschten Formatstring (MM, MON, MONTH), um den Monat festzulegen. Beispiel: Der folgende Ausdruck ändert die Monatskomponente in allen Datumsangaben im Port SHIP_DATE zu Oktober:

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

- **Tag.** Ändern Sie den Tag, indem Sie für das Argument *value* eine positive Ganzzahl zwischen 1 und 31 (außer für die Monate, die weniger als 31 Tage haben: Februar, April, Juni, September und November) eingeben. Verwenden Sie den gewünschten Formatstring (D, DD, DDD, DY und DAY), um den Tag festzulegen. Beispiel: Der folgende Ausdruck ändert die Tageskomponente in allen Datumsangaben im Port SHIP_DATE zu 10:

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

- **Stunde.** Ändern Sie die Stunde, indem Sie für das Argument *value* eine positive Ganzzahl zwischen 0 und 24 (0 = 12AM, 12 = 12PM, 24 = 12AM) eingeben. Verwenden Sie den gewünschten Formatstring (HH, HH12, HH24), um die Stunde festzulegen. Beispiel: Der folgende Ausdruck ändert die Stundenkomponente in allen Datumsangaben im Port SHIP_DATE zu 14:00:00 (bzw. 2:00:00PM):

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

- **Minute.** Ändern Sie die Minuten, indem Sie für das Argument *value* eine positive Ganzzahl zwischen 0 und 59 eingeben. Verwenden Sie zum Festlegen der Minuten den Formatstring MI. Beispiel: Der folgende Ausdruck ändert die Minutenkomponente in allen Datumsangaben im Port SHIP_DATE zu 25:

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

- **Sekunden.** Ändern Sie die Sekunden, indem Sie für das Argument *value* eine positive Ganzzahl zwischen 0 und 59 eingeben. Verwenden Sie zum Festlegen der Sekunden den Formatstring SS. Beispiel: Der folgende Ausdruck ändert die Sekundenkomponente in allen Datumsangaben im Port SHIP_DATE zu 59:

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

- **Millisekunden**. Ändern Sie die Millisekunden, indem Sie für das Argument *value* eine positive Ganzzahl zwischen 0 und 999 eingeben. Verwenden Sie zum Festlegen der Millisekunden den Formatstring MS. Beispiel: Der folgende Ausdruck ändert die Millisekundenkomponente in allen Datumsangaben im Port SHIP_DATE zu 125:

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

- **Mikrosekunden.** Ändern Sie die Mikrosekunden, indem Sie für das Argument *value* eine positive Ganzzahl zwischen 1000 und 999999 eingeben. Verwenden Sie zum Festlegen der Mikrosekunden den Formatstring US. Beispiel: Der folgende Ausdruck ändert die Mikrosekundenkomponente in allen Datumsangaben im Port SHIP_DATE zu 12555:

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

- **Nanosekunden.** Ändern Sie die Nanosekunden, indem Sie für das Argument *value* eine positive Ganzzahl zwischen 1000000 und 999999999 eingeben. Verwenden Sie zum Festlegen der Nanosekunden den Formatstring NS. Beispiel: Der folgende Ausdruck ändert die Nanosekundenkomponente in allen Datumsangaben im Port SHIP_DATE zu `12555555`:

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

Syntax

```
SET_DATE_PART( date, format, value )
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
Datum	Erforderlich	Datum/Zeit-Datentyp. Das Datum, das geändert werden soll. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.
format	Erforderlich	Formatstring, der festlegt, welcher Teil des Datums geändert wird. Bei Formatstrings muss nicht auf Groß-/Kleinschreibung geachtet werden.
value	Erforderlich	Ein positiver Ganzzahlenwert, der dem angegebenen Teil des Datums zugewiesen wird. Die Ganzzahl muss ein gültiger Wert für die zu ändernde Datumskomponente sein. Wenn Sie einen ungeeigneten Wert wie den 30. Februar eingeben, schlägt die Sitzung fehl.

Rückgabewert

Datum im selben Format wie das Ausgangsdatum, bei dem der angegebene Teil geändert wurde.

NULL, falls ein an die Funktion übergebener Wert NULL ist.

Beispiele

Die folgenden Ausdrücke ändern die Stundenkomponente aller Datumsangaben im Port DATE_PROMISED zu 4PM:

```
SET_DATE_PART( DATE_PROMISED, 'HH', 16 )
SET_DATE_PART( DATE_PROMISED, 'HH12', 16 )
SET_DATE_PART( DATE_PROMISED, 'HH24', 16 )
```

DATE_PROMISED	RETURN VALUE
Jan 1 1997 12:15:56AM	Jan 1 1997 4:15:56PM
Feb 13 1997 2:30:01AM	Feb 13 1997 4:30:01PM
Mar 31 1997 5:10:15PM	Mar 31 1997 4:10:15PM
Dec 12 1997 8:07:33AM	Dec 12 1997 4:07:33PM
NULL	NULL

Die folgenden Ausdrücke ändern die Monatskomponente der Datumsangaben im Port DATE_PROMISED zu Juni: Data Integration Service zeigt einen Fehler an, wenn Sie versuchen, ein nicht existentes Datum zu erstellen, etwa beim Ändern von 31. März zu 31. Juni:

```
SET_DATE_PART( DATE_PROMISED, 'MM', 6 )
SET_DATE_PART( DATE_PROMISED, 'MON', 6 )
SET_DATE_PART( DATE_PROMISED, 'MONTH', 6 )
```

DATE_PROMISED	RETURN VALUE
Jan 1 1997 12:15:56AM	Jun 1 1997 12:15:56AM
Feb 13 1997 2:30:01AM	Jun 13 1997 2:30:01AM
Mar 31 1997 5:10:15PM	*Error. Integration Service doesn't write row.*
Dec 12 1997 8:07:33AM	Jun 12 1997 8:07:33AM
NULL	NULL

Die folgenden Ausdrücke ändern die Jahreskomponente der Datumsangaben im Port DATE_PROMISED zu 2000:

```
SET_DATE_PART( DATE_PROMISED, 'Y', 2000 )
SET_DATE_PART( DATE_PROMISED, 'YY', 2000 )
SET_DATE_PART( DATE_PROMISED, 'YYY', 2000 )
SET_DATE_PART( DATE_PROMISED, 'YYYY', 2000 )
```

DATE_PROMISED	RETURN VALUE
Jan 1 1997 12:15:56AM	Jan 1 2000 12:15:56AM
Feb 13 1997 2:30:01AM	Feb 13 2000 2:30:01AM
Mar 31 1997 5:10:15PM	Mar 31 2000 5:10:15PM

DATE_PROMISED	RETURN VALUE
Dec 12 1997 8:07:33AM	Dec 12 2000 4:07:33PM
NULL	NULL

Tipp

Wenn Sie mehrere Teile eines Datums ändern möchten, können Sie mehrere SET_DATE_PART-Funktionen innerhalb des Arguments *date* verschachteln. Beispiel: Zum Ändern aller Datumsangaben im DATE_ENTERED zu 1. Juli 1998 können Sie folgenden Ausdruck formulieren:

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

SETMAXVARIABLE

Setzt den aktuellen Wert einer Mapping-Variable auf den höheren von zwei Werten, indem entweder der aktuelle Wert der Variable beibehalten oder der angegebene Wert verwendet wird. Gibt den neuen aktuellen Wert zurück. Die Funktion wird nur dann ausgeführt, wenn eine Zeile zum Einfügen markiert ist. SETMAXVARIABLE ignoriert alle anderen Zeilentypen. Der aktuelle Wert wird unverändert beibehalten.

Am Ende einer erfolgreichen Sitzung speichert Data Integration Service den endgültigen aktuellen Wert im Repository. In Sitzungen mit mehreren Partitionen generiert Data Integration Service einen eigenen aktuellen Wert für jede Partition. Am Ende der Sitzung wird der höchste aktuelle Wert aus allen Partitionen im Repository gespeichert. Sofern er nicht überschrieben wird, dient der gespeicherte Wert als Startwert der Variable für die nächste Ausführung dieser Sitzung.

Bei Verwendung mit einer String-Mapping-Variablen gibt SETMAXVARIABLE den höheren String zurück, basierend auf der für die Sitzung festgelegten Sortierreihenfolge.

Verwenden Sie SETMAXVARIABLE nur einmal pro Mapping-Variable in einer Pipeline. Data Integration Service verarbeitet Variablenfunktionen in der Reihenfolge, in der sie angetroffen werden. Diese Reihenfolge Data Integration Serviceim Mapping kann von Sitzung zu Sitzung unterschiedlich ausfallen. Dies kann zu inkonsistenten Ergebnissen führen, wenn Sie die gleiche Variablenfunktion in einem Mapping mehrfach verwenden.

Verwenden Sie SETMAXVARIABLE in Mapping-Variablen mit dem Max-Aggregationstyp. Verwenden Sie SETMAXVARIABLE in folgenden Umwandlungen:

- Ausdruck
- Filter
- Router
- Update-Strategie

Data Integration Service speichert den endgültigen Wert von Mapping-Variablen nicht im Repository, wenn eine der folgenden Situationen eintritt:

- Die Sitzung kann nicht abgeschlossen werden.
- Die Sitzung ist für eine Testladung konfiguriert.
- Die Sitzung ist eine Debug-Sitzung.
- Die Sitzung wird im Debug-Modus ausgeführt, bei dem die Sitzungsausgabe verworfen wird.

Syntax

```
SETMAXVARIABLE( $$Variable, value )
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
$$Variable	Erforderlich	Name der Mapping-Variable, die festgelegt werden soll. Verwenden Sie Mapping-Variablen mit dem Max-Aggregationstyp.
value	Erforderlich	Der Wert, den Data Integration Service mit dem aktuellen Wert der Variable vergleichen soll. Sie können jeden beliebigen Umwandlungsausdruck eingeben, dessen Auswertung einen mit der Variable kompatiblen Datentyp ergibt.

Rückgabewert

Der höhere von zwei Werten: der aktuelle Wert der Variable oder der angegebene. Der Rückgabewert ist der neue aktuelle Wert der Variable.

Wenn *value* NULL ist, gibt Data Integration Service den aktuellen Wert von *$$Variable* zurück.

Beispiele

Der folgende Ausdruck vergleicht die Anzahl der in jeder Transaktion gekauften Artikel mit der Mapping-Variable $$MaxItems. Er setzt $$MaxItems auf den höheren von zwei Werten und gibt die höchste Anzahl der in einer Einzeltransaktion gekauften Artikel an den Port MAX_ITEMS zurück. Der Anfangswert von $ $MaxItems aus der vorherigen Ausführung der Sitzung beträgt 22.

```
SETMAXVARIABLE ($$MAXITEMS, ITEMS)
```

TRANSACTION	ITEMS	MAX_ITEMS
0100002	12	22
0100003	5	22
0100004	18	22
0100005	35	35
0100006	5	35
0100007	14	35

Am Ende der Sitzung speichert Data Integration Service den maximalen aktuellen Wert von $$MaxItems im Repository: in diesem Fall 35. Bei der nächsten Ausführung der Sitzung wertet Data Integration Service den Anfangswert für $$MaxItems mit 35 aus.

Bei Sitzungen mit drei Partitionen wertet Data Integration Service $$MaxItems für jede Partition aus. Anschließend wird der höchste Wert im Repository gespeichert. Beispiel: Der letzte ermittelte Wert für $ $MaxItems in jeder Partition lautet wie folgt:

Partition	Final Current Value for $$MaxItems
Partition 1	35
Partition 2	23
Partition 3	22

SETMINVARIABLE

Setzt den aktuellen Wert einer Mapping-Variable auf den kleineren von zwei Werten, indem entweder der aktuelle Wert der Variable beibehalten oder der angegebene Wert verwendet wird. Gibt den neuen aktuellen Wert zurück. Die Funktion SETMINVARIABLE wird nur dann ausgeführt, wenn eine Zeile zum Einfügen markiert ist. SETMINVARIABLE ignoriert alle anderen Zeilentypen. Der aktuelle Wert wird unverändert beibehalten.

Am Ende einer erfolgreichen Sitzung speichert Data Integration Service den endgültigen aktuellen Wert im Repository. In Sitzungen mit mehreren Partitionen generiert Data Integration Service einen eigenen aktuellen Wert für jede Partition. Am Ende der Sitzung wird der niedrigste aktuelle Wert aus allen Partitionen im Repository gespeichert. Sofern er nicht überschrieben wird, dient der gespeicherte Wert als Startwert der Variable für die nächste Ausführung dieser Sitzung.

Bei Verwendung mit einer String-Mapping-Variablen gibt SETMINVARIABLE den niedrigeren String zurück, basierend auf der für die Sitzung festgelegten Sortierreihenfolge.

Verwenden Sie SETMINVARIABLE nur einmal pro Mapping-Variable in einer Pipeline. Data Integration Service verarbeitet Variablenfunktionen in der Reihenfolge, in der sie angetroffen werden. Diese Reihenfolge Data Integration Serviceim Mapping kann von Sitzung zu Sitzung unterschiedlich ausfallen. Dies kann zu inkonsistenten Ergebnissen führen, wenn Sie die gleiche Variablenfunktion in einem Mapping mehrfach verwenden.

Verwenden Sie SETMINVARIABLE in Mapping-Variablen mit dem Min-Aggregationstyp. Verwenden Sie SETMINVARIABLE in folgenden Umwandlungen:

- Ausdruck
- Filter
- Router
- Update-Strategie

Data Integration Service speichert den endgültigen Wert von Mapping-Variablen nicht im Repository, wenn eine der folgenden Situationen eintritt:

- Die Sitzung kann nicht abgeschlossen werden.
- Die Sitzung ist für eine Testladung konfiguriert.
- Die Sitzung ist eine Debug-Sitzung.
- Die Sitzung wird im Debug-Modus ausgeführt, bei dem die Sitzungsausgabe verworfen wird.

Syntax

```
SETMINVARIABLE( $$Variable, value )
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
$$Variable	Erforderlich	Name der Mapping-Variable, die festgelegt werden soll. Verwenden Sie SETMINVARIABLE in Mapping-Variablen mit dem Min-Aggregationstyp.
value	Erforderlich	Der Wert, den Data Integration Service mit dem aktuellen Wert der Variable vergleichen soll. Sie können jeden beliebigen Umwandlungsausdruck eingeben, dessen Auswertung einen mit der Variable kompatiblen Datentyp ergibt.

Rückgabewert

Der niedrigere von zwei Werten: der aktuelle Wert der Variable oder der angegebene. Der Rückgabewert ist der neue aktuelle Wert der Variable.

Wenn *value* NULL ist, gibt Data Integration Service den aktuellen Wert von *$$Variable* zurück.

Beispiel

Der folgende Ausdruck vergleicht den Preis eines Artikels mit der Mapping-Variable $$MinPrice. Er setzt $ $MinPrice auf den niedrigeren von zwei Werten und gibt den niedrigsten Artikelpreis an den Port MIN_PRICE zurück. Der Anfangswert von $$MinPrice aus der vorherigen Ausführung der Sitzung beträgt 22.50.

```
SETMINVARIABLE ($$MinPrice, PRICE)
```

DATE	PRICE	MIN_PRICE
05/01/2000 09:00:00	23.50	22.50
05/01/2000 10:00:00	27.00	22.50
05/01/2000 11:00:00	26.75	22.50
05/01/2000 12:00:00	25.25	22.50
05/01/2000 13:00:00	22.00	22.00
05/01/2000 14:00:00	22.75	22.00
05/01/2000 15:00:00	23.00	22.00
05/01/2000 16:00:00	24.25	22.00
05/01/2000 17:00:00	24.00	22.00

Am Ende der Sitzung speichert Data Integration Service den niedrigsten aktuellen Wert von $$MinPrice im Repository: in diesem Fall 22.00. Bei der nächsten Ausführung der Sitzung wertet Data Integration Service den Anfangswert für $$MinPrice mit 22.00 aus.

Bei Sitzungen mit drei Partitionen wertet Data Integration Service $$MinPrice für jede Partition aus. Anschließend wird der niedrigste Wert im Repository gespeichert. Beispiel: Der letzte ermittelte Wert für $ $MinPrice in jeder Partition lautet wie folgt:

Partition	Final Current Value for $$MinPrice
Partition 1	22.00
Partition 2	22.50
Partition 3	22.50

SETVARIABLE

Setzt den aktuellen Wert einer Mapping-Variable auf einen angegebenen Wert. Gibt den angegebenen Wert zurück. Die Funktion SETVARIABLE wird nur ausgeführt, wenn eine Zeile zum Einfügen oder Aktualisieren markiert ist. SETVARIABLE ignoriert alle anderen Zeilentypen. Der aktuelle Wert wird unverändert beibehalten.

Am Ende einer erfolgreichen Sitzung vergleicht Data Integration Service den endgültigen aktuellen Wert der Variable mit ihrem Startwert. Je nach Aggregattyp der Variable wird der endgültige aktuelle Wert im Repository gespeichert. Sofern er nicht überschrieben wird, dient der gespeicherte Wert als Startwert der Variable für die nächste Ausführung dieser Sitzung.

Verwenden Sie SETVARIABLE nur einmal pro Mapping-Variable in einer Pipeline. Data Integration Service verarbeitet Variablenfunktionen in der Reihenfolge, in der sie angetroffen werden. Diese Reihenfolge Data Integration Serviceim Mapping kann von Sitzung zu Sitzung unterschiedlich ausfallen. Dies kann zu inkonsistenten Ergebnissen führen, wenn Sie die gleiche Variablenfunktion in einem Mapping mehrfach verwenden.

Verwenden Sie SETVARIABLE in folgenden Umwandlungen:

- Ausdruck
- Filter
- Router
- Update-Strategie

Data Integration Service speichert den endgültigen Wert von Mapping-Variablen nicht im Repository, wenn eine der folgenden Situationen eintritt:

- Die Sitzung kann nicht abgeschlossen werden.
- Die Sitzung ist für eine Testladung konfiguriert.
- Die Sitzung ist eine Debug-Sitzung.
- Die Sitzung wird im Debug-Modus ausgeführt, bei dem die Sitzungsausgabe verworfen wird.

Syntax

```
SETVARIABLE( $$Variable, value )
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
$$*Variable*	Erforderlich	Name der Mapping-Variable, die festgelegt werden soll. Verwenden Sie SETVARIABLE in Mapping-Variablen mit dem Max/Min-Aggregationstyp.
value	Erforderlich	Der Wert, auf den der aktuelle Wert der Variable gesetzt werden soll. Sie können jeden beliebigen Umwandlungsausdruck eingeben, dessen Auswertung einen mit der Variable kompatiblen Datentyp ergibt.

Rückgabewert

Der aktuelle Wert der Variable.

Wenn *value* NULL ist, gibt Data Integration Service den aktuellen Wert von *$$Variable* zurück.

Beispiele

Der folgende Ausdruck setzt die Mapping-Variable $$Time auf das Systemdatum des Zeitpunkts, an dem Data Integration Service die Zeile auswertet, und gibt das Systemdatum an den Port SET_$$TIME zurück:

```
SETVARIABLE ($$Time, SYSDATE)
```

TRANSACTION	TOTAL	SET_$$TIME
0100002	534.23	10/10/2000 01:34:33
0100003	699.01	10/10/2000 01:34:34
0100004	97.50	10/10/2000 01:34:35
0100005	116.43	10/10/2000 01:34:36
0100006	323.95	10/10/2000 01:34:37

Am Ende der Sitzung speichert Data Integration Service den letzten aktuellen Wert von $$Time im Repository: in diesem Fall 10/10/2000 01:34:37. Bei der nächsten Ausführung der Sitzung wertet Data Integration Service alle Verweise auf $$Time mit 10/10/2000 01:34:37 aus.

Der folgende Ausdruck setzt die Mapping-Variable $$Timestamp auf den Zeitstempel der Zeile und gibt ihn dem Port SET_$$TIMESTAMP zurück:

```
SETVARIABLE ($$Time, TIMESTAMP)
```

TRANSACTION	TIMESTAMP	TOTAL	SET_$$TIMESTAMP
0100002	10/01/2000 12:01:01	534.23	10/01/2000 12:01:01
0100003	10/01/2000 12:10:22	699.01	10/01/2000 12:10:22
0100004	10/01/2000 12:16:45	97.50	10/01/2000 12:16:45
0100005	10/01/2000 12:23:10	116.43	10/01/2000 12:23:10
0100006	10/01/2000 12:40:31	323.95	10/01/2000 12:40:31

Am Ende der Sitzung speichert Data Integration Service den letzten aktuellen Wert von $$Timestamp im Repository: in diesem Fall 10/01/2000 12:40:31.

Bei der nächsten Ausführung der Sitzung wertet Data Integration Service den Anfangswert für $$Timestamp mit 10/01/2000 12:40:31 aus.

SIGN

Gibt zurück, ob ein numerischer Wert positiv, negativ oder 0 ist.

Syntax

```
SIGN( numeric_value )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
numeric_value	Erforderlich	Numerischer Wert. Übergibt die Werte, die ausgewertet werden sollen. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

-1 für negative Werte.

0 für 0.

1 für positive Werte.

NULL, wenn NULL.

Beispiel

Der folgende Ausdruck ermittelt, ob der Port SALES negative Werte enthält:

```
SIGN( SALES )
```

SALES	RETURN VALUE
100	1
-25.99	-1
0	0
NULL	NULL

SIN

Gibt den Sinus eines numerischen Werts zurück (ausgedrückt in Radianten).

Syntax

```
SIN( numeric_value )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
numeric_value	Erforderlich	Numerischer Datentyp. Numerische Daten, ausgedrückt in Radianten (Grad multipliziert mit Pi durch 180). Übergibt die Werte, für die der Sinus berechnet werden soll. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben. Sie können auch mithilfe von Operatoren numerische Werte in Radianten konvertieren oder mathematische Berechnungen innerhalb einer SIN-Berechnung durchführen.

Rückgabewert

Double-Wert

NULL, falls ein an die Funktion übergebener Wert NULL ist.

Beispiel

Der folgende Ausdruck wandelt die Werte im Port DEGREES in Radianten um und berechnet anschließend den Sinus jedes einzelnen Radianten:

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

DEGREES	RETURN VALUE
0	0
90	1
70	0.939692620785936
30	0.50000000000003
5	0.0871557427476639
89	0.999847695156393
NULL	NULL

Sie können anhand der an SIN übergebenen Werte mathematische Berechnungen durchführen, bevor die Funktion den Sinus berechnet. Beispiel:

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

SINH

Gibt den hyperbolischen Sinus des numerischen Werts zurück.

Syntax

```
SINH( numeric_value )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
numeric_value	Erforderlich	Numerischer Datentyp. Numerische Daten, ausgedrückt in Radianten (Grad multipliziert mit Pi durch 180). Übergibt die Werte, für die der hyperbolische Sinus berechnet werden soll. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

Double-Wert

NULL, falls ein an die Funktion übergebener Wert NULL ist.

Beispiel

Der folgende Ausdruck gibt den hyperbolischen Sinus für die Werte im Port ANGLES zurück:

```
SINH( ANGLES )
```

ANGLES	RETURN VALUE
1.0	1.1752011936438
2.897	9.03225804884884
3.66	19.4178051793031
5.45	116.376934801486
0	0.0
0.345	0.35188478309993
NULL	NULL

Tipp

Sie können anhand der an SINH übergebenen Werte mathematische Berechnungen durchführen, bevor die Funktion den hyperbolischen Sinus berechnet. Beispiel:

```
SINH( MEASURES.ARCS / 180 )
```

SOUNDEX

Kodiert einen Stringwert in einen String aus vier Zeichen.

SOUNDEX ist für die Zeichen des englischen Alphabets (A-Z) ausgelegt. Die Funktion fügt das erste Zeichen des Eingabestrings als erstes Zeichen in den Rückgabewert ein und kodiert die übrigen eindeutigen Konsonanten als Nummern.

Die Kodierung durch SOUNDEX erfolgt nach den folgenden Regeln:

- Das erste Zeichen im Argument *string* ist das erste Zeichen im Rückgabewert, kodiert in Großbuchstaben. Beispiel: Sowohl SOUNDEX('John') als auch SOUNDEX('john') geben „J500" zurück.
- Die ersten drei eindeutigen Konsonanten nach dem ersten Zeichen in *string* werden kodiert. Der Rest wird ignoriert. Beispiel: Sowohl SOUNDEX('JohnRB') als auch SOUNDEX('JohnRBCD') geben „J561" zurück.
- Ähnlich klingenden Konsonanten wird derselbe Code zugewiesen.

  Die folgende Tabelle beschreibt die Kodierungsrichtlinien für Konsonanten in SOUNDEX:

  Tabelle 2. SOUNDEX-Kodierungsrichtlinien für Konsonanten

Code	Konsonant
1	B, P, F, V
2	C, S, G, J, K, Q, X, Z
3	D, T
4	L
5	M, N
6	R

- Die Zeichen A, E, I, O, U, H und W werden übersprungen, es sei denn, es handelt sich dabei um das erste Zeichen in *string*. Beispiel: SOUNDEX('A123') gibt „A000" zurück, SOUNDEX('MAeiouhwC') gibt „M000" zurück.
- Wenn *string* weniger als vier Zeichen ergibt, füllt SOUNDEX den Ergebnisstring mit Nullen auf. Beispiel: SOUNDEX('J') gibt „J000" zurück.
- Wenn *string* eine Reihe aufeinanderfolgender Konsonanten enthält, denen in "SOUNDEX" auf Seite 192 derselbe Code zugewiesen ist, kodiert SOUNDEX den ersten dieser Konsonanten und überspringt die restlichen. Beispiel: SOUNDEX('AbbpdMN') gibt „A135" zurück.
- Zahlen in *string* werden übersprungen. Beispiel: Sowohl SOUNDEX('Joh12n') als auch SOUNDEX('1John') geben „J500" zurück.
- Die Rückgabe lautet NULL, wenn *string* NULL ist oder keine Zeichen in *string* Buchstaben aus dem englischen Alphabet sind.

Syntax

```
SOUNDEX( string )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
string	Erforderlich	Zeichenstring. Übergibt den Stringwert, der kodiert werden soll. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

String.

NULL, wenn eine der folgenden Bedingungen eintritt:

- Der an die Funktion übergebene Wert ist NULL.
- Kein Zeichen in *string* ist ein Buchstabe des englischen Alphabets.
- Das Argument *string* ist leer.

Beispiel

Der folgende Ausdruck kodiert die Werte im Port EMPLOYEE_NAME:

```
SOUNDEX( EMPLOYEE_NAME )
```

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

SQL_LIKE

Gibt zurück, ob ein Wert mit einem regulären Ausdrucksmuster übereinstimmt. Hiermit können Sie Datumsmuster, wie z. B. IDs, Telefonnummern, Postleitzahlen und Namen von Bundesländern validieren.

Syntax

```
SQL_LIKE(subject, pattern, escape character)
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
Thema	Erforderlich	Datentyp „Zeichenfolge“. Übergibt den Wert, der mit dem regulären Ausdruck abgeglichen werden soll. Schließen Sie den Wert in einfache Anführungszeichen ein.
Muster	Erforderlich	Datentyp „Zeichenfolge“. Regulärer Ausdruck, der abgeglichen werden soll. Schließen Sie das Muster in einfache Anführungszeichen ein.
Escape-Zeichen	Optional	Datentyp „Zeichenfolge“. Die Funktion SQL_LIKE unterstützt das Prozentzeichen (%) und den Unterstrich (_) als Escape-Zeichen. Schließen Sie das Escape-Zeichen in einfache Anführungszeichen ein.

Rückgabewert

TRUE, wenn die Daten mit dem Muster übereinstimmen.

FALSE, wenn die Daten nicht mit dem Muster übereinstimmen.

NULL, wenn die Eingabe ein Nullwert ist oder das Muster NULL ergibt.

Beispiel

Sie verwenden SQL_LIKE unter Umständen in einem Ausdruck, um nach Namen zu suchen, die einem Muster entsprechen. Der folgende Ausdruck gleicht beispielsweise Namen mit dem Muster „A_#%“ mit dem Escape-Zeichen '#' ab:

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

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

SQRT

Gibt die Quadratwurzel eines positiven numerischen Werts zurück.

Syntax

```
SQRT( numeric_value )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
numeric_value	Erforderlich	Positiver numerischer Wert. Übergibt die Werte, für die die Quadratwurzel berechnet werden soll. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

Double-Wert

NULL, falls ein an die Funktion übergebener Wert NULL ist.

Beispiel

Der folgende Ausdruck gibt die Quadratwurzel der Werte im Port NUMBERS zurück:

```
SQRT( NUMBERS )
```

NUMBERS	RETURN VALUE
100	10
-100	*Error. Data Integration Service does not write row.*
NULL	NULL
60.54	7.78074546557076

Der Wert -100 führt zu einem Fehler, da die Funktion SQRT nur positive numerische Werte auswertet. Wenn Sie einen negativen Wert oder einen Zeichenwert übergeben, zeigt Data Integration Service einen Fehler bei der Umwandlungsauswertung an und schreibt die Zeile nicht.

Sie können anhand der an SQRT übergebenen Werte mathematische Berechnungen durchführen, bevor die Funktion die Quadratwurzel berechnet.

STDDEV

Gibt die Standardabweichung der numerischen Werte zurück, die an die Funktion übergeben werden. STDDEV dient zur Analyse statistischer Daten. Sie können eine weitere Aggregatfunktion in STDDEV schachteln, und die verschachtelte Funktion muss einen numerischen Datentyp zurückgeben.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
numeric_value	Erforderlich	Numerische Datentypen. Diese Funktion übergibt die Werte, für die eine Standardabweichung berechnet werden soll, oder die Ergebnisse einer Funktion. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben. Mit Operatoren können Sie den Durchschnitt der Werte in verschiedenen Ports ermitteln.
filter_condition	Optional	Begrenzt die Zeilen in der Suche. Die Filterbedingung muss ein numerischer Wert sein oder mit TRUE, FALSE oder NULL ausgewertet werden. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

Numerischer Wert.

NULL, wenn alle übergebenen Werte NULL sind oder keine Zeilen ausgewählt wurden (z. B. wenn die Filterbedingung in allen Zeilen FALSE oder NULL ergibt).

Hinweis: Wenn der Rückgabewert eine Dezimalmal mit Präzision höher als 15 ist, können Sie „Hohe Präzision“ aktivieren, um Dezimalgenauigkeit bis zu 28 Stellen zu gewährleisten.

Nullen

Wenn nur ein Wert NULL ist, wird er ignoriert. Wenn jedoch alle Werte NULL sind, gibt STDDEV NULL zurück.

Hinweis: Standardmäßig behandelt Data Integration Service Nullwerte in Aggregatfunktionen als NULL. Wenn Sie einen Port oder eine Gruppe mit ausschließlich Nullwerten übergeben, gibt die Funktion NULL zurück. Beim Konfigurieren von Data Integration Service können Sie jedoch entscheiden, wie mit Nullwerten in Aggregatfunktionen umgegangen werden soll. Sie können Nullwerte in Aggregatfunktionen als 0 oder als NULL verarbeiten.

Gruppieren nach

STDDEV gruppiert die Werte nach der Einstellung „Gruppieren nach Ports“, die Sie in der Umwandlung festlegen, und gibt pro Gruppe ein Ergebnis zurück.

Wenn „Gruppieren nach Ports“ nicht festgelegt wurde, behandelt STDDEV alle Zeilen als eine einzige Gruppe und gibt nur einen Wert zurück.

Beispiele

Der folgende Ausdruck berechnet die Standardabweichung aller Zeilen größer als 2000.00 USD im Port TOTAL_SALES:

```
STDDEV( SALES, SALES > 2000.00 )
```

SALES
2198.0
1010.90
2256.0
153.88
3001.0
NULL
8953.0

RETURN VALUE: 3254.60361129688

Die Werte 1010.90 und 153.88 werden bei der Berechnung nicht berücksichtigt, da für das Argument *filter_condition* Umsatzzahlen über 2.000 USD angegeben wurden.

Der folgende Ausdruck berechnet die Standardabweichung aller Zeilen im Port SALES:

```
STDDEV(SALES)
```

SALES
2198.0
2198.0
2198.0

```
SALES
```

```
2198.0
```

```
RETURN VALUE: 0
```

Der Rückgabewert ist 0, weil alle Zeilen die gleiche Zahl aufweisen (keine Standardabweichung vorhanden). Wenn es keine Standardabweichung gibt, lautet der Rückgabewert 0.

SUBSTR

Gibt einen Teil eines Strings zurück. SUBSTR startet am Anfang des Strings und zählt alle Zeichen einschließlich Leerzeichen.

Syntax

```
SUBSTR( string, start [,length] )
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
string	Erforderlich	Muss ein Zeichenstring sein. Übergibt die Strings, die durchsucht werden sollen. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben. Wenn Sie einen numerischen Wert übergeben, konvertiert ihn die Funktion in einen Zeichenstring.
start	Erforderlich	Muss eine Ganzzahl sein. Die Position im String, an der Sie zu zählen beginnen. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben. Wenn die Startposition eine positive Zahl ist, ermittelt sie SUBSTR durch Zählen vom Stringanfang. Wenn die Startposition eine negative Zahl ist, wird die vom Stringende aus ermittelt. Bei Startposition 0 beginnt die Suche mit dem ersten Zeichen im String.
length	Optional	Muss eine Ganzzahl größer als Null sein. Die Anzahl von Zeichen, die SUBSTR zurückgeben soll. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben. Wenn Sie das Argument „length“ nicht angeben, gibt SUBSTR alle Zeichen zwischen Startposition und Stringende zurück. Wenn Sie eine negative Ganzzahl oder 0 übergeben, gibt die Funktion einen leeren String zurück. Bei der Übergabe von Dezimalzahlen rundet die Funktion auf den nächsten Ganzzahlwert.

Rückgabewert

String.

Leerer String, wenn „length“ negativ oder 0 ist.

NULL, falls ein an die Funktion übergebener Wert NULL ist.

Beispiele

Die folgenden Ausdrücke geben die Ortsnetzkennzahl aus jeder Zeile des Ports PHONE zurück:

```
SUBSTR( PHONE, 0, 3 )
```

PHONE	RETURN VALUE
809-555-0269	809
357-687-6708	357
NULL	NULL

```
SUBSTR( PHONE, 1, 3 )
```

PHONE	RETURN VALUE
809-555-3915	809
357-687-6708	357
NULL	NULL

Die folgenden Ausdrücke geben die Telefonnummer ohne Ortsnetzkennzahl aus jeder Zeile des Ports PHONE zurück:

```
SUBSTR( PHONE, 5, 8 )
```

PHONE	RETURN VALUE
808-555-0269	555-0269
809-555-3915	555-3915
357-687-6708	687-6708
NULL	NULL

Sie können auch einen negativen Startwert übergeben, um die Telefonnummer aus jeder Zeile des Ports PHONE zurückzugeben. Der Ausdruck liest den Quellstring auch dann von links nach rechts, wenn er das Ergebnis des Arguments *length* zurückgibt:

```
SUBSTR( PHONE, -8, 3 )
```

PHONE	RETURN VALUE
808-555-0269	555
809-555-3915	555
357-687-6708	687
NULL	NULL

In den Argumenten *start* und *length* kann INSTR verschachtelt werden, um einen bestimmten String zu suchen und seine Position zurückzugeben.

Der folgende Ausdruck wertet einen String aus, beginnend beim Stringende. Der Ausdruck sucht das letzte (rechteste) Leerzeichen im String und gibt alle Zeichen davor zurück:

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

CUST_NAME	RETURN VALUE
PATRICIA JONES	PATRICIA
MARY ELLEN SHAH	MARY ELLEN

Der folgende Ausdruck entfernt das Zeichen # aus einem String:

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

Wenn das Argument *length* länger als der String ist, gibt SUBSTR alle Zeichen zwischen Startposition und Stringende zurück. Betrachten Sie das folgende Beispiel:

```
SUBSTR('abcd', 2, 8)
```

Der Rückgabewert lautet „bcd“. Vergleichen Sie dieses Ergebnis mit dem folgenden Beispiel:

```
SUBSTR('abcd', -2, 8)
```

Der Rückgabewert lautet „cd“.

SUM

Gibt die Summe aller Werte im ausgewählten Port zurück. Optional können Sie einen Filter anwenden, um die Anzahl der Zeilen zu beschränken, aus denen die Summe berechnet wird. Sie können eine weitere Aggregatfunktion in SUM schachteln, und die verschachtelte Funktion muss einen numerischen Datentyp zurückgeben.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
numeric_value	Erforderlich	Numerischer Datentyp. Übergibt die zu summierenden Werte. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben. Mithilfe von Operatoren können Sie Werte in verschiedenen Ports angeben.
filter_condition	Optional	Begrenzt die Zeilen in der Suche. Die Filterbedingung muss ein numerischer Wert sein oder mit TRUE, FALSE oder NULL ausgewertet werden. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

Numerischer Wert.

NULL, wenn alle übergebenen Werte NULL sind oder keine Zeilen ausgewählt wurden (z. B. wenn die Filterbedingung in allen Zeilen FALSE oder NULL ergibt).

Hinweis: Wenn der Rückgabewert eine Dezimalmal mit Präzision höher als 15 ist, können Sie „Hohe Präzision“ aktivieren, um Dezimalgenauigkeit bis zu 28 Stellen zu gewährleisten.

Nullen

Wenn ein einzelner Wert NULL ist, wird er ignoriert. Wenn jedoch alle vom Port übergebenen Werte NULL ergeben, gibt SUM NULL zurück.

Hinweis: Standardmäßig behandelt Data Integration Service Nullwerte in Aggregatfunktionen als NULL. Wenn Sie einen Port oder eine Gruppe mit ausschließlich Nullwerten übergeben, gibt die Funktion NULL zurück. Beim Konfigurieren von Data Integration Service können Sie jedoch entscheiden, wie mit Nullwerten in Aggregatfunktionen umgegangen werden soll. Sie können Nullwerte in Aggregatfunktionen als 0 oder als NULL verarbeiten.

Gruppieren nach

SUM gruppiert die Werte nach der Einstellung „Gruppieren nach Ports“, die Sie in der Umwandlung festlegen, und gibt pro Gruppe ein Ergebnis zurück.

Wenn „Gruppieren nach Ports“ nicht festgelegt wurde, behandelt SUM alle Zeilen als eine einzige Gruppe und gibt nur einen Wert zurück.

Beispiel

Der folgende Ausdruck gibt die Summe aller Werte größer als 2000 im Port SALES zurück:

```
SUM( SALES, SALES > 2000 )
```

SALES
2500.0
1900.0
1200.0
NULL
3458.0
4519.0

RETURN VALUE: 10477.0

Tipp

Sie können anhand der an SUM übergebenen Werte mathematische Berechnungen durchführen, bevor die Funktion die Summe berechnet. Beispiel:

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

SYSTIMESTAMP

Gibt das aktuelle Datum und die aktuelle Uhrzeit auf dem Knoten, auf dem sich Data Integration Service befindet, mit Nanosekunden-Präzision zurück. Die Präzision für die Anzeige von Datum und Uhrzeit hängt von der Plattform ab.

Der Rückgabewert der Funktion variiert je nach Konfiguration des Arguments:

- Wenn das Argument von SYSTIMESTAMP als Variable konfiguriert ist, wertet Data Integration Service die Funktion für jede Zeile in der Umwandlung aus.
- Wenn das Argument von SYSTIMESTAMP als Konstante konfiguriert ist, wertet Data Integration Service die Funktion einmal aus und behält den Wert für jede Zeile in der Umwandlung bei.

Syntax

```
SYSTIMESTAMP( [format] )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
format	Optional	Präzision, die im Zeitstempel angegeben werden soll. Sie können die Präzision auf Sekunden (SS), Millisekunden (MS), Mikrosekunden (US) oder Nanosekunden (NS) einstellen. Setzen Sie den Formatstring zwischen einfache Anführungszeichen. Bei Formatstrings muss nicht auf Groß-/Kleinschreibung geachtet werden. Beispiel: Zum Anzeigen von Datum und Uhrzeit mit Millisekunden-Präzision geben Sie folgende Syntax an: SYSTIMESTAMP('MS'). Die Standardpräzision ist Mikrosekunden (US).

Rückgabewert

Zeitstempel. Gibt Datum und Uhrzeit mit der angegebenen Präzision zurück.

Beispiele

Ihr Unternehmen betreibt einen Online-Bestelldienst und verarbeitet Echtzeitdaten. Mit der Funktion SYSTIMESTAMP können Sie für jede Transaktion in der Zieldatenbank einen Primärschlüssel generieren.

Erstellen Sie eine Ausdrucksumwandlung mit den folgenden Ports und Werten:

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

Während der Laufzeit generiert Data Integration Service die Systemzeit mit Mikrosekunden-Präzision für jede Zeile:

Customer_Name	Order_Qty	Transaction_Id
Vani Deed	14	07/06/2007 18:00:30.701015000
Kalia Crop	3	07/06/2007 18:00:30.701029000
Vani Deed	6	07/06/2007 18:00:30.701039000
Harry Spoon	32	07/06/2007 18:00:30.701048000

TAN

Gibt den Tangens eines numerischen Werts zurück (ausgedrückt in Radianten).

Syntax

```
TAN( numeric_value )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
numeric_value	Erforderlich	Numerischer Datentyp. Numerische Daten, ausgedrückt in Radianten (Grad multipliziert mit Pi durch 180). Übergibt die numerischen Werte, für die der Tangens berechnet werden soll. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

Double-Wert

NULL, falls ein an die Funktion übergebener Wert NULL ist.

Beispiel

Der folgende Ausdruck gibt den Tangens aller Werte im Port DEGREES zurück:

```
TAN( DEGREES * 3.14159 / 180 )
```

DEGREES	RETURN VALUE
70	2.74747741945531
50	1.19175359259435
30	0.577350269189672
5	0.0874886635259298
18	0.324919696232929
89	57.2899616310952
NULL	NULL

TANH

Gibt den hyperbolischen Tangens des numerischen Werts zurück, der an die Funktion übergeben wurde.

Syntax

```
TANH( numeric_value )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
numeric_value	Erforderlich	Numerischer Datentyp. Numerische Daten, ausgedrückt in Radianten (Grad multipliziert mit Pi durch 180). Übergibt die numerischen Werte, für die der hyperbolische Tangens berechnet werden soll. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

Double-Wert

NULL, falls ein an die Funktion übergebener Wert NULL ist.

Beispiel

Der folgende Ausdruck gibt den hyperbolischen Tangens der Werte im Port ANGLES zurück:

```
TANH( ANGLES )
```

ANGLES	RETURN VALUE
1.0	0.761594155955765
2.897	0.993926947790665
3.66	0.998676551914886
5.45	0.999963084213409
0	0.0
0.345	0.331933853503641
NULL	NULL

Tipp

Sie können anhand der an TANH übergebenen Werte mathematische Berechnungen durchführen, bevor die Funktion den hyperbolischen Tangens berechnet. Beispiel:

```
TANH( ARCS / 360 )
```

TO_BIGINT

Wandelt einen String oder numerischen Wert in einen Bigint-Wert um. Die Syntax von TO_BIGINT umfasst ein optionales Argument, mit dem Sie die Zahl auf die nächste Ganzzahl runden oder die Dezimalstellen abschneiden können. TO_BIGINT ignoriert vorangestellte Leerzeichen.

Syntax

```
TO_BIGINT( value [, flag] )
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
value	Erforderlich	String- oder numerischer Datentyp. Übergibt den gewünschten Wert, der in einen Bigint-Wert umgewandelt werden soll. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.
flag	Optional	Legt fest, ob die Dezimalstellen abgeschnitten oder gerundet werden. Das Flag muss ein Ganzzahl-Literal sein oder den Konstanten TRUE oder FALSE entsprechen. TO_BIGINT schneidet die Dezimalstellen ab, wenn für das Flag TRUE oder eine andere Zahl als 0 eingegeben wurde. TO_BIGINT rundet den Wert auf die nächste Ganzzahl, wenn das Flag FALSE oder 0 lautet bzw. nicht angegeben ist. Das Flag weist keine Standardeinstellung auf.

Rückgabewert

Bigint.

NULL, wenn der an die Funktion übergebene Wert NULL ist.

0, wenn der an die Funktion übergebene Wert alphanumerische Zeichen enthält.

Wenn der an die Funktion übergebene Wert Daten enthält, die für einen Bigint-Wert nicht gültig sind, markiert der Datenintegrationsdienst die Zeile als Fehlerzeile oder das Mapping schlägt fehl.

Beispiele

Die folgenden Ausdrücke verwenden Werte aus dem Port IN_TAX:

```
TO_BIGINT( IN_TAX, TRUE )
```

IN_TAX	RETURN VALUE
'7245176201123435.6789'	7245176201123435
'7245176201123435.2'	7245176201123435
'7245176201123435.2.48'	7245176201123435
NULL	NULL
'A12.3Grove'	0
'A12.3Grove'	*Error. Integration Service skips this row.*
' 176201123435.87'	176201123435
'-7245176201123435.2'	-7245176201123435
'-7245176201123435.23'	-7245176201123435

IN_TAX	RETURN VALUE
-9223372036854775806.9	-9223372036854775806
9223372036854775806.9	9223372036854775806

```
TO_BIGINT( IN_TAX )
```

IN_TAX	RETURN VALUE
'7245176201123435.6789'	7245176201123436
'7245176201123435.2'	7245176201123435
'7245176201123435.348'	7245176201123435
NULL	NULL
'A12.3Grove'	0
'A12.3Grove'	*Error. Integration Service skips this row.*
' 176201123435.87'	176201123436
'-7245176201123435.6789'	-7245176201123436
'-7245176201123435.23'	-7245176201123435
-9223372036854775806.9	-9223372036854775807
9223372036854775806.9	9223372036854775807

TO_CHAR (Datum)

Konvertiert Datumsangaben in Zeichenstrings. TO_CHAR wandelt auch numerische Werte in Strings um. Mithilfe der TO_CHAR-Formatstrings können Sie das Datum in jedes beliebige Format konvertieren.

TO_CHAR (date [,format]) wandelt einen Datentyp oder internen Wert vom Datentyp „Datum“, „Zeitstempel“, „Zeitstempel mit Zeitzone“ oder „Zeitstempel mit lokaler Zeitzone“ in einen Wert vom Datentyp „Zeichenfolge“ um, der durch die Formatzeichenfolge angegeben wird.

Syntax

```
TO_CHAR( date [,format] )
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
date	Erforderlich	Datum/Zeit-Datentyp. Übergibt die Datumswerte, die in Zeichenfolgen umgewandelt werden sollen. Sie können einen beliebigen gültigen Umwandlungsausdruck eingeben.
format	optional	Geben Sie eine gültige TO_CHAR-Formatzeichenfolge ein. Die Formatzeichenfolge definiert das Format des Rückgabewerts, nicht aber das Format der Werte im Date-Argument. Wenn Sie die Formatzeichenfolge auslassen, gibt die Funktion eine Zeichenfolge zurück, die auf dem in der Sitzung definierten Datumsformat basiert. Geben Sie eine gültige TO_CHAR-Formatzeichenfolge ein. Die Formatzeichenfolge definiert das Format des Rückgabewerts, nicht aber das Format der Werte im Date-Argument. Wenn Sie die Formatzeichenfolge auslassen, gibt die Funktion eine Zeichenfolge zurück, die auf dem in der Zuordnungskonfiguration festgelegten Datumsformat basiert.

Rückgabewert

Zeichenfolge.

NULL, falls ein an die Funktion übergebener Wert NULL ist.

Beispiele

Der folgende Ausdruck wandelt die Daten im Port DATE_PROMISED in Text im Format MON DD YYYY um:

```
TO_CHAR( DATE_PROMISED, 'MON DD YYYY' )
```

DATE_PROMISED	RETURN VALUE
Apr 1 1998 12:00:10AM	'Apr 01 1998'
Feb 22 1998 01:31:10PM	'Feb 22 1998'
Oct 24 1998 02:12:30PM	'Oct 24 1998'
NULL	NULL

Wenn Sie *format* nicht angeben, gibt TO_CHAR den String im Datumsformat der Sitzung zurück, standardmäßig MM/DD/YYYY HH24:MI:SS.US:

Wenn Sie *format* nicht angeben, gibt TO_CHAR den String im Datumsformat der Mapping-Konfiguration zurück, standardmäßig MM/DD/YYYY HH24:MI:SS.US:

```
TO_CHAR( DATE_PROMISED )
```

DATE_PROMISED	RETURN VALUE
Apr 1 1998 12:00:10AM	'04/01/1998 00:00:10.000000'
Feb 22 1998 01:31:10PM	'02/22/1998 13:31:10.000000'

DATE_PROMISED	RETURN VALUE
Oct 24 1998 02:12:30PM	'10/24/1998 14:12:30.000000'
NULL	NULL

Die folgenden Ausdrücke geben den Wochentag aller Datumsangaben in einem Port zurück:

```
TO_CHAR( DATE_PROMISED, 'D' )
```

DATE_PROMISED	RETURN VALUE
04-01-1997 12:00:10AM	'3'
02-22-1997 01:31:10PM	'7'
10-24-1997 02:12:30PM	'6'
NULL	NULL

```
TO_CHAR( DATE_PROMISED, 'DAY' )
```

DATE_PROMISED	RETURN VALUE
04-01-1997 12:00:10AM	'Tuesday'
02-22-1997 01:31:10PM	'Saturday'
10-24-1997 02:12:30PM	'Friday'
NULL	NULL

Der folgende Ausdruck gibt den Tag des Monats aller Datumsangaben in einem Port zurück:

```
TO_CHAR( DATE_PROMISED, 'DD' )
```

DATE_PROMISED	RETURN VALUE
04-01-1997 12:00:10AM	'01'
02-22-1997 01:31:10PM	'22'
10-24-1997 02:12:30PM	'24'
NULL	NULL

Der folgende Ausdruck gibt den Tag des Jahres aller Datumsangaben in einem Port zurück:

```
TO_CHAR( DATE_PROMISED, 'DDD' )
```

DATE_PROMISED	RETURN VALUE
04-01-1997 12:00:10AM	'091'

DATE_PROMISED	RETURN VALUE
02-22-1997 01:31:10PM	'053'
10-24-1997 02:12:30PM	'297'
NULL	NULL

Die folgenden Ausdrücke geben die Stunde des Tages aller Datumsangaben in einem Port zurück:

```
TO_CHAR( DATE_PROMISED, 'HH' )
TO_CHAR( DATE_PROMISED, 'HH12' )
```

DATE_PROMISED	RETURN VALUE
04-01-1997 12:00:10AM	'12'
02-22-1997 01:31:10PM	'01'
10-24-1997 02:12:30PM	'02'
NULL	NULL

```
TO_CHAR( DATE_PROMISED, 'HH24' )
```

DATE_PROMISED	RETURN VALUE
04-01-1997 12:00:10AM	'00'
02-22-1997 01:31:10PM	'13'
10-24-1997 11:12:30PM	'23'
NULL	NULL

Der folgende Ausdruck konvertiert Datumswerte in MJD-Werte, ausgedrückt als Strings:

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

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

Der folgende Ausdruck konvertiert Daten in Strings im Format „MM/DD/YY“:

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

SHIP_DATE	RETURN_VALUE
12/31/1999 01:02:03AM	12/31/99

SHIP_DATE	RETURN_VALUE
09/15/1996 03:59:59PM	09/15/96
05/17/2003 12:13:14AM	05/17/03

Sie können auch den SSSSS-Formatstring in TO_CHAR-Ausdrücken einsetzen. Beispiel: Der folgende Ausdruck konvertiert die Daten im Port SHIP_DATE in Strings mit Angabe der Gesamtzahl der Sekunden seit Mitternacht:

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

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

In TO_CHAR-Ausdrücken erzielt der YY-Formatstring dieselben Ergebnisse wie der RR-Formatstring.

Der folgende Ausdruck konvertiert Daten in Strings im Format „MM/DD/YY“:

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

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

Der folgende Ausdruck gibt die Woche des Monats aller Datumsangaben in einem Port zurück:

```
TO_CHAR( DATE_PROMISED, 'W' )
```

DATE_PROMISED	RETURN VALUE
04-01-1997 12:00:10AM	'01'
02-22-1997 01:31:10AM	'04'
10-24-1997 02:12:30PM	'04'
NULL	NULL

Der folgende Ausdruck gibt die Woche des Jahres aller Datumsangaben in einem Port zurück:

```
TO_CHAR( DATE_PROMISED, 'WW' )
```

DATE_PROMISED	RETURN VALUE
04-01-1997 12:00:10PM	'18'
02-22-1997 01:31:10AM	'08'

DATE_PROMISED	RETURN VALUE
10-24-1997 02:12:30AM	'43'
NULL	NULL

Tipp

TO_CHAR und TO_DATE können kombiniert werden, z. B. zum Konvertieren eines numerischen Werts für einen Monat in einen Textwert für einen Monat mithilfe einer Funktion:

```
TO_CHAR( TO_DATE( numeric_month, 'MM' ), 'MONTH' )
```

TO_CHAR (Zahlen)

Konvertiert numerische Werte in Textstrings. TO_CHAR konvertiert auch Datumsangaben in Strings.

TO_CHAR konvertiert doppelte Werte folgendermaßen in Textstrings:

- Konvertiert doppelte Werte mit bis zu 16 Stellen in Strings und bietet eine Genauigkeit von bis zu 15 Stellen. Wenn Sie eine Zahl mit mehr als 15 Stellen übergeben, rundet TO_CHAR die Zahl auf Basis der 16. Stelle und gibt die Stringdarstellung in wissenschaftlicher Schreibweise zurück. Beispiel: Der doppelte Wert 1234567890123456 wird in den Stringwert '1.23456789012346e+015' konvertiert.
- Gibt die Dezimalschreibweise für Zahlen in den Bereichen (-1e16, -1e-16] und [1e-16, 1e16) zurück. Zahlen außerhalb dieser Bereiche werden von TO_CHAR in wissenschaftlicher Schreibweise zurückgegeben. Beispiel: Der doppelte Wert 10842764968208837340 wird in den Stringwert '1.08427649682088e+019' konvertiert.

TO_CHAR konvertiert Dezimalwerte folgendermaßen in Textstrings:

- Im Hochgenauigkeitsmodus konvertiert TO_CHAR Dezimalwerte mit bis zu 38 Stellen in Strings. Wenn Sie einen Dezimalwert mit mehr als 38 Stellen übergeben, gibt TO_CHAR Zahlen mit mehr als 38 Stellen in wissenschaftlicher Schreibweise zurück.
- Im Hochgenauigkeitsmodus konvertiert TO_CHAR Dezimalwerte mit bis zu 28 Stellen in Strings. Wenn Sie einen Dezimalwert mit mehr als 28 Stellen übergeben, gibt TO_CHAR Zahlen mit mehr als 28 Stellen in wissenschaftlicher Schreibweise zurück.
- Im Niedriggenauigkeitsmodus behandelt TO_CHAR Dezimalwerte wie doppelte Werte.

Syntax

```
TO_CHAR( numeric_value )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
numeric_value	Erforderlich	Numerischer Datentyp. Der numerische Wert, der in einen String konvertiert werden soll. Sie können einen beliebigen gültigen Umwandlungsausdruck eingeben.

Rückgabewert

String.

NULL, falls ein an die Funktion übergebener Wert NULL ist.

Doppelte Umwandlung - Beispiel

Der folgende Ausdruck wandelt die doppelten Werte im SALES-Port in Strings um:

```
TO_CHAR( SALES )
```

SALES	RETURN VALUE
1010.99	'1010.99'
-15.62567	'-15.62567'
10842764968208837340	'1.08427649682088e+019' *(rounded based on the 16th digit and returns the value in scientific notation)*
236789034569723	'236789034569723'
0	'0'
33.15	'33.15'
NULL	NULL

Beispiel für eine Dezimalkonvertierung

Der folgende Ausdruck wandelt die Dezimalwerte im SALES-Port im Hochgenauigkeitsmodus in Strings um:

```
TO_CHAR( SALES )
```

SALES	RETURN VALUE
2378964536789761	'2378964536789761'
12345678901234567890123456 79	'12345678901234567890123456 79'
1.2345789454696493458761234 56	'1.2345789454696493458761234 56'
0.99999999999999999999999999	'0.99999999999999999999999999'
12345678901234567890123456 799	'12345678901234567890123456 799'
2345678899223345667845893456 7123465239	'2345678899223345667845893456 7123465239'
4234567890123456789012345679 91234567899	'4.23456789012346e+038'

(größer als 38)

SALES	RETURN VALUE
2378964536789761	'2378964536789761'
12345678901234567890123456 79	'12345678901234567890123456 79'
1.2345789454696493458761234 56	'1.2345789454696493458761234 56'

SALES	RETURN VALUE
0.9999999999999999999999999999	'0.9999999999999999999999999999'
12345678901234567890123456799 *(größer als 28)*	'1.23456789012346e+028'

TO_DATE

Konvertiert einen Zeichenstring in einen Datum/Zeit-Datentyp. Mit den TO_DATE-Formatstrings geben Sie das Format der Quellstrings an.

Bei TO_DATE-Ausdrücken muss der Ausgabeport den Datum/Zeit-Datentyp aufweisen.

Beim Konvertieren von zweistelligen Jahreszahlen mit TO_DATE verwenden Sie die Formatstrings RR oder YY. Der YYYY-Formatstring ist dazu nicht geeignet.

Syntax

```
TO_DATE( string [, format] )
```

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
string	Erforderlich	Muss ein String-Datentyp sein. Übergibt die Werte, die in Datumsangaben konvertiert werden sollen. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.
format	Optional	Geben Sie einen gültigen TO_DATE-Formatstring ein. Der Formatstring muss den Teilen des Arguments *string* entsprechen. Beispiel: Beim Übergeben des Strings „Mar 15 1998 12:43:10AM“ müssen Sie den Formatstring DD MON YYYY HH12:MI:SSAM angeben. Wenn Sie den Formatstring auslassen, muss der Stringwert in dem Datumsformat vorliegen, das in der Sitzung definiert wurde.

Rückgabewert

Datum.

TO_DATE gibt immer ein Datum und eine Uhrzeit zurück. Wenn Sie einen String ohne Zeitwert übergeben, gibt das Rückgabedatum die Uhrzeit immer mit 00:00:00.000000000 an. Sie können die Ergebnisse dieser Funktion mit einer Zielspalte des Datetime-Datentyps verknüpfen. Wenn die Präzision der Zielspalte unter Nanosekunden liegt, schneidet Data Integration Service den Datetime-Wert beim Schreiben in das Ziel ab, damit er mit der Präzision der Zielspalte übereinstimmt.

NULL, wenn Sie der Funktion einen Nullwert übergeben.

Warnung: Das Format des TO_DATE-Strings muss mit dem Formatstring übereinstimmen, einschließlich der Trennzeichen für Datumsangaben. Andernfalls kann Data Integration Service ungenaue Werte zurückgeben oder den Datensatz überspringen.

Beispiele

Der folgende Ausdruck gibt Datumswerte für die Strings im Port DATE_PROMISED zurück. TO_DATE gibt immer ein Datum und eine Uhrzeit zurück. Wenn Sie einen String ohne Zeitwert übergeben, gibt das Rückgabedatum die Uhrzeit immer mit 00:00:00.000000000 an. Wenn Sie eine Sitzung im 20. Jahrhundert ausführen, lautet das Jahrhundert 19. In diesem Beispiel ist das aktuelle Jahr auf dem Knoten, auf dem Data Integration Service ausgeführt wird, 1998. Das Datetime-Format der Zielspalte lautet MON DD YY HH24:MI SS, was bedeutet, dass Data Integration Service den Datetime-Wert beim Schreiben in das Ziel auf Sekunden beschneidet:

Der folgende Ausdruck gibt Datumswerte für die Strings im Port DATE_PROMISED zurück. TO_DATE gibt immer ein Datum und eine Uhrzeit zurück. Wenn Sie einen String ohne Zeitwert übergeben, gibt das Rückgabedatum die Uhrzeit immer mit 00:00:00.000000000 an. Wenn Sie ein Mapping im 20. Jahrhundert ausführen, lautet das Jahrhundert 19. In diesem Beispiel ist das aktuelle Jahr auf dem Knoten, auf dem Data Integration Service ausgeführt wird, 1998. Das Datetime-Format der Zielspalte lautet MON DD YY HH24:MI SS, was bedeutet, dass Data Integration Service den Datetime-Wert beim Schreiben in das Ziel auf Sekunden beschneidet:

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

DATE_PROMISED	RETURN VALUE
'01/22/98'	Jan 22 1998 00:00:00
'05/03/98'	May 3 1998 00:00:00
'11/10/98'	Nov 10 1998 00:00:00
'10/19/98'	Oct 19 1998 00:00:00
NULL	NULL

Der folgende Ausdruck gibt Datums- und Zeitwerte für die Strings im Port DATE_PROMISED zurück. Wenn Sie einen String ohne Zeitwert übergeben, gibt Data Integration Service eine Fehlermeldung aus. Wenn Sie eine Sitzung im 20. Jahrhundert ausführen, lautet das Jahrhundert 19. Das aktuelle Jahr auf dem Knoten, auf dem Data Integration Service ausgeführt wird, ist 1998:

Der folgende Ausdruck gibt Datums- und Zeitwerte für die Strings im Port DATE_PROMISED zurück. Wenn Sie einen String ohne Zeitwert übergeben, gibt Data Integration Service eine Fehlermeldung aus. Wenn Sie ein Mapping im 20. Jahrhundert ausführen, lautet das Jahrhundert 19. Das aktuelle Jahr auf dem Knoten, auf dem Data Integration Service ausgeführt wird, ist 1998:

```
TO_DATE( DATE_PROMISED, 'MON DD YYYY HH12:MI:SSAM' )
```

DATE_PROMISED	RETURN VALUE
'Jan 22 1998 02:14:56PM'	Jan 22 1998 02:14:56PM
'Mar 15 1998 11:11:11AM'	Mar 15 1998 11:11:11AM
'Jun 18 1998 10:10:10PM'	Jun 18 1998 10:10:10PM
'October 19 1998'	*Error. Integration Service skips this row.*
NULL	NULL

Der folgende Ausdruck konvertiert Strings im Port SHIP_DATE_MJD_STRING in Datumswerte:

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

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

Da der J-Formatstring die Zeitkomponente der Datumsangabe nicht berücksichtigt, wird die Uhrzeit in den Rückgabewerten auf 00: 00: 00.000000000 gesetzt.

Der folgende Ausdruck konvertiert einen String in ein vierstelliges Jahresdatumsformat. Das aktuelle Jahr ist 1998:

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

DATE_STR	RETURN VALUE
'04/01/98'	04/01/1998 00:00:00.000000000
'08/17/05'	08/17/2005 00:00:00.000000000

Der folgende Ausdruck konvertiert einen String in ein vierstelliges Jahresdatumsformat. Das aktuelle Jahr ist 1998:

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

DATE_STR	RETURN VALUE
'04/01/98'	04/01/1998 00:00:00.000000000
'08/17/05'	08/17/1905 00:00:00.000000000

Hinweis: In der zweiten Zeile gibt RR das Jahr 2005, YY hingegen das Jahr 1905 zurück.

Der folgende Ausdruck konvertiert einen String in ein vierstelliges Jahresdatumsformat. Das aktuelle Jahr ist 1998:

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

DATE_STR	RETURN VALUE
'04/01/8'	04/01/1998 00:00:00.000000000
'08/17/5'	08/17/1995 00:00:00.000000000

Der folgende Ausdruck konvertiert einen String in ein vierstelliges Jahresdatumsformat. Das aktuelle Jahr ist 1998:

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

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

Der folgende Ausdruck konvertiert Strings, die eine Angabe der Sekunden seit Mitternacht enthalten, in Datumswerte:

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

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

Wenn das Ziel unterschiedliche Datumsformate akzeptiert, verwenden Sie TO_DATE und IS_DATE mit der Funktion DECODE zum Ermitteln der zulässigen Formate. Beispiel:

```
DECODE( TRUE,

 --test first format
   IS_DATE( CLOSE_DATE,'MM/DD/YYYY HH24:MI:SS' ),

 --if true, convert to date
      TO_DATE( CLOSE_DATE,'MM/DD/YYYY HH24:MI:SS' ),

 --test second format; if true, convert to date
   IS_DATE( CLOSE_DATE,'MM/DD/YYYY'), TO_DATE( CLOSE_DATE,'MM/DD/YYYY' ),

 --test third format; if true, convert to date
   IS_DATE( CLOSE_DATE,'MON DD YYYY'), TO_DATE( CLOSE_DATE,'MON DD YYYY'),

 --if none of the above
   ERROR( 'NOT A VALID DATE') )
```

TO_CHAR und TO_DATE können kombiniert werden, z. B. zum Konvertieren eines numerischen Werts für einen Monat in einen Textwert für einen Monat mithilfe einer Funktion:

```
TO_CHAR( TO_DATE( numeric_month, 'MM' ), 'MONTH' )
```

Verwandte Themen:

- "Regeln und Richtlinien für Datumsformatstrings" auf Seite 49

TO_DECIMAL

Konvertiert eine Zeichenfolge oder numerischen Wert in einen Dezimalwert. TO_DECIMAL ignoriert vorangestellte Leerzeichen.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
value	Erforderlich	Muss ein Zeichenfolgen- oder numerischer Datentyp sein. Übergibt die Werte, die in Dezimalwerte umgewandelt werden sollen. Sie können einen beliebigen gültigen Umwandlungsausdruck eingeben.
scale	Optional	Muss ein Ganzzahl-Literal zwischen 0 und einschließlich 28 sein. Gibt die maximale Anzahl der Ziffern nach dem Dezimalzeichen an. Wenn Sie dieses Argument auslassen, gibt die Funktion einen Wert mit der gleichen Größenordnung (scale) wie der Eingabewert (value) zurück.

Rückgabewert

Dezimalzahl mit Präzision und Größenordnung zwischen 0 und einschließlich 28.

NULL, falls ein an die Funktion übergebener Wert NULL ist.

Wenn die Zeichenfolge nicht-numerische Zeichen enthält, wird der numerische Teil der Zeichenfolge bis hin zum ersten nicht-numerischen Zeichen konvertiert.

Wenn das erste Zeichen nicht numerisch ist, lautet die Rückgabe 0.

Wenn der an die Funktion übergebene Wert Daten enthält, die für einen Dezimalwert nicht gültig sind, markiert der Datenintegrationsdienst die Zeile als Fehlerzeile.

Hinweis: Wenn der Rückgabewert eine Dezimalmal mit Präzision höher als 15 ist, können Sie „Hohe Präzision“ aktivieren, um Dezimalgenauigkeit bis zu 28 Stellen zu gewährleisten.

Beispiel

Dieser Ausdruck verwendet Werte aus dem Port IN_TAX. IN_TAX ist ein Zeichenfolgendatentyp mit einer Genauigkeit von 44 Stellen. RETURN VALUE ist ein Dezimaldatentyp mit einer Genauigkeit von 28 und einer Größenordnung von 3:

```
TO_DECIMAL( IN_TAX, 3 )
```

IN_TAX	RETURN VALUE
'15.6789'	15.679
'60.2'	60.200
'118.348'	118.348
NULL	NULL
'A12.3Grove'	0
'711A1'	711
'1234567890.123'	1234567890.123
'123456789012345678901234567890.123'	Error. Integration Service skips this row.

IN_TAX	RETURN VALUE
'12345678901234567890123456789012345678 90.123	Error. Integration Service skips this row.

IN_TAX	RETURN VALUE
'15.6789'	15.679
'60.2'	60.200
'118.348'	118.348
NULL	NULL
'A12.3Grove'	*Error. Integration Service skips this row.*
'711A1'	*Error. Integration Service skips this row.*
'1234567890.123'	1234567890.123
'12345678901234567890123456789 0.123'	Error. Integration Service skips this row.
'12345678901234567890123456789012345678 90.123	Error. Integration Service skips this row.

TO_DECIMAL38

Konvertiert eine Zeichenfolge oder numerischen Wert in einen Dezimalwert. TO_DECIMAL38 ignoriert vorangestellte Leerzeichen.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
value	Erforderlich	Muss ein Zeichenfolgen- oder numerischer Datentyp sein. Übergibt die Werte, die in Dezimalwerte umgewandelt werden sollen. Sie können einen beliebigen gültigen Umwandlungsausdruck eingeben.
scale	Optional	Muss ein Ganzzahlliteral zwischen 0 und einschließlich 38 sein. Gibt die maximale Anzahl der Ziffern nach dem Dezimalzeichen an. Wenn Sie dieses Argument auslassen, gibt die Funktion einen Wert mit der gleichen Größenordnung (scale) wie der Eingabewert (value) zurück.

Rückgabewert

Dezimalzahl mit Genauigkeit und Größenordnung zwischen 0 und einschließlich 38.

NULL, falls ein an die Funktion übergebener Wert NULL ist.

Wenn der an die Funktion übergebene Wert Daten enthält, die für einen Dezimalwert nicht gültig sind, markiert der Datenintegrationsdienst die Zeile als Fehlerzeile. Wenn Sie beispielsweise `TO_DECIMAL38("12345678901234567890123456789012345678 90.12")` übergeben, weist der Datenintegrationsdienst die Zeile zurück.

Hinweis: Wenn der Rückgabewert eine Dezimalzahl mit einer Genauigkeit höher als 15 ist, können Sie „Hohe Genauigkeit“ aktivieren, um Dezimalgenauigkeit bis zu 38 Stellen zu gewährleisten.

Beispiel

Dieser Ausdruck verwendet Werte aus dem Port IN_TAX. IN_TAX ist ein Zeichenfolgendatentyp mit einer Genauigkeit von 44 Stellen. RETURN VALUE ist ein Dezimaldatentyp mit einer Genauigkeit von 38 und einer Größenordnung von 3:

```
TO_DECIMAL38( IN_TAX, 3 )
```

IN_TAX	RETURN VALUE
'15.6789'	15.679
'60.2'	60.200
'118.348'	118.348
NULL	NULL
'A12.3Grove'	*Error. Integration Service skips this row.*
'1234567890.123'	1234567890.123
'12345678901234567890123456789 0.123'	12345678901234567890123456789 0.123
'12345678901234567890123456789 01234567890.123'	*Error. Integration Service skips this row.*
'711A1'	*Error. Integration Service skips this row.*

TO_FLOAT

Konvertiert einen String oder numerischen Wert in eine Gleitkommazahl mit Double-Präzision (Double-Datentyp). TO_FLOAT ignoriert vorangestellte Leerzeichen.

Syntax

```
TO_FLOAT( value )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
value	Erforderlich	Muss ein String- oder numerischer Datentyp sein. Übergibt die Werte, die in Double-Werte konvertiert werden sollen. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

Double-Wert

NULL, falls ein an die Funktion übergebener Wert NULL ist.

0, wenn der Wert im Port leer oder ein nicht-numerisches Zeichen ist.

Wenn der an die Funktion übergebene Wert Daten enthält, die für einen Floatwert nicht gültig sind, markiert der Datenintegrationsdienst die Zeile als Fehlerzeile oder das Mapping schlägt fehl.

Beispiel

Dieser Ausdruck verwendet Werte aus dem Port IN_TAX:

```
TO_FLOAT( IN_TAX )
```

IN_TAX	RETURN VALUE
'15.6789'	15.6789
'60.2'	60.2
'118.348'	118.348
NULL	NULL
'A12.3Grove'	0
'A12.3Grove'	*Error. Integration Service skips this row.*

TO_INTEGER

Konvertiert einen String oder numerischen Wert in eine Ganzzahl. Die Syntax von TO_INTEGER umfasst ein optionales Argument, mit dem Sie die Zahl auf die nächste Ganzzahl runden oder die Dezimalstellen abschneiden können. TO_INTEGER ignoriert vorangestellte Leerzeichen.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
value	Erforderlich	String- oder numerischer Datentyp. Übergibt den Wert, der in eine Ganzzahl konvertiert werden soll. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.
flag	Optional	Legt fest, ob die Dezimalstellen abgeschnitten oder gerundet werden. Das Flag muss ein Ganzzahl-Literal sein oder den Konstanten TRUE oder FALSE entsprechen. TO_INTEGER schneidet die Dezimalstellen ab, wenn für das Flag TRUE oder eine andere Zahl als 0 eingegeben wurde. TO_INTEGER rundet den Wert auf die nächste Ganzzahl, wenn das Flag FALSE oder 0 lautet bzw. nicht angegeben ist.

Rückgabewert

Ganzzahl.

NULL, wenn der an die Funktion übergebene Wert NULL ist.

0, wenn der an die Funktion übergebene Wert alphanumerische Zeichen enthält.

Wenn der an die Funktion übergebene Wert Daten enthält, die für einen Ganzzahlwert nicht gültig sind, markiert der Datenintegrationsdienst die Zeile als Fehlerzeile oder das Mapping schlägt fehl.

Beispiele

Die folgenden Ausdrücke verwenden Werte aus dem Port IN_TAX. Data Integration Service zeigt eine Fehlermeldung an, wenn die Konvertierung einen numerischen Overflow verursacht:

```
TO_INTEGER( IN_TAX, TRUE )
```

IN_TAX	RETURN VALUE
'15.6789'	15
'60.2'	60
'118.348'	118
'5,000,000,000'	*Error. Integration Service skips this row.*
NULL	NULL
'A12.3Grove'	0
'A12.3Grove'	*Error. Integration Service skips this row.*
' 123.87'	123

IN_TAX	RETURN VALUE
'-15.6789'	-15
'-15.23'	-15

```
TO_INTEGER( IN_TAX, FALSE)
```

IN_TAX	RETURN VALUE
'15.6789'	16
'60.2'	60
'118.348'	118
'5,000,000,000'	*Error. Integration Service skips this row.*
NULL	NULL
'A12.3Grove'	0
'A12.3Grove'	*Error. Integration Service skips this row.*
' 123.87'	124
'-15.6789'	-16
'-15.23'	-15

TO_TIMESTAMP_TZ

Wandelt eine Zeichenfolge in einen „Zeitstempel mit Zeitzone“-Wert um. Die Funktion gibt den Datentyp „Zeitstempel mit Zeitzone“ zurück. Mit den TO_TIMESTAMP_TZ-Formatzeichenfolgen geben Sie das Format der Quellzeichenfolgen an.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
Zeichenfolge	Erforderlich	Muss vom Datentyp „Zeichenfolge“ sein. Übergibt die Werte, die Sie in Zeitstempel mit Zeitzone umwandeln möchten. Sie können einen beliebigen gültigen Umwandlungsausdruck eingeben. Die Zeichenfolge muss eine Zeichenfolge sein.
format	Optional	Geben Sie eine gültige TO_TIMESTAMP_TZ-Formatzeichenfolge ein. Die Formatzeichenfolge muss den Teilen des Zeichenfolgenarguments entsprechen. Wenn Sie beispielsweise die Zeichenfolge 'Mar 15 1997 12:43:10AM ASIA/CALCUTTA' übergeben, müssen Sie die Formatzeichenfolge 'MON DD YYYY HH12:MI:SSAM TZR' verwenden. Wenn Sie die Formatzeichenfolge nicht angeben, verwendet die Funktion das Standardformat für Datum/Uhrzeit im Dialogfeld „Konfigurationen ausführen“.

Rückgabewert

Gibt einen Zeitstempel mit dem Datentyp „Zeitzone“ zurück.

NULL, wenn die Eingabe ein Nullwert ist.

Wenn der an die Funktion übergebene Wert Daten enthält, die für einen Zeitstempel mit Zeitzone-Wert nicht gültig sind, markiert der Datenintegrationsdienst die Zeile als Fehlerzeile oder die Zuordnung schlägt fehl.

Beispiel

INPUT VALUE	RETURN VALUE
‘1947-08-05 10:45:00.221111000 AM America/Los_Angeles’, 'YYYY-MM-DD HH:MI:SS.NS AM TZR'	Gibt den Datentyp „Zeitstempel mit Zeitzone“ mit den folgenden Daten zurück: ‘1947-08-05 10:45:00.221111000 AM AMERICA/LOS_ANGELES’
‘1947-08-05 10:45:00.221111000 AM America/Los_Angeles’, 'YYYY-MM-DD HH:MI:SS.NS AM'	Gibt den Datentyp „Zeitstempel mit Zeitzone“ auch dann zurück, wenn die Zeitzonenregion nicht im Format der Zeitzonenregion angegeben wird: ‘1947-08-05 10:45:00.221111000 AM AMERICA/LOS_ANGELES'
‘1947-08-05 10:45:00.221111000 AM America/Los_Angeles’	Gibt den Datentyp „Zeitstempel mit Zeitzone“ auch dann zurück, wenn das Format „Zeitstempel mit Zeitzone“ nicht angegeben wird. '1947-08-05 10:45:00.221111000 AM AMERICA/LOS_ANGELES'
‘1947-08-05 10:45:00.221111000 AM America/Los_Angeles’, ’MM-DD-YYYY HH:MI:SS.NS AM'	Das Standardformat für Datum/Uhrzeit im Dialogfeld „Konfigurationen ausführen“ wird verwendet, wenn das Format nicht auf der Funktionsebene angegeben wird. Standardformat für Datum/Uhrzeit: 'YYYY-MM-DD HH:MI:SS.NS AM TZR' Wenn „Zeitstempel mit Zeitzone“-Daten nicht mit dem angegebenen Format übereinstimmen, wird folgender Fehler angezeigt: Process row failed for function [TO_TIMESTAMP_TZ]: Failed to convert the string to timestamp with time zone value. Verify that the specified date format string is valid. Verify that the timestamp with time zone string used in the first argument is compatible with the specified date format.

TRUNC (Datum)

Schneidet Datumsangaben nach Jahr, Monat, Tag, Stunde, Minute, Sekunde, Millisekunde oder Mikrosekunde ab. TRUNC kann auch zum Abschneiden von Zahlen eingesetzt werden.

Sie können folgende Teile eines Datums abschneiden:

- **Jahr.** Wenn Sie an der Jahreskomponente eines Datums abschneiden, gibt die Funktion den 1. Januar des Eingabejahres mit Uhrzeit 00:00:00.000000000 zurück. Beispiel: Der folgende Ausdruck ergibt 1/1/1997 00:00:00.000000000:

  ```
  TRUNC(12/1/1997 3:10:15, 'YY')
  ```

- **Monat.** Wenn Sie an der Monatskomponente eines Datums abschneiden, gibt die Funktion den ersten Tag des Monats mit Uhrzeit 00:00:00.000000000 zurück. Beispiel: Der folgende Ausdruck ergibt 4/1/1997 00:00:00.000000000:

  ```
  TRUNC(4/15/1997 12:15:00, 'MM')
  ```

- **Tag.** Wenn Sie an der Tageskomponente eines Datums abschneiden, gibt die Funktion das Datum mit Uhrzeit 00:00:00.000000000 zurück. Beispiel: Der folgende Ausdruck ergibt 6/13/1997 00:00:00.000000000:

  ```
  TRUNC(6/13/1997 2:30:45, 'DD')
  ```

- **Stunde.** Wenn Sie an der Stundenkomponente eines Datums abschneiden, gibt die Funktion das Datum mit auf 0 gestellten Minuten, Sekunden und Subsekunden zurück. Beispiel: Der folgende Ausdruck ergibt 4/1/1997 11:00:00.000000000:

  ```
  TRUNC(4/1/1997 11:29:35, 'HH')
  ```

- **Minute.** Wenn Sie an der Minutenkomponente eines Datums abschneiden, gibt die Funktion das Datum mit auf 0 gestellten Sekunden und Subsekunden zurück. Beispiel: Der folgende Ausdruck ergibt 5/22/1997 10:15:00.000000000:

  ```
  TRUNC(5/22/1997 10:15:29, 'MI')
  ```

- **Sekunde.** Wenn Sie an der Sekundenkomponente eines Datums abschneiden, gibt die Funktion das Datum mit auf 0 gestellten Millisekunden zurück. Beispiel: Der folgende Ausdruck ergibt 5/22/1997 10:15:29.000000000:

  ```
  TRUNC(5/22/1997 10:15:29.135, 'SS')
  ```

- **Millisekunde.** Wenn Sie an der Millisekundenkomponente eines Datums abschneiden, gibt die Funktion das Datum mit auf 0 gestellten Mikrosekunden zurück. Beispiel: Der folgende Ausdruck ergibt 5/22/1997 10:15:30.135000000:

  ```
  TRUNC(5/22/1997 10:15:30.135235, 'MS')
  ```

- **Mikrosekunde.** Wenn Sie an der Mikrosekundenkomponente eines Datums abschneiden, gibt die Funktion das Datum mit auf 0 gestellten Nanosekunden zurück. Beispiel: Der folgende Ausdruck ergibt 5/22/1997 10:15:30.135235000:

  ```
  TRUNC(5/22/1997 10:15:29.135235478, 'US')
  ```

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
Datum	Erforderlich	Datum/Zeit-Datentyp. Das Datumswerte, die abgeschnitten werden sollen. Sie können jeden beliebigen Umwandlungsausdruck eingeben, dessen Auswertung ein Datum ergibt.
format	Optional	Geben Sie einen gültigen Formatstring ein. Bei Formatstrings muss nicht auf Groß-/Kleinschreibung geachtet werden. Wenn Sie den Formatstring nicht angeben, schneidet die Funktion das Datum an der Zeitkomponente ab und setzt sie auf 00:00:00.000000000.

Rückgabewert

Datum.

NULL, falls ein an die Funktion übergebener Wert NULL ist.

Beispiele

Die folgenden Ausdrücke schneiden die Datumsangaben im Port DATE_SHIPPED an der Jahreskomponente ab:

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

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:30AM	Jan 1 1998 12:00:00.000000000
Apr 19 1998 1:31:20PM	Jan 1 1998 12:00:00.000000000
Jun 20 1998 3:50:04AM	Jan 1 1998 12:00:00.000000000
Dec 20 1998 3:29:55PM	Jan 1 1998 12:00:00.000000000
NULL	NULL

Die folgenden Ausdrücke schneiden die Datumsangaben im Port DATE_SHIPPED an der Monatskomponente ab:

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

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:30AM	Jan 1 1998 12:00:00.000000000AM
Apr 19 1998 1:31:20PM	Apr 1 1998 12:00:00.000000000AM
Jun 20 1998 3:50:04AM	Jun 1 1998 12:00:00.000000000AM

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

Die folgenden Ausdrücke die Datumsangaben im Port DATE_SHIPPED an der Tageskomponente ab:

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

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:30AM	Jan 15 1998 12:00:00.000000000AM
Apr 19 1998 1:31:20PM	Apr 19 1998 12:00:00.000000000AM
Jun 20 1998 3:50:04AM	Jun 20 1998 12:00:00.000000000AM
Dec 20 1998 3:29:55PM	Dec 20 1998 12:00:00.000000000AM
Dec 31 1998 11:59:59PM	Dec 31 1998 12:00:00.000000000AM
NULL	NULL

Die folgenden Ausdrücke schneiden die Datumsangaben im Port DATE_SHIPPED an der Stundenkomponente ab:

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

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:31AM	Jan 15 1998 2:00:00.000000000AM
Apr 19 1998 1:31:20PM	Apr 19 1998 1:00:00.000000000PM
Jun 20 1998 3:50:04AM	Jun 20 1998 3:00:00.000000000AM
Dec 20 1998 3:29:55PM	Dec 20 1998 3:00:00.000000000PM
Dec 31 1998 11:59:59PM	Dec 31 1998 11:00:00.000000000AM
NULL	NULL

Der folgende Ausdruck schneidet die Datumsangaben im Port DATE_SHIPPED an der Minutenkomponente ab:

```
TRUNC( DATE_SHIPPED, 'MI' )
```

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:30AM	Jan 15 1998 2:10:00.000000000AM

DATE_SHIPPED	RETURN VALUE
Apr 19 1998 1:31:20PM	Apr 19 1998 1:31:00.000000000PM
Jun 20 1998 3:50:04AM	Jun 20 1998 3:50:00.000000000AM
Dec 20 1998 3:29:55PM	Dec 20 1998 3:29:00.000000000PM
Dec 31 1998 11:59:59PM	Dec 31 1998 11:59:00.000000000PM
NULL	NULL

TRUNC (Zahlen)

Schneidet Zahlen nach einer bestimmten Stelle ab. TRUNC kann auch zum Abschneiden von Datumsangaben eingesetzt werden.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
numeric_value	Erforderlich	Numerischer Datentyp. Übergibt die Werte, die abgeschnitten werden sollen. Sie können jeden beliebigen Umwandlungssausdruck eingeben, dessen Auswertung einen numerischen Datentyp ergibt.
precision	Optional	Kann eine positive oder negative Ganzzahl sein. Sie können jeden beliebigen Umwandlungsausdruck eingeben, dessen Auswertung eine Ganzzahl ergibt. Die Ganzzahl gibt die Anzahl der Stellen an, nach denen abgeschnitten wird.

Wenn *precision* eine positive Ganzzahl ist, gibt TRUNC den *numeric_value* mit der von *precision* angegebenen Anzahl von Dezimalstellen zurück. Wenn *precision* eine negative Ganzzahl ist, ändert TRUNC die angegebene Anzahl von Ziffern links vom Dezimalzeichen zu Nullen. Wenn Sie das Argument *precision* auslassen, schneidet TRUNC den gesamten Dezimalbereich von *numeric_value* ab und gibt eine Ganzzahl zurück.

Wenn Sie für *precision* einen Dezimalwert angeben, Data Integration ServicePowerCenter Integration Service*numeric_value* auf die nächste Ganzzahl, bevor der Ausdruck ausgewertet wird.

Bei Sitzungen im hohen Präzisionsmodus sollten Sie vor dem Abschneiden die Funktion ROUND ausführen.

Bei Mappings im hohen Präzisionsmodus sollten Sie vor dem Abschneiden die Funktion ROUND ausführen.

Beispiel: Angenommen, die Werte im Port QTY werden mithilfe des folgenden Ausdrucks abgeschnitten:

```
TRUNC ( QTY / 15 )
```

Wenn der Wert für QTY = 15000000, gibt die Sitzung den Wert 999999 zurück. Das erwartete Ergebnis ist jedoch 1000000.

Zur Laufzeit wertet Data Integration Service zunächst den Konstantenteil des Ausdrucks aus und dann erst den Variablenteil.

Im Ausdruck oben ist QTY der Variablenwert, (1/15) der Konstantenwert.

Wenn QTY = 15000000, wird der Ausdruck wie folgt ausgewertet:

```
TRUNC ( 15000000 * (1/15)
TRUNC ( 15000000 * (1/15)
= TRUNC ( 15000000 * 0.0666666666666666)
= TRUNC ( 15000000 * 0.0666666666666666)
= TRUNC ( 999999.99999999)
= 999999
```

Wenn Sie vor dem Abschneiden die Funktion ROUND ausführen, wird der Ausdruck folgendermaßen ausgewertet:

```
TRUNC (ROUND (QTY/15, .999999999999999999999999999)).
```

Rückgabewert

Numerischer Wert.

NULL, wenn eines der Argumente NULL ist.

Hinweis: Wenn der Rückgabewert eine Dezimalmal mit Präzision höher als 15 ist, können Sie „Hohe Präzision“ aktivieren, um Dezimalgenauigkeit bis zu 28 Stellen zu gewährleisten.

Beispiele

Die folgenden Ausdrücke schneiden die Werte im Port PRICE ab:

```
TRUNC( PRICE, 3 )
```

PRICE	RETURN VALUE
12.9995	12.999
-18.8652	-18.865
56.9563	56.956
15.9928	15.992
NULL	NULL

```
TRUNC( PRICE, -1 )
```

PRICE	RETURN VALUE
12.99	10.0
-187.86	-180.0
56.95	50.0
1235.99	1230.0
NULL	NULL

```
TRUNC( PRICE )
```

PRICE	RETURN VALUE
12.99	12.0

PRICE	RETURN VALUE
-18.99	-18.0
56.95	56.0
15.99	15.0
NULL	NULL

UPPER

Konvertiert die Kleinbuchstaben in einem String in Großbuchstaben.

Syntax

```
UPPER( string )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
string	Erforderlich	String-Datentyp. Übergibt die Werte, die zu Großbuchstaben geändert werden sollen. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

String aus Großbuchstaben. Wenn die Daten Multibyte-Zeichen enthalten, hängt der Rückgabewert von der Codepage und dem Datenverschiebungsmodus von Data Integration Service ab.

NULL, falls ein an die Funktion übergebener Wert NULL ist.

Beispiel

Der folgende Ausdruck ändert alle Namen im Port FIRST_NAME zu Großbuchstaben:

```
UPPER( FIRST_NAME )
```

FIRST_NAME	RETURN VALUE
Ramona	RAMONA
NULL	NULL
THOMAS	THOMAS
PierRe	PIERRE
Bernice	BERNICE

UUID4

Gibt einen zufällig generierten binären 16-Byte-Wert zurück, der mit Variante 4 der in RFC 4122 beschriebenen UUID-Spezifikation übereinstimmt. UUID4 übernimmt kein Argument.

Syntax

```
UUID4()
```

Rückgabewert

Binär.

UUID4 gibt niemals einen Nullwert oder einen Fehler zurück.

UUID_UNPARSE

Wandelt einen binären 16-Byte-Wert in eine Zeichenfolgendarstellung bestehend aus 36 Zeichen wie in RFC 4122 angegeben um.

Syntax

```
UUID_UNPARSE( binary )
```

In der folgenden Tabelle wird das Argument für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
Binär	Erforderlich	Binärer Datentyp. Alle binären 16-Byte-Werte, die Sie in Zeichenfolgen bestehend aus 36 Zeichen umwandeln möchten.

Rückgabewert

Zeichenfolge bestehend aus 36 Zeichen.

Gibt NULL zurück, wenn das Argument NULL ist und einen Fehler, wenn das Argument kein binärer 16-Byte-Wert ist.

Beispiel

Der folgende Ausdruck gibt möglicherweise einen Wert von 6948DF80-14BD-4E04-8842-7668D9C001F5 zurück:

```
UUID_UNPARSE(UUID4())
```

VARIANCE

Gibt die Varianz eines übergebenen Werts zurück. VARIANCE dient zur Analyse statistischer Daten. Sie können eine weitere Aggregatfunktion in VARIANCE schachteln, und die verschachtelte Funktion muss einen numerischen Datentyp zurückgeben.

Syntax

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

In der folgenden Tabelle werden die Argumente für diesen Befehl beschrieben:

Argument	Erforderlich/ Optional	Beschreibung
numeric_value	Erforderlich	Numerischer Datentyp. Übergibt die Werte, deren Varianz berechnet werden soll. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.
filter_condition	Optional	Begrenzt die Zeilen in der Suche. Die Filterbedingung muss ein numerischer Wert sein oder mit TRUE, FALSE oder NULL ausgewertet werden. Sie können einen beliebigen gültigen Umwandlungausdruck eingeben.

Rückgabewert

Double-Wert

NULL, wenn alle übergebenen Werte NULL sind oder keine Zeilen ausgewählt wurden (z. B. wenn *filter_condition* in allen Zeilen FALSE oder NULL ergibt).

Nullen

Wenn nur ein Wert NULL ist, wird er ignoriert. Wenn jedoch alle der Funktion übergebenen Werte NULL sind oder keine Zeilen ausgewählt werden, gibt VARIANCE NULL zurück.

Hinweis: Standardmäßig behandelt Data Integration Service Nullwerte in Aggregatfunktionen als NULL. Wenn Sie einen Port oder eine Gruppe mit ausschließlich Nullwerten übergeben, gibt die Funktion NULL zurück. Beim Konfigurieren von Data Integration Service können Sie jedoch entscheiden, wie mit Nullwerten in Aggregatfunktionen umgegangen werden soll. Sie können Nullwerte in Aggregatfunktionen als 0 oder als NULL verarbeiten.

Gruppieren nach

VARIANCE gruppiert die Werte nach der Einstellung „Gruppieren nach Ports“, die Sie in der Umwandlung festlegen, und gibt pro Gruppe ein Ergebnis zurück.

Wenn „Gruppieren nach Ports“ nicht festgelegt wurde, behandelt VARIANCE alle Zeilen als eine einzige Gruppe und gibt nur einen Wert zurück.

Beispiel

Der folgende Ausdruck berechnet die Varianz aller Zeilen im Port TOTAL_SALES:

```
VARIANCE( TOTAL_SALES )
```

TOTAL_SALES
2198.0
2256.0
3001.0
NULL

TOTAL_SALES
8953.0

RETURN VALUE: 10592444.6666667

KAPITEL 7

Erstellen von benutzerdefinierten Funktionen

Dieses Kapitel umfasst die folgenden Themen:

- Erstellen von benutzerdefinierten Funktionen – Übersicht, 233
- Schritt 1. Abrufen von Repository-ID-Attributen, 234
- Schritt 2. Erstellen einer Header-Datei, 235
- Schritt 3. Erstellen einer Implementierungsdatei, 237
- Schritt 4. Erstellen der Module, 246
- Schritt 5. Erstellen der Repository-Plug-In-Datei, 248
- Schritt 6. Testen von benutzerdefinierten Funktionen, 252
- Installieren von benutzerdefinierten Funktionen, 253
- Erstellen von Ausdrücken mit benutzerdefinierten Funktionen, 254

Erstellen von benutzerdefinierten Funktionen – Übersicht

Benutzerdefinierte Funktionen erweitern die Bibliothek der Funktionen in PowerCenter. Dabei handelt es sich um Funktionen, die Sie erstellen, um sie in Umwandlungs- und Arbeitsablaufsausdrücken einzusetzen. Benutzerdefinierte Funktionen werden außerhalb von PowerCenter mit der API für benutzerdefinierte Funktionen erstellt. Um diese API für benutzerdefinierte Funktionen nutzen zu können, müssen Sie die Informatica Developer-Plattform von der Informatica Developer-Community-Website herunterladen.

Die API für benutzerdefinierte Funktionen verwendet die Programmiersprache C. Sie können benutzerdefinierte Funktionen mit anderen Benutzern gemeinsam verwenden. Benutzer können die Funktionen ihrem Repository hinzufügen und wie eine PowerCenter-Umwandlungssprachfunktion verwenden.

Dieses Kapitel enthält eine einfache Funktion, die zeigt, wie eine benutzerdefinierte Funktion mit der Pushdown-Optimierung erstellt und eingesetzt werden kann. Mithilfe der Schritte in diesem Kapitel erstellen Sie die Funktion ECHO. Diese Funktion nimmt ein Argument als Eingabe und gibt den Eingabewert an den Benutzer zurück. Der Beispielcode für die Funktion ECHO befindet sich im Verzeichnis `CustomFunctionAPI\samples\echo` der Informatica Development Platform-Installation.

Sie können auch ein komplexeres Beispiel für eine benutzerdefinierte Funktion anzeigen. Die benutzerdefinierte Funktion „SampleLoanPayment“ enthält Funktionen, die in C nicht zur Verfügung stehen.

„SampleLoanPayment“ steht im Verzeichnis `\CustomFunctionAPI\samples\loanpayment` der Informatica Developer Platform-Installation zur Verfügung.

Schritte zum Erstellen von benutzerdefinierten Funktionen

Führen Sie die folgenden Schritte zum Erstellen von benutzerdefinierten Funktionen aus:

1. **Rufen Sie die Repository-ID-Attribute ab.** Rufen Sie die Repository-ID-Attribute ab, die in das Repository-Plug-In aufgenommen werden sollen.
2. **Erstellen Sie die Header-Datei.** Definieren Sie eine oder mehrere benutzerdefinierte Funktionen in der Header-Datei.
3. **Erstellen Sie die Implementierungsdatei.** Definieren Sie eine oder mehrere benutzerdefinierte Funktionen in der Implementierungsdatei.
4. **Erstellen Sie die Module.** Erstellen Sie die Module, um die DLLs und die gemeinsam genutzten Bibliotheken zu erstellen.
5. **Erstellen Sie die Repository-Plug-In-Datei.** Definieren Sie die Metadaten der benutzerdefinierten Funktionen.
6. **Testen Sie die benutzerdefinierten Funktionen.** Installieren Sie die benutzerdefinierten Funktionen und verwenden Sie sie in einem Mapping und einem Arbeitsablauf, um sie zu überprüfen.

Installieren von benutzerdefinierten Funktionen

Um benutzerdefinierte Funktionen verwenden zu können, müssen Sie sie der PowerCenter-Umgebung hinzufügen.

Verwandte Themen:

- “Installieren von benutzerdefinierten Funktionen” auf Seite 253

Schritt 1. Abrufen von Repository-ID-Attributen

Bevor Sie eine benutzerdefinierte Funktion entwickeln können, müssen Sie die Repository-ID-Attribute für das Repository-Plug-In der benutzerdefinierten Funktion festlegen. Mithilfe der Repository-ID-Attribute können Sie beim Definieren der Plug-In-Metadaten das betreffende Plug-In identifizieren.

Zum Abrufen der Repository-ID-Attribute führen Sie einen der folgenden Schritte aus:

- Wenn Sie benutzerdefinierte Funktionen außerhalb Ihrer Organisation verteilen müssen, wenden Sie sich an Informatica. Informatica weist jedem Plug-In eindeutige Repository-ID-Attribute zu. Repository-ID-Attribute sind ungültig, wenn Sie mit jenen eines Drittanbieters in Konflikt treten. Um die Repository-ID-Attribute zu erhalten, gehen Sie zu https://community.informatica.com/community/marketplace/repositoryidattributes und klicken Sie auf **Übermitteln**.
- Wenn Sie lediglich benutzerdefinierte Funktionen innerhalb Ihrer Organisation verwenden, definieren Sie Repository-ID-Attribute, ohne sich an Informatica zu wenden. Wenn Sie ein Plug-In für Ihre Organisation entwickeln, das mit anderen Plug-Ins in PowerCenter verwendet wird, weisen Sie den Repository-ID-Attributen eindeutige Werte für jedes Plug-In zu.

Die folgende Tabelle zeigt die XML-Attribute, die für die Definition eines Plug-In eindeutige Werte benötigen:

Repository-ID-Attribut	Beschreibung
Plug-In-ID	Identifiziert die ID des Plug-In. Dieser Wert entspricht dem Attribut ID des Elements PLUGIN.
Lieferanten-ID	Gibt den Anbieter an, der das Plug-In entwickelt hat. Dieser Wert entspricht dem Attribut VENDORID des Elements PLUGIN.
Funktionsgruppen-ID	Gibt die ID der Funktionsgruppe an. Dieser Wert entspricht dem Attribut ID des Elements FUNCTION_GROUP.
Funktions-ID	Gibt die ID der Funktion an. Dieser Wert entspricht dem Attribut ID des Elements FUNCTION.

Hinweis: Repository-ID-Attribute sind ungültig, wenn Konflikte zwischen ihnen bestehen.

Verwandte Themen:

- "Das Element PLUGIN" auf Seite 249
- "Das Element FUNCTION_GROUP" auf Seite 249
- "Das Element FUNCTION" auf Seite 250

Schritt 2. Erstellen einer Header-Datei

Erstellen Sie eine Header-Datei in C, um alle Funktionen zu deklarieren. Sie können eine einzige Header-Datei für eine oder für mehrere benutzerdefinierte Funktionen verwenden.

Das folgende Beispiel zeigt die Header-Datei „echo.h" der benutzerdefinierten Funktion ECHO:

```
#ifndef __ECHO_PLUGIN_HPP
#define __ECHO_PLUGIN_HPP

#if defined(WIN32)
    #if defined SAMPLE_EXPR_EXPORTS
        #define  SAMPLE_EXPR_SPEC __declspec(dllexport)
    #else
        #define  SAMPLE_EXPR_SPEC __declspec(dllimport)
    #endif
#else
    #define SAMPLE_EXPR_SPEC
#endif

// method to get description of Echo function
extern "C" SAMPLE_EXPR_SPEC IUNICHAR * getDescriptionEcho(IUNICHAR* ns, IUNICHAR*
sFuncName);

// method to get prototype of Echo function
extern "C" SAMPLE_EXPR_SPEC IUNICHAR * getPrototypeEcho(IUNICHAR* ns, IUNICHAR*
sFuncName);

// method to validate usage of Echo function
extern "C" SAMPLE_EXPR_SPEC INFA_EXPR_STATUS validateFunctionEcho(IUNICHAR* ns,
IUNICHAR* sFuncName,
                              IUINT32 numArgs, INFA_EXPR_OPD_METADATA** inputArgList,
                              INFA_EXPR_OPD_METADATA* retValue);

//method to generate SQL code for pushdown optimization
```

```
extern "C" SAMPLE_EXPR_SPEC INFA_EXPR_STATUS pushdownFunctionEcho(IUNICHAR* sNameSpace,
                     IUNICHAR* sFuncName,
                     IUINT32 numArgs,
                     INFA_EXPR_OPD_METADATA** inputArgList,
                     EDatabaseType dbType,
                     EPushdownMode pushdownMode,
                     IUNICHAR** sGenSql);

// method to process row for Echo function
extern "C" SAMPLE_EXPR_SPEC INFA_EXPR_ROWSTATUS
processRowEcho(INFA_EXPR_FUNCTION_INSTANCE_HANDLE *fnInstance, IUNICHAR **errMsg);

// method to do module level initialization for Echo function
extern "C" SAMPLE_EXPR_SPEC INFA_EXPR_STATUS moduleInitEcho(INFA_EXPR_MODULE_HANDLE
*modHandle);

// method to do module level deinitialization for Echo function
extern "C" SAMPLE_EXPR_SPEC INFA_EXPR_STATUS moduleDeinitEcho(INFA_EXPR_MODULE_HANDLE
*modHandle);

// method to do function level initialization for Echo function
extern "C" SAMPLE_EXPR_SPEC INFA_EXPR_STATUS functionInitEcho(INFA_EXPR_FUNCTION_HANDLE
*funHandle);

// method to do function level deinitialization for Echo function
extern "C" SAMPLE_EXPR_SPEC INFA_EXPR_STATUS
functionDeinitEcho(INFA_EXPR_FUNCTION_HANDLE *funHandle);

// method to do function instance level initialization for Echo function
extern "C" SAMPLE_EXPR_SPEC INFA_EXPR_STATUS
functionInstInitEcho(INFA_EXPR_FUNCTION_INSTANCE_HANDLE *funInstHandle);

// method to do function instance level deinitialization for Echo function
extern "C" SAMPLE_EXPR_SPEC INFA_EXPR_STATUS
functionInstDeinitEcho(INFA_EXPR_FUNCTION_INSTANCE_HANDLE *funInstHandle);

/**
    These are all plugin callbacks, which have been implemented to get various module,
    function level interfaces
*/
// method to get plugin version
extern "C" SAMPLE_EXPR_SPEC INFA_EXPR_STATUS INFA_EXPR_GetPluginVersion(INFA_VERSION*
sdkVersion, INFA_VERSION* pluginVersion);

// method to delete the string allocated by this plugin. used for deleting the error
// messages
extern "C" SAMPLE_EXPR_SPEC void INFA_EXPR_DestroyString(IUNICHAR *);

// method to get validation interfaces
extern "C" SAMPLE_EXPR_SPEC INFA_EXPR_STATUS
INFA_EXPR_ValidateGetUserInterface( IUNICHAR* ns, IUNICHAR* sFuncName,
INFA_EXPR_VALIDATE_METHODS* functions);

// method to get module interfaces
extern "C" SAMPLE_EXPR_SPEC INFA_EXPR_STATUS
INFA_EXPR_ModuleGetUserInterface(INFA_EXPR_LIB_METHODS* functions);

// method to get function interfaces
extern "C" SAMPLE_EXPR_SPEC INFA_EXPR_STATUS
INFA_EXPR_FunctionGetUserInterface(IUNICHAR* nameSpaceName, IUNICHAR* functionName,
INFA_EXPR_FUNCTION_METHODS* functions);

// method to get function instance interfaces
extern "C" SAMPLE_EXPR_SPEC INFA_EXPR_STATUS
INFA_EXPR_FunctionInstanceGetUserInterface(IUNICHAR* nameSpaceName, IUNICHAR*
functionName, INFA_EXPR_FUNCTION_INSTANCE_METHODS* functions);

#endif
```

Schritt 3. Erstellen einer Implementierungsdatei

Die Implementierungsdatei enthält die Definitionen der Funktionen, aus denen Sie Ihre benutzerdefinierte Funktion erstellen. Erstellen Sie die Implementierungsdatei in C. Sie können eine einzige Implementierungsdatei für eine oder für mehrere benutzerdefinierte Funktionen verwenden. Sie können auch die Validierungs- und die Laufzeitfunktion einer benutzerdefinierten Funktion in einer einzigen Implementierungsdatei definieren.

Das folgende Beispiel zeigt die Implementierungsdatei „echo.c“ der benutzerdefinierten Funktion ECHO:

```
/*************************************************************************
 * ECHO function Procedure File
 *
 * This file contains code that creates the ECHO function, which the
 * Integration Service calls during a workflow.
 *************************************************************************/

/* Informatica ECHO function example developed using the Custom Function
 * API.

 * Filename: Echo.c

 * An example of a custom function developed using PowerCenter
 *
 * The purpose of the ECHO function is to return the input value to the user.
 *
 */

/*************************************************************************
                    Includes
 *************************************************************************/
#include <stdio.h>
#include <string.h>
#include "sdkexpr/exprsdk.h"

#define SAMPLE_EXPR_EXPORTS
#include "SampleExprPlugin.hpp"


static IUNICHAR ECHO_STR[80];


/*************************************************************************
                    Functions
 *************************************************************************/

/*************************************************************************
    Function: INFA_EXPR_GetPluginVersion

Description: Defines the version of the plug-in. It must be the same as the
Custom Function API version. Returns ISUCCESS if the plug-in version
matches the Custom Function API version.  Otherwise, returns IFAILURE.

Input:  sdkVersion - Current version of the Custom Function API.
Output: pluginVersion - Set the version of the plug-in.
Remarks: Custom Function API checks for compatibility between itself and the
plug-in version.
 *************************************************************************/

 extern "C" SAMPLE_EXPR_SPEC
  INFA_EXPR_STATUS INFA_EXPR_GetPluginVersion(INFA_VERSION* sdkVersion, INFA_VERSION*
pluginVersion)
{
    pluginVersion->m_major = 1;
    pluginVersion->m_minor = 0;
    pluginVersion->m_patch = 0;

    INFA_EXPR_STATUS retStatus;
```

```
    retStatus.status = ISUCCESS;
    retStatus.errMsg = NULL;
    return retStatus;
}

/***************************************************************************
    Function: INFA_EXPR_DestroyString

Description: Destroys all strings the plug-in returns. For example, it
destroys error messages or the return value of other function calls, such
as getFunctionDescription. Returns no value.

Input:  The pointer to the allocated string.
Output: N/A
Remarks: Frees the memory to avoid issues with multiple heaps.
 ***************************************************************************/

extern "C" SAMPLE_EXPR_SPEC void INFA_EXPR_DestroyString(IUNICHAR *strToDelete)
{
    delete [] strToDelete;
}


/***************************************************************************
    Function: INFA_EXPR_ValidateGetUserInterface

Description: Returns function pointers to the validation functions. Returns
ISUCCESS when the plug-in implemented the function. Returns IFAILURE
when the plug-in did not implement the function or another error occurred.

Input: Namespace and name of function.
Output: Functions. The plug-in needs to set various function pointers.
Remarks: Check the namespace and function name for validaity. Set the various
function pointers appropriately.
 ***************************************************************************/

extern "C" SAMPLE_EXPR_SPEC
    INFA_EXPR_STATUS INFA_EXPR_ValidateGetUserInterface(IUNICHAR* ns, IUNICHAR*
sFuncName,
                                                INFA_EXPR_VALIDATE_METHODS* functions)
{
    INFA_EXPR_STATUS retStatus;
    retStatus.errMsg = NULL;

    // check function name is not null
    if (!sFuncName)
    {
        retStatus.status = IFAILURE;
        return retStatus;
    }

    // set the appropriate function pointers
    functions->validateFunction = validateFunctionEcho;
    functions->getFunctionDescription = getDescriptionEcho;
    functions->getFunctionPrototype = getPrototypeEcho;
    functions->pushdownFunction = pushdownFunctionEcho;

    retStatus.status = ISUCCESS;
    return retStatus;
}

/***************************************************************************
    Function: INFA_EXPR_ModuleGetUserInterface

Description: Sets the function pointers for module-level interaction.
Returns ISUCCESS when functions pointers are set appropriately. Otherwise,
returns IFAILURE.

Input: N/A
Output: Functions. The plug-in needs to set various function pointers.
Remarks: Set the module init/deinit function pointers.
```

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

extern "C" SAMPLE_EXPR_SPEC
    INFA_EXPR_STATUS INFA_EXPR_ModuleGetUserInterface(INFA_EXPR_LIB_METHODS* functions)
{
    functions->module_init = moduleInitEcho;
    functions->module_deinit = moduleDeinitEcho;

    INFA_EXPR_STATUS retStatus;
    retStatus.status = ISUCCESS;
    retStatus.errMsg = NULL;
    return retStatus;
}

/***************************************************************************
    Function: INFA_EXPR_FunctionGetUserInterface

Description: Sets the function pointers for function-level interaction.
PowerCenter calls this function for every custom function this library
implements. Returns ISUCCESS when The plugin implements this function and
sets the function pointers correctly. Otherwise, returns IFAILURE.

Input:  Namespace and name of function.
Output: Functions. The plug-in needs to set function pointers for function
init/deinit.
Remarks: Set the function init/deinit function pointers.
 ***************************************************************************/

extern "C" SAMPLE_EXPR_SPEC
    INFA_EXPR_STATUS INFA_EXPR_FunctionGetUserInterface(IUNICHAR* nameSpaceName,
                                                IUNICHAR* functionName,
                                                INFA_EXPR_FUNCTION_METHODS* functions)
{
    functions->function_init = functionInitEcho;
    functions->function_deinit = functionDeinitEcho;

    INFA_EXPR_STATUS retStatus;
    retStatus.status = ISUCCESS;
    retStatus.errMsg = NULL;
    return retStatus;
}

/***************************************************************************
    Function: INFA_EXPR_FunctionInstanceGetUserInterface

Description: Sets the function pointers for function instance-level
interaction. PowerCenter calls this function for every custom function this
library implements. Returns ISUCCESS when The plugin implements this
function and sets the function pointers correctly. Otherwise, returns
IFAILURE.

Input:  Namespace and name of function.
Output: Functions. The plug-in needs to set function pointers for instance
init/deinit/processrow.
Remarks: Set the function instance init/deinit/processrow function pointers.
 ***************************************************************************/

extern "C" SAMPLE_EXPR_SPEC
    INFA_EXPR_STATUS INFA_EXPR_FunctionInstanceGetUserInterface(IUNICHAR* nameSpaceName,
                                                        IUNICHAR* functionName,

INFA_EXPR_FUNCTION_INSTANCE_METHODS* functions)
{
    functions->fnInstance_init = functionInstInitEcho;
    functions->fnInstance_processRow = processRowEcho;
    functions->fnInstance_deinit = functionInstDeinitEcho;

    INFA_EXPR_STATUS retStatus;
    retStatus.status = ISUCCESS;
    retStatus.errMsg = NULL;
    return retStatus;
```

```
}

/*************************************************************************
    Function: INFA_EXPR_getDescriptionEcho

Description: Gets the description of the ECHO function. It calls
destroyString to delete the arguments from memory when usage is complete.
The return value must be a null-terminated string.

Input:  Namespace and name of function.
Output: Description of the function.
Remarks: Returns the description of function. The Custom Functions API calls
destroy string to free the allocated memory.
 *************************************************************************/

extern "C" SAMPLE_EXPR_SPEC
    IUNICHAR * getDescriptionEcho(IUNICHAR* ns, IUNICHAR* sFuncName)
{
    static IUNICHAR *uniDesc = NULL;
    const char *description = "Echoes the input";

    if (uniDesc)
        return uniDesc;

    int i, len;
    len = strlen(description);
    uniDesc = new IUNICHAR[2*len+2];
    for (i=0; i<len; i++)
    {
        uniDesc[i] = description[i];
    }
    uniDesc[i] = 0;
    return uniDesc;
}

/*************************************************************************
    Function: INFA_EXPR_getPrototypeEcho

Description: Gets the arguments of the ECHO function in the Expression
Editor. It calls destroyString to delete the arguments from memory when
usage is complete. The return value must be a null-terminated string. The
function returns NULL if there is no value for the arguments.

Input:  Namespace and name of the function.
Output: Prototype of the function
Remarks: Returns the prototype of function.  The Custom Functions API calls
destroy string to free the allocated memory.
 *************************************************************************/

extern "C" SAMPLE_EXPR_SPEC
    IUNICHAR * getPrototypeEcho(IUNICHAR* ns, IUNICHAR* sFuncName)
{
    static IUNICHAR *uniProt = NULL;
    const char *prototype = "Echo(x), where x can be any type, returns x";

    if (uniProt)
        return uniProt;

    int i, len;
    len = strlen(prototype);
    uniProt = new IUNICHAR[2*len+2];
    for (i=0; i<len; i++)
    {
        uniProt[i] = prototype[i];
    }
    uniProt[i] = 0;
    return uniProt;
}


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

```
    Function: validateFunctionEcho

Description: Validates the arguments in the ECHO function. Provides the
name, datatype, precision, and scale of the arguments in the ECHO function.
Provides the datatype of the return value of the ECHO function. PowerCenter
calls this function once for each instance of the ECHO function used in a
mapping or workflow. Returns ISUCCESS when function usage is valid as per
the syntax.

The ECHO function takes exactly one argument of any datatype. The return
datatype is the same as the input datatype, because the function echoes the
input. Otherwise, returns IFAILURE.

Input: Namespace and name of the function, the number of arguments being
passed, and the metadata (datatype, scale, precision) of each argument.
Output: retValue. Set the metadata for return type.
Remarks: Called by the Custom Functions API to validate the usage of the
function and the input argument metadata to be passed. The plug-in needs
to verify the number of arguments for this function, the expected metadata
for each argument, etc. The plug-in can optionally change the expected datatype
of the input arguments. The plug-in needs to set the return type metadata.
The plugin can specify if the return value of this function is constant,
depending on whether or not all input arguments are constant.
 ***************************************************************************/

extern "C" SAMPLE_EXPR_SPEC
    INFA_EXPR_STATUS validateFunctionEcho(IUNICHAR* ns, IUNICHAR* sFuncName,
                                          IUINT32 numArgs,
                                          INFA_EXPR_OPD_METADATA** inputArgList,
                                          INFA_EXPR_OPD_METADATA* retValue)
{
    INFA_EXPR_STATUS exprStatus;

    // Check number of arguments.
    if (numArgs != 1)
    {
        static const char *err = "Echo function takes one argument.";
        IUNICHAR *errMsg = NULL;

        unsigned int len = strlen(err);
        errMsg = new IUNICHAR[2*len+2];
        unsigned int i;
        for (i=0; i<len; i++)
        {
            errMsg[i] = err[i];
        }
        errMsg[i] = 0;

        exprStatus.status = IFAILURE;
        exprStatus.errMsg = errMsg;
        return exprStatus;
    }


    // This is an echo function.
    // It returns the input value.
    retValue->datatype = inputArgList[0]->datatype;
    retValue->precision = inputArgList[0]->precision;
    retValue->scale = inputArgList[0]->scale;

    // If the input value is constant,
    // the return value is also constant.
    if (inputArgList[0]->isValueConstant)
        retValue->isValueConstant = ITRUE;
    else
        retValue->isValueConstant = IFALSE;


    exprStatus.status = ISUCCESS;
    return exprStatus;
}
```

```
/*************************************************************************
    Function: processRowEcho

Description: Called when an input row is available to an ECHO function
instance. The data for the input arguments of the ECHO function is bound and
accessed through fnInstance-inputOPDHandles. Set the data, length, and
indicator for the output and return ports in fnInstance->retHandle.
PowerCenter calls the function-level initialization function before calling
this function.

Returns INFA_ROWSUCCESS when the function successfully processes the row of
data. Returns INFA_ROWERROR when the function encounters an error for the row
of data. The Integration Service increments the internal error count.
Only returns this value when the data access mode is row. Returns
INFA_FATALERROR when the function encounters a fatal error for the row of
data or the block of data. The Integration Service fails the session.

Input:  Function instance handle, which has the input data.
Output: return value
Remarks: The plug-in needs to get various input arguments from the function
instance handle, perform calculations, and set the return value.
 *************************************************************************/

extern "C" SAMPLE_EXPR_SPEC
    INFA_EXPR_ROWSTATUS processRowEcho(INFA_EXPR_FUNCTION_INSTANCE_HANDLE *fnInstance,
IUNICHAR **errMsg)
{

    INFA_EXPR_OPD_RUNTIME_HANDLE* arg1 = fnInstance->inputOPDHandles[0];
    INFA_EXPR_OPD_RUNTIME_HANDLE* retHandle = fnInstance->retHandle;

    // Check if the input argument has a null indicator.
    // If yes, the return value is also null.
    if (INFA_EXPR_GetIndicator(arg1) == INFA_EXPR_NULL_DATA)
    {
        INFA_EXPR_SetIndicator(retHandle, INFA_EXPR_NULL_DATA);
        return INFA_EXPR_SUCCESS;
    }

    short sval;
    long lval;
    int ival;
    char *strval;
    IUNICHAR *ustrval;
    void *rawval;
    float fval;
    double dval;
    INFA_EXPR_DATE *infaDate = NULL;
    int len;

    // Depending on the datatype,
    // get the input argument
    // and set the same value in the return value.
    // Also, set the same indicator.
    switch (arg1->pOPDMetadata->datatype)
    {
        case eCTYPE_SHORT:
            sval = INFA_EXPR_GetShort(arg1);
            INFA_EXPR_SetShort(retHandle, sval);
            INFA_EXPR_SetIndicator(retHandle, INFA_EXPR_GetIndicator(arg1));
            break;

        case eCTYPE_LONG:
        case eCTYPE_LONG64:
            lval = INFA_EXPR_GetLong(arg1);
            INFA_EXPR_SetLong(retHandle, lval);
            INFA_EXPR_SetIndicator(retHandle, INFA_EXPR_GetIndicator(arg1));
            break;

        case eCTYPE_INT32:
```

```
            ival = INFA_EXPR_GetInt(arg1);
            INFA_EXPR_SetInt(retHandle, ival);
            INFA_EXPR_SetIndicator(retHandle, INFA_EXPR_GetIndicator(arg1));
            break;

        case eCTYPE_CHAR:
            strval = INFA_EXPR_GetString(arg1);
            len = INFA_EXPR_GetLength(arg1);
            strcpy((char *)retHandle->pUserDefinedPtr, strval);
            INFA_EXPR_SetString(retHandle, retHandle->pUserDefinedPtr);
            INFA_EXPR_SetLength(retHandle, INFA_EXPR_GetLength(arg1));
            INFA_EXPR_SetIndicator(retHandle, INFA_EXPR_GetIndicator(arg1));
            break;

        case eCTYPE_RAW:
            rawval = INFA_EXPR_GetRaw(arg1);
            len = INFA_EXPR_GetLength(arg1);
            memcpy(retHandle->pUserDefinedPtr, rawval, len);
            INFA_EXPR_SetRaw(retHandle, retHandle->pUserDefinedPtr);
            INFA_EXPR_SetLength(retHandle, len);
            INFA_EXPR_SetIndicator(retHandle, INFA_EXPR_GetIndicator(arg1));
            break;

        case eCTYPE_UNICHAR:
            ustrval = INFA_EXPR_GetUniString(arg1);
            len = INFA_EXPR_GetLength(arg1);
            memcpy(retHandle->pUserDefinedPtr, ustrval, 2*(len+1));
            INFA_EXPR_SetUniString(retHandle, retHandle->pUserDefinedPtr);
            INFA_EXPR_SetLength(retHandle, len);
            INFA_EXPR_SetIndicator(retHandle, INFA_EXPR_GetIndicator(arg1));
            break;

        case eCTYPE_TIME:
            infaDate = INFA_EXPR_GetDate(arg1);
            *((INFA_EXPR_DATE *)retHandle->pUserDefinedPtr) = *infaDate;
            INFA_EXPR_SetDate(retHandle, retHandle->pUserDefinedPtr);
            INFA_EXPR_SetIndicator(retHandle, INFA_EXPR_GetIndicator(arg1));
            break;

        case eCTYPE_FLOAT:
            fval = INFA_EXPR_GetFloat(arg1);
            INFA_EXPR_SetFloat(retHandle, fval);
            INFA_EXPR_SetIndicator(retHandle, INFA_EXPR_GetIndicator(arg1));
            break;

        case eCTYPE_DOUBLE:
            dval = INFA_EXPR_GetDouble(arg1);
            INFA_EXPR_SetDouble(retHandle, dval);
            INFA_EXPR_SetIndicator(retHandle, INFA_EXPR_GetIndicator(arg1));
            break;

        default:
            return INFA_EXPR_ROWERROR;
            break;
    }
    return INFA_EXPR_SUCCESS;
}

/*************************************************************************
    Function: moduleInitEcho

Description: Called once for each module to initialize any global data
structure in the function. Called before calling any function-level
functions. Returns ISUCCESS when module initialization is successful.
Otherwise, returns IFAILURE.

Input:  module handle
Output: status
Remarks: The plug-in can optionally implement this method for one-time
initialization.
 *************************************************************************/
```

```
extern "C" SAMPLE_EXPR_SPEC
    INFA_EXPR_STATUS moduleInitEcho(INFA_EXPR_MODULE_HANDLE *modHandle)
{
    INFA_EXPR_STATUS exprStatus;

    // initialize the ECHO_STR
    const char *fnName = "Echo";
    int len = strlen(fnName);
    int i;
    for (i=0;i<len;i++)
        ECHO_STR[i] = fnName[i];

    exprStatus.status = ISUCCESS;
    return exprStatus;
}

/*************************************************************************
    Function: moduleDeinitEcho

Description: Called once for each module to deinitialize any data structure
in this function. Called after all function-level interactions are complete.
Returns ISUCCESS when module deinitialization is successful. Otherwise,
returns IFAILURE.

Input:  module handle
Output: status
Remarks: The plug-in can optionally implement this method for one-time
deinitialization.
 *************************************************************************/

extern "C" SAMPLE_EXPR_SPEC
    INFA_EXPR_STATUS moduleDeinitEcho(INFA_EXPR_MODULE_HANDLE *modHandle)
{
    INFA_EXPR_STATUS exprStatus;
    exprStatus.status = ISUCCESS;
    return exprStatus;

}

/*************************************************************************
    Function: functionInitEcho

Description: Called once for each custom function to initialize any
structure related to the custom function. Module-level initialization
function is called before this function. Returns ISUCCESS when function
init is successful. Otherwise, returns IFAILURE.

Input:  function handle
Output: status
Remarks: The plug-in can optionally implement this method for one-time function
initialization.
 *************************************************************************/

extern "C" SAMPLE_EXPR_SPEC
    INFA_EXPR_STATUS functionInitEcho(INFA_EXPR_FUNCTION_HANDLE *funHandle)
{
    INFA_EXPR_STATUS exprStatus;
    exprStatus.status = ISUCCESS;
    return exprStatus;

}

/*************************************************************************
    Function: functionDeinitEcho

Description: Called once for each function level to deinitialize any
structure related to the ECHO function. Returns ISUCCESS when function deinit
is successful. Otherwise, returns IFAILURE.

Input:  function handle
```

```
Output: status
Remarks: The plug-in can optionally implement this method for one-time function
deinitialization.
 ***************************************************************************/

extern "C" SAMPLE_EXPR_SPEC
    INFA_EXPR_STATUS functionDeinitEcho(INFA_EXPR_FUNCTION_HANDLE *funHandle)
{
    INFA_EXPR_STATUS exprStatus;
    exprStatus.status = ISUCCESS;
    return exprStatus;

}

/***************************************************************************
    Function: functionInstInitEcho

Description: Called once for each custom function instance to initialize
any structure related to the an instance of the ECHO function. If there are
two instances of ECHO in a mapping or workflow, PowerCenter calls this
function twice. PowerCenter calls the module-level initialization function
before calling this function.  Returns ISUCCESS when function instance
initialization is successful. Otherwise, returns IFAILURE.

Input:  function instance handle
Output: status
Remarks: The plug-in can optionally implement this method for one-time
function instance initialization.
 ***************************************************************************/

extern "C" SAMPLE_EXPR_SPEC
    INFA_EXPR_STATUS functionInstInitEcho(INFA_EXPR_FUNCTION_INSTANCE_HANDLE
*funInstHandle)
{
    INFA_EXPR_STATUS exprStatus;
    exprStatus.status = ISUCCESS;

    INFA_EXPR_OPD_RUNTIME_HANDLE *retHandle = funInstHandle->retHandle;

    // Allocate memory depending on the datatype.
    if (retHandle->pOPDMetadata->datatype == eCTYPE_CHAR)
        retHandle->pUserDefinedPtr = new char[retHandle->pOPDMetadata->precision+1];
    else if (retHandle->pOPDMetadata->datatype == eCTYPE_UNICHAR)
        retHandle->pUserDefinedPtr = new IUNICHAR[retHandle->pOPDMetadata->precision+1];
    else if (retHandle->pOPDMetadata->datatype == eCTYPE_RAW)
        retHandle->pUserDefinedPtr = new unsigned char[retHandle->pOPDMetadata-
>precision];
    else if (retHandle->pOPDMetadata->datatype == eCTYPE_TIME)
        retHandle->pUserDefinedPtr = new INFA_EXPR_DATE();
    return exprStatus;
}

/***************************************************************************
    Function: functionInstDeinitEcho

Description: Called once for each function level during deinitialization.
Can deinitialize any structure related to the ECHO function. Returns ISUCCESS
when deinitialization is successful. Otherwise, returns IFAILURE.

Input:  function instance handle
Output: status
Remarks: The plug-in can optionally implement this method for one-time
function instance deinitialization.
 ***************************************************************************/

extern "C" SAMPLE_EXPR_SPEC INFA_EXPR_STATUS
functionInstDeinitEcho(INFA_EXPR_FUNCTION_INSTANCE_HANDLE *funInstHandle)
{
    INFA_EXPR_STATUS exprStatus;
    exprStatus.status = ISUCCESS;
    INFA_EXPR_OPD_RUNTIME_HANDLE *retHandle = funInstHandle->retHandle;
```

```
    if (retHandle->pOPDMetadata->datatype == eCTYPE_CHAR)
        delete [] (char *)retHandle->pUserDefinedPtr;
    else if (retHandle->pOPDMetadata->datatype == eCTYPE_UNICHAR)
        delete [] (IUNICHAR *)retHandle->pUserDefinedPtr;
    else if (retHandle->pOPDMetadata->datatype == eCTYPE_RAW)
        delete [] (unsigned char *)retHandle->pUserDefinedPtr;
    else if (retHandle->pOPDMetadata->datatype == eCTYPE_TIME)
        delete (INFA_EXPR_DATE *)retHandle->pUserDefinedPtr;
    return exprStatus;
}


/*************************************************************************
    Function: pushdownFunctionEcho

Description: Method to generate SQL code for pushdown optimization.

Input:  Namespace and name of the function, the number of arguments being passed,
        and the metadata (datatype, scale, precision) of each argument, database type,
        Pushdown mode.
Output: Generated SQL.
Remarks: The plug-in can optionally implement this method to enable pushdown
optimization.
 *************************************************************************/

//method to generate SQL code for pushdown optimization
extern "C" SAMPLE_EXPR_SPEC INFA_EXPR_STATUS pushdownFunctionEcho(IUNICHAR* sNameSpace,
                     IUNICHAR* sFuncName,
                     IUINT32 numArgs,
                     INFA_EXPR_OPD_METADATA** inputArgList,
                     EDatabaseType dbType,
                     EPushdownMode pushdownMode,
                     IUNICHAR** sGenSql)
{
    INFA_EXPR_STATUS retStatus;
    static const char *sql_str = "{1}";

    // Construct the SQL: "{1}"
    unsigned int len = strlen(sql_str);

    IUNICHAR *pGenSql = new IUNICHAR[len+1];
    unsigned int i;

    for (i=0; i<len; i++)
    {
        pGenSql[i] = sql_str[i];
    }

    pGenSql[len] = 0;

    // Return the generated SQL
    *sGenSql = pGenSql;

    retStatus.status = ISUCCESS;
    retStatus.errMsg = NULL;
    return retStatus;
}
```

Schritt 4. Erstellen der Module

Sie können die Module in Windows oder UNIX erstellen. Erstellen Sie ein Modul für jede Plattform, auf der PowerCenter ausgeführt wird. Sie müssen auf jeden Fall ein Modul in Windows erstellen, da der PowerCenter-

Client auf Windows ausgeführt wird. Möglicherweise müssen Sie auch Module in UNIX oder Linux erstellen, je nach dem Knoten, auf dem sich Integration Service befindet.

In der folgenden Tabelle finden Sie eine Auflistung der Bibliotheksdateinamen für jede Plattform zum Erstellen der Module:

Plattform	Moduldateiname
Windows	<module_identifier>.dll
AIX	lib<module_identifier>.a
HP-UX	lib<module_identifier>.sl
Linux	lib<module_identifier>.so
Solaris	lib<module_identifier>.so

Die Module werden in der Repository-Plug-In-XML-Datei deklariert.

Verwandte Themen:

- "Schritt 5. Erstellen der Repository-Plug-In-Datei" auf Seite 248

Erstellen der Module in Windows

In Windows können Sie das Modul in Microsoft Visual C++ erstellen.

So erstellen Sie das Modul in Windows:

1. Starten Sie Visual C++.
2. Klicken Sie auf „Datei > Neu"
3. Klicken Sie im Dialogfeld „Neu" auf die Registerkarte „Projekte" und wählen Sie „Win32 Dynamic-Link Library".
4. Geben Sie ihren Speicherort ein.

   Für das ECHO-Beispiel geben Sie folgendes Verzeichnis an:

   ```
   <Informatica Development Platform installation directory>\CustomFunctionAPI\samples
   \echo
   ```

5. Geben Sie den Namen des Projekts ein.

   Als Projektnamen müssen Sie den Modulnamen verwenden, der für die benutzerdefinierte Funktion angegeben wurde. Für das ECHO-Beispiel geben Sie „EchoDemo" ein:

6. Klicken Sie auf „OK".

   Visual C++ öffnet einen Assistenten zum Definieren der Projektkomponenten.

7. Wählen Sie im Assistenten ein leeres DLL-Projekt aus und klicken Sie auf „Fertigstellen". Klicken Sie im Dialogfeld der Informationen zum neuen Projekt auf „OK".

   Visual C++ erstellt die Projektdateien im angegebenen Verzeichnis.

8. Klicken Sie auf „Projekt > Zu Projekt hinzufügen > Dateien".
9. Gehen Sie im Verzeichnis eine Ebene nach oben. Dieses Verzeichnis enthält die Prozedurdateien, die Sie erstellt haben. Wählen Sie alle Dateien mit der Erweiterung „*.c" und klicken Sie auf „OK".

   Für das ECHO-Beispiel fügen Sie die Datei „echo.c" hinzu.

10. Klicken Sie auf „Projekt > Einstellungen".

11. Klicken Sie auf die Registerkarte „C/C++“ und wählen Sie im Feld „Kategorie“ den Eintrag „Präprozessor“.
12. Geben Sie im Feld der zusätzlich einzuschließenden Verzeichnisse folgenden Pfad ein und klicken Sie auf „OK“:

    ```
    ..; <Informatica Development Platform installation directory>\CustomFunctionAPI
    \samples\echo; ...
    ```

13. Klicken Sie auf „Erstellen > Erstellen < Modulname>.dll“ oder drücken Sie F7, um das Projekt zu erstellen.

    Visual C++ erstellt die DLL und fügt sie in das Debug- oder Versionsverzeichnis im Projektverzeichnis ein.

Erstellen der Module in UNIX

In UNIX können Sie einen beliebigen C-Compiler zum Erstellen des Moduls verwenden.

So erstellen Sie das Modul in UNIX:

1. Setzen Sie die Umgebungsvariable INFA_HOME auf das Installationsverzeichnis von PowerCenter-Integrationsdienst.

   Hinweis: Wenn Sie einen falschen Verzeichnispfad für die Umgebungsvariable INFA_HOME angeben, kann PowerCenter-Integrationsdienst nicht gestartet werden.

   Starten Sie den Knoten neu, um die Änderungen zu übernehmen.
2. Geben Sie einen Befehl aus der folgenden Tabelle ein, um das Projekt zu erstellen.

UNIX-Version	Befehl
AIX (32 Bit)	make -f makefile.aix
AIX (64 Bit)	make -f makefile.aix64
HP-UX (32 Bit)	make -f makefile.hp
HP-UX (64 Bit)	make -f makefile.hp64
HP-UX PA-RISC	make -f makefile.hpparisc64
Linux	make -f makefile.linux
Solaris	make -f makefile.sol

Schritt 5. Erstellen der Repository-Plug-In-Datei

Erstellen Sie eine XML-Datei, um die Funktionsmetadaten einer oder mehrerer benutzerdefinierter Funktionen zu definieren. Verwenden Sie die Struktur der Plug-In-DTD-Datei beim Erstellen oder Ändern der Plug-In-XML-Datei. Die Plug-In-DTD-Datei (plugin.dtd) befindet sich im Verzeichnis des PowerCenter-Client. Sie können jedes beliebige Tool verwenden, mit dem Sie eine XML-Datei erstellen können. Beim Erstellen der Repository-Plug-In-Datei geben Sie ihr einen neuen Namen.

Folgende Abbildung zeigt die Struktur von „plugin.dtd“:

```
POWERMART
└ REPOSITORY
  └─ PLUGIN
     └ FUNCTION_GROUP
       ├─ FUNCTION
       └─ LIBRARY
```

Das Element PLUGIN

Verwenden Sie das Element PLUGIN zum Definieren der Metadaten für das Plug-In, das Sie erstellen möchten. Die Attribute des Elements PLUGIN dienen zur eindeutigen Identifizierung der Plug-In-Metadaten.

Die folgende Tabelle zeigt die Attribute des Elements PLUGIN:

Attribut	Erforderlich / Optional	Beschreibung
NAME	Erforderlich	Name des Plug-In; der Plug-In-Name wird auf der Registerkarte „Plug-In“ in PowerCenter Repository Service angezeigt.
ID	Erforderlich	ID des Plug-In; damit können sie zwischen verschiedenen Plug-Ins mit derselben VENDORID unterscheiden.
VENDORNAME	Erforderlich	Name des Lieferanten; der Lieferantenname wird auf der Registerkarte „Plug-In“ in PowerCenter Repository Service angezeigt.
VENDORID	Erforderlich	Lieferanten-ID; holen Sie sich die Lieferanten-ID von Informatica, wenn Sie benutzerdefinierte Funktionen zur Verteilung außerhalb Ihres Unternehmens entwickeln. Weitere Informationen hierzu finden Sie unter “Schritt 1. Abrufen von Repository-ID-Attributen” auf Seite 234.
DESCRIPTION	Optional	Beschreibung des Plug-In; die Plug-In-Beschreibung wird auf der Registerkarte „Plug-In“ in PowerCenter Repository Service angezeigt.
VERSION	Erforderlich	Version des Plug-In; dient zum Verfolgen der Aktualisierungen der Plug-In-Metadaten.

Das Element FUNCTION_GROUP

Mit dem Element FUNCTION_GROUP definieren Sie die Gruppe, der die benutzerdefinierte Funktion angehört.

Die folgende Tabelle zeigt die Attribute des Elements FUNCTION_GROUP:

Attribut	Erforderlich/ Optional	Beschreibung
NAME	Erforderlich	Name der benutzerdefinierten Funktionsgruppe, die Sie definieren möchten; der Funktionsgruppenname wird auf der Registerkarte „Plug-In“ in PowerCenter Repository Service angezeigt.
ID	Erforderlich	ID der Funktionsgruppe; holen Sie sich die Funktionsgruppen-ID von Informatica, wenn Sie benutzerdefinierte Funktionen zur Verteilung außerhalb Ihres Unternehmens entwickeln. Weitere Informationen hierzu finden Sie unter “Schritt 1. Abrufen von Repository-ID-Attributen” auf Seite 234. die Funktionsgruppen-ID wird auf der Registerkarte „Plug-In“ in PowerCenter Repository Service angezeigt.
COMPONENTVERSION	Erforderlich	Versionsnummer der Funktionsgruppe; dient zum Verfolgen der Aktualisierungen des Elements FUNCTION_GROUP.
DESCRIPTION	Optional	Beschreibung der Funktionsgruppe; die Funktionsgruppenbeschreibung wird auf der Registerkarte „Plug-In“ in PowerCenter Repository Service angezeigt.
NAMESPACE	Erforderlich	Namespace der Funktionsgruppe; der Ausdruckseditor zeigt benutzerdefinierte Funktionen mit dem Namespace in einem eigenen Ordner auf der Registerkarte „Funktionen“ an. Namespaces unterscheiden nicht zwischen Groß- und Kleinschreibung. Der Namespace „infa" darf nicht verwendet werden. Es ist reserviert. Der Namespace darf auch nicht leer sein.

Festlegen eines Namespace

Sie können einen Namespace für alle Funktionen wählen, die Sie erstellen. Der Namespace darf jedoch nicht in Konflikt mit den Namespaces der benutzerdefinierten Funktionen anderer Anbieter stehen. Sie müssen daher einen eindeutigen Namespace wählen. Verwenden Sie z. B. einen Namespace, der direkt mit Ihrem Unternehmen verbunden ist, etwa sein Börsenkürzel.

Das Element FUNCTION

Mit dem Element FUNCTION definieren Sie die Eigenschaften der benutzerdefinierten Funktion.

Die folgende Tabelle zeigt die Attribute des Elements FUNCTION:

Attribut	Erforderlich/ Optional	Beschreibung
NAME	Erforderlich	Name der Drittanbieter-Funktion, die Sie definieren möchten
ID	Erforderlich	ID des Elements FUNCTION; identifiziert die Funktion. Holen Sie sich die Funktions-ID von Informatica, wenn Sie benutzerdefinierte Funktionen zur Verteilung außerhalb Ihres Unternehmens entwickeln. Weitere Informationen hierzu finden Sie unter "Schritt 1. Abrufen von Repository-ID-Attributen" auf Seite 234.
FUNCTION_CATEGORY	Optional	Kategorie der Funktion, die Sie definieren möchten; verwenden Sie eine der folgenden Kategorien: - Zeichen - Umwandlung - Datenbereinigung - Datum - Numerisch - Wissenschaftlich - Spezial - Test Im Ausdruckseditor wird die benutzerdefinierte Funktion in dieser Kategorie angezeigt.

Das Element LIBRARY

Mit dem Element LIBRARY geben Sie die kompilierten gemeinsam genutzten Bibliotheken für die benutzerdefinierte Funktion an.

Die folgende Tabelle zeigt die Attribute des Elements LIBRARY:

Attribut	Erforderlich/ Optional	Beschreibung
NAME	Erforderlich	Name der kompilierten gemeinsam genutzten Bibliothek
OSTYPE	Erforderlich	Betriebssystem, für das die gemeinsam genutzte Bibliothek kompiliert wurde
TYPE	Erforderlich	Typ der gemeinsam genutzten Bibliothek; geben Sie einen der folgenden Typen an: - VALIDATION: die Bibliothek, die der PowerCenter-Client verwendet, um die Beschreibung der benutzerdefinierten Funktion abzurufen und den Funktionsaufruf zu validieren, z. B. den Rückgabetyp und die Anzahl von Argumenten - SERVER: die Bibliothek, die PowerCenter Integration Service zum Ausführen des Funktionsaufrufs verwendet

Plug-In-XML-Beispieldatei

Das folgende Beispiel zeigt die Repository-Plug-In-Datei, in der die benutzerdefinierte Funktion ECHO definiert wird:

```
<?xml version="1.0" encoding="us-ascii"?>
<!DOCTYPE POWERMART SYSTEM "plugin.dtd">
<POWERMART>
    <REPOSITORY CODEPAGE="us-ascii">
        <PLUGIN NAME="Echo" ID="506001" VENDORNAME="Informatica"
            VENDORID="1"
            DESCRIPTION="Plugin for Expressions from Informatica">
            <FUNCTION_GROUP ID="506002" NAME="INFA Function Group1"
                COMPONENTVERSION="1.0.0"
                DESCRIPTION="The functions group for my own Echo function"
                NAMESPACE="">
               <FUNCTION ID="506004" NAME="ECHO" FUNCTION_CATEGORY="Data Cleansing"/>
               <LIBRARY NAME="pmecho.dll" OSTYPE="NT" TYPE="VALIDATION"/>
               <LIBRARY NAME="llibpmecho.sl" OSTYPE="HPUX" TYPE="VALIDATION"/>
               <LIBRARY NAME="libpmecho.so" OSTYPE="SOLARIS" TYPE="VALIDATION"/>
               <LIBRARY NAME="libpmecho.so " OSTYPE="LINUX" TYPE="VALIDATION"/>
               <LIBRARY NAME="libpmecho.a" OSTYPE="AIX" TYPE="VALIDATION"/>
               <LIBRARY NAME="pmecho.dll" OSTYPE="NT" TYPE="SERVER"/>
               <LIBRARY NAME="libpmecho.sl" OSTYPE="HPUX" TYPE="SERVER"/>
               <LIBRARY NAME="libpmecho.so" OSTYPE="SOLARIS" TYPE="SERVER"/>
               <LIBRARY NAME="libpmecho.so" OSTYPE="LINUX" TYPE="SERVER"/>
               <LIBRARY NAME="libpmecho.a" OSTYPE="AIX" TYPE="SERVER"/>
</FUNCTION_GROUP>
        </PLUGIN>
    </REPOSITORY>
</POWERMART>
```

Schritt 6. Testen von benutzerdefinierten Funktionen

Während der Entwicklung können Sie die benutzerdefinierte Funktionen auch testen. Führen Sie folgende Aufgaben aus, um benutzerdefinierte Funktionen in PowerCenter zu testen:

- Validieren Sie die Repository-Plug-In-XML-Datei.
- Stellen Sie sicher, dass die benutzerdefinierten Funktionen in einem Ausdruck präzise Daten liefern.

Zum Testen der benutzerdefinierten Funktionen müssen Sie sie in einer PowerCenter-Umgebung installieren.

Validieren der Repository-Plug-In-Datei

Sie können die Repository-Plug-In-Datei validieren, indem Sie sie in einem PowerCenter-Repository registrieren. Beim Registrieren eines Plug-In wird die Struktur der Datei von der zugehörigen DTD-Datei namens „plugin.dtd“ validiert. Die Datei muss der Struktur der zugeordneten Datei „plugin.dtd“ entsprechen. Die Datei „plugin.dtd“ befindet sich im Verzeichnis des PowerCenter-Client.

Beim Entwickeln von benutzerdefinierten Funktionen können Sie ein Repository-Plug-In erstellen und es registrieren, bevor Sie die Header- und Implementierungsdateien für die Funktionen fertigstellen. Beim Registrieren der Datei fügen Sie Metadaten der benutzerdefinierten Funktion hinzu, z. B. Plug-In-ID, Namespace und Funktionsnamen. Damit werden die betreffenden Daten im Repository reserviert.

Nach der Registrierung der Repository-Plug-In-Datei können Sie mit der Entwicklung von benutzerdefinierten Funktionen fortfahren. Wenn Sie fertig sind, registrieren Sie die Repository-Plug-In-Datei erneut, um die Metadaten im Repository zu aktualisieren.

Nach der Registrierung des Repository-Plug-In können Sie die Plug-In-Metadaten anzeigen.

Anzeigen der Plug-In-Metadaten

Die folgende Tabelle zeigt die Metadaten auf der Registerkarte „Plug-Ins“ in Informatica Administrator:

Repository Service, Attribut	XML, Element und Attribut
Name	PLUGIN NAME
Lieferantenname	PLUGIN VENDORNAME
Beschreibung	PLUGIN DESCRIPTION
Gruppenname	FUNCTION_GROUP NAME
Gruppen-ID	FUNCTION_GROUP ID
Gruppenbeschreibung	FUNCTION_GROUP DESCRIPTION

Überprüfen der Funktionsgenauigkeit

Um die Genauigkeit einer benutzerdefinierten Funktion zu prüfen, erstellen Sie einen Ausdruck mit der Funktion und nehmen ihn in ein Mapping und einen Arbeitsablauf auf. Führen Sie die folgenden Schritte zum Überprüfen der Genauigkeit einer benutzerdefinierten Funktion aus:

1. Erstellen Sie die Testdaten.
2. Erstellen Sie ein Mapping.
3. Fügen Sie die benutzerdefinierte Funktion einem Ausdruck im Mapping hinzu.
4. Erstellen Sie ein Mapping.
5. Führen Sie den Debugger aus (optional). Alternativ können Sie auch eine Sitzung und einen Arbeitsablauf für das Mapping erstellen.
6. Führen Sie den Arbeitsablauf aus.
7. Zeigen Sie die Ergebnisse an.

Installieren von benutzerdefinierten Funktionen

Führen Sie die folgenden Schritte zum Installieren der benutzerdefinierten Funktionen aus:

1. Kopieren Sie die Bibliotheken der benutzerdefinierten Funktion in die PowerCenter-Umgebung.
2. Registrieren Sie das Repository-Plug-In.

Nachdem Sie die benutzerdefinierten Funktionen installiert haben, verwenden Sie sie in Umwandlungs- und Arbeitsablaufsausdrücken.

Schritt 1. Kopieren benutzerdefinierter Funktionsbibliotheken nach PowerCenter

Kopieren Sie die Bibliotheken der benutzerdefinierten Funktion und die Repository-Plug-In-XML-Datei in die Verzeichnisse von PowerCenter-Client und Integration Service. Anweisungen dazu erhalten Sie vom Entwickler der benutzerdefinierten Funktion.

Bei hoher Verfügbarkeit oder Ausführung von Sitzungen in einem Gitter legen Sie die Bibliotheken an einem gemeinsamen Speicherort ab und setzen ihn als erforderliche Ressource fest.

Schritt 2. Registrieren des Plug-In

Registrieren Sie die Repository-Plug-In-XML-Datei über das Administrator-Tool.

Erstellen von Ausdrücken mit benutzerdefinierten Funktionen

Benutzerdefinierte Funktionen können einem Ausdruck hinzugefügt werden. Wenn Sie eine benutzerdefinierte Funktion bei der manuellen Erstellung eines Ausdrucks einfügen, muss ihr der Namespace vorangestellt werden, den Sie vom Entwickler der benutzerdefinierten Funktion erhalten. Beim Erstellen eines Ausdrucks im Ausdrucksseditor werden benutzerdefinierte Funktionen in der Liste aller Funktionen zusammen mit dem jeweiligen Funktionstyp angezeigt. Sie verwenden benutzerdefinierte Funktionen genau gleich wie alle anderen Funktionen.

Beim Validieren des Ausdrucks wird die benutzerdefinierte Funktion in Designer oder Workflow Manager nicht validiert. Nur der Ausdruck selbst wird validiert. Die benutzerdefinierte Funktion wird über das Plug-In validiert.

KAPITEL 8

Referenz für die API für benutzerdefinierte Funktionen

Dieses Kapitel umfasst die folgenden Themen:

- Referenz für die API für benutzerdefinierte Funktionen - Übersicht, 255
- Allgemeine APIs, 255
- Laufzeit-APIs, 260

Referenz für die API für benutzerdefinierte Funktionen - Übersicht

Mit der API für benutzerdefinierte Funktionen können Sie benutzerdefinierte Funktionen zur Verwendung in Umwandlungs- oder Arbeitsablaufausdrücken entwickeln. Die API für benutzerdefinierte Funktionen ist ein Framework zum Erstellen von benutzerdefinierten Funktionen. Sie enthält gemeinsame und Laufzeit-APIs. Die APIs ermöglichen PowerCenter, Ausdrücke mit benutzerdefinierten Funktionen zu validieren und in Arbeitsabläufen zu verwenden.

Verwenden Sie die APIs in Header- und Implementierungsdateien zum Entwickeln von benutzerdefinierten Funktionen. Mit den Header- und Implementierungsdateien müssen Sie gemeinsam genutzte Bibliotheken erstellen. Die gemeinsam genutzten Bibliotheken spezifizieren Sie in einer Repository-Plug-In-Datei, die Sie in PowerCenter registrieren. Sie kopieren die gemeinsam genutzten Bibliotheken auch in die PowerCenter-Umgebung.

Allgemeine APIs

PowerCenter Client Integration Service und Repository Service rufen die gemeinsamen APIs auf, um Ausdrücke zu validieren und Funktionsrückgaben, -beschreibungen und -Prototypen nach Verwendung aus dem Speicher zu löschen.

Die gemeinsamen APIs weisen folgende Struktur auf:

```
INFA_EXPR_VALIDATE_METHODS
└─ INFA_EXPR_ValidateGetUserInterface()
   ├── validateFunction
   ├── getFunctionDescription
   └── getFunctionPrototype
INFA_EXPR_OPD_METADATA
INFA_EXPR_GetPluginVersion
INFA_EXPR_DestroyString
```

Validierungs-Handle

Das Handle INFA_EXPR_VALIDATE_METHODS ist ein Validierungs-Handle. PowerCenter ruft INFA_EXPR_ValidateGetUserInterface auf, um Funktionszeiger in diesem Validierungs-Handle abzurufen.

Validierungsfunktion für Schnittstellen

Wenn PowerCenter INFA_EXPR_ValidateGetUserInterface aufruft, gibt das Plug-In die Funktionszeiger für die Validierungsfunktionen zurück.

Verwenden Sie folgende Syntax:

```
INFA_EXPR_STATUS INFA_EXPR_ValidateGetUserInterface( IUNICHAR* sNamespace, IUNICHAR*
sFuncName, INFA_EXPR_VALIDATE_METHODS* functions);
```

Argument	Datentyp	Eingabe / Ausgabe	Beschreibung
sNamespace	IUNICHAR	Eingabe	Namespace der benutzerdefinierten Funktion
sFuncName	IUNICHAR	Eingabe	Name der benutzerdefinierten Funktion
Funktionen	INFA_EXPR_VALIDATE_METHODS	Ausgabe	Zeiger für andere, während der Validierung und Berichtserstellung aufgerufene Funktionen;

Rückgabe-Datentyp ist `INFA_EXPR_STATUS`. Verwenden Sie ISUCCESS und IFAILURE als Rückgabewert. Wenn die Funktion IFAILURE zurückgibt, hat das Plug-In die Funktion nicht implementiert oder es ist ein anderer Fehler aufgetreten.

INFA_EXPR_ValidateGetUserInterface gibt die folgenden Funktionen zurück:

- **validateFunction:** Validiert eine benutzerdefinierte Funktion.
- **getFunctionDescription:** Beschreibt eine benutzerdefinierte Funktion.
- **getFunctionPrototype:** Liefert den Prototyp für eine benutzerdefinierte Funktion.
- **pushdownFunction.** Generiert SQL-Code für die Pushdown-Optimierung.

Validierungsfunktion für benutzerdefinierte Funktionen

PowerCenter ruft validateFunction auf, um die Argumente in der benutzerdefinierten Funktion zu validieren. Die Funktion liefert Namen, Datentyp, Präzision und Größenordnung der Argumente in der benutzerdefinierten

Funktion. Außerdem nutzt PowerCenter diese Funktion, um den Datentyp des Rückgabewerts der benutzerdefinierten Funktion bereitzustellen.

PowerCenter ruft diese Funktion einmal für jede Instanz der benutzerdefinierten Funktion auf, die in einem Mapping oder Arbeitsablauf verwendet wird.

Verwenden Sie folgende Syntax:

```
INFA_EXPR_STATUS *(validateFunction)(IUNICHAR* sNamespace, IUNICHAR* sFuncName, IUINT32
numArgs, INFA_EXPR_OPD_METADATA** inputArgList, INFA_EXPR_OPD_METADATA* retValue);
```

Argument	Datentyp	Eingabe / Ausgabe	Beschreibung
sNamespace	IUNICHAR	Eingabe	Namespace der Funktion
sFuncName	IUNICHAR	Eingabe	Name der zu validierenden benutzerdefinierten Funktion
numArgs	IUINT32	Eingabe	Anzahl der Argumente in der benutzerdefinierten Funktion
inputArgList	INFA_EXPR_OPD_METADATA	Eingabe	Eingabeargumente der benutzerdefinierten Funktion
retValue	INFA_EXPR_OPD_METADATA	Ausgabe	Metadaten des Rückgabeports der benutzerdefinierten Funktion; Datentyp, Präzision und Größenordnung des Rückgabewerts müssen in diesem Argument festgelegt werden.

Rückgabe-Datentyp ist `INFA_EXPR_STATUS`. Verwenden Sie ISUCCESS und IFAILURE als Rückgabewert. Wenn die Funktion IFAILURE zurückgibt, zeigt PowerCenter eine Fehlermeldung an.

Beschreibungsfunktion für benutzerdefinierte Funktionen

PowerCenter ruft getFunctionDescription auf, um eine Beschreibung der benutzerdefinierten Funktion abzurufen. Mit destroyString löscht PowerCenter die Beschreibung aus dem Speicher, wenn sie nicht mehr benötigt wird.

Verwenden Sie folgende Syntax:

```
IUNICHAR* *(getFunctionDescription) (IUNICHAR* sNamespace, IUNICHAR* sFuncName);
```

Argument	Datentyp	Eingabe/ Ausgabe	Beschreibung
sNamespace	IUNICHAR	Eingabe	Namespace der Funktion
sFuncName	IUNICHAR	Eingabe	Name der benutzerdefinierten Funktion, die das Plug-In beschreiben soll;

Rückgabe-Datentyp ist IUNICHAR. Der Rückgabewert muss ein mit Null beendeter String sein.

Prototyp-Funktion für benutzerdefinierte Funktionen

PowerCenter ruft getFunctionPrototype auf, um die Argumente in der benutzerdefinierten Funktion im Ausdrucksseditor abzurufen. Mit destroyString löscht PowerCenter die Argumente aus dem Speicher, wenn sie nicht mehr benötigt werden.

Verwenden Sie folgende Syntax:

```
IUNICHAR* *(getFunctionPrototype) (INICHAR* sNamespace, IUNICHAR* sFuncName);
```

Argument	Datentyp	Eingabe/ Ausgabe	Beschreibung
sNamespace	IUNICHAR	Eingabe	Namespace der Funktion
sFuncName	IUNICHAR	Eingabe	Name der benutzerdefinierten Funktion, die das Plug-In beschreiben soll;

Rückgabe-Datentyp ist IUNICHAR. Der Rückgabewert muss ein mit Null beendeter String sein. Die Funktion gibt NULL zurück, wenn kein Wert für die Argumente gefunden wurde.

Benutzerdefinierte Funktion mit der Pushdown-Funktion

PowerCenter ruft pushdownFunction auf, um den SQL-Code für die Pushdown-Optimierung zu erstellen.

Verwenden Sie folgende Syntax:

```
INFA_EXPR_STATUS pushdownFunctionEcho(IUNICHAR* sNameSpace,
                    IUNICHAR* sFuncName,
                    IUINT32 numArgs,
                    INFA_EXPR_OPD_METADATA** inputArgList,
                    EDatabaseType dbType,
                    EPushdownMode pushdownMode,
                    IUNICHAR** sGenSql)
```

Argument	Datentyp	Eingabe/ Ausgabe	Beschreibung
sNameSpace	IUNICHAR	Eingabe	Namespace der Funktion
sFuncName	IUNICHAR	Eingabe	Name der zu validierenden benutzerdefinierten Funktion
numArgs	IUNINT32	Eingabe	Anzahl der Argumente in der benutzerdefinierten Funktion
inputArgList	INFA_EXPR_OPD_METADATA	Eingabe	Eingabeargumente der benutzerdefinierten Funktion
dbType	EDatabaseType	Eingabe	Datenbanktyp.
pushdownMode	EPushdownMode	Eingabe	Typ der Pushdown-Optimierung.
sGenSql	IUNICHAR	Ausgabe	Von der benutzerdefinierten Funktion erstellter SQL-Code.

Rückgabe-Datentyp ist `INFA_EXPR_STATUS`. Verwenden Sie ISUCCESS und IFAILURE als Rückgabewert. Wenn die Funktion IFAILURE zurückgibt, zeigt PowerCenter eine Fehlermeldung an.

INFA_EXPR_OPD_METADATA-Struktur

Diese Struktur definiert die Metadaten der Ausdrucksoperanden, einschließlich der an die Funktion übergebenen Argumente und des Rückgabetyps.

Die Struktur enthält die folgenden Metadaten:

- **Datentyp:** Datentyp des Arguments
- **Präzision:** Genauigkeit des Arguments
- **Größenordnung:** Größenordnung des Arguments
- **isValueConstant:** Gibt an, ob das Argument eine Konstante ist. Wenn ja, wird das Argument einmal für jeden Funktionsaufruf ausgewertet. Das Framework nutzt isValueConstant zur Leistungsoptimierung. Bei Eingabeargumenten, die Konstanten sind, kann das Plug-In die Argumentwerte während der Initialisierung der Funktionsinstanz abrufen und die Leistung dadurch optimieren. Für Ausgabewerte setzt das Plug-in isValueConstant auf TRUE.

Funktion zum Abrufen der Plug-In-Version

Diese Funktion definiert die Version des Plug-In. Es muss die gleiche Version aufweisen wie die API für benutzerdefinierte Funktionen (Version 1.0.0).

Verwenden Sie folgende Syntax:

```
INFA_EXPR_STATUS INFA_EXPR_GetPluginVersion(INFA_VERSION *sdkVersion, INFA_VERSION
*pluginVersion);
```

Argument	Datentyp	Eingabe/ Ausgabe	Beschreibung
sdkVersion	INFA_VERSION	Eingabe	Version der API für benutzerdefinierte Funktionen; verwenden Sie 1.0.0.
pluginVersion	INFA_VERSION	Ausgabe	Version des Plug-In, das Sie erstellen möchten;

Rückgabe-Datentyp ist INFA_EXPR_STATUS. Verwenden Sie ISUCCESS und IFAILURE als Rückgabewert. Wenn die Funktion IFAILURE ausgibt, schlägt die Sitzung bzw. der Arbeitsablauf fehl.

Funktion zum Löschen von Strings

Diese Funktion vernichtet alle Strings, die vom Plug-In zurückgegeben werden. Beispiel: Fehlermeldungen oder Rückgabewerte von anderen Funktionsaufrufen, z. B. getFunctionDescription, werden gelöscht.

Verwenden Sie folgende Syntax:

```
void *(DestroyString)(IUNICHAR* str);
```

Argument	Datentyp	Eingabe/ Ausgabe	Beschreibung
str	IUNICHAR	Eingabe	Der Eingabestring, der von dieser Funktion gelöscht wird;

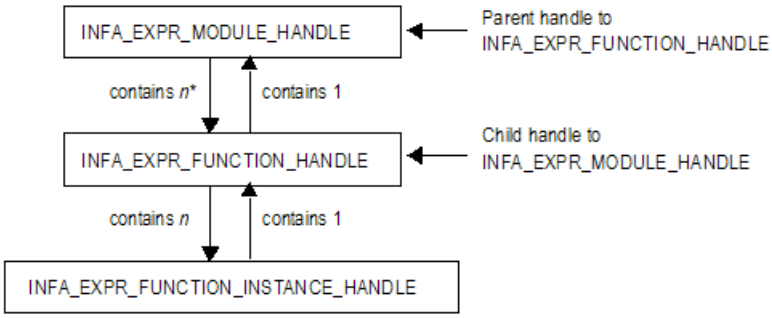
die Funktion gibt keinen Wert zurück.

Laufzeit-APIs

PowerCenter Integration Service ruft während einer Sitzung Laufzeit-APIs auf, um den Ausdruck auszuwerten, der die benutzerdefinierte Funktion enthält. Dabei wird das Plug-In auf Modul-, Funktions- und Funktionsinstanzebene initialisiert.

Jede Ebene enthält einen Satz von Funktionen. Diese Funktionen sind einem Handle zugeordnet, z. B. INFA_EXPR_MODULE_HANDLE. Der erste Parameter für diese Funktionen ist das Handle für betroffene Funktionen. Handles in der API für benutzerdefinierte Funktionen weisen untereinander hierarchische Beziehungen auf. Zwischen übergeordneten und untergeordneten Handles besteht eine 1:*n*-Beziehung.

Die folgende Abbildung zeigt die Handles in der API für benutzerdefinierte Funktionen:



Die folgende Tabelle beschreibt die Laufzeit-Handles:

Handle-Name	Beschreibung
INFA_EXPR_MODULE_HANDLE	Bezieht sich auf die gemeinsam genutzte Bibliothek oder DLL. Das Plug-In kann nur auf Modul-Handles in der eigenen gemeinsam genutzten Bibliothek oder DLL zugreifen. Es hat keinen Zugriff auf Modul-Handles in anderen gemeinsam genutzten Bibliotheken oder DLLs.
INFA_EXPR_FUNCTION_HANDLE	Bezieht sich auf eine benutzerdefinierte Funktion innerhalb der gemeinsam genutzten Bibliothek oder DLL.
INFA_EXPR_FUNCTION_INSTANCE_HANDLE	Bezieht sich auf eine bestimmte Instanz einer benutzerdefinierten Funktion.

Funktionen auf Modulebene

PowerCenter ruft Funktionen auf Modulebene einmal für jede gemeinsam genutzte Bibliothek oder DLL auf.

Funktion zum Abrufen von Schnittstellen (Modulebene)

Diese Funktion legt die Funktionszeiger für die Interaktion auf Modulebene fest.

Verwenden Sie folgende Syntax:

```
INFA_EXPR_STATUS INFA_EXPR_ModuleGetUserInterface(INFA_EXPR_LIB_METHODS* functions);
```

Argument	Datentyp	Eingabe / Ausgabe	Beschreibung
Funktionen	INFA_EXPR_LIB_METHODS	Ausgabe	Module rufen Schnittstellenfunktionen ab.

Rückgabe-Datentyp ist INFA_EXPR_STATUS. Verwenden Sie ISUCCESS und IFAILURE als Rückgabewert. Wenn die Funktion IFAILURE ausgibt, schlägt die Sitzung bzw. der Arbeitsablauf fehl.

Diese Funktion gibt die folgenden Funktionen zurück:

- **function_init:** Initialisiert die Funktion.
- **function_deinit:** Deinitialisiert die Funktion.

Initialisierungsfunktion (Modulebene)

PowerCenter ruft module_init einmal für jedes Modul auf, um alle globalen Datenstrukturen in der Funktion zu initialisieren. Der Aufruf dieser Funktion erfolgt vor dem Aufruf aller Funktionen auf Funktionsebene.

Verwenden Sie folgende Syntax:

```
INFA_EXPR_STATUS (*module_init) (INFA_EXPR_MODULE_HANDLE module);
```

Argument	Datentyp	Eingabe / Ausgabe	Beschreibung
Modul	INFA_EXPR_MODULE_HANDLE	Eingabe	Speichert die Daten, die auf Funktionsebene abgerufen werden können.

Rückgabe-Datentyp ist INFA_EXPR_STATUS. Verwenden Sie ISUCCESS und IFAILURE als Rückgabewert. Wenn die Funktion IFAILURE ausgibt, schlägt die Sitzung bzw. der Arbeitsablauf fehl.

Deinitialisierungsfunktion (Modulebene)

PowerCenter ruft module_deinit einmal für jedes Modul auf, um alle Datenstrukturen in der Funktion zu deinitialisieren. Der Aufruf dieser Funktion erfolgt nach Abschluss aller Interaktionen auf Funktionsebene.

Verwenden Sie folgende Syntax:

```
INFA_EXPR_STATUS (*module_deinit) (INFA_EXPR_MODULE_HANDLE module);
```

Argument	Datentyp	Eingabe / Ausgabe	Beschreibung
Modul	INFA_EXPR_MODULE_HANDLE	Eingabe	Handle auf Modulebene, das vom Framework an das Plug-In übergeben wird, wenn module_init aufgerufen wird;

Rückgabe-Datentyp ist INFA_EXPR_STATUS. Verwenden Sie ISUCCESS und IFAILURE als Rückgabewert. Wenn die Funktion IFAILURE ausgibt, schlägt die Sitzung bzw. der Arbeitsablauf fehl.

Funktionen auf Funktionsebene

PowerCenter ruft die Funktionen auf Funktionsebene einmal für jede benutzerdefinierte Funktion und einmal für jede gemeinsam genutzte Bibliothek oder DLL auf, die die Parameter für die benutzerdefinierte Funktion bereitstellen.

Funktion zum Abrufen von Schnittstellen (Funktionsebene)

Diese Funktion legt die Funktionszeiger für die Interaktion auf Funktionsebene fest. PowerCenter ruft diese Funktion einmal für jede benutzerdefinierte Funktion auf, die von dieser Bibliothek implementiert wird.

Verwenden Sie folgende Syntax:

```
INFA_EXPR_STATUS INFA_EXPR_FunctionGetUserInterface (IUNICHAR* nameSpaceName, IUNICHAR*
functionName, INFA_EXPR_FUNCTION_METHODS* functions);
```

Argument	Datentyp	Eingabe / Ausgabe	Beschreibung
nameSpaceName	IUNICHAR	Eingabe	Namespace der Funktion
functionName	IUNICHAR	Eingabe	Name der benutzerdefinierten Funktion, die das Plug-In beschreiben soll;
Funktion	INFA_EXPR_FUNCTION_METHODS	Eingabe	Platzhalter für die Funktionszeiger, die auf Funktionsinstanzebene aufgerufen werden;

Rückgabe-Datentyp ist INFA_EXPR_STATUS. Verwenden Sie ISUCCESS und IFAILURE als Rückgabewert. Wenn die Funktion IFAILURE ausgibt, schlägt die Sitzung bzw. der Arbeitsablauf fehl.

Diese Funktion gibt die folgenden Funktionen zurück:

- **function_init:** Initialisiert die Funktion.
- **function_deinit:** Deinitialisiert die Funktion.

Initialisierungsfunktion (Funktionsebene)

PowerCenter ruft function_init einmal für jede benutzerdefinierte Funktion auf, um alle mit der benutzerdefinierten Funktion verbundenen Strukturen zu initialisieren. Vor dieser Funktion wird die Initialisierungsfunktion auf Modulebene aufgerufen.

Verwenden Sie folgende Syntax:

```
INFA_EXPR_STATUS (*function_init) (INFA_EXPR_FUNCTION_HANDLE fnInstance);
```

Argument	Datentyp	Eingabe / Ausgabe	Beschreibung
fnInstance	INFA_EXPR_FUNCTION_HANDLE	Eingabe	Führt folgende Aufgaben aus: - Speichert die benutzerdefinierten Zeiger, damit sie während der Laufzeit oder bei der Deinitialisierung abgerufen werden können. - Initialisiert die Datenstrukturen für die Funktionsinstanzebene. - Wenn das Eingabeargument eine Konstante ist, ruft das Plug-In diesen konstanten Wert ab und führt alle erforderlichen Vorverarbeitungsschritte aus.

Rückgabe-Datentyp ist INFA_EXPR_STATUS. Verwenden Sie ISUCCESS und IFAILURE als Rückgabewert. Wenn die Funktion IFAILURE ausgibt, schlägt die Sitzung bzw. der Arbeitsablauf fehl.

Deinitialisierungsfunktion (Funktionsebene)

PowerCenter ruft diese Funktion einmal für jede Funktionsebene auf, um alle mit der benutzerdefinierten Funktion verbundenen Strukturen zu deinitialisieren.

Verwenden Sie folgende Syntax:

```
INFA_EXPR_STATUS (*function_deinit) (INFA_EXPR_FUNCTION_HANDLE function);
```

Argument	Datentyp	Eingabe / Ausgabe	Beschreibung
fnInstance	INFA_EXPR_FUNCTION_HANDLE	Eingabe	Handle auf Funktionsebene, das vom Framework an die Plug-Ins übergeben wird, wenn die Init-Funktion für die Funktionsinstanzebene aufgerufen wird;

Rückgabe-Datentyp ist INFA_EXPR_STATUS. Verwenden Sie ISUCCESS und IFAILURE als Rückgabewert. Wenn die Funktion IFAILURE ausgibt, schlägt die Sitzung bzw. der Arbeitsablauf fehl.

Funktionen auf Funktionsinstanzebene

PowerCenter ruft diese Funktionen bei jeder Verwendung einer benutzerdefinierten Funktion in einem Mapping oder Arbeitsablauf auf.

Funktion zum Abrufen von Schnittstellen (Funktionsebene)

Diese Funktion legt die Funktionszeiger für die Interaktion auf Funktionsebene fest. PowerCenter ruft diese Funktion einmal für jede benutzerdefinierte Funktion auf, die von dieser Bibliothek implementiert wird.

Verwenden Sie folgende Syntax:

```
INFA_EXPR_STATUS INFA_EXPR_FunctionInstanceGetUserInterface(IUNICHAR* functionName,
INFA_EXPR_FUNCTION_INSTANCE_METHODS* functions)
```

Argument	Datentyp	Eingabe / Ausgabe	Beschreibung
functionName	IUNICHAR	Eingabe	Namespace der Funktion
Funktionen	INFA_EXPR_FUNCTION_INSTANCE_METHODS	Eingabe	Platzhalter für die Funktionszeiger, die auf Funktionsinstanzebene aufgerufen werden;

Diese Funktion gibt die folgenden Funktionen zurück:

- **fnInstance_init:** Initialisiert eine Instanz einer benutzerdefinierten Funktion.
- **fnInstance_processRow:** Verarbeitet die Daten einer Instanz der benutzerdefinierten Funktion.
- **fnInstance_deinit:** Deinitialisiert eine Instanz einer benutzerdefinierten Funktion.

Initialisierungsfunktion (Funktionsinstanzebene)

PowerCenter ruft fnInstance_init einmal für jede Instanz einer benutzerdefinierten Funktion auf, um alle mit der Instanz verbundenen Strukturen zu initialisieren. Wenn in einem Mapping oder Arbeitsablauf zwei Instanzen einer benutzerdefinierten Funktion vorhanden sind, ruft PowerCenter diese Funktion zweimal auf. Vor dieser Funktion wird die Initialisierungsfunktion auf Modulebene aufgerufen.

Verwenden Sie folgende Syntax:

```
INFA_EXPR_STATUS (*fnInstance_init)(INFA_EXPR_FUNCTION_INSTANCE_HANDLE fnInstance);
```

Argument	Datentyp	Eingabe / Ausgabe	Beschreibung
fnInstance	INFA_EXPR_FUNCTION_HANDLE	Eingabe	Führt folgende Aufgaben aus: - Speichert die benutzerdefinierten Zeiger, damit sie während der Laufzeit oder bei der Deinitialisierung abgerufen werden können. - Initialisiert die Datenstrukturen für die Funktionsinstanzebene. - Wenn das Eingabeargument eine Konstante ist, ruft das Plug-In diesen konstanten Wert ab und führt alle erforderlichen Vorverarbeitungsschritte aus.

Rückgabe-Datentyp ist INFA_EXPR_STATUS. Verwenden Sie ISUCCESS und IFAILURE als Rückgabewert. Wenn die Funktion IFAILURE ausgibt, schlägt die Sitzung bzw. der Arbeitsablauf fehl.

Datenverarbeitungsfunktion (Funktionsinstanzebene)

PowerCenter ruft fnInstance_processRow auf, wenn für eine Instanz einer benutzerdefinierten Funktion eine Eingabezeile verfügbar ist. Die Daten für die Eingabeargumente der benutzerdefinierten Funktion sind gebunden; der Zugriff erfolgt über fnInstance-inputOPDHandles. Legen Sie Daten, Länge und Indikator für die

Ausgabe- und Rückgabeports in „fnInstance->retHandle“ fest. Vor dieser Funktion wird die Initialisierungsfunktion auf Funktionsebene aufgerufen.

Verwenden Sie folgende Syntax:

```
INFA_EXPR_ROWSTATUS (*fnInstance_processRow) (INFA_EXPR_FUNCTION_INSTANCE_HANDLE
fnInstance);
```

Argument	Datentyp	Eingabe / Ausgabe	Beschreibung
fnInstance	INFA_EXPR_FUNCTION_HANDLE	Eingabe	Handle auf Funktionsebene, für die Daten verfügbar sind;

Datentyp des Rückgabewerts lautet INFA_EXPR_ROWSTATUS. Verwenden Sie für den Rückgabewert folgende Werte:

- **INFA_ROWSUCCESS:** Gibt an, dass die Funktion die Datenzeile erfolgreich verarbeitet hat.
- **INFA_ROWERROR:** Gibt an, dass die Funktion in der Datenzeile einen Fehler gefunden hat. PowerCenter Integration Service erhöht den internen Fehlerzähler. Geben Sie diesen Wert nur dann zurück, wenn der Datenzugriffsmodus auf Zeile eingestellt ist.
- **INFA_FATALERROR:** Gibt an, dass die Funktion in der Datenzeile oder dem Datenblock einen schwerwiegenden Fehler gefunden hat. PowerCenter Integration Service lässt die Sitzung fehlschlagen.

Deinitialisierungsfunktion (Funktionsinstanzebene)

PowerCenter ruft fnInstance_deinit während der Deinitialisierung einmal für jede Funktionsebene auf. Diese Funktion kann aufgerufen werden, um alle mit der benutzerdefinierten Funktion verbundenen Strukturen zu deinitialisieren.

Verwenden Sie folgende Syntax:

```
INFA_EXPR_STATUS (*fnInstance_deinit)(INFA_EXPR_FUNCTION_INSTANCE_HANDLE fnInstance);
```

Argument	Datentyp	Eingabe / Ausgabe	Beschreibung
fnInstance	INFA_EXPR_FUNCTION_INSTANCE_HANDLE	Eingabe	Handle auf Funktionsebene, das vom Framework an die Plug-Ins übergeben wird, wenn die Initialisierungsfunktion für die Funktionsinstanzebene aufgerufen wird;

Rückgabe-Datentyp ist INFA_EXPR_STATUS. Verwenden Sie ISUCCESS und IFAILURE als Rückgabewert. Wenn die Funktion IFAILURE ausgibt, schlägt die Sitzung bzw. der Arbeitsablauf fehl.

INDEX

A

ABORT, Funktion
  Beschreibung 60
ABS, Funktion
  Beschreibung 61
Abschneiden
  Datumsangaben 224
  Zahlen 227
Absolute Werte
  Anfordern 61
ADD_TO_DATE, Funktion
  Beschreibung 62
Advanced Encryption Standard, Algorithmus
  Beschreibung 65
AES_DECRYPT, Funktion
  Beschreibung 65
AES_ENCRYPT, Funktion
  Beschreibung 65
Aggregatfunktionen
  ANY 66
  AVG 68
  Beschreibung 52
  COUNT 79
  FIRST 95
  LAST 119
  MAX (Datum) 132
  MAX (String) 135
  MAX (Zahlen) 133
  MEDIAN 137
  MIN (Datum) 143
  MIN (Zahlen) 144, 145
  Nullwerte 24, 54
  PERCENTILE 152
  STDDEV 196
  SUM 200
  VARIANCE 230
AND
  Reserviertes Wort 19
Anführungszeichen
  Einfügen einfacher Anführungszeichen mithilfe der Funktion CHR 16
Anhalten
  Sitzungen 60
ANY-Funktion
  Beschreibung 66
Arbeitsablaufsvariablen
  Beschreibung 13
  Integrierte Variablen 31
ASCII
  CHR, Funktion 72
  Konvertieren in Unicode-Werte 73
  Konvertieren von ASCII-Werten 72
  Konvertieren von Zeichen in ASCII-Werte 67
ASCII, Funktion
  Beschreibung 67
Ausdrücke
  Erstellen mit benutzerdefinierten Funktionen 254
  Hinzufügen von Kommentaren 18
  Konditional 23
  Mit Operatoren 26
  Syntax-Notation 15
  Übersicht 13
Ausdrucksseditor
  Verwenden mit benutzerdefinierten Funktionen 254
AVG, Funktion
  Beschreibung 68

B

Beispielfunktion
  ECHO 233
  SampleLoanPayment 233
Benutzerdefinierte Umwandlung
  Funktionen 255
Benutzerspezifische Funktionen
  Erstellen 234
  Erstellen einer Header-Datei 235
  Erstellen einer Implementierungsdatei 237
  Erstellen von Modulen 246
  Im Ausdrucksseditor 254
  Installieren 234, 253
  Übersicht 233
bigint
  Konvertieren von Werten 204

C

CEIL, Funktion
  Beschreibung 70
CHOOSE, Funktion
  Beschreibung 71
CHR, Funktion
  Beschreibung 72
  Einfügen einfacher Anführungszeichen 16, 72
CHRCODE, Funktion
  Beschreibung 73
COBOL-Syntax
  Konvertieren in Perl-Syntax 158
COMPRESS, Funktion
  Beschreibung 74
CONCAT, Funktion
  Beschreibung 75
  Einfügen einfacher Anführungszeichen 75
CONVERT_BASE, Funktion
  Beschreibung 76
COS, Funktion
  Beschreibung 77
COSH, Funktion
  Beschreibung 78

COUNT, Funktion
Beschreibung 79
CRC32, Funktion
Beschreibung 81
CREATE_TIMESTAMP_TZ, Funktion
Beschreibung 82
CUME, Funktion
Beschreibung 82

D

DATE_COMPARE, Funktion
Beschreibung 84
DATE_DIFF, Funktion
Beschreibung 85
Datenbereinigungsfunktionen
Beschreibung 55
GREATEST 102
IN 106
LEAST 122
Datenintegrationsdienst
Behandlung von Nullen in Vergleichsausdrücken 23
Datentypen
Date/Time 38
Datum/Zeit-Werte
Hinzufügen 62
Datumsangaben
Abschneiden 224
Durchführen von Berechnungen 51
Einfachdateien 41
Formatstrings 42
Funktionen 56
Jahr 2000 39
Julianisch 39
Konvertieren in Zeichenstrings 206
Modifiziert Julianisch 39
Relationale Datenbanken 41
Runden 170
Standardformat für Datum/Zeit 41
Übersicht 38
Datumsfunktionen
ADD_TO_DATE 62
DATE_COMPARE 84
DATE_DIFF 85
GET_DATE_PART 98
LAST_DAY 120
MAKE_DATE_TIME 131
MAX (Datum) 132
MIN (Datum) 143
ROUND 170
SET_DATE_PART 181
SYSTIMESTAMP 201
TRUNC (Datum) 224
DD_DELETE, Konstante
Beschreibung 20
Reserviertes Wort 19
Update-Strategie, Beispiel 20
DD_INSERT, Konstante
Beschreibung 21
Reserviertes Wort 19
Update-Strategie, Beispiel 21
DD_REJECT, Konstante
Beschreibung 21
Reserviertes Wort 19
Update-Strategie, Beispiel 21
DD_UPDATE, Konstante
Beschreibung 22
DD_UPDATE, Konstante *(Fortsetzung)*
Reserviertes Wort 19
Update-Strategie, Beispiel 22
DEC_BASE64, Funktion
Beschreibung 88
DECODE, Funktion
Beschreibung 89
Internationalisierung 14
DECOMPRESS, Funktion
Beschreibung 91
Dekodieren
DEC_BASE64, Funktion 88
Dezimalwerte
Konvertieren 82, 100, 101, 216, 218, 222
Division
Zurückgeben des Rests 147
DLL
Kompilieren für benutzerdefinierte Funktionen 246
Double-Präzision, Werte
Gleitkommazahlen 219
Durchschnittswerte
Aggregatfunktionen zur Ermittlung 68
Zurückgeben 148

E

:EXT, Referenzqualifikator
Beschreibung 16
Reserviertes Wort 19
ECHO, Beispielfunktion
Beschreibung 233
Einfachdateien
Datumsangaben 41
Einfache Anführungszeichen in String-Literalen
CHR, Funktion 72
Verwendung der Funktionen CHR und CONCAT 75
Elemente
FUNCTION 250
FUNCTION_GROUP 249
LIBRARY 251
PLUGIN 249
ENC_BASE64, Funktion
Beschreibung 92
Entschlüsselung
AES_DECRYPT, Funktion 65
ERROR, Funktion
Beschreibung 93
Standardwert 93
Erstellen
Benutzerspezifische Funktionen 234
Header-Datei für benutzerdefinierte Funktionen 235
Implementierungsdatei für benutzerdefinierte Funktionen 237
Module für benutzerdefinierte Funktionen 246
EXP, Funktion
Beschreibung 94
Exponentwerte
Berechnen 94
Zurückgeben 155

F

FALSE, Konstante
Beschreibung 23
Reserviertes Wort 19
Filterbedingungen
Aggregatfunktionen 54

Filterbedingungen *(Fortsetzung)*
Nullwerte 24
Filterumwandlung
Mit ISNULL-Funktion 113
Finanzfunktionen
Beschreibung 57
FV, Funktion 97
NPER, Funktion 151
PMT, Funktion 154
PV, Funktion 156
RATE, Funktion 157
FIRST, Funktion
Beschreibung 95
FLOOR, Funktion
Beschreibung 96
FLOOR, Funktion (Ausdrücke)
Beschreibung 96
format
Von Datum in Zeichenstring 206
Von Zeichenstring in Datum 213
Formatstrings
Datumsangaben 42
Definition 38
IS_DATE, Funktion 47
Julianisches Datum 44, 47
Modifiziertes Julianisches Datum 44, 47
TO_CHAR, Funktion 44
TO_DATE, Funktion 47
Übereinstimmung 49
FUNCTION_GROUP, Element
Beschreibung 249
FUNCTION, Element
Beschreibung 250
Funktionen
Aggregat 52
Beschreibung 13
Datenbereinigung 55
Datum 56
Finanz 57
Internationalisierung 14
Kategorien 52
Kodierung 57
Konvertierung 55
Numerisch 58
Spezial 58
string 59
Test 59
variable 59
Wissenschaftlich 58
Zeichen 54
FV, Funktion
Beschreibung 97

G

Ganzzahlen
Konvertieren von Werten 220
Gemeinsam genutzte Bibliotheken
Kompilieren für benutzerdefinierte Funktionen 246
GET_DATE_PART, Funktion
Beschreibung 98
GET_TIMESTAMP, Funktion
Beschreibung 101
GET_TIMEZONE, Funktion
Beschreibung 100
GREATEST, Funktion
Beschreibung 102
Gregorianischer Kalender
In Datumsfunktionen 39
Groß-/Kleinschreibung
Konvertieren in Großbuchstaben 229
Großschreibung
Strings 109, 128, 229

H

Header-Datei
Erstellen 235
Hohe Präzision
ABS 61
ABS, Funktion 61
AVG 68
AVG, Funktion 68
CEIL 70
CREATE_TIMESTAMP_TZ, Funktion 82
CUME 82
CUME, Funktion 82
EXP 94
GET_TIMESTAMP, Funktion 101
GET_TIMEZONE, Funktion 100
LOG 125
Mathematische Operatoren 27
MAX (Zahlen) 133
MAX, Funktion 133
MEDIAN 137
MEDIAN, Funktionen 137
MIN (Zahlen) 144
MIN, Funktion 144
MOD 147
MOVINGAVG 148
MOVINGAVG, Funktion 148
MOVINGSUM 150
MOVINGSUM, Funktion 150
PERCENTILE 152
PERCENTILE, Funktion 152
POWER 155
ROUND (Zahlen) 174
ROUND, Funktion 174
SIGN 190
SIN 190
STDDEV, Funktion 196
SUM 200
SUM, Funktion 200
TO_DECIMAL, Funktion 216
TO_DECIMAL38, Funktion 218
TO_TIMESTAMP_TZ, Funktion 222
TRUNC, Funktion 227
Hyperbolisch
Kosinus-Funktion 78
Sinus-Funktion 191
Tangens-Funktion 203

I

IIF, Funktion
Beschreibung 103
Internationalisierung 14
Implementierungsdatei
Erstellen 237
IN, Funktion
Beschreibung 106
INDEXOF, Funktion
Beschreibung 108

:INFA, Referenzqualifikator
Reserviertes Wort 19
INITCAP, Funktion
Beschreibung 109
Internationalisierung 14
Installieren
Benutzerspezifische Funktionen 234, 253
INSTR, Funktion
Beschreibung 110
Integrierte Variablen
Beschreibung 31
Internationalisierung
Betroffene Funktionen 14
Sortierreihenfolge 14
Ungültiger Ausdruck 14
IS_DATE, Funktion
Beschreibung 114
Formatstrings 47
IS_NUMBER, Funktion
Beschreibung 116
IS_SPACES, Funktion
Beschreibung 118
ISNULL, Funktion
Beschreibung 113

J

J-Formatstring
Verwendung mit IS_DATE 50
Verwendung mit TO_CHAR 46
Verwendung mit TO_DATE 50
Jahr 2000
Datumsangaben 39
Julianisches Datum
Formatstring 44, 47
In Datumsfunktionen 39

K

Kalender
Unterstützte Datumstypen 39
Kodierung
ENC_BASE64, Funktion 92
Zeichen 139, 192
Kodierungsfunktionen
AES_DECRYPT 65
AES_ENCRYPT 65
Beschreibung 57
COMPRESS 74
CRC32 81
DEC_BASE64 88
DECOMPRESS 91
ENC_BASE64 92
MD5 136
Kommentare
Hinzufügen zu Ausdrücken 18
Kompilieren
Module für benutzerdefinierte Funktionen 246
Komponenten der Umwandlungssprache
Übersicht 13
Kompression
Dekomprimieren von Daten 91
Komprimieren von Daten 74
Konstanten
Beschreibung 13
DD_INSERT 21
Konstanten *(Fortsetzung)*
DD_REJECT 21
DD_UPDATE 22
FALSE 23
NULL 23
TRUE 25
Konvertieren
Datumstrings 40
Konvertierungsfunktionen
Beschreibung 55
CREATE_TIMESTAMP_TZ 82
GET_TIMESTAMP 101
GET_TIMEZONE 100
TO_CHAR (Datum) 206
TO_CHAR (Zahlen) 211
TO_DATE 213
TO_DECIMAL 216
TO_DECIMAL38 218
TO_FLOAT 219
TO_INTEGER 220
TO_TIMESTAMP_TZ 222
Kosinus
Berechnen 77
Berechnen des hyperbolischen Kosinus 78

L

:LKP, Referenzqualifikator
Beschreibung 16
Reserviertes Wort 19
LAST_DAY, Funktion
Beschreibung 120
LAST, Funktion
Beschreibung 119
Laufender Kontostand
Zurückgeben 82
LEAST, Funktion
Beschreibung 122
Leere Strings
Testen 123
Leerzeichen
Entfernen mit DD_REJECT 22
Vermeiden in Zeilen 118
LENGTH, Funktion
Beschreibung 123
Leere Strings, Test 123
LIBRARY, Element
Beschreibung 251
Literale
Einfache Anführungszeichen 72, 75
Einfache Anführungszeichen, Anforderung 16
LN, Funktion
Beschreibung 124
LOG, Funktion
Beschreibung 125
Logarithmus
Zurückgeben 124, 125
Logische Operatoren
Beschreibung 29
Lokale Variablen
Beschreibung 13
LOOKUP, Funktion
Beschreibung 126
LOWER, Funktion
Beschreibung 128
Internationalisierung 14

LPAD, Funktion
Beschreibung 128
LTRIM, Funktion
Beschreibung 130

M

MAKE_DATE_TIME, Funktion
Beschreibung 131
Mapping-Parameter
Definition 13
Mapping-Variablen
Beschreibung 13
Integrierte Variablen 31
Mathematisch
Datum/Zeit-Werte 51
Mathematische Operatoren
Beschreibung 27
Verwendung von Strings in Ausdrücken 27
Zum Konvertieren von Daten 27
MAX (Datum), Funktion
Beschreibung 132
Internationalisierung 14
MAX (String), Funktion
Beschreibung 135
MAX (Zahlen), Funktion
Beschreibung 133
Internationalisierung 14
:MCR, Referenzqualifikator
Reserviertes Wort 19
MD5, Funktion
Beschreibung 136
MEDIAN, Funktionen
Beschreibung 137
Mehrere Suchvorgänge
Beispiel für Konstante TRUE 25
METAPHONE
Beschreibung 139
MIN (Datum), Funktion
Beschreibung 143
Internationalisierung 14
MIN (Zahlen), Funktion
Beschreibung 144, 145
Internationalisierung 14
Minimal
Wert, Rückgabe 143
MOD, Funktion
Beschreibung 147
Modifiziertes Julianisches Datum
Formatstring 44, 47
Module
Erstellen für benutzerdefinierte Funktionen 246
Monat
Zurückgeben des letzten Tages 120
MOVINGAVG, Funktion
Beschreibung 148
MOVINGSUM, Funktion
Beschreibung 150

N

Namespaces
Auswählen 250
Negative Werte
SIGN 190
NOT
Reserviertes Wort 19
NPER, Funktion
Beschreibung 151
NULL, Konstante
Beschreibung 23
Reserviertes Wort 19
Nullwerte
Aggregatfunktionen 24, 54
Filterbedingungen 24
In Vergleichsausdrücken 23
ISNULL 113
Logische Operatoren 30
Operatoren 25
String-Operator 28
Suchen 113
Numerische Funktionen
ABS 61
Beschreibung 58
CEIL 70
CONVERT_BASE 76
CUME 82
EXP 94
FLOOR 96
LN 124
LOG 125
MOD 147
MOVINGAVG 148
MOVINGSUM 150
POWER 155
RAND 157
ROUND (Zahlen) 174
SIGN 190
SQRT 195
TRUNC (Zahlen) 227
Numerische Werte
Konvertieren in Textstrings 211
Rückgabe des hyperbolischen Kosinus 78
Rückgabe des Kosinus 77
Rückgabe von absoluten Werten 61
SIGN 190
Zurückgeben der Quadratwurzel 195
Zurückgeben der Standardabweichung 196
Zurückgeben des hyperbolischen Sinus 191
Zurückgeben des hyperbolischen Tangens 203
Zurückgeben des Mindestwerts 144
Zurückgeben des Sinus 190
Zurückgeben des Tangens 203
Zurückgeben von Logarithmen 124, 125

O

Operatoren
Beschreibung 13
Logische Operatoren 29
Mathematisch 27
Nullwerte 25
String-Operatoren 28
Vergleichsoperatoren 29
Verwendung von Strings in mathematischen Berechnungen 27
Verwendung von Strings in Vergleichen 29
OR
Reserviertes Wort 19

P

$PMFolderName
  Beschreibung 34
$PMIntegrationServiceName
  Beschreibung 34
$PMMappingName
  description 34
$PMRepositoryServiceName
  description 34
$PMRepositoryUserName
  description 34
$PMSessionName
  description 34
$PMSessionRunMode
  description 35
$PMSourceName@TableName
  Beschreibung 33
$PMTargetName@TableName
  Beschreibung 33
$PMWorkflowName
  description 35
$PMWorkflowRunId
  description 35
$PMWorkflowRunInstanceName
  description 35
PERCENTILE, Funktion
  Beschreibung 152
Perl-kompatible Syntax für reguläre Ausdrücke
  In REG_EXTRACT-Funktion 158
  In REG_MATCH-Funktion 158
Plug-In-XML-Datei
  FUNCTION_GROUP, Element 249
  FUNCTION, Element 250
  LIBRARY, Element 251
  PLUGIN, Element 249
PLUGIN, Element
  Beschreibung 249
PMT, Funktion
  Beschreibung 154
Ports
  Syntax-Notation 16
Positive Werte
  SIGN 190
POWER, Funktion
  Beschreibung 155
PowerCenter-Integrationsdienst
  Behandlung von Nullen in Vergleichsausdrücken 23
Primärschlüssel, Einschränkung
  Nullwerte 23
PROC_RESULT, Variable
  Reserviertes Wort 19
PV, Funktion
  Beschreibung 156

Q

Quadratwurzel
  Zurückgeben 195

R

RAND, Funktion
  Beschreibung 157
Rangordnung von Operatoren
  Ausdrücke 26
RATE, Funktion
  Beschreibung 157
Referenzqualifikatoren
  Beschreibung 16
REG_EXTRACT, Funktion
  Beschreibung 158
  Mit Perl-Syntax 158
REG_MATCH, Funktion
  Beschreibung 161
  Mit Perl-Syntax 158
REG_REPLACE, Funktion
  Beschreibung 162
Registrieren
  Repository-Plug-In 254
Relationale Datenbanken
  Datumsangaben 41
REPLACECHR, Funktion
  Beschreibung 163
REPLACESTR, Funktion
  Beschreibung 166
Repository-ID-Attribute
  Abrufen 234
Repository-Plug-In
  Abrufen der Repository-ID-Attribute 234
  Registrieren 254
Reservierte Wörter
  list 19
REVERSE, Funktion
  Beschreibung 169
ROUND (Datum), Funktion
  Beschreibung 170
  Verarbeiten von Subsekunden 170
ROUND (Zahlen), Funktion
  Beschreibung 174
RPAD, Funktion
  Beschreibung 177
RR-Formatstring
  Beschreibung 40
  Unterschied zwischen YY und RR 40
  Verwendung mit IS_DATE 50
  Verwendung mit TO_CHAR 46
  Verwendung mit TO_DATE 50
RTRIM, Funktion
  Beschreibung 178
Rückgabewerte
  Beschreibung 13
  Syntax-Notation 16
Runden
  Datumsangaben 170
  Zahlen 174

S

:SEQ, Referenzqualifikator
  Beschreibung 16
  Reserviertes Wort 19
:SD, Referenzqualifikator
  Beschreibung 16
  Reserviertes Wort 19
$$$SessStartTime
  Verwendung in Ausdrücken 36
:SP, Referenzqualifikator
  Beschreibung 16
  Reserviertes Wort 19
SampleLoanPayment, Beispielfunktion
  Beschreibung 233

SESSSTARTTIME, Variable
Beschreibung 35
Reserviertes Wort 19
Verwendung in Datumsfunktionen 51
SET_DATE_PART, Funktion
Beschreibung 181
SETCOUNTVARIABLE, Funktion
Beschreibung 179
SETMAXVARIABLE, Funktion
Beschreibung 184
SETMINVARIABLE, Funktion
Beschreibung 186
SETVARIABLE, Funktion
Beschreibung 188
SIGN, Funktion
Beschreibung 190
SIN, Funktion
Beschreibung 190
SINH, Funktion
Beschreibung 191
Sinus
Zurückgeben 190, 191
Sitzungen
Anhalten 60
Sortierreihenfolge
Internationalisierung 14
SOUNDEX, Funktion
Beschreibung 192
Spezialfunktionen
ABORT 60
Beschreibung 58
DECODE 89
ERROR 93
IIF 103
LOOKUP 126
SPOUTPUT
Reserviertes Wort 19
SQL IS_CHAR, Funktion
Mit REG_MATCH 161
SQL LIKE, Funktion
Mit REG_MATCH 161
SQL_LIKE, Funktion
Beschreibung 194
SQL-Syntax
Konvertieren in Perl-Syntax 158
SQRT, Funktion
Beschreibung 195
SSSSS-Formatstring
Verwendung mit IS_DATE 51
Verwendung mit TO_CHAR 46
Verwendung mit TO_DATE 51
Standardabweichung
Zurückgeben 196
Standardformat für Datum/Zeit
Einstellen 41
Standardwerte
ERROR, Funktion 93
STDDEV, Funktion
Beschreibung 196
String-Konvertierung
Datumsangaben 40
String-Literale
Einfache Anführungszeichen 72, 75
Einfache Anführungszeichen, Anforderung 16
String-Operatoren
Beschreibung 28
Stringfunktionen
Beschreibung 59
Stringfunktionen *(Fortsetzung)*
CHOOSE 71
INDEXOF 108
REVERSE 169
Strings
Ändern der Länge 177
Anzahl von Zeichen 123
Entfernen von Leerzeichen 130
Entfernen von Leerzeichen und Zeichen 178
Entfernen von Zeichen 130
Ersetzen eines einzelnen Zeichens 163
Ersetzen mehrerer Zeichen 166
Großschreibung 109, 128, 229
Hinzufügen von Leerzeichen 128
Hinzufügen von Zeichen 128
Konvertieren numerischer Werte in Textstrings 211
Konvertieren von Datumsangaben in Zeichen 206
Konvertieren von Zeichenstrings in Datumsangaben 213
Verketten 28, 75
Zeichensatz 110
Zurückgeben eines Teils 198
Stringwerte
Zurückgeben des Mindestwerts 145
Zurückgeben von maximalen Werten 135
Subsekunden
Verarbeiten in ROUND (Datum) 170
Verarbeiten in TRUNC (Datum) 224
SUBSTR, Funktion
Beschreibung 198
SUM, Funktion
Beschreibung 200
Summe
Zurückgeben 150, 200
Syntax-Notation
Allgemeine Regeln 17
expression 15
Ports 16
Rückgabewerte 16
SYSDATE, Variable
Beschreibung 36
Reserviertes Wort 19
Verwendung in Ausdrücken 36
Systemvariablen 31
SYSTIMESTAMP, Funktion
Beschreibung 201

T

:TD, Referenzqualifikator
Beschreibung 16
Reserviertes Wort 19
TAN, Funktion
Beschreibung 203
Tangens
Zurückgeben 203
TANH, Funktion
Beschreibung 203
TC_COMMIT_AFTER, Variable
Beschreibung 37
TC_COMMIT_BEFORE, Variable
Beschreibung 37
TC_CONTINUE_TRANSACTION, Variable
Beschreibung 37
TC_ROLLBACK_BEFORE, Variable
Beschreibung 37
Testfunktionen
Beschreibung 59

Testfunktionen *(Fortsetzung)*
IS_DATE 114
IS_NUMBER 116
IS_SPACES 118
ISNULL 113
Textstrings
Konvertieren numerischer Werte 211
TO__TIMESTAMP_TZ, Funktion
Beschreibung 222
TO_CHAR (Datum), Funktion
Beispiele 46
Beschreibung 206
Formatstrings 44
TO_CHAR (Zahlen), Funktion
Beschreibung 211
TO_DATE, Funktion
Beispiele 50
Beschreibung 213
Formatstrings 47
TO_DECIMAL, Funktion
Beschreibung 216
TO_DECIMAL38, Funktion
Beschreibung 218
TO_FLOAT, Funktion
Beschreibung 219
TO_INTEGER, Funktion
Beschreibung 220
Transaktionssteuerungsvariablen
Beschreibung 37
TRUE, Konstante
Beschreibung 25
Reserviertes Wort 19
TRUNC (Datum), Funktion
Beschreibung 224, 227
Verarbeiten von Subsekunden 224

U

Überspringen
Zeilen 93
Umwandlungsausdrücke
Null, Einschränkungen 23
Übersicht 13
Umwandlungssprache
Operatoren 26
Reservierte Wörter 19
Vergleich mit SQL 15
Unicode
Konvertieren in ASCII-Werte 73
Konvertieren von Unicode-Werten 72
Konvertieren von Zeichen in Unicode-Werte 67
UNIX
Kompilieren gemeinsam genutzter Bibliotheken für benutzerdefinierte Funktionen 248
Update-Strategie
DD_DELETE, Beispiel 20
DD_INSERT, Beispiel 21
DD_REJECT, Beispiel 21
DD_UPDATE, Beispiel 22
Updates der Umwandlungssprache
Boolesche Ausdrücke 23
Vergleichsausdrücke 23
UPPER, Funktion
Beschreibung 229
Internationalisierung 14
UUID_UNPARSE-Funktion
Beschreibung 230
UUID4-Funktion
Beschreibung 230

V

Variablen
$PMFolderName 34
$PMIntegrationServiceName 34
$PMMappingName 34
$PMRepositoryServiceName 34
$PMRepositoryUserName 34
$PMSessionName 34
$PMSessionRunMode 35
$PMSourceName@TableName 33
$PMTargetName@TableName 33
$PMWorkflowName 35
$PMWorkflowRunId 35
$PMWorkflowRunInstanceName 35
Integrierte Variablen 31
SESSSTARTTIME 35
SYSDATE 36
TC_COMMIT_AFTER 37
TC_COMMIT_BEFORE 37
TC_CONTINUE_TRANSACTION 37
TC_ROLLBACK_BEFORE 37
Transaktionssteuerungsvariablen 37
WORKFLOWSTARTTIME 36
Variablenfunktionen
Beschreibung 59
Mit mehreren Partitionen 59
SETCOUNTVARIABLE 179
SETMAXVARIABLE 184
SETMINVARIABLE 186
SETVARIABLE 188
VARIANCE, Funktion
Beschreibung 230
Vergleichsoperatoren
Beschreibung 29
Verwendung von Strings in Ausdrücken 29
Verketten
Strings 28, 75
Verschachtelte Ausdrücke
Operatoren 26
Verschlüsselung
AES_ENCRYPT, Funktion 65
Mit dem AES-Algorithmus 65

W

Windows-Betriebssystem
Kompilieren einer DLL für benutzerdefinierte Funktionen 247
Wissenschaftliche Funktionen
Beschreibung 58
COS 77
COSH 78
SIN 190
SINH 191
TAN 203
TANH 203
WORKFLOWSTARTTIME, Variable
Beschreibung 36
Reserviertes Wort 19
Verwendung in Ausdrücken 36

Y

YY-Formatstring
Unterschied zwischen RR und YY 40
Verwendung mit IS_DATE 50
Verwendung mit TO_CHAR 47
Verwendung mit TO_DATE 50

Z

Zahlen
Abschneiden 227
Runden 174
Zeichen
ASCII-Zeichen 67, 72
Entfernen von Strings 130, 178
Ersetzen einzelner Zeichen 163
Ersetzen mehrerer Zeichen 166
Großschreibung 109, 128, 229
Hinzufügen zu Strings 128, 177
Kodierung 139, 192
Unicode-Zeichen 67, 72, 73
Zählen 198
Zurückgeben der Anzahl 123
Zeichenfunktionen
ASCII 67
CHR 72
CHRCODE 73
CONCAT, Funktion 75
INITCAP 109
Zeichenfunktionen *(Fortsetzung)*
INSTR 110
LENGTH 123
Liste 54
LOWER 128
LPAD 128
LTRIM 130
METAPHONE 139
REG_EXTRACT 158
REG_MATCH 161
REG_REPLACE 162
REPLACECHR 163
REPLACESTR 166
RPAD 177
RTRIM 178
SOUNDEX 192
SUBSTR 198
UPPER 229
Zeichenstrings
Konvertieren in Datumsangaben 213
Konvertieren von Datumsangaben 206
Zeilen
Beliebige Zeile wird zurückgegeben 66
Laufender Kontostand 82
Überspringen 93
Vermeiden von Leerzeichen 118
Zählen 79
Zurückgeben der ersten Zeile 95
Zurückgeben der letzten Zeile 119
Zurückgeben der Summe 150
Zurückgeben des Durchschnitts 148